AAR5227
Trab
ITC2008
U6

ING. LUIS G. URIBE C.

Trabajo de Ascenso, Universidad Católica Andrés Bello

Ingeniería Digital: Aspectos Técnicos

ING. LUIS GUILLERMO URIBE CATAÑO

Aspectos Técnicos de la Ingeniería Digital

© 2008 Ing. Luis G. Uribe C.
Universidad Católica Andrés Bello
Caracas, Venezuela

Contenido

PRÓLOGO 6

➢ **ABSTRACT** 6

Un nombre como telón de fondo 6

➢ **INTRODUCCIÓN** 7

A QUÉ SE ENFRENTA UN INGENIERO DEL AREA DE DIGITALES 7

Observaciones sobre algunos términos empleados 10

➢ **PLAN DEL DOCUMENTO** 11

HISTORIA 13

➢ **INTRODUCCIÓN** 13

HISTORIA DE LA COMPUTACIÓN 14

¿CUÁL FUE EL PRIMER PROGRAMA QUE SE ENSAMBLÓ? 14

FALLA EN TOMÓGRAFOS AXIALES COMPUTARIZADOS EN CARACAS, 1980 15

Qué hacer cuando se interrumpe una operación 15

Qué hacer en casos de rearranque 15

Las desviaciones 17

LAS FALLAS DEL MARS CLIMATE ORBITER Y DEL MARS POLAR LANDER, 1999 17

LA FALLA DEL COHETE EUROPEO ARIANE 5 17

Introducción 17

Cadena Inversa de Eventos Técnicos 18

Comentarios Sobre el Escenario de la Falla 19

Los Procedimientos de Prueba y Calificación 21

Conclusiones 24

FALLAS EN NUESTROS MODELOS MENTALES 25

➢ **BIBLIOGRAFÍA** 28

PROYECTOS I 29

➢ **INTRODUCCIÓN** 29

➢ **ASPECTOS GENERALES** 29

PRODUCTO, MERCADO, ADMINISTRACIÓN 29

THINK BIG... OR SMALL? 30

➢ **INTERROGANTES PARA UNA INVESTIGACIÓN** 33

➢ **CÓMO SE PLANIFICA UN PROYECTO** 34

ASPECTOS PREVIOS 34

DISEÑO CONCEPTUAL 34

CRONOGRAMA (TENTATIVO) DE TRABAJO 35

ENMIENDAS AL CRONOGRAMA DE TRABAJO 35

DISEÑO *FUNCIONAL*: HARDWARE Y SOFTWARE 35

DESCRIPCIÓN 35

ENMIENDAS AL DISEÑO FUNCIONAL 35
DISEÑO *FUNCIONAL*: HARDWARE 36
DISEÑO *FUNCIONAL*: SOFTWARE 36
DISEÑO DE *DETALLE*: HARDWARE Y SOFTWARE 37
DESCRIPCIÓN 38
DISEÑO DE DETALLE DE HARDWARE 38
DISEÑO DE DETALLE DE SOFTWARE 38
Ambiente de Desarrollo 39
VARIOS 39
PRUEBAS DE HARDWARE Y SOFTWARE: PROTOTIPO 40
PROTOTIPO 40
PRUEBAS DE SOFTWARE 40
PRUEBAS DE HARDWARE 40
PRUEBAS DE SISTEMA 40
➢ DOCUMENTACIÓN DEL SISTEMA41
A NIVEL DE PRODUCTO: 41
A NIVEL DE FABRICACIÓN: 41
A NIVEL DE CADA EQUIPO, FICHA TÉCNICA IDENTIFICANDO: 42
➢ INSTALACIÓN Y GARANTÍA 42
➢ RESGUARDO DEL PRODUCTO 42
RESPALDO 43
➢ DETALLE (CHECK-LIST √) 44
➢ COMUNICACIÓN 58
➢ FLUJO DE CAJA: ZERO CASHFLOW 58
➢ FAST PROTOTYPING 59
➢ QUE FUNCIONE 60
➢ KISS 61
➢ TEMAS VARIOS QUE SON "PURO ORO" 61
CONCEPTOS BÁSICOS DE ELECTRÓNICA.61
ARMADO DE CIRCUITOS. 61
BLINDAJE Y CABLEADO 62
EXPERIMENTACIÓN 62
QUÉ CHIPS USAR 63
DOS LINKS FABULOSOS 63
"C" 64
PROGRAMAR LA FLASH 68
INTEN, INTDSB; TRAZAS 68
A FAVOR DE "C" 69
"LA PESADILLA ¿HARDWARE O SOFTWARE?" 70
PREMISA DE URIBE 71
"PERDER LOS ARCHIVOS" 71
PROYECTOS II 73
➢ DOCUMENTACIÓN 73
IDEAS PARA MANUALES DEL PROYECTO "ESTACIÓN METEOROLÓGICA DE ALTAS PRESTACIONES" 73
MANUAL DE USO E INSTALACIÓN DEL PROTOTIPO 73
MANUAL TÉCNICO 74
EXPLICACIÓN 74
CONCLUSIÓN 74
BIBLIOGRAFÍA CONSULTADA Y SITIOS WEB EN LOS QUE SE SOPORTÓ 74
ANEXOS 74
EXTRAS 75
➢ SCADA 76
INTRODUCCIÓN 76
SCADA 76
CONSTITUCIÓN 76
ESPECIFICACIONES DEL PROYECTO 77
ESTACIÓN MAESTRA 77

SUBSISTEMA DE ADQUISICIÓN DE DATOS 77

SUBSISTEMA DE PRESENTACIÓN DE INFORMACIÓN 78

GENERACIÓN Y PROCESAMIENTO DE ALARMAS 78

CONTROL REMOTO 78

COMUNICACIÓN CON LAS RTU DEL CAMPO 79

RTU 79

ADQUISICIÓN DE DATOS ANALÓGICOS (AI) 80

ADQUISICIÓN DE DATOS DIGITALES (DI) 80

CONTROL (DO) 80

COMUNICACIÓN CON LA ESTACIÓN MAESTRA 81

ANTEPROYECTO 81

- **DATA LOGGER 83**

TÍTULO DEL PROYECTO 83

CONTRATANTE DEL PROYECTO 83

OBJETIVOS DEL PROYECTO 83

DESCRIPCIÓN DE LOS EQUIPOS A SER DESARROLLADOS 84

CARACTERÍSTICAS FUNCIONALES MÍNIMAS DEL SISTEMA 84

SUBSISTEMA DE APLICACIÓN FINAL 86

ASPECTOS EXTRAS QUE DEBEN CONSIDERAR 86

- **FAT, SAT: FORMULARIOS DE EJEMPL087**

FAT: DOCUMENTO DE ACEPTACIÓN EN FÁBRICA (PARA EL "DATA LOGGER") 87

CARACTERÍSTICAS GENERALES 87

ESPECIFICACIONES PARTICULARES 89

SAT: DOCUMENTO DE ACEPTACIÓN EN SITIO (PARA EL "DATA LOGGER") 90

CARACTERÍSTICAS GENERALES 90

ESPECIFICACIONES PARTICULARES 90

DOCUMENTACIÓN DEL SISTEMA91

- **BIBLIOGRAFÍA 92**

CIRCUITOS DIGITALES 93

- **INTRODUCCIÓN 93**
- **REALIDAD Y ELECTRÓNICA ANALÓGICA 94**

¿POR QUÉ EL NOMBRE DE "ANALÓGICO"?94

EJEMPLO DE TRANSDUCCIÓN 94

- **ELECTRÓNICA DIGITAL: CARACTERÍSTICAS IMPORTANTES 95**

1) REPRESENTACIÓN NUMÉRICA 95

2) VARIABLES DISCRETAS, INMUNIDAD AL RUIDO 95

3) EXACTITUD, PRECISIÓN Y RESOLUCIÓN 96

4) CONTROL DE ERRORES: DETECCIÓN Y CORRECCIÓN 97

5) ALMACENAMIENTO DE INFORMACIÓN98

6) PROCESAMIENTO DE LA VARIABLE TIEMPO 98

7) ESTANDARIZACIÓN 98

- **CONVERSIÓN ENTRE BASES 99**

1) CONVERSIÓN DE DECIMAL A BINARIO O A HEXADECIMAL 99

MÉTODO CONVENCIONAL 99
MÉTODO RECOMENDADO 99
2) CONVERSIÓN DE BINARIO A DECIMAL 100
MÉTODO CONVENCIONAL 100
MÉTODO RECOMENDADO 100
3) CONVERSIÓN DE HEXADECIMAL A DECIMAL 101
MÉTODO CONVENCIONAL 101
MÉTODO RECOMENDADO 101
4) CONVERSIÓN DE OCTAL A DECIMAL 101
➢ LAB0 102
PROTOBOARD 102
CIRCUITOS INTEGRADOS Y ALAMBRES 103
FUENTES DE ALIMENTACIÓN 104
INTERRUPTORES 105
FAN OUT 105
ENTRADAS NO USADAS 106
CUIDADOS CON LOS CMOS 106
CONDENSADORES DE DESACOPLE 107
EQUIPO Y SALIDAS 108
QUÉ COSAS VAN A FALLAR EN SUS PROYECTOS DE LABORATORIO!!! 108
1) FAN OUT 108
2) MULTIPLICACIÓN (AND) POR EL PULSO DE RELOJ 109
3) ESTADO INICIAL 110
4) SEÑALES NO SINCRONIZADAS 110
5) SEÑALES MÁS RÁPIDAS QUE EL PULSO DE RELOJ 110
6) FALTA DE SOPORTE PARA "DEBUGGING" 110
7) SWITCHES CON REBOTE 110
8) LED. CARGAS NO TTL 111
9) ALIMENTACIÓN DEL PROTOBOARD 111
10) SOLDADURA!!! 111
11) CONDENSADORES DE DESACOPLE 111
12) DIAGRAMAS 112
13) IMPROVISACIÓN E INGENIO... Vs. ¡ MÉTODO ! 113
14) MAL USO DE LOS INSTRUMENTOS 114
15) LA JUSTIFICACIÓN 115
➢ HÁGASE LA LUZ 117
1) RECONECTADOR 117
2) COMPORTAMIENTO ASINCRÓNICO 118
3) "ARRANCADOR" (STARTER) 118
4) CONVERTIR FLIP-FLOPS CON "CLEAR", EN BIESTABLES CON "PRESET" 119
➢ EJERCICIOS 120
MOTIVACIÓN 120
REDES COMBINATORIAS 120
1) DISPOSITIVO DE ALARMA PARA LAS CÁPSULAS APOLO 120
2) LA SOCIEDAD Y LOS ACCIONISTAS 122
3) CONTROL AUTOMÁTICO DE VELOCIDAD 123
4) FRENAR LOS COHETES 124
5) MONTACARGAS 125

6) CALDERA 126

7) CALCULADORA BRAILLE 126

REDES SECUENCIALES 127

1) JUEGO DE PING-PONG (BASIC) [Solución] 127

2) CERRADURA DE COMBINACIÓN 129

3) UN ASCENSOR INTERESANTE129

4) REPRESENTACIÓN NUMÉRICA129

- **BIBLIOGRAFÍA 130**

PROGRAMACIÓN GENERAL 131

- **INTRODUCCIÓN 131**
- **DÍGITOS DE VERIFICACIÓN 131**
- **PARECE SIMPLE 137**
- **ROMANOS Y ARÁBIGOS 141**
- **PI 143**
- **POLÍTICA Y MECANISMOS 144**
- **REALLY SIMPLY HASH 161**
- **SASSERT 163**
- **ROUNDS UP TO A POWER OF 2 164**
- **PROGRAMACIÓN POR REGLAS172**
- **ANTI REBOTE (Debouncing) 179**
- **DEBOUNCING Y MÁQUINAS DE ESTADO 180**
- **DEBOUNCING DE VARIOS BITS183**
- **SECUENCIADOR DE EVENTOS, SOE 184**
- **BIBLIOGRAFÍA 190**

MICROCONTROLADORES 191

- **INTRODUCCIÓN 191**
- **CONDITION CODE REGISTER: C, Z, N y V 193**

COMPARAR ES RESTAR 193

UNSIGNED 194

SIGNED 197

- **COMPARACIONES PARA EL PIC200**
- **LIBRERÍA DE TIMERS PARA HC08 207**
- **PRUEBA LIBRERÍA DE TIMERS210**
- **EJERCICIOS ADICIONALES 212**

PLANTEAMENTO INICIAL 212

1) EJERCICIOS DE REEMPLAZO 212

2) MANEJO DE UNA IMPRESORA GPIB213

LEVEL 1 215

SOME GENERAL PIC INIT ROUTINES.. 215

INITPRINTER() 215

MORE GENERIC PIC INIT ROUTINES.. 215

PRINTLINE("FELIZ AÑO\N") 215

3) EL JUEGO DE LA VIDA [Solución] 215

PLANTEAMIENTO 215

SOLUCIÓN (Programa life2.c) 216

- **BIBLIOGRAFÍA 230**

PRÓLOGO

Aspectos prácticos que complementan los conocimientos de Ingeniería Digital.

ABSTRACT

SI usted trabaja en el campo de los *circuitos digitales*, los *microcomputadores* (incluidos como componentes), los *sistemas operativos*, las *telecomunicaciones de datos* y similares, estas notas pueden resultarle de utilidad, pues su propósito es el de ir más allá de los conocimientos impartidos en los planes de estudio, que por lo común tratan estos temas sólo a nivel de sistemas y omiten referirse en detalle al hardware, al software, a los algoritmos y a los problemas prácticos.

Este material incursiona en las áreas ya mencionadas pero, desde luego, presupone en el lector el conocimiento básico de dichos temas, adquirido con la formación académica convencional.

UN NOMBRE COMO TELÓN DE FONDO

Al comenzar mi actividad fuera del "campus" universitario, después de 15 años a dedicación completa como alumno de electrónica y profesor de "digitales", encontré un pequeño libro, ***"Technical Aspects of Data Communication"***, por John E. McNamara, de cuya lectura obtuve enormes beneficios. Mis conocimientos en esa área eran a nivel de "sistemas" (ecuaciones y bloques). Cuando yo hablaba de línea de comunicación, el libro me enseñaba lazos de corriente; las marcas y espacios, como se llamaban desde Morse, ahora eran –12 y +12 voltios. Los cables pasaron de ser simples líneas a tener longitudes limitantes, ¡y acotaban las velocidades de transmisión! Las conexiones seguían normas y estándares nominados; los puntos de "tierra" fijaban sus posiciones, y no era igual conectarlos en un lugar que en otro. El sistema asíncrono de mi viejo teleimpresor se materializó en los UART; la paridad se vistió de polinomio y los CRC serpenteaban entre registros de desplazamiento y compuertas ExOr. Los protocolos dejaron sus vestiduras tabulares y mostraron desnudas sus secuencias de bits. Hasta el confort logrado cuando las centrales telefónicas eliminaron el "retorno del sonido", hubo que aprender a deshabilitarlo con "supresores del eliminador de eco"; claro, a las personas les convenía porque solo

parlamentaba una de las dos, mientras la otra escuchaba, ¡ pero las computadoras insistían en comunicarse al mismo tiempo!

Estas notas, emulando un poco aquella referencia tan útil, ofrecen aspectos técnicos que complementan los conocimientos teóricos de los cursos regulares y sus libros de texto. Se incluyen temas provocativos, ejercicios y problemas, resueltos y por resolver, en las áreas de la Ingeniería Digital. Se cubre el aspecto técnico, pero también ciertas facetas gerenciales relacionadas con los proyectos.

No hay forma tan eficaz como Aprender Haciendo (ah!), pero ese método no está exento de problemas. Por ejemplo, para aplicarlo al estudio de los circuitos digitales, la programación de microcontroladores o los sistemas operativos, se requiere manejar con precisión una enorme cantidad de detalles. Esa es una diferencia entre el enfoque del aula, y el del laboratorio; y entre comprender el meollo de los conceptos, y ostentar la destreza del Maestro. Estas notas pretenden mostrar una pequeña pero importante fracción de esos detalles, que se necesitan para la praxis, y que no se consiguen con facilidad ni en claustros ni bibliotecas. Desde luego, siendo ilimitado el ámbito de lo que desconocemos, este aporte resultará siempre insuficiente, con independencia de su extensión.

INTRODUCCIÓN

A QUÉ SE ENFRENTA UN INGENIERO DEL AREA DE DIGITALES

La siguiente categorización es tomada de mi experiencia profesional –que en su mayoría ha sido en el área de Desarrollo de Proyectos– y pretende plasmar el cúmulo de actividades que vi enfrentar a los ingenieros especializados en Sistemas Digitales. El propósito es el de ayudar a ubicar al estudiante y al joven profesional en lo que pueden ser líneas de acción del egresado con esta especialización.

Qué hace un Ingeniero con mención en Sistemas Digitales...

- Detecta **necesidades** concretas,
- **Solubles** en el ámbito de la especialidad de Sistemas Digitales
- Elabora especificaciones de **compra** de:
  - ✓ partes y
  - ✓ componentes electrónicos
- Obtiene especificaciones (electrónicas) de **diseño**, contemplando factores tan disímiles como:
  - ✓ **ubicación** y alineación de antenas,
  - ✓ normas y **estándares** aplicables,
  - ✓ **seguridad** intrínseca,
  - ✓ consideraciones de **consumo** de energía y de generación de calor,
  - ✓ tipos de **gabinetes** (housing),
  - ✓ diseño para una **vida útil** determinada...
  - ✓ de **equipos** realizables bajo capacidades disponibles de procesamiento **limitadas**
- Produce especificaciones **funcionales** de:
  - ✓ sistemas,
  - ✓ equipos y
  - ✓ subsistemas electrónicos
- **Diseña** circuitos, y programas asociados a la manipulación del hardware (firmware)
- Elabora **prototipos**, maquetas y modelos

- **Prueba**, verifica y valida, circuitos y sus programas asociados, pertenecientes a subsistemas electrónicos
- Especifica, para la **producción**:
  - ✓ módulos,
  - ✓ ensamblaje,
  - ✓ cableado,
  - ✓ gabinetes,
  - ✓ interconexión entre susbsistemas,
  - ✓ interfaces eléctricas,
  - ✓ **protocolos** de intercambio de información
- Genera **protocolos** para pruebas en fábrica (**FAT**), y en el sitio de instalación (**SAT**)
- **Realiza pruebas** en fábrica y en sitio, para garantizar parámetros tales como la disponibilidad, y demás consideraciones de diseño:
  - ✓ rendimiento,
  - ✓ escalabilidad,
  - ✓ **extremos** ambientales,
- **Instala** equipos
- Determina la logística de **transporte** y **almacenaje** de subsistemas y equipos
- Lleva a cabo de **pruebas de campo**
- **Redacta**:
  - ✓ Documentación,
  - ✓ Manuales técnicos,
  - ✓ Manuales de operación,
  - ✓ Manuales de usuario,
  - ✓ "Brochures"
- Presta la **garantía** de equipos y sistemas
- **Resguarda** apropiadamente el producto: Respalda documentos, programas, ampara los ambientes de desarrollo
- **Lidera Ingenieros** y demás personal técnico
- **Entrena** y ejerce la docencia, tanto "in house" como para entidades externas
- Lleva y entiende procesos contables y administrativos relacionados con la **Gerencia de Proyectos**
- Elabora **planes** detallados y precisos para el diseño de hardware y software
- Genera rutinas de autodiagnóstico y diseña **dispositivos a prueba de fallas**
- Aplica metodologías apropiadas para el diseño de **circuitos impresos**
- Se vincula al proceso de realimentación, desarrollo y administración de **versiones** mejoradas del producto
- Evalúa desastres y catástrofes; hace análisis **post mortem**
- Realiza **mantenimiento** en general; y define las políticas de reparación, la estrategia y su logística
- Maneja **proveedores** principales y suplidores alternos
- **Integra** sistemas
- Calibra y **afiere** instrumentos
- Ejecuta la prueba y **validación** de normas
- Hace **consultoría**
- Determina en forma concreta y objetiva las necesidades de mano de obra para los proyectos
- Es preciso en la estimación de cuánto tiempo van a tomar las diversas fases de un diseño o proyecto
- Garantiza la **escalabilidad** del producto
- Administra y **gerencia proyectos**
- Asume la **Gerencia Técnica**
- Aprovecha las redes y el **Internet** para la interconexión moderna de subsistemas

En algunos casos, personal técnico de nivel inferior al de Ingeniero puede abocarse a algunas de estas actividades, que deberán en todo caso ser supervisadas. Este solapamiento de responsabilidades es común en todas las profesiones, y es de grado, no de competencia. Hay otras zonas de interacción entre carreras, como la Ingeniería Eléctrica y la Electrónica, o las que surgen con el Ingeniero en Computación, o con el de Instrumentación, y hasta con el Administrador y el Economista.

El Ingeniero en Sistemas Digitales es un profesional capacitado para:

- Analizar,
- Especificar
  - ✓ para una ubicación,
  - ✓ según normas y estándares,
  - ✓ bajo restricciones de consumo,
    - generación de calor,
    - producción de ruido,
    - confiabilidad,
    - rendimiento

sistemas, subsistemas y equipos, con la clase apropiada de gabinete (housing), y para una vida útil especificada.

En cuanto a los equipos con los que se relaciona, el Ingeniero con mención en Sistemas Digitales puede:

- Configurarlos,
- Componerlos,
- Diseñarlos,
- Normalizarlos,
- Investigar sobre ellos,
- Hacerlos evolucionar,
- Operarlos,
- Planear,
- Programarlos,
- Mantenerlos,
- Adaptarlos,
- Producirlos,
- Interconectarlos,
- Repararlos,
- Documentarlos e
- Instalarlos;
- Debe generar protocolos para pruebas en fábrica y en sitio;
- Verificar la disponibilidad y demás consideraciones de diseño
  - ✓ rendimiento,
  - ✓ escalabilidad,
  - ✓ extremos ambientales;
- Transportarlos,
- Almacenarlos;
- Administrar el grupo de Ingenieros y demás personal técnico,
- Dar entrenamiento tanto "in house" como externamente,
- Debe elaborar planes para el diseño de hardware y software;
- Circuitos impresos;
- Evaluar mejoras al producto;
- Hacer análisis post mortem,
- Mantenimiento preventivo,
- Mantenimiento correctivo;
- Manejar proveedores;
- Calibrar y Aferir equipos e instrumentos;
- Demostrar y validar el cumplimiento de normas;
- Hacer Consultorías;
- Calibrar tiempos,
- Costos y recursos de los proyectos;
- Llevar una Gerencia técnica y
- Aprovechar el Internet para la interconexión de subsistemas.

Quedan muchos aspectos por tocar, como la Venta Técnica, y el contacto con otras disciplinas como la electrónica, de la cual se nutre; la electromedicina, la instrumentación, el control industrial, las telecomunicaciones, la computación, y todas aquellas áreas a las cuales la especialización en Digitales les sirve de pilar y sustento.

Lo más deseable sería que para lograr un título de Ingeniero con especialización en esta área, se cubriera la mayor cantidad de los elementos expuestos en este perfil. En todo caso, los primeros años de trabajo deben servir para complementar las deficiencias que pueda manifestar el profesional.

OBSERVACIONES SOBRE ALGUNOS TÉRMINOS EMPLEADOS

Siendo el inglés la *lingua mater* de la investigación electrónica, resulta en ocasiones difícil elaborar en esa materia un texto en castellano que no incurra en anglicismos, y aquellos que lo hacen pueden terminar en un galimatías. Recuerdo algún impreso que traducía "Master Clear" como Limpieza Maestra, o "buffer" como Memorias Tampón, y "flip-flops" por básculas.

Incluyo algunas definiciones tomadas del diccionario de la Real Academia Española en línea, para ayudar a explicar el uso que en este texto se les da:

hardware.

(Voz ingl.).

1. m. *Inform.* Conjunto de los componentes que integran la parte material de una computadora.

software.

(Voz ingl.).

1. m. *Inform.* Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

técnico, ca.

5. f. Conjunto de **procedimientos y recursos** de que se sirve una ciencia o un arte.

Real Academia Española © Todos los derechos reservados

PLAN DEL DOCUMENTO

Nuestra presentación se encuentra distribuida a lo largo de cinco capítulos. En el **primero** se hace una introducción histórica en la que se reseñan algunos de los accidentes más notables relacionados con la Ingeniería Digital, resaltando las causas que los ocasionaron, y se describe un mecanismo perverso que afecta nuestros propios modelos mentales. Disfruté enormemente el análisis del impecable documento técnico: "Report by the Inquiry Board", sobre las fallas del vuelo 501 del Ariane 5, y el estudio sobre el accidente aeronáutico de Kegworth, que resalta el fenómeno que yo llamo: "Contra-evidencia", enemigo número uno del método científico y yo espero que la lectura de todos los casos sirva para abrir nuestras mentes y mantenernos alertas.

El **segundo** capítulo enfoca el tema de los Proyectos, en general, motivando una metodología para la planificación de proyectos de Ingeniería Digital, aspectos prácticos como el financiamiento de los mismos, y unas interminables listas de detalles que es necesario no olvidar cuando se va a presentar un proyecto. La aproximación ha servido como guía en mis cursos de Laboratorio de Proyectos II y III; el notable detalle de mi presentación surgió como respuesta a un grupo de estudiantes que, al ser conminados a discriminar las distintas etapas y facetas de un diseño, no llegaron ni a seis. Cuando los forcé a imaginarse al menos cien ítems dijeron que eso era imposible, y el grupo que hizo el mayor esfuerzo no pasó de 25. Por eso me senté y lo desglosé en cien (100) pasos, advirtiéndoles que proyectos serios pueden tener centenares y hasta miles de actividades. El propósito fundamental es establecer una metodología que permita, entre otras cosas, dar a conocer el nivel de progreso al que se ha llegado en el desarrollo, lo que es fundamental para establecer pautas de corrección, tanto en tiempos como en el uso del dinero y demás recursos involucrados. Si un proyecto no se ve como un proceso realimentado, resultará una casualidad si se llega a cumplir con algunos de los objetivos: Tiempo de entrega, funcionalidades desarrolladas, costos presupuestados, calidad del sistema desarrollado.

El **tercer** capítulo hace una introducción a los circuitos digitales, presentando sus características más resaltantes, en el entendido de que muchos de nuestros estudiantes ni siquiera saben el por qué del nombre de los circuitos **analógicos**, ni diferencian muy bien entre conceptos básicos como resolución, precisión y exactitud. Asimismo, hay un apartado denominado **Laboratorio Cero**, en el que se señalan aspectos elementales qué considerar para la elaboración de prácticas y proyectos de circuitos digitales, el uso del protoboard, condensadores de desacople, planos de tierra y similares. Sería un avance notable si lográramos que nuestros estudiantes se hicieran su propia Lista de Verificación, o Check-List, como las que son de obligatorio cumplimiento en profesiones tales como las de Piloto Aeronáutico o Marítimo, en las que no se puede dejar a la memoria la verificación y constatación de las condiciones de los equipos y el seguimiento de los procedimientos requeridos para la seguridad de sus misiones.

El **cuarto** capítulo trata de provocar al lector con aspectos modernos, a veces no muy bien entendidos, de la programación, como la aplicación de reglas, "static asserts", representación apropiada de Máquinas de Estado, filtraje en entradas digitales (eliminadores de rebote; secuenciadores de eventos), sistemas de verificación para entrada manual de información, vía teclado, y diferenciación entre políticas y mecanismos en la concepción moderna de la programación.

El **quinto** capítulo sirve como complemento a los cursos de programación de microcontroladores, con una serie de ejercicios propuestos y algunos resueltos, como el Juego de la Vida, para la familia PIC 16Fxxx, y otros para el Freescale (antes Motorola) HC08.

Se ha dejado de lado para una futura entrega, la presentación de algunos temas de la parte más importante del estudio de los microcomputadores: La Entrada y Salida, como soporte a los cursos de arquitectura, y que incluirían conceptos y detalles del manejo de Colas e Interrupciones, enmarcados dentro del campo de las comunicaciones seriales en los PC y también referidos específicamente a los microcontroladores PIC y Freescale, pero el tamaño actual del documento hace necesaria esta limitación.

Es de esperar, por lo que aquí se prologa, que este documento sea de utilidad como referencia a los interesados en simplificar sus vidas, que buena falta nos hace, y de complemento a la literatura cotidiana en la materia.

El autor, que ha tenido la fortuna de haber transitado durante 39 años el camino del desarrollo tecnológico, y ha recorriendo desde el relé y el tubo de vacío hasta el Pentium y el Internet, ofrece con el mayor placer esta pequeña contribución.

Ing. Luis G. Uribe C.

Caracas, julio de 2008

Capítulo

1

HISTORIA

"Quien no conoce la historia está condenado a repetirla", Jorge Santayana.

INTRODUCCIÓN

ESTAS pequeñas referencias históricas se incluyen aquí porque relatan acontecimientos de los que hay que extraer importantes conclusiones para incorporarlas a nuestra actividad profesional cotidiana. El método científico, sustento del impresionante desarrollo tecnológico actual, es un procedimiento iterativo que se basa en corregir errores: "*trial and error*"; en general, se observan los hecho pertinentes, se presenta una hipótesis que los explique; se predicen sucesos futuros, que ocurrirían de ser ciertas nuestras suposiciones; se diseñan pruebas experimentales para comprobar los vaticinios que, si se producen, le dan solidez al postulado y lo promueven, categorizándolo como teoría o ley natural; en caso de encontrar desviaciones en los eventos predichos, se rechaza o se modifica la anterior hipótesis, se genera una nueva y se repite el ciclo. Los procedimientos y resultados de las pruebas deben ser públicos, al menos entre la comunidad científica, para permitir la verificación de investigadores independientes.

En un lapso tan amplio es muy probable que ya se haya escrito todo lo que puede decirse como introducción al tema. Sin embargo, es importantísimo y resulta siempre conveniente, poner en **Perspectiva** la Ingeniería Digital para quienes van a estudiarla; tal es el propósito de la siguiente presentación, escrita como ambientación hacia nuestros lectores y alumnos que se adentran en el torbellino actual de las Técnicas Digitales.

HISTORIA DE LA COMPUTACIÓN

¿CUÁL FUE EL PRIMER PROGRAMA QUE SE ENSAMBLÓ?

Se había logrado ya cierto avance desde los iniciales mastodontes (Eniac [Von Neumann], Mark I [Harvard]); ya la programación no se hacía empleando "patch-cords", sino que un código pequeño, el "Bootstrap", llevaba a memoria uno más completo (el Loader, Absolute or Relocating), que se encargaba de leer de los programas, sus imágenes binarias de memoria, y almacenarlas en la computadora (se seguía codificando en binario, pero desde Von Neumann ya no se hacía con cables...).

Grace Murray Hopper, mujer de gran inteligencia, Almirante de la armada norteamericana, fue pionera entre las programadoras y la descubridora del primer "bug", una polilla que quedó aprisionada en medio de los contactos de un relé; ella hizo el primer "debugging" al extraer la mariposa con una pinza de depilar, para pegarla en la bitácora de la computadora.

Grace analizó el procedimiento que se empleaba para codificar un problema, que consistía en escribir en papel el programa: Símbolos que representaban instrucciones, y los que identificaban localidades de memoria; luego, en forma manual se buscaban las equivalencias con los valores binarios que la máquina entendía. La científica pensó básicamente en hacer un programa que automatizara ese proceso de traducción. Ella hizo el código, ***en binario***, para leer tarjetas perforadas con los símbolos y generar un módulo objeto, en binario.

Cuando ya su ensamblador, escrito en binario, a mano, estaba funcionando, lo primero que hizo, muy sagazmente, fue ejecutarlo para convertir las fuentes simbólicas de su propio código, ¡ **a binario** !, lo que produjo una copia exacta del programa que estaba haciendo la traducción. Ahora, ya tenía cómo generar automáticamente el binario del Ensamblador, a partir de las fuentes, así que las posteriores modificaciones y agregados que le hizo a su programa **las realizó en Simbólico** (en Assembler, como se llamaría luego), y con el ensamblador anterior obtuvo el siguiente y el siguiente, en una cadena que aún no se detiene. **Ya nunca más se escribió un programa en binario (excepto por necedad)**.

¡ En otras palabras, el primer programa que se "ensambló" fue el mismo ensamblador !

De ahí en adelante el procedimiento se ha repetido siempre: Uno de los objetivos de cada compilador nuevo es procesarse a sí mismo. Interesante... Así, el primer C, por ejemplo, se codificó en Assembler, y su trabajo inicial fue ese, a partir de las fuentes de C, escritas en C. De esta manera, cuando hubo que corregirlo o agregarle características al lenguaje, la actividad ya no se hizo en Assembler, sino en C. El compilador viejo de C crea la versión nueva, corregida y aumentada. El Perl se compila en Perl...

FALLA EN TOMÓGRAFOS AXIALES COMPUTARIZADOS EN CARACAS, 1980

En 1978 abandoné la comodidad de 15 años de vida universitaria y asumí la gerencia técnica de la compañía que representaba, para Venezuela, a DEC (Digital Equipment Corporation), segunda empresa de importancia en el mundo en materia de computadoras digitales, después de IBM. El gerente técnico de uno de mis clientes, exalumno mío de la USB, con el que me relacionaba pues le prestábamos servicio de mantenimiento a las computadoras de sus tomógrafos, me narró un percance ocurrido a una paciente a la que le hicieron un estudio en una clínica caraqueña. Lo cuento tal como lo recuerdo.

La unidad que hacía las radiaciones y recababa la información para estructurar el **TAC** estaba gobernada en esa época por una minicomputadora PDP-11/34, que recibía instrucciones del técnico radiólogo a través de un teclado similar al de los actuales PC. El operador introdujo los datos específicos para esa exploración y un par de órdenes complementarias, y el tomógrafo comenzó su proceso de excitación y lectura. En un momento el radiólogo vaciló y decidió detener el examen, para lo cual canceló la secuencia oprimiendo la combinación de teclas "***Control-C***", lo que era común en esa época pre-PC. El software estaba escrito para "abortar" toda actividad ulterior y rearrancar si recibía la señal de interrupción, por lo que se posicionó otra vez en su estado inicial pero *sin desenergizar el irradiador*; así, se sometió a la paciente a dosis de radiación por encima de los niveles aceptables.

Varias cosas pueden aprenderse de aquí:

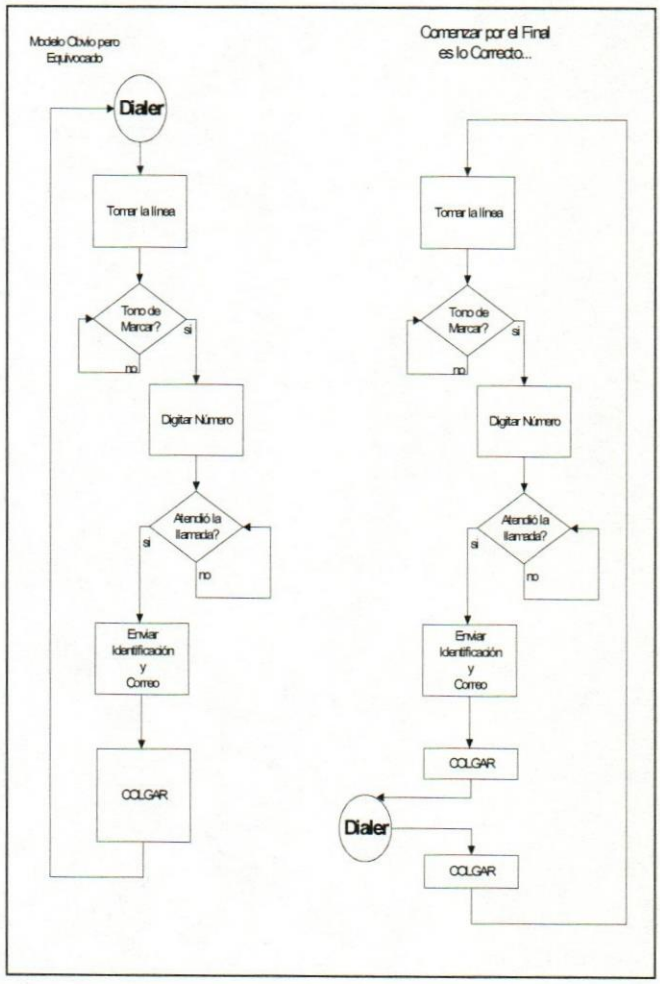
Qué hacer cuando se interrumpe una operación

Un secuenciador, de equipos que interactúan con procesos como el descrito (u otros industriales; ejemplo, el control de una grúa en una factoría), no puede simplemente retornar a la posición de recibir comandos si se lo instruye para terminar anticipadamente su actividad; hay que garantizar que todos los *elementos* bajo su control queden en la posición inicial correcta, la cual tiene que ser estudiada y establecida con absoluta precisión y escrupulosa meticulosidad para cada uno de ellos.

Qué hacer en casos de rearranque

Esta condición hay que extenderla a cualquier otra clase de posibles reinicio, por anomalías internas o exteriores. Ejemplo de una falla interior sería cuando se produce un "overlfow" en alguna operación, o una división por cero (esta última eventualidad hace que DOS y Windows terminen [aborten] un proceso, con un mensaje en la pantalla; ese comportamiento no es tolerable, como ya se explicó, y el programador tiene que tomar todas las previsiones que garanticen que deja las unidades bajo su control, en ese estado inicial del que se habló). Ejemplo de un error externo sería una falla de energía eléctrica; en tal caso, al rearrancar, el controlador debe asegurarse de dejar todo de acuerdo al estado inicial. Si, por ejemplo, su sistema está trasladando la grúa hacia un extremo de la fábrica, y una intermitencia en la electricidad "resetea" su microcontrolador, su programa no puede simplemente quedarse esperando órdenes del operador, porque el motor continuará trasladando su carga y ocasionará un accidente (como tumbar la pared del extremo y destrozar la fábrica, dependiendo de la instalación. *Cfr. switches de fin de carrera* en un capítulo posterior)

En una situación bien diferente que aprovecho para comentar, aparecen los mismos inconvenientes. Casi todo estudiante que tiene que resolver el problema de enviar de manera automatizada, desde su dispositivo microcontrolado, un correo electrónico con información destinada a un servidor, plantea la solución mediante esta secuencia de actividades: » comandar al sistema (modem) para que tome la línea ("levantar el auricular"); » aguardar el tono de marcar; » digitar el número telefónico del servidor; » verificar que se atienda la llamada; » intercambiar la identificación y la clave de acceso; » enviar el correo; » esperar la confirmación; » terminar la transacción ("colgar el teléfono")..., **¡ y está mal !**



Ese es el mismo orden que seguía también el "*dialer*" de Win95/98; yo lo usaba hacia 1995 y esa era su secuencia. Sin embargo, a veces se distinguía, por el sonido, que el modem encontraba dificultades al realizar la comunicación; siendo que a mi equipo le llevaba minutos enterarse, yo interrumpía a mano el proceso mediante un punto sensible en la pantalla, "*Cancel*". Ahora, al tratar de repetir la llamada, el "*dialer*" comenzaba, inmutable, la misma cadencia: » levantar el auricular; » esperar el tono de marcar... pero resulta que al cancelar la secuencia anterior, el driver simplemente salía de memoria, *y dejaba el modem esperando el resto de las acciones* (que contestaran, etc). Así, al reintentar tomar la línea, lo que mi PC encontraba no era un tono de marcar, sino la señal de ocupado... y ya no podía establecerse, **nunca más**, ninguna llamada ***¡ hasta reiniciarar el PC .!*** En una versión, bastante posterior, Microsoft corrigió el problema haciendo que **lo primero que ejecuta su "dialer" cuando uno lo invoca es COLGAR EL TELÉFONO**, que era exactamente el último paso de la secuencia original. Por eso dijo Blaise Pascal, al terminar de fabricar su calculadora mecánica, llena de engranajes:

"La última cosa que se aprende al armar algo, es qué colocar de primero"

Pasos similares deben darse, por ejemplo, en el controlador de la grúa industrial: Al encenderlo o arrancarlo, hay que comandar los motores a fin de que se detengan. Para el tomógrafo, si el sistema comenzara *apagando* el radiador (y detuviera el mecanismo que mueve la cama, o que hace girar el cilindro, etc.) la paciente hubiera resultado ilesa a pesar de tan sensible falla de programación.

Suena como si un error corrigiera otro. ¡ Y a lo mejor ! La ley de *Murphy sobre la Incertidumbre* dice:

"Usted puede saber que algo ha fallado, solo cuando comete un número impar de equivocaciones".

Las desviaciones

Una observación recurrente a lo largo de esta obra es: Cuando a un programador se le da una especificación, probablemente tenga perfectamente definida la serie normal de pasos de la operación (como en la llamada telefónica). Y es casi seguro que esa secuencia estará probada (es impensable que usted no verifique, al menos, ¡ lo que se le pidió !), así que su programa funcionará... ¡ siempre que no haya desviaciones ! Allí, en las circunstancias excepcionales al comportamiento normal, es donde **el ingeniero debe lucirse**: ¿Qué pasa si la llamada se termina? ¿Y cuando el ISP no contesta?, ¿Si el microcontrolador se muere y la grúa se aproxima a 30 km/h a la pared de la fábrica? Todas estas preguntas, y muchas más, tienen que encontrar una **acción apropiada en su diseño**; ese es precisamente su trabajo.

LAS FALLAS DEL MARS CLIMATE ORBITER Y DEL MARS POLAR LANDER, 1999

Estas dos naves formaron parte del Mars Surveyor '98 Program; ambas se destruyeron, en septiembre y diciembre del 99, debido a un error de navegación ocasionado porque la famosísima compañía **Lockheed Martin**, subcontratista de la NASA, usó en sus programas ***unidades imperiales*** (pound-seconds [millas]) **y el resto de los participantes empleó el sistema métrico decimal (newton-seconds** [kilómetros]**)**. El Mars Climate Orbiter, en particular, tenía que entrar en órbita a 150 km por encima de Marte, pero confundiendo millas con kilómetros a toda lo largo de su travesía, acumuló una diferencia en el cálculo de la distancia que lo hizo comenzar a maniobrar cuando ya estaba muy cerca, a solo 57 km. La atmósfera marciana lo quemó.

¿Cómo puede pasar un error tan grosero en instituciones tan avanzadas?

LA FALLA DEL COHETE EUROPEO ARIANE 5

"Las suposiciones son la madre de los grandes desaciertos" Refranero popular.

Esta descripción se basa en el "Report by the Inquiry Board" sobre las fallas del vuelo 501 del Ariane 5. Todo ingeniero debería leer el documento original (*cfr. bibliografía*), reflexionar sobre él y extraer profundas verdades; es espectacular por su enfoque metodológico, sus análisis y conclusiones; resulta muy ilustrativo sobre lo que **no** debe hacerse, y nos ilumina acerca de por qué a veces se toman ciertas decisiones absurdas.

La narración original tiene muchos términos que se repiten una y otra vez, pero eso resulta inevitable por ser eminentemente técnica. Mis disculpas, pero que eso no opaque la importancia del artículo.

Introducción

El 4 de junio de 1996 el vuelo de inauguración del cohete Ariane 5 terminó en desastre. 40 segundos después de iniciarse la secuencia de despegue, a una altura de 3700 m, el transportador se salió de su ruta, se fracturó y explotó. La secuencia inversa de sucesos fue la siguiente:

Cadena Inversa de Eventos Técnicos

- El cohete comenzó a desintegrarse a las H0+39 segundos debido a los grandes esfuerzos aerodinámicos producidos por encontrarse en un ángulo de ataque de más de 20 grados, lo que ocasionó la separación de los propulsores del cohete principal, activando la secuencia de autodestrucción.

- El inconveniente con el ángulo de ataque se debió a la deflexión completa de los propulsores de combustible sólido y del motor principal, Vulcano.

- Dichas deflexiones fueron ordenadas por el software del computador de abordo ("On-Board Computer", OBC), en base a los datos transmitidos por el sistema de referencia inercial (SRI 2). Parte de esa información no correspondía a lecturas apropiadas, sino a secuencias de diagnóstico del computador del SRI 2, ***que fueron interpretadas como información de vuelo...***

- La razón por la que el SRI 2 activo no envió información correcta de altitud fue porque esa unidad había declarado una falla debido a una excepción de software.

- El OBC no pudo comunicarse con el computador de respaldo, SRI 1, porque esa unidad había dejado de trabajar 72 milisegundos antes por la misma razón que SRI 2.

- La excepción interna del software SRI se originó durante una conversión de datos de un valor de punto flotante de 64 bits a uno entero de 16, que resultó en Operand Error. Estas conversiones, escritas en leguaje Ada, no estaban protegidas contra generación de Operand Errors, aunque las demás, comparables y en el mismo lugar del código, sí lo estaban.

- El error ocurrió en una sección del software que solo realiza alineación inicial de la plataforma de lanzamiento, y únicamente produce resultados significativos **antes** del despegue, después de lo cual esta función no tiene ningún propósito **¡!**

- La función de alineación se dejó operativa 50 segundos después de comenzar el modo de vuelo del SRI, que ocurre a H0-3 segundos en el Ariane 5; por eso, cuando se realiza el despegue sigue operando 40 segundos más. Ese requisito era del Ariane 4 y no se necesitaba para el 5.

- El Operand Error sucedió debido a que se obtuvo un valor inesperadamente alto de una función interna de alineación llamada BH, Horizontal Bias, relacionada con la velocidad horizontal leída desde la plataforma. Este valor se calcula como indicador de la precisión de la Alineación en el tiempo.

- El valor de BH fue mucho mayor a lo esperado porque la parte inicial de la trayectoria del Ariane 5 difiere de la del 4, lo que produce como resultado velocidades horizontales considerablemente más altas.

Los eventos internos del SRI que condujeron a la falla se reprodujeron por medio de la simulación. Más aún, ambos SRI se recuperaron y el contexto de la falla se determinó con precisión leyendo las

memorias. Y al examinar el código fuente del software demostró consistencia con el escenario de la falla.

Comentarios Sobre el Escenario de la Falla

En resumidas cuentas, las principales causas técnicas que ocasionaron la falla fueron ➢ el Operand Error al cambiar la representación de la variable BH, ➢ y la falta de protección en el mecanismo de conversión, lo que ocasionó que los computadores SRI se detuvieran.

La razón por la cual esta conversión no tenía protección es porque no se esperaba una carga mayor al 80% en el computador SRI. Para determinar las vulnerabilidades del código desprotegido se analizó *cada una* de las operaciones que podían generar una excepción, incluyendo el Operand Error. En particular, se revisaron todas las conversiones de valores de punto flotante a enteros, encontrándose que ese riesgo existía en 7 variables, de las cuales solo 4 se protegieron (lo que consiste en activar un mecanismo en el lenguaje ADA). En la documentación directa del código fuente no se encontró ninguna justificación para dejar esas tres variables desprotegidas...

La razón, en todo caso, fue que ellas estaban limitadas físicamente, o que había un gran margen de seguridad, lo cual resultó ser una ***suposición falsa*** en el caso de la variable BH. La decisión de proteger unas variables sí y otras no, fue tomada de manera conjunta por compañeros del proyecto situados en varios niveles de la escala jerárquica contractual.

¡ No se emplearon datos reales de ninguna trayectoria para analizar el comportamiento de las variables desprotegidas y, peor aún, se acordó no incluir la trayectoria del Ariane 5 entre los requisitos y especificaciones del SRI !

Aunque se identificó la fuente el Operand Error, esto por sí mismo no causó la falla de la misión... La especificación del mecanismo de atención y manejo de excepciones también contribuyó, puesto que allí se indica que, ante la ocurrencia de cualquier clase de excepción, debería ➢ indicarse la falla en el bus de datos, ➢ almacenarse el contexto de la falla en memoria EEPROM (que se recobró de los despojos del Ariane 5 y se leyó) y, finalmente, ➢ el SRI debería desactivarse (*shut down*).

Al final esta decisión de hacer que el procesador cesara de operar **fue la que resultó fatal**. El rearranque no es factible porque la altitud es muy difícil de recalcular después de que se lo ha desactivado; por tanto, el Sistema Inercial de Referencia se vuelve inútil. La razón detrás de esta acción tan drástica (*shut down*) subyace en la cultura existente dentro del programa Ariane de **solo atender desperfectos del hardware que ocurran al azar**; por tanto, el manejo de excepciones –o de errores– está diseñado para ese tipo de fallas aleatorias, ***que bien podrían resolverse por un sistema apropiado de respaldo.***

Aunque la falla en realidad ocurrió por un error sistemático en el diseño del software, *se habrían podido introducir mecanismos que mitigaran esa clase de problemas.* Por ejemplo, los computadores dentro de los SRIs podrían haber continuado suministrando sus mejores estimados de la información de altitud que se requería (Nota de Uribe: Muchos sistemas industriales y de control establecen esta clase de estimados en aquellos valores que dejan de poderse leer o evaluar; se emplean técnicas matemáticas que lo permiten, en base a la redundancia que existe en las diversas topologías. Por ejemplo, no es necesario

medir los voltajes en todos los nodos de una red eléctrica, porque las leyes de Kirchhoff permiten establecer muchos valores no medidos en base a las lecturas hechas en los demás puntos)

Hay razones para preocuparse de que a una excepción de software se le permita, o inclusive requiera, provocar un paro en un procesador mientras está a cargo de equipo crítico para la misión. De hecho la pérdida de una función del software es riesgosa porque el mismo programa corre en ambas unidades SRI. En el caso del Ariane 501, esto ocasionó la desconexión de dos unidades de equipo críticas, ***que aún estaban operativas***.

El requisito original para continuar la operación del software de Alineación después del despegue se estableció 10 años antes en los primeros modelos del Ariane, para permitir encarar un evento muy improbable: Que se detuviera la cuenta regresiva entre -9 segundos, cuando comienza el modo de vuelo en los SRI del Ariane 4, y -5 segundos cuando se inicializan en el equipo de despegue ciertos procesos, que toman horas para detenerlos. Los 50 segundos seleccionados para esta operación de Alineación continuada se basaron en el tiempo que le llevaría a los dispositivos de tierra reasumir el control completo del equipo de despegue, en la eventualidad de un "hold". Esta característica especial hizo posible rearrancar la cuenta regresiva en las primeras versiones del Ariane sin esperar por el alineamiento normal, que toma 45 minutos o más.

Pero ese requisito no aplica al Ariane 5, cuya secuencia de preparación es *diferente*, y se mantuvo muy probablemente por razones de sentido común, basadas presumiblemente en la opinión de que, a menos que se demostrara lo contrario, ***no resultaba acertado hacer cambios en software que funcionaba bien en el Ariane 4***.

Aún en los casos en los que el requisito fuera válido, sigue siendo cuestionable que la función de Alineación continúe operando después de despegar. La Alineación envuelve la aplicación de filtros matemáticos complejos para hacer coincidir el eje x con el eje de gravedad, y localizar la dirección norte de la tierra. La presunción de la Alineación previa al vuelo es que se conoce con exactitud la posición de la plataforma y que ésta es fija. Por eso la función de Alineación se ve completamente destrozada cuando se realiza durante el vuelo, pues las mediciones de los movimientos se interpretan como valores fuera de posición (offsets) de los sensores y de otros coeficientes que caracterizan el comportamiento de ellos.

Retornando al error de programación, hay que enfatizar que el software es una expresión de un diseño altamente detallado y no falla en el mismo sentido en que lo hace un sistema mecánico. Más aún, el software es flexible y expresivo, y fomenta la incorporación de requisitos de muy altas expectativas, que a su vez inducen implementaciones complejas, muy difíciles de evaluar.

Un tema subyacente en el desarrollo del Ariane 5 es la polarización hacia mitigar las fallas aleatorias. El suplidor de los SRI se limitó simplemente a seguir al pie de la letra las especificaciones que se le dieron, que estipulaban que en el evento de cualquier excepción detectada debería detener el procesador. La que ocurrió **no** se debió a eventos **al azar**, sino a un ***error de diseño***. Se la detectó bien pero el **manejo fue equivocado**, por el punto de vista asumido de que "***el software debe considerarse correcto hasta que se demuestre lo contrario***". Hay razones para creer también que esta visión es aceptada en otras áreas del diseño del software del Ariane 5, cuando lo más acertado es la concepción opuesta, de que **el**

software debe suponerse incorrecto hasta que, aplicando los mejores métodos prácticos aceptados en la actualidad, pueda demostrarse que es correcto.

Esto significa que el software crítico, en el sentido de que si falla pone en riesgo la misión, tiene que estar identificado a un nivel muy detallado, que el comportamiento excepcional debe quedar confinado, y que una política de respaldo razonable también necesita tomar en consideración las fallas de software.

Los Procedimientos de Prueba y Calificación

La calificación del Sistema de Control de Vuelo para el Ariane 5 sigue un procedimiento estándar y se realiza en los siguientes niveles:

- Calificación del Equipo
- Calificación del Software (On Board Computer Software)
- Etapa de Integración
- Tests para la Validación del Sistema

La lógica aplicada consiste en verificar en cada nivel lo que no pudo realizarse en el anterior; así se obtiene, eventualmente, cobertura completa de las pruebas de cada subsistema y del sistema integrado.

En el caso del SRI se condujo la prueba a nivel de equipo de manera rigurosa con relación a todos los factores ambientales y, en realidad, más allá de lo que se esperaba para el Ariane 5. Sin embargo, no se realizó ninguna prueba para verificar que el SRI se comportaría correctamente al ser sujeto a la cuenta regresiva, a la secuencia de vuelo y a la trayectoria del Ariane 5.

Hay que notar que por razones físicas no es posible probar el SRI como una "caja negra" en el ambiente de vuelo, a menos que uno haga pruebas completamente realistas, pero sí es posible hacerlas en tierra inyectando señales simuladas de los acelerómetros, de acuerdo a los parámetros de vuelo predichos, empleando mesas giratorias para simular los movimientos angulares del equipo. Si el suplidor hubiera hecho estas verificaciones de manera independiente, o como parte de la prueba de aceptación, se ***hubiera descubierto el mecanismo de falla***.

La explicación principal para la ausencia de tan vital prueba es, como ya se mencionó, que la especificación del SRI no incluye la trayectoria del Ariane 5 como un requisito funcional.

El comité de evaluación también notó que la especificación del sistema del SRI no indica restricciones operacionales que emerjan de la implementación escogida. Tal declaración de limitaciones, que debería ser mandatoria para cada dispositivo crítico para la misión, habría servido para identificar cualquier incumplimiento con la trayectoria del Ariane 5.

La otra oportunidad que se tuvo de detectar el mecanismo de falla con anticipación, fue durante las numerosas pruebas y simulaciones que se efectuaron en las Instalaciones para la Simulación Funcional, ISF, que está al lado del Arquitecto Industrial. El alcance de las pruebas del ISF califica:

- El comportamiento del guía, la navegación y el control dentro del contexto completo del vuelo,
- La operación redundante de los sensores,
- Las funciones dedicadas de las etapas,
- El cumplimiento del software de vuelo (On Board Computer) con todo el equipamiento del Sistema Eléctrico del Control de Vuelo

Se corrieron muchas simulaciones a lazo cerrado del vuelo completo, incluyendo la operación del segmento de tierra, flujo de la telemetría y la dinámica del lanzamiento para verificar:

- La trayectoria nominal
- Las degradaciones de la trayectoria con respecto a los parámetros internos del sistema de lanzamiento
- ... y en relación a los parámetros atmosféricos
- Fallas en los equipos, los consecuentes aislamientos, y su recuperación

Durante esas pruebas muchos dispositivos estaban presentes físicamente y se ejercitaron, pero **no** los dos SRI, que se simularon mediante módulos de software especialmente desarrollados. Algunas pruebas de lazo abierto, para verificar el cumplimiento del On Board Computer y el SRI se ejecutaron con los SRI reales. Pero solo pruebas de integración a nivel eléctrico, y de cumplimiento a bajo nivel (comunicación entre buses).

No es mandatorio, aunque sí preferible, que todas las partes de un subsistema estén presentes en todas las pruebas en algún cierto nivel. Algunas veces esto no es físicamente factible, o resulta imposible ejercitarlos completamente, o de alguna manera que sea representativa. En esos casos resulta lógico reemplazarlos con simuladores, pero solo después de una revisión minuciosa para garantizar que las pruebas previas han cubierto el escenario completo.

Este procedimiento es especialmente importante para la prueba final del sistema, antes de que pueda usárselo de forma operativa. Para poder entender la explicación dada para no incluir los SRIs en la simulación a lazo cerrado hay que describir la configuración de las pruebas que han debido realizarse.

Como resulta imposible simular las grandes aceleraciones lineales que se producen en el despegue, en los tres ejes, en una mesa de pruebas, hay dos maneras de colocar el SRI en el lazo:

A. Colocarlo en una mesa dinámica de 3 ejes (para estimular el Anillo de Láser Giroscópicos) y sustituir la salida análoga de los acelerómetros (que no pueden estimularse en forma mecánica) por simulación, mediante un conector de prueba, dedicado, y una tarjeta electrónica diseñada para este propósito.

B. Sustituir tanto la salida analógica de los acelerómetros como el Anillo de Láser Giroscópicos mediante un conector de prueba, dedicado, con señales producidas vía simulación.

La primera aproximación puede suministrar una simulación exacta, y es bastante costosa; la segunda es más barata y su comportamiento depende esencialmente de la exactitud de la simulación. En ambos casos se prueba gran parte de la electrónica y el software completo, en ambientes reales de operación.

Cuando se definió la filosofía de prueba para el proyecto, se reconoció la importancia de tener los SRIs en el anillo, y se seleccionó el método B de arriba. Pero con posterioridad se cambió de opinión y ***se decidió no tener los SRIs concretos en el lazo***, por los siguientes motivos:

- Los SRIs pueden considerarse completamente calificados, al nivel de Calificación del Equipo
- La precisión del software de navegación en el On Board Computer depende de manera crítica de la precisión de las medidas del SRI. En las Instalaciones para la Simulación Funcional, ISF, no puede obtenerse tal precisión creando las señales de prueba por medios electrónicos.
- Es imposible simular los modos de falla con equipo real, y solo puede hacerse con modelos.
- El período base del SRI es de 1 milisegundo en tanto que el del simulador ISF es de 6 milisegundos, lo cual complica la electrónica de las interfaces y puede reducir más aún la precisión de la simulación.

La opinión del comité de evaluación es que estos argumentos son válidos técnicamente, pero como el propósito de las pruebas del sistema simulado no es únicamente verificar las interfaces, **sino el sistema como un todo** para una aplicación particular, había en definitiva un riesgo al suponer que equipo tan crítico como los SRI habrían sido validados mediante la Calificación del Equipo solamente, o por su uso previo en el Ariane 4.

A pesar de que es deseable una alta precisión en la simulación, en las pruebas de sistema del ISF, resulta claramente mejor un compromiso en la precisión, pero **conseguir todos los demás objetivos**, entre ellos el de probar la acertada integración en el sistema de equipos tales como el SRI. La precisión del sistema de navegación y guía **puede demostrarse efectivamente mediante análisis y simulación** por computadora.

Finalmente hay que hacer notar que el **método definitivo de prevenir fallas** son las revisiones, **que forman parte integral del diseño** y del proceso de calificación, y que se ejecutan a todos los niveles y envuelven a la mayoría de los involucrados en el proyecto, así como a los expertos externos. En un programa de esta naturaleza, **se arreglan literalmente miles de problemas y de fallas potenciales, de una manera exitosa, en el proceso de revisión**, y es obvio que resulta difícil detectar errores de diseño de software del tipo que fue la causa técnica principal de la falla del 501. Sin embargo, también resulta evidente que **las limitaciones del software del SRI no se analizaron en toda su extensión mediante revisiones**, y que **ni siquiera se cayó en cuenta de que la cobertura de las pruebas realizadas fue inadecuada para exponer dichas limitaciones**. **Ni se pensó por un momento en las posibles implicaciones de permitir que el software de Alineación operara durante el vuelo.**

En este sentido, el proceso de revisión y prueba fue un factor importante que contribuyó a la falla.

Conclusiones

a. No hubo eventos relacionados con la falla, surgidos durante la preparación del vuelo y la cuenta regresiva.

b. Las condiciones meteorológicas en el momento del lanzamiento eran aceptables y no jugaron parte en la falla. Tampoco se encontraron otros factores externos que fueran de relevancia.

c. El encendido de los motores y el despegue fueron nominales en esencia, y algunos efectos ambientales (ruido y vibración) no resultaron de relevancia alguna para la falla. Se encontró que el comportamiento de los propulsores estuvo dentro de las especificaciones.

d. 36.7 segundos después de H0 (aproximadamente a los 30 segundos de despegar) el computador dentro del Sistema Inercial de Referencia de Respaldo, que estaba trabajando en stand-by para el control de trayectoria y altitud se volvió inoperativo. La causa fue una variable interna relacionada con la velocidad horizontal, y que excedió un límite que existía en el software de este computador.

e. Aproximadamente 0.05 segundos después el Sistema Inercial de Referencia Activo, idéntico al de respaldo en hardware y software, falló por la misma razón. Como el sistema de respaldo ya estaba inoperativo, no pudo obtenerse información correcta del sistema de control de trayectoria y altitud, y la pérdida de la misión fue inevitable.

f. Como resultado de esta falla, el sistema activo de referencia inercial transmitió, esencialmente, información de diagnóstico al computador principal de la misión, ***que la interpretó como información de vuelo y donde se usó para realizar cálculos para el control de vuelo***.

g. En base a dichos cálculos el computador principal comandó los motores para hacer grandes correcciones, para unas desviaciones de altitud que en realidad no habían ocurrido...

h. Debido a fuerzas aerodinámicas ocurrió un cambio rápido de altitud que hizo que el cohete se desintegrara a los 39 segundos después de H0.

i. El mecanismo de autodestrucción se inició automáticamente después de la desintegración, como está diseñado, a una altitud de 4 Km. y una distancia de 1 Km. de la plataforma de lanzamiento.

j. La chatarra se regó en un área de 5 x 2.5 km2. Entre los equipos recuperados se encuentran los dos sistemas de referencia inercial, que se usaron para el análisis.

k. El sistema de referencia inercial del Ariane 5 es en esencia común al sistema que vuela en la actualidad en el Ariane 4. La parte del software que causó la interrupción en la operación de los computadores del sistema inercial se usa antes del lanzamiento para alinear los sistemas de referencia inercial y, en el Ariane 4, también para habilitar cualquier rápido realineamiento en caso de una suspensión tardía en la cuenta regresiva. Esta función de re-Alineación, que **no sirve ningún propósito en el Ariane 5**, se retuvo de todas maneras por ***razones de comodidad***, y se permitió extender su operación, 40 segundos más allá del lanzamiento.

l. Durante el diseño del software del sistema de referencia inercial usado en el Ariane 4 y el 5, se tomó la decisión de no proteger el computador del sistema inercial contra valores excesivos relacionados con la velocidad horizontal, protección que sí se suministró a otras secciones del software de Alineación. Al tomar esta decisión de diseño **no se analizó ni se entendió** por completo qué valores podía asumir esta variable en particular, si se dejaba operar el software más allá del despegue.

m. En los vuelos del Ariane 4 que usaron el mismo tipo de sistema de referencia inercial, nunca hubo tales fallas, porque la trayectoria durante los primeros 40 segundos de vuelo es tal que esa variable, en particular, relacionada con la velocidad horizontal, no puede alcanzar, dentro de márgenes operacionales adecuados, valores más allá de los límites presentes en el software.

n. El Ariane 5 tiene una aceleración inicial más alta y una trayectoria que consigue llevar la velocidad horizontal a **5 veces más que la del Ariane 4**. Esto hace que, dentro de los 40 segundos en cuestión, el valor excesivo haga que los computadores del sistema inercial **cesen de operar**.

o. El propósito de los procesos de revisión, que involucran a los socios más grandes en el programa Ariane 5 es validar decisiones de diseño y obtener calificaciones de vuelo. En este proceso, las limitaciones del software de Alineación no se analizaron en profundidad y no se tomó conciencia de las posibles implicaciones que tenía permitir que continuara operando durante el vuelo.

p. La especificación del sistema inercial de referencia y las pruebas que se hicieron a nivel de equipo, no incluyeron información sobre la trayectoria particular del Ariane 5. En consecuencia no se probó la función de re-Alineación bajo condiciones de vuelo simuladas para el Ariane 5, y por tanto no se descubrió el error de diseño.

q. Hubiera sido técnicamente posible incluir casi todo el sistema de referencia inercial íntegro en las pruebas de simulación del sistema completo que se efectuaron. Pero por una serie de razones se decidió usar la salida simulada del sistema de referencia inercial, en vez del sistema en sí mismo, o de una simulación mucho más detallada. Si se hubiera incluido el sistema se habría detectado la falla.

r. Luego del vuelo se hicieron simulaciones en computadores, con el software del sistema de referencia inercial y en un ambiente simulado, incluyendo la trayectoria actual del vuelo del Ariane 501, y se reprodujo la cadena fatal de eventos que ocasionó la falla en el sistema de referencia inercial.

En resumen, la falla del Ariane 501 se debió a una pérdida completa de información de navegación y altitud, 37 segundos después de la secuencia de encendido del motor principal, debida a errores en la especificación y diseño del software del sistema de referencia inercial.

Las revisiones y pruebas extensivas que se realizaron durante el programa de desarrollo del Ariane 5 no incluyeron análisis y verificaciones adecuados del sistema de referencia inercial ni del sistema de control de vuelo completo, que hubieran detectado la falla potencial.

FALLAS EN NUESTROS MODELOS MENTALES

A veces lo más difícil de tener en cuenta no es ni el hardware ni el software: ***Somos nosotros mismos!***

Hay un fenómeno psicológico que afecta profundamente la aplicabilidad de los modelos mentales que nos hacemos automáticamente ante las diversas situaciones de la vida: Cuando dos eventos consecutivos ocurren de acuerdo a las expectativas de un operador, esa situación tiende a reforzar la confianza depositada en el modelo mental que hemos elaborado de la particular situación; sin embargo, dichos eventos pueden surgir en forma consecutiva por azar, debido a razones que en realidad difieren de las que el operador les otorga. A pesar de ello, dada la consistencia entre los datos del entorno y las expectativas del operador, éste llega a suponer que uno de los eventos es

causa del otro. Cuando esta falsa creencia ocurre, se asume de manera equivocada que el modelo mental es válido y este error puede producir consecuencias desastrosas.

Así pues, los humanos tienden a considerar que su visión de la realidad es correcta cuando hay eventos que acaecen de acuerdo a sus expectativas; sin embargo, como ya hemos dicho, muchas veces dos eventos pueden darse en la secuencial esperada, sin que tengan por ello ninguna relación de causa a efecto. Cuando este es el caso los humanos tienden a tratar la evidencia de que disponen como si en realidad reflejara el mundo de manera exhaustiva, convencidos, equivocadamente, de que han logrado entender a cabalidad el problema o fenómeno que les concierne.

En procesos dinámicos puede ocurrir que al comienzo el modelo sea válido, pero con el tiempo, y posiblemente con la degradación de una determinada situación (por un accidente o falla, por ejemplo), el modelo resulte sobre simplificado, y por la urgencia, lleguemos a tomar en consideración solo los mejores predictores de los estados del sistema, e ignoremos la información periférica que no se ajusta. Desde el concepto de la "racionalidad acotada" de Simon (1957) se acepta que preferimos soluciones económicas y adecuadas antes que soluciones perfectas pero costosas. En otras palabras, la gente tiende a satisfacer, más que a optimizar, acogiéndose a una solución que parece suficientemente buena, aún si ésta resulta sub-óptima.

El siguiente es un ejemplo, tomado de un episodio de la aeronáutica (el accidente de Kegworth; 1989) para ilustrar este tipo de inconveniente. La calamidad, con 47 víctimas mortales, fue ocasionada por el manejo equivocado que se le dio a un incidente en el **motor izquierdo** (#1). Un aspa se desprendió del motor, ocasionando vibraciones severas en el avión, y humo que se introdujo por el sistema de aire acondicionado de la aeronave. *Pero la tripulación creyó que el motor dañado* ***era el derecho*** *(#2).* En la grabadora de la cabina de mando se escucha indecisión; el primer oficial le responde al capitán: "...es el **izq**..., ¡***es el derecho***!"..., lo apagaron... y esto resultó en una disminución de la vibración del motor izquierdo, ¡el que en realidad estaba dañado! (**¿?**), **y hasta se terminó el humo**.

En base a lo anterior, la tripulación dedujo que había hecho la decisión correcta y se comenzó un aterrizaje de emergencia. En realidad el motor izquierdo continuó con un nivel anormal de vibración durante algunos minutos, *pero los pilotos no lo notaron...* Como además le bajaron la potencia a ese motor para comenzar el descenso, la vibración se redujo bastante, quedando solo un poco por encima del valor normal. A los 10 minutos, al incrementar otra vez la potencia para mantener la altitud durante la etapa final del descenso, otra vez aumentaron muchísimo las vibraciones, el motor perdió potencia y se declaró una alarma de incendio en esa turbina. Con afán comenzaron entonces a tratar de rearrancar el motor derecho, pero ya era tarde y no hubo tiempo suficiente; el ahora inevitable impacto ocurrió a media milla náutica de la pista.

Otros factores también contribuyeron a este accidente, además de los errores ya planteados de la tripulación. Ninguno de los pilotos reportó, cuando los entrevistaron, haber visto indicaciones de las vibraciones en el Engine Instrument System (EIS). Es más, el capitán *declaró que rara vez miraba los indicadores de vibración* porque según su propia experiencia en otros aviones, éstos no eran confiables (se les "*pegaban*" las agujas). Hay que notar que en este modelo de avión los indicadores son digitales (numéricos), pero el 64% de los pilotos reportó en una encuesta que estos indicadores modernos no eran tan efectivos como los viejos para llamar su atención ante cambios

rápidos en los motores, y el 74% dijo preferir los indicadores antiguos. Para colmo, estos dispositivos no incluyen ni indicativos sonoros ("beeps") ni medios visuales complementarios (parpadeos) para alertar sobre lecturas excesivas...

Además, la carga de trabajo de la tripulación se incrementó fuera de control; el piloto trató de mantenerse en fase con la evolución del incidente: Al apagar el motor #2 (el bueno) se oye en la grabación que decía: "Ok; qué indicaciones en realidad tuvimos..., grandes vibraciones en el avión, humo...", pero al llegar a este punto las comunicaciones de radio los interrumpieron desde la torre de control, la aeromoza entró para reportar que los pasajeros tenían pánico, lo cual distrajo más aún al piloto quien tuvo que hacer un anuncio por el micrófono para calmarlos, y tanto el capitán como el copiloto tuvieron que hacer más llamadas a la torre, en preparación al aterrizaje de emergencia; todo esto afectó negativamente el grado de control de la situación.

Por último hay que anotar que a pesar de que tanto el capitán como el primer oficial eran experimentados pilotos, con 13,000 y 3,200 horas de vuelo, respectivamente, entre ambos solo tenían 76 horas de experiencia en ese modelo de avión (Boeing 737-400).

Lo que ocurre con los modelos mentales en un mundo cambiante es que los involucrados, tratando de ahorrar en recursos cognoscitivos, se polarizan de tal manera que llegan a aceptar con facilidad, confirmaciones que son solo parciales, y en vez de darles un alto valor a las **evidencias contradictorias**, las personas tienden a ***aceptar solo información consistente con lo que esperan***; este fenómeno se conoce como polarización en la confirmación (Klayman & Ha, 1989). El corolario es que la gente no le presta atención a la información que contradice el modelo; para colmo, en el caso del accidente narrado la decisión (equivocada) coincidió con la reducción en el nivel de los síntomas por veinte minutos, lo cual hizo más difícil aún de integrar la evidencia contraria disponible a continuación. Los operadores, en general, son más propensos a rechazar cualquier información que no es consistente con sus expectativas, que a actualizar sus modelos mentales, pues esto tiene costos en los que ellos no siempre pueden incurrir en situaciones que se tornan críticas y muy exigentes en el uso del tiempo. Al final lo que sucede es que los datos se reinterpretan abusivamente para ajustarlos al modelo que los operadores tienen de la situación, siguiendo la línea del menor esfuerzo. Así, se les asigna significado a datos aleatorios, que no lo tienen, si se ajustan a su visión del mundo para ese momento, con lo que equivocadamente mantienen como válidas, representaciones que se alejan de la realidad.

En situaciones dinámicas una razón para ésto es que los operadores tratan de evitar el costo de revisar sus modelos mentales, porque esos modelos son los que les dan la sensación de estar más en control. Es decir, ***satisfacen***. Y la urgencia del momento hace que se mantengan los modelos mentales inclusive después de detectar contradicciones.

A este fenómeno yo lo llamo: "***Contra-evidencia***", y no se limita a situaciones de emergencia; en ocasiones tomamos decisiones en cierta dirección pero, al revisarlas en perspectiva, se nos hacen patente hechos que desdeñamos, y que de haber sido tomados en cuenta nos habrían hecho girar 180 grados…

La "Contra-evidencia" es la enemiga número uno del método científico.

BIBLIOGRAFÍA

http://ei.cs.vt.edu/~history/50th/30.minute.show.html

http://en.wikipedia.org/wiki/Computer_bug

http://en.wikipedia.org/wiki/Mars_Climate_Orbiter (unidades métricas e inglesas)

ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf

http://en.wikipedia.org/wiki/Mars_Polar_Lander

http://americanhistory.si.edu/collections/object.cfm?key=35&objkey=30

http://en.wikipedia.org/wiki/Ariane_5_Flight_501

http://homepages.inf.ed.ac.uk/perdita/Book/ariane5rep.html

Capítulo

2

PROYECTOS I

"Perfection is finally attained, not when there is no longer anything to add, but when there is no longer anything to take away..." (Antoine De Saint-Exupéry)

INTRODUCCIÓN

EL tema de los proyectos no solo cubriría un libro entero, sino toda una especialización. Y la Gerencia de Proyectos **no** es la parte menos importante, si se la compara en relación al aspecto técnico intrínseco. Pero como no pienso rescribir lo que ya está impreso, quiero aportar solo algunas ideas generales, extraídas de mi propia experiencia de 20 años de dirigir grupos de ingenieros (1982-2002) en el desarrollo de diversos tipos de proyectos, desde la implantación de sistemas foráneos y su adaptación al medio nacional (SCADAS, computadoras industriales), hasta el desarrollo de tecnología vernácula en la rama de los sistemas de supervisión y control para la industria eléctrica venezolana (RTUs, Registradores de Fallas, SCADAS); desde sistemas de télex y procesadores de palabras, cuando el vocablo "PC" no había sido inventado, hasta sistemas en que Clientes microcomputarizados se comunican con Servidores en Internet, para formar una red de supervisión de la calidad de la energía.

ASPECTOS GENERALES

PRODUCTO, MERCADO, ADMINISTRACIÓN

Aprendí que el aspecto técnico, el **Producto**, **no** era suficiente para establecer una empresa con porvenir, que eran necesarios también el Mercado, y la Administración. En **Mercado** se agrupan las ventas, la publicidad, las relaciones públicas, su investigación, y de la mayor importancia, la existencia de necesidades genuinas qué satisfacer y, por sobre todo, **El Cliente**. En **Administración** se engloban las cobranzas, las finanzas, los recursos humanos y la detestable pero necesaria atención a la miríada de obligaciones públicas, impuestos y regulaciones (en ese orden descendente de importancia).

La dirección de una compañía pequeña, más que una Gerencia General, precisa de una Gerencia Técnica que maneje esos tres aspectos fundamentales: Producto, Mercado y Administración.

Vi a mi competencia caer, ostentando un buen producto, regular administración, ***y un manejo pésimo del Mercado y de El Cliente***. La vi venirse abajo con un cliente cautivo en el bolsillo, una administración común y corriente, ***y un producto pésimo***. Y yo mismo desaparecí, con productos de grandes prestaciones técnicas, una pésima administración..., y un mercado que se desmoronó en 2002… *Por cierto, algo por lo que pagué muy caro por aprender es que no hay socio bueno.*

Cuando un ingeniero encara sus primeros proyectos y quiere sopesar las facetas más importantes a las que se enfrenta, encuentra un telón de fondo recurrente: "¿Debo pensar en grande, y lanzarme a las más aventuradas empresas, o tengo que atender a los detalles?".

Desde luego, ahora sé que **ambos** aspectos son importantes, y que el fracaso cuando se emprenden proyectos ambiciosos puede buscarse en el descuido a los detalles, pero que también el ser minucioso no conduce a nada si no se tiene una visión en grande, progresista, del negocio. También encontré que es muy difícil que en una misma persona coincidan las cualidades necesarias para cubrir los dos aspectos al mismo tiempo. Los soñadores suelen adorar la composición y menospreciar las pinceladas, y los talladores pueden materializar un relicario de la nada e ignorar el equilibrio de formas, luces y colores.

THINK BIG... OR SMALL?

Pues no estamos solos en tan grotesca encrucijada. Les cuento en breve, lo que viví.

En 1978 asumí la Gerencia Técnica de la compañía que representaba a ***Digital Equipment Corporation*** para Venezuela. DEC era entonces, con sus 15,000 empleados, el segundo suplidor de computadoras en el mundo, después de IBM, que para la fecha tenía alrededor de 50,000 personas en su nómina. Para que noten la diferencia entre las dos empresas, las ventas de DEC igualaban a lo que IBM invertía en **R&D**: Investigación y Desarrollo. DEC había comenzado sus operaciones produciendo módulos que realizaban diversas funciones, el equivalente de nuestros circuitos integrados MSI actuales; había profundizado en una innovación de la tecnología que se ha mantenido como disciplina hasta nuestros días: **RTL**, o *Register Transfer Logic*, los *Algoritmic State Machines* modernos. Su principal logro en el campo de las computadoras, después de unos sistemas mas o menos grandes (DecSystem-10 y DecSystem-20) fue la introducción de las primeras Minicomputadoras, siendo sus estrellas la PDP-8 primero, la PDP-11 luego (del cual los micros modernos copiaron ***gran*** cantidad de conceptos) y el sistema VAX-780 después.

Hacia 1980 algún directivo innovador en DEC decidió realizar un gigantesco salto hacia adelante en la organización, tratando de aproximarse más a IBM... Como las grandes corporaciones norteamericanas, DEC se encargaba de hacer fluir sus políticas por toda la organización (una especie de adoctrinamiento, o *lavado de cerebro*); recuerdo que un día nos llegaron unos cuadros, enmarcados, destinados a ser colocados en la pared, detrás del escritorio de los gerentes, con un letrero inmenso que decía:

Think Big

Cada división, área y unidad en DEC debía cambiar su cultura para implantar estrategias, programas y planes que elevaran su "performance" (rendimiento) a toda costa y, fundamentalmente, **sus ganancias** (***profits***). Fue así que decidieron **no** producir computadoras… completas, sino que cada local se

especializaría: Habría fábricas de discos, factorías de CPUs, manufactureras de impresoras, ensambladoras de monitores...

¡ Tampoco se integrarían los sistemas en la fábrica !, sino que se "unirían" en los ***muelles*** desde donde se despachaban los camiones con la mercancía (***On the deck merge***).

Excelente..., muy bien. Pero recuerdo que una de las computadoras que me llegó en esos días tenía las tarjetas **enchufadas al revés**. ***¡ Mentira que ese equipo hubiera sido probado !***

Siendo lo más importante **despachar y facturar**, empezaron a faltar pequeñas partes en cada caja (¡ trampa !), como los cables de alimentación de monitores e impresoras. En USA eso no era mucho problema: El técnico que estaba instalando la computadora simplemente llamaba a la fábrica por teléfono, y vía "Overnight delivery" (*Federal Express*) recibía los faltantes al día siguiente a las 8 a.m. En Venezuela... yo podía estar 45 y hasta 60 días esperando la importación de dichas piezas. Lo cierto es que **yo no podía cobrarle al disgustado cliente** (***centenares de miles de dólares***), pues casi nunca se conseguían en el país los cables y conectores (los de alimentación de los CPU, en particular, eran bastante ***especiales*** de DEC; los de las impresoras tampoco los había localmente, y pare usted de contar)

No se detuvo ahí la innovación del "Piense en Grande"...

Otro sector de Digital, el de la Documentación Técnica, no tuvo mejor idea para comenzar a hacer ganancias inmediatas, que ***cobrar 15 veces más los manuales técnicos*** (antes se pagaban $10 por un conjunto de planos, que de la noche a la mañana pasó a costar $150). Y el área de Entrenamiento, para no quedarse atrás, elevó el precio de sus cursos. Ambos factores hicieron que mis propios costos de servicio (como empresa independiente, representante local de DEC en Venezuela) subieran astronómicamente, mientras que a IBM de Venezuela, mi aplastante competidor, le salía el mantenimiento por una cifra nominal (***transfer costs***). Para nuestros clientes, el mantenimiento pasó a ser prohibitivo. Nótese que adentro de DEC no importaba demasiado, ya que ***unas unidades le facturaban a otras los servicios*** cuando un gerente quería entrenar un técnico (eso era sacar dinero de un bolsillo para ponerlo en el otro). Igual daba un valor bajo que alto. Eso sí, ***creían ellos***..., desde el punto de "resultados", cada unidad producía ahora 15 veces más, **de un solo plumazo**... En nuestro país, en cambio, si una computadora no funcionaba, ¡ al cliente poco le importaba si era por el CPU dañado (los nuestros nunca fallaban; los de IBM sí), **o por que le faltaba un cable** !

Para empeorar el panorama (los males nunca llegan solos), el sector de mercadeo de DEC, siempre pensando en grande, decidió que si ofrecía infinidad de modelos de cada opción se abrirían más y más mercados y oportunidades de venta. Pero ahora, yo (en Venezuela) sólo podía vender 2 ó 3 equipos de cada modelo (unidades de disco, por ejemplo, o CPUs) antes de que fueran obsoletos y Digital sacara unos más modernos (exactamente como ocurre en la actualidad con los Pentium y los PC). Como mi contrato de representación me exigía tener un (1) kit de repuestos por cada 10 unidades (o fracción), pues el costo de esas partes en lugar de promediarse entre 10 **lo hacía entre 3!** Y mis técnicos, obligados a tomar cursos de entrenamiento en fábrica, para ***cada modelo*** (siempre según mi contrato), pues se la pasaban en USA dizque estudiando la pléyade de opciones y modelos, en lugar de andar en el mantenimiento e instalación de equipos en Venezuela, que era su función y vía única y real de ingresos de la compañía venezolana.

En año y medio, DEC estaba al borde de la quiebra, gracias a su flamante:
"Think Big"

Llegó entonces otro director... que reaccionando según la dialéctica, impulsó la ***contrarreforma***, y un día nos enviaron otros cuadros enmarcados... destinados a reemplazar a los anteriores... con un letrero inmenso que decía:

Think Small

Se decretó ahora que ***lo fundamental eran el detalle y la minuciosidad...***

DEC ***desapareció*** luego de unos 35 años de existencia.

IBM aún sobrevive, superando los 100 años

Resulta, pues, que lo importante es la ***armonía*** *entre todos los factores involucrados. Esta es la postura "****Sistémica Holística****" actual (grandes palabras...)*

INTERROGANTES PARA UNA INVESTIGACIÓN

"El Pastelero"

Hay por lo menos 10 puntos importantes que **siempre** deben tenerse en cuenta y **definirse para cada faceta** de un proyecto. He aquí mi nemotecnia para mi amigo Quique (Enrique):

"**Qui**(én)**Qué**: ¡ He reñido a un pastelero ! **Por_qué Cuándo Dónde Cómo**, **Para_Qué** hablan **Acerca_de Cuánto**, si no hay **Con_qué**?"

¿Qué? ¿Quién?	¿Por qué? ¿Cuándo?	¿Dónde? ¿Cómo?	¿Para qué? ¿Acerca de?	¿Cuánto? ¿Con qué?

¿Acerca de? Se refiere a los temas o contenidos en los cuáles se centrará el proyecto.

¿Para qué? Comprende los objetivos, o lo que se quiere lograr con el proyecto (a corto, mediano y largo plazo). Constituye el punto de llegada del trabajo y los logros sucesivos a lo largo del proceso.

¿Qué?, Se refiere al tipo de actividad que va a realizarse. Se debe precisar exactamente lo que va a hacerse, sin que quede lugar a dudas... ***Parece de Perogrullo, pero es muy difícil.***

¿Quién? (o quiénes), Asigna sin ambigüedades las diversas responsabilidades y actividades del proyecto

¿Por qué? Es la justificación de cada actividad. Todo tiene un por qué.

¿Cuándo? Constituye la programación en tiempo y en etapas; se fijan fechas y períodos destinados a cada fase del proyecto (**Duración**). Debe especificarse cada paso, la secuencia, la prioridad, y el momento en el que se realizará. No se crean que porque han llenado unos cuantos campos en "MSProject" u otra herramienta similar, ya con eso es suficiente. Hace falta CRITERIO. ***Uno de los puntos más notables en donde falla un proyecto es en el Cuándo.***

¿Dónde? Se refiere al alcance geográfico o ubicación espacial donde se llevará a cabo cada actividad (diseño, programación, pruebas, instalación, etc.) ¿Obvio? Uno de mis proyectos iba a realizarse en la planta Tacoa, Caracas. Y así se cotizó. Luego resultó que, bueno..., Tacoa no está, en realidad, en Caracas. Había como ***hora y media*** de trayecto (ida y regreso: **Tres** horas), con lo que hubo que alojar el personal en un hotel de Catia La Mar, y se pagaron viáticos, residencia, alimentación, transporte, alquiler de vehículos, etc., SIN que ésto pudiera cobrársele al cliente (las cotizaciones son a precio fijo). Gastos extras se restan directamente del ***profit*** (las ganancias), y pueden llevar a pérdida un proyecto.

¿Cómo? El cómo define los métodos, las técnicas; las tácticas y las estrategias que van a utilizarse en las diversas etapas del proyecto, para alcanzar los objetivos. ¡ Es el "**know-how**" **por el que le pagan** !

¿Cuánto? Implica la elaboración del estudio de costos y el presupuesto del proyecto, así como las alternativas de financiamiento. Debe incluir la determinación del factor Costo/Beneficio del proyecto. Esta pregunta es **vital** para la sobrevivencia de una compañía, **y es su razón de ser**. Y un cliente al que se le hace ver que no está gastando, sino que va a ahorrar..., es un cliente dispuesto a comprar.

¿Con qué? Define los recursos o materiales necesarios para el desarrollo del proyecto.

CÓMO SE PLANIFICA UN PROYECTO

Hay que repetir que este no es el tema completo de la planificación de proyectos; son solo guías que intentan sobre todo motivar y orientar el trabajo en las asignaturas que tienen laboratorio y diseño en la universidad.

ASPECTOS PREVIOS

Todo comienza identificando una necesidad; el usuario determina sus requisitos, y dependiendo del tamaño del problema, y las normas internas de la empresa que hace la solicitud, se generan unas Especificaciones, un Pliego de Licitación y se hace una Licitación o a lo mejor, una Negociación Directa. Como resultado, el cliente tiene una Oferta; si fue una licitación se hace la adjudicación al mejor oferente, de acuerdo a los parámetros de evaluación, que suelen incluir aspectos económicos y técnicos.

Normalmente, después de la adjudicación, en donde hay por una parte un pliego de licitación, que contiene entre otras cosas las Especificaciones (lo que el cliente quiere), y por otra la oferta del ganador (lo que él propone), se pasa por una etapa de consolidación, en donde se genera un documento conocido como **Statement of Work** (**SOW**; cfr. IEEE/EIA standard 12207, 1996), que es el que se emplea para hacer el diseño definitivo, y para generar otros documentos, tales como la lista de "deliverables" (qué cosas concretas se le van a entregar al cliente), documento de Factory Acceptance Test (**FAT**, cómo van a ser las pruebas que se harán en la fábrica, antes de que el sistema se traslade al lugar del cliente), Site Acceptance Test (**SAT**, qué pruebas se realizarán con los equipos ya instalados en el campo) y otros documentos más, entre los que está el "**COMMISIONING**" (puesta en marcha del sistema, en el sitio final).

Todo lo anterior se encuentra fuera de los objetivos de esta obra, por lo que continuamos con el primer paso del diseño propiamente dicho.

DISEÑO CONCEPTUAL

Esta etapa es ***Fundamental***; deben analizarse diferentes aproximaciones conceptuales al diseño del proyecto. Esto incluye la evaluación al menos de los siguientes factores:

- cuáles partes deben realizarse en Hardware y cuáles en Software;
- consideraciones de tamaño (si debe caber en un espacio predeterminado),
- costo (cuáles son los límites),
- materiales que deben emplearse (tecnología de semiconductores; materiales especiales para la caja),
- sistemas de comunicación (email, FTP, RS-232 normal),
- alternativas de interconexión (entre equipos; con las interfaces; con la maestra: Lazo de corriente, fibra óptica),
- análisis de las dificultades de la instalación,
- tiempos de ejecución (cuáles son los parámetros; esto indicará la cantidad de recursos que habrá que asignar), etc.

Como resultado de esta actividad, el Líder del Proyecto genera un ***Documento de Diseño Conceptual***. Es importante obtener la aceptación del cliente (a través de la Gerencia de Operaciones) del documento de *Diseño Conceptual*, ya que **éste es el momento más apropiado para debatir cualquier discrepancia de parte del Cliente**, ***antes de continuar con el proyecto.***

El *Diseño Conceptual* tomará un tiempo muy corto, porque es muy probable que esta actividad ya se haya adelantado con anterioridad en la empresa, durante la etapa de cotización e interacción con el cliente, previa a la orden de compra.

CRONOGRAMA (TENTATIVO) DE TRABAJO

Terminada la etapa de *Diseño Conceptual* puede realizarse un *Cronograma de Trabajo*, indicando las duraciones estimadas de las etapas siguientes del proyecto. Este cronograma SIEMPRE es **tentativo**, y será evaluado semanalmente entre el Líder del Proyecto y la gerencia, a fin de determinar las posibles desviaciones y tomar los correctivos necesarios. Las siguientes etapas pueden refinarlo y modificarlo…

ENMIENDAS AL CRONOGRAMA DE TRABAJO

Cualquier cambio en el cronograma deberá ser nuevamente aprobado por la gerencia como **enmiendas** al cronograma original.

DISEÑO *FUNCIONAL*: HARDWARE Y SOFTWARE

DESCRIPCIÓN

Se descompone el proyecto, definido según el documento de *Diseño Conceptual*, en los diferentes módulos de Hardware y de Software que realizarán **FUNCIONES** específicas. El documento de *Diseño Funcional* deberá contener **diagramas que muestren *las interrelaciones* entre las diferentes partes**.

Terminada esta actividad se realizará una revisión del *Cronograma de Trabajo*, afinando algunos tiempos, según pueda determinarse con la nueva óptica que se tiene al finalizar el *Diseño Funcional*, mediante **enmiendas** al cronograma original.

ENMIENDAS AL DISEÑO FUNCIONAL

Discrepancias que surjan en el *Diseño Funcional* al realizar algunas de las etapas subsiguientes, deberán ser reportadas inmediatamente a la gerencia a fin de tomar una decisión y recibir la aceptación de los posibles desvíos. Esto se hará como **enmiendas** al *Diseño Funcional.*

También una modificación en el *Diseño Conceptual* puede generar (hacia abajo) una **enmienda** al *Diseño Funcional.*

Los **cambios** al *Diseño Funcional* deben generar las **enmiendas** que correspondan en toda la cadena subsecuente comenzando por el *Diseño de Detalle*, y al *Diseño Conceptual*, si fuera pertinente. Si cambian algunas funciones de alto nivel (las que "ve" el usuario), de manera que impacten al cliente, es imperativo obtener su aprobación explícita para evitar, al momento de la entrega, modificaciones en esta instancia que impliquen: Recodificaciones, repeticiones de tarjetas, etc.

DISEÑO *FUNCIONAL*: HARDWARE

La parte del *Diseño* ***Funcional*** correspondiente al Hardware debe incluir la **definición** completa, precisa, exacta, detallada; sin vaguedades (todo debe estar cuantificado) de **todos y cada uno de los siguientes puntos** (los que correspondan):

- CPU (características de velocidad, longitud de palabra, familia, cantidad de CPU's y función de cada uno)
- Memorias (cantidad, tipo)
- Almacenamiento secundario (tipo, tamaño)
- Interfaces varias (seriales, paralelas: Cantidad, tipo)
- Módulos de conversión A/D (número de bits, velocidad, necesidad de filtraje, voltajes o corrientes que van a manejarse, necesidades de Sample&Hold)
- Módulos de entrada digital (cantidad, definición de los niveles de tensión y corriente de entrada que van a manejarse, filtrajes necesarios, aislamientos)
- Unidades acondicionadoras de analógicos (AIM)
- Unidades acondicionadoras de digitales (DIM)
- Unidades de interfaz con las señales analógicas provenientes del campo (AFI)
- Unidades de interfaz con las señales digitales provenientes del campo (DFI)
- Aisladores galvánicos para señales analógicas (ISOs)
- Optoaisladores para señales digitales
- Modems
- Otros optoacopladores
- Relés (capacidades de los contactos, tiempos de conmutación, "duty cycle")
- Gabinetes (tamaño, norma NEMA, capacidad de entrada, salida y control)
- Previsiones de alambrado de señal, de control, de comunicaciones, etc.

Si hubiera restricciones que limiten la agrupación de unas funciones con otras, en una misma tarjeta o "backplane", deberán señalarse claramente aquí. Si pueden emplearse elementos suplidos por otros (*Fuentes, Modems, Computadoras, Tarjetas con Microprocesadores, etc.*) o diseñados previamente en la compañía, debe también indicarse a este nivel. Si es necesario ***modificar*** algunos elementos empleados por la compañía anteriormente, deberán señalarse aquí las diferencias que deben introducirse.

El Líder de Proyecto producirá un ***Documento de Diseño Funcional del Hardware***, con la definición completa, precisa, exacta y detallada de todas las funciones necesarias para el proyecto, ateniéndose siempre al Documento de *Diseño Conceptual.*

DISEÑO *FUNCIONAL*: SOFTWARE

El *Diseño* ***Funcional*** *del Software* incluirá la **descripción** de las diferentes prestaciones de *alto nivel* (las funciones que "ve" el usuario): Protocolo de comunicaciones, cantidad de información que va a transmitirse y su velocidad, categorización de las informaciones (analógicos rápidos, lentos; scan normal, scan de seguridad), capacidades de control, funciones de diagnóstico, interfaz con el usuario, interfaz de

mantenimiento, funciones de respaldo (en caso de que haya necesidad), funciones de cálculo específicas (FFT, etc), necesidades de multitarea, spooling, comunicación en back-ground, etc.

Deben incluirse diagramas de despliegues (si algo tiene que mostrarse en la pantalla, aquí hay que incluir un diagrama, o foto, de dicha pantalla), editores, menús, reportes (si algo va a imprimirse, aquí tiene que mostrar un facsímil). Obsérvese que los facsímiles se incluyen ANTES de haber generado ni una sola línea de código para producirlos), etc. Deben identificarse los programas que van a ser ***modificados*** (en caso de que ya la compañía haya hecho programas similares con anterioridad) o creados **nuevos**.

Funciones que **no** ve el usuario, pero que tienen que estar incluidas en la actual etapa son, por ejemplo, las rutinas de autodiagnóstico. En este caso es bien importante definir qué acciones deberán tomarse en caso de detectarse una falla: Reportes, señales, etc. Cada posible falla detectable deberá tener su especificación correspondiente. (Como de costumbre, no tiene sentido incluir una rutina de diagnóstico si no está claramente especificada qué acción inteligente y apropiada deberá adoptarse en caso de detectar una falla. Normalmente **¡ apagar el equipo es lo menos indicado !**)

Si hubiera funciones que han de operar en CPU's diferentes (8031, HC08, etc.), hay que indicarlo. Si pueden emplearse rutinas suplidas por otros (Spoolers de impresión, rutinas estándar de comunicación [Xmodem], FFT, librerías "**.VI**", de CRC, etc.) o codificadas previamente en la compañía, debe también indicarse a este nivel. Si es necesario modificar algunos de los anteriores elementos deberán señalarse las diferencias que se necesita introducirles.

Al evaluar ciertas funciones como impresión de reportes y adquisición de información, el *Comité de Diseño* debe determinar claramente si van a realizarse como multitareas, en cuyo caso hay que definir las prioridades, o si la adquisición de información va a realizarse en background, con garantía de que no se pierda información, o si va a realizarse una actividad con **exclusión** de la otra, sopesándose lo apropiado de esta decisión. Cuando se trata de una maestra, que no tiene necesidad de tiempo real para la adquisición de la información, ya que el dispositivo que actúa en tiempo real es el que está en las instalaciones, esta definición puede hacerse de una cierta manera; en cambio, si se trata de una unidad de campo, debe tenerse muy presente que su actividad primordial es la de adquisición y almacenamiento de información en tiempo real; las demás deben supeditársele.

Cambios fundamentales en el software estándar de la compañía (como por ejemplo en sus Rutinas de Tiempo Real), deben seguir un proceso mucho más riguroso aún, debido a lo sensibles que resultan pues son el núcleo de la gran mayoría de sus productos (es decir, si están funcionando, y probadas, RESÍSTASE A MODIFICARLAS ni siquiera levemente).

El Líder de Proyecto producirá un documento de *Diseño Funcional del Software*, con la definición completa de todas las funciones necesarias, ateniéndose siempre al documento de *Diseño Conceptual.*

DISEÑO DE *DETALLE*: HARDWARE Y SOFTWARE

Noten que hasta aquí:

NO HEMOS ESCRITO NI UNA LÍNEA DE CÓDIGO,
NI HEMOS ARMADO CIRCUITO ALGUNO...

Las mayores economías y mejores prestaciones en un equipo, no se consiguen cuando se lo está programando, o se lo arma:

SE LOGRAN EN LA ETAPA DE DISEÑO

Allí es donde tiene que aplicar la creatividad y el ingenio, ingeniero...

DESCRIPCIÓN

Esta etapa se basa en el documento de *Diseño Funcional*. Cualquier discrepancia que surja en el *Diseño de Detalle* con aquél, deberá ser reportada inmediatamente al *Comité de Diseño* y a la gerencia a fin de tomar una decisión y recibir la aprobación de la posible desviación. Esto se hará como **enmiendas** al documento de *Diseño Funcional*, que hay que anexar, debidamente aprobadas por la gerencia, a su correspondiente **Carpeta de Trabajo**.

Si el proyecto es de envergadura, habrá un *Líder de Hardware* y un *Líder de Software*, bajo la coordinación del *Gerente del Proyecto*.

DISEÑO DE DETALLE DE HARDWARE

La actividad comienza generando un ➢ ***Diseño de Implementación***, que muestra en el Hardware ***las relaciones*** entre las diversas tarjetas, ➢ definición de interfaces entre ellas (cómo se van a conectar unas con otras), ➢ mostrando el funcionamiento mediante diagramas electrónicos. El *Diseño de Implementación* será finalmente revisado para constatar que se cumplen todos los requisitos de las etapas anteriores; como resultado se obtiene el ➢ ***Diseño de Detalle*** de cada tarjeta, que permite producir la **descripción detallada (diseño electrónico)** de cada una, que ya no admite cambios posteriores.

Como **anexo A** al documento de *Diseño de Detalle de Hardware*, se señalarán todas las **pruebas** que deben realizarse para la **verificación** de la operabilidad de cada tarjeta. Las **pruebas** deben ser completas y describirse en profundidad, de manera de permitir su empleo en otras etapas de la operación, tales como: Control de calidad, pruebas de aceptación, etc. Debe incluirse la descripción de equipos de prueba necesarios en esta etapa. Asimismo, deben *anexarse* los ***Programas de Prueba*** que permitan constatar la operación de cada tarjeta. Estos programas servirán posteriormente en la etapa de producción.

Como **anexo B** deben incluirse todas las **indicaciones de error** (leds encendidos, etc.) que sean producidos por cada tarjeta cuando se detecte una falla en ellas.

DISEÑO DE DETALLE DE SOFTWARE

La actividad comienza generando un ➢ ***Diseño de Implementación de Software*** mostrando las ➢ relaciones entre las diversas rutinas, ➢ definición de variables (locales y externas; nombres, significados, uso), ➢ indicando el lenguaje en que se hará el desarrollo, ➢ mostrando un diagrama de flujo o, en su defecto, un listado de pseudo código. El *Diseño de Implementación* será finalmente revisado para constatar que se cumplen todos los requisitos exigidos. Como resultado se obtiene el ➢ ***Diseño de***

Detalle de cada Función, con la **programación** detallada de las rutinas, ***no sujeta a cambios posteriores***.

Como **anexo A** al documento de *Diseño de Detalle de Software*, se señalarán todas las **pruebas** que deben realizarse para verificar la operabilidad de cada función. Las **pruebas** deben ser completas, describirse en profundidad y deben anexarse los *Programas de Prueba* que permitan constatar la operación de cada función, si fueran necesarios.

Como **anexo B** deben incluirse todos los **mensajes de error** y las demás **señalizaciones de error** (leds encendidos, marcas en la pantalla, etc.) que sean producidos por cada función de software cuando se detecte una falla interna.

Ambiente de Desarrollo

Será homogéneo: Cada máquina de la compañía tendrá la estructura de directorios ***mínima común*** para los discos, definida en el **Documento de Estandarización de Computadoras**.

Se emplearán los mismos lenguajes (C, MASM x86, Assembler 8047, CodeWarrior, WinIDE, etc.), con los compiladores estándar de la compañía (no se aceptarán implementaciones en otros lenguajes, ni con otros compiladores)

En caso de ser necesario usar Rutinas de Tiempo Real, se emplearán las que son estándar de la compañía, las cuales serán verificadas para establecer su uniformidad y evitar productos contemporáneos establecidos sobre versiones variadas (e inapropiadas) de dichas rutinas.

Los archivos **"make"** deberán formar parte de la documentación y si no se usa un formato autodescriptivo (como en Unix), habrá de anexarse un **"make.doc"** que explique suficientemente el comportamiento del archivo "make". No es aceptable entregar el diseño de detalle sin los correspondientes archivos "make" y "make.doc".

VARIOS

Si surgieran problemas en la operación del Hardware, en desacuerdo al diseño original y éstos pudieran ser corregidos por Software, empleando técnicas como promediación de señales ante presencia de ruidos no previstos en los conversores, filtrajes de oscilaciones no esperadas (ni especificadas), etc. (la vida real es así), éstos procedimientos tienen que ser sometidos a estricta aprobación de la gerencia. De lo contrario termina el implementador de turno derrumbando todo el castillo de naipes al imponer sus propios criterios técnicos, que no tienen que ser, y casi nunca son, los de la compañía. A toda costa hay que evitar que las ocurrencias del programador de menor jerarquía prevalezca sobre la organización.

De manera análoga, errores de operación en el Software (*c'est la vie*), en desacuerdo al diseño original y que "pudieran" ser apropiadamente corregidos o socorridos por Hardware, empleando elementos tales como restauraciones de la operación (***Reboots***), mediante el uso de temporizadores electrónicos (Watch Dog Timers), etc., tienen que ser sometidos también a la más estricta aprobación de la gerencia.

Asimismo, la puesta en marcha de equipos sin la inclusión de una o más funciones originalmente concebidas y decididas (tales como no activar las tarjetas de "Watch-Dog Timer", o dejar las rutinas de diagnóstico desactivadas, u otros despropósitos similares), tienen que ser aprobados por la gerencia.

PRUEBAS DE HARDWARE Y SOFTWARE: PROTOTIPO

Como parte del *Diseño de Detalle* debe generarse un *Plan de Pruebas*, que incluye tanto el Hardware como el Software, que debe formar parte del procedimiento de la aceptación del *Diseño de Detalle*, y que debe cubrir no solo las facetas más elementales, sino que ha de ser lo más completo posible, a fin de establecer cualquier inconveniente ***antes*** de comenzar la producción en serie.

PROTOTIPO

Para verificar el *Diseño de Detalle* se necesitan *prototipos* (que pueden ser completos o parciales) **antes** de realizar la elaboración completa y la prueba definitiva de las tarjetas, y antes de dar por aceptada la codificación completa de las rutinas y la prueba definitiva de las mismas. Además, el prototipo *pretende demostrar* ***el sistema*** *trabajando como un todo.*

Es importante establecer una frontera clara entre Producción y Desarrollo en cuanto al prototipo, a fin de evitar que se entremezclen responsabilidades: Echar a andar un ***equipo*** está plenamente bajo la égida de Producción; poner a punto ***un prototipo*** *es claramente una obligación de Desarrollo.*

PRUEBAS DE SOFTWARE

El plan de pruebas de software será detallado y completo, y pretende evitar la ocurrencia de potenciales problemas tales como rutinas de interrupción que no preserven el valor de algún registro y que ante una secuencia de interrupciones pueda producir una falla, intermitente o esporádica. En la eventualidad de descubrirse errores, debe generarse una solución conjuntamente con el *Comité de Diseño.*

PRUEBAS DE HARDWARE

El plan de pruebas de hardware será detallado y completo, y pretende verificar la operatividad del equipamiento y el trabajo en ambiente hostil. En la eventualidad de descubrirse errores, debe generarse una solución conjuntamente con el *Comité de Diseño.*

El plan de pruebas de hardware debe evitar que en la etapa de producción se produzcan parches, techitos, "piggy-back's", cablecitos, producción de circuitos impresos nuevos, etc., medidas que a ese nivel hay que realizárselas a ***toda*** la producción.

De ninguna manera la prueba verdadera se realizará en el sitio definitivo de instalación del equipo (se realizará *antes*, en el laboratorio).

PRUEBAS DE SISTEMA

Los sistemas nuevos deben *demostrarse trabajando como un todo.* Esto incluye la verificación de la operatividad bajo las condiciones de operación extremas previsibles. Pruebas especiales solicitadas por el cliente, como por ejemplo "Surge Withstand Capability" (cfr. SWC, Ansi standard C37.90.1, 2002,

IEEE standard 472, 1974), deben realizarse a nivel de prototipo, **antes** de pasar a la etapa de producción, por los mismos motivos indicados atrás.

HAY QUE DUPLICAR EL AMBIENTE DE TRABAJO REAL o, en su defecto, llevar un prototipo (***"plan piloto"***) al sitio de trabajo final; lo primero puede ser muy difícil de realizar en sí mismo, dependiendo de las circunstancias, y en ocasiones genera un proyecto de mayor envergadura que lo que estamos haciendo. Suponga que va a instalar un equipo de control para una planta de la industria petrolera. Usted **no** puede simplemente llegar con su equipo, encenderlo, y ver qué pasa. Pero simular el comportamiento de la planta, y que su equipo interactúe con el simulador, a fin de verificarlo, pueden ser palabras mayores, muy mayores…

Un día me llamó una compañía de Arizona solicitando mi soporte técnico para instalar un sistema que habían desarrollado para la planta de Jose, Lagoven (subsidiaria de PDVSA en aquel entonces, cerca de Barcelona). Lo habían instalado y lo único que lograban hacer era: **¡** "shut-down" de la planta **!** (es decir, levantaban el sistema, y éste apagaba toda la instalación). Apagar una planta de procesamiento de petroleo, como se comprenderá, implica costos multi-millonarios en dólares. Desde luego, la empresa diseñadora NO había probado a cabalidad su sistema, pues carecían de un simulador de la planta. Eso siempre pasa, pero la solución no puede ser venir a probar en las instalaciones del cliente.

Un Alto Horno de una siderúrgica como Sidor, por ejemplo, NO puede apagarse jamás, pues hay que volverlo a construir. Así que no se permiten fallas en el sistema, que lo reinicialicen.

DOCUMENTACIÓN DEL SISTEMA

A NIVEL DE PRODUCTO:

- Descripción del sistema (Cómo opera; cómo lo va usar el cliente; cómo se lo va a manipular)
- Especificación funcional del sistema y subsistemas
- Documento de diseño
- Diseño de detalle
- Plan de pruebas
- Definición de módulos de software
- Listados de codificación comentados
- Manual de usuario (Cómo se instala; cómo lo mantiene)
- Recomendaciones y procedimientos de calibración
- Recomendaciones, frecuencias y procedimientos de mantenimiento preventivo
- Recomendaciones y procedimientos de mantenimiento correctivo
- Instrucciones de instalación
- Revisiones de código y de diseño de hardware
- Copias de ejecutables y ROMs

A NIVEL DE FABRICACIÓN:

- Especificación de Producción
- Lista de partes y subassemblies (kits)

A NIVEL DE CADA EQUIPO, FICHA TÉCNICA IDENTIFICANDO:

- Lote de producción
- Fecha
- Serial
- Modificaciones hechas después de control de calidad, justificando quién y por qué
- Base de datos de cableado y sus modificaciones, si las hubiera
- Reparaciones hechas después de instalación, justificando quién y por qué
- Versiones de hardware y de software; modificaciones hechas después de manufactura

INSTALACIÓN Y GARANTÍA

Puede ser que usted requiera instalar su sistema (*commisioning*), o no, dependiendo del tipo de proyecto. En todo caso viene luego una etapa de garantía y corrección de fallas en el sitio en donde finalmente estará operativo.

Nada de esto será reseñado aquí.

RESGUARDO DEL PRODUCTO

Los componentes fundamentales de cada producto, que permiten duplicarlo sin mayor esfuerzo, incluyen, en papel y en archivos de computadora (documentos legibles electrónicamente):

- Partes y piezas, absolutamente detalladas; con copia de las órdenes de compra que se realizaron
- Instrucciones completas de ensamblaje
- Diseños de AutoCad y planos de fabricación metalmecánica
- Bibliotecas de AutoCad empleadas
- Diseños de OrCad completos
- Bibliotecas de OrCad empleadas
- Archivos para fabricación de impresos
- Fotolitos
- Programas ejecutables, en ROM
- Fuentes de los programas
- Ambiente de desarrollo:
  - Compilador
  - Bibliotecas

La información, además, debería ser ***auditada***, como mínimo, en relación a su **consistencia**; es decir, que sin depender ni de una máquina en particular, ni de las personas directamente involucradas, un ingeniero independiente tiene que poder lograr, de manera autónoma:

- Que los códigos fuente generen correctamente todos los ejecutables (binarios idénticos a los que producen los ingenieros)

- Que los archivos OrCad y AutoCad generen exactamente los planos e instrucciones de manufactura respectivos
- Que los DOC produzcan los manuales y demás información técnica
- Que los Kits puedan comprarse en los suplidores indicados, a los precios estipulados, y con las identificaciones disponibles, en los plazos señalados en los documentos de "despiece" (system break-down).

Uno se encuentra con MUCHAS SORPRESAS: ***Lo anterior, casi nunca sucede...*** Como gerente de proyectos, primero, y luego como vicepresidente de investigación y desarrollo, siempre terminaban en mi escritorio los proyectos de aquellos ingenieros que decidían tomar rumbos fuera de la empresa. Y era con mucho esfuerzo que podían marcharse sin quebrantar el día a día de la compañía. Lograr la repetibilidad de un proyecto era **fundamental** para mi equilibrio psíquico y la estabilidad de la organización.

RESPALDO

Debe haber procedimientos de respaldo, responsabilidad de cada empleado, para:

- Cotizaciones
- Ofertas
- Programas
- Manuales
- Archivos de AutoCad y OrCad
- Bibliotecas

DETALLE (CHECK-LIST √)

Es común que en mis cursos de Proyectos les advierta a los estudiantes que su plan no puede ser una copia del cronograma de evaluación que sigue el profesor. Para poder hacer la planificación de las actividades, hay que saber qué va a hacerse, y estimar cuánto tiempo va a llevar (y en los proyectos reales hay que contestar el decálogo del ***El Pastelero***: Cuánto va a costar, dónde se realizará, cuáles recursos se emplearán, y así con todas ellas)

Es entonces evidente que si no logramos visualizar un proyecto estratificado en muchas partes (actividades, responsabilidades, etapas), no será posible tener un plan de trabajo adecuado. Así que siempre les pido que dividan el proyecto EN CIEN PARTES. Esto jamás ocurre... A lo máximo que llegan es por allá a veintitantas...

Para que vean que sí es posible, les anexo un ejercicio que hice, donde hay 86 ítems diferentes, todos susceptibles de una adecuada planificación. Esto, para un proyecto de laboratorio sin grandes pretensiones. Para uno real, la itemización requerirá centenares y hasta varios miles de actividades. Además, la versión que uso en la actualidad tiene 98 ítems; no la anexo porque es menos genérica.

Muchos de los siguientes apartados tienen que dar lugar a múltiples aspectos más para considerar, porque se aplican a diversidad de unidades, o secciones, que hay que trabajar por separado. En el diseño, cada apartado diferente es un ítem separado. Hay que comprender ***acerca de*** qué versa el proyecto y ***para qué*** será, y en cada aspecto establecer ***qué*** es con exactitud, ***quién*** lo realizará, ***por qué***, ***cuándo*** y con qué ***duración*** *(Pert)*; en ***dónde*** se ejecutará; ***cómo*** (los métodos, las técnicas, las tácticas y las estrategias que van a utilizarse), cuál será su costo (***cuánto***) y qué recursos se necesitarán (***con qué***).

Todo comienza con la **ESPECIFICACIÓN DEL SISTEMA** o producto que el cliente, tanto exterior como interior de nuestra compañía, debe suministrar (muchos proyectos son *internos*). Hay ocasiones en que aquel conoce sus necesidades pero no está en capacidad de generar la Especificación para un proyecto; en ese caso se atraviesa por una etapa anterior, en la cual un ente externo (asesor o compañía de ingeniería) analiza el problema y produce la especificación. Otras veces el cliente piensa que puede cambiar algún aspecto de su actividad, o emprender una nueva, pero no tiene idea de qué podría ser; entonces se retrocede más aún para estudiar la operación de la compañía y proponer alternativas para una nueva operación, lo que genera una lista de necesidades y de ahí la Especificación.

DISEÑO CONCEPTUAL:

√

1. Analizar y exponer diferentes *aproximaciones* conceptuales al diseño del proyecto (deben ser *VARIAS*; más de 2)Evaluar qué partes deben realizarse en Hardware y cuáles en Software. Explicar por qué (por ejemplo, podría resultar mejor generar algunas señales con osciladores del tipo 555 que con el micro; o podría ser imperativo usar conversores A/D externos, por razones de resolución o velocidad. También podría ser que NO hubiera que incluir ciertas piezas de hardware, como determinados filtros, si pudieran realizarse por software). Para cada aspecto del diseño (cada parte que usted incluya) *tiene* que haber una justificación explícita de por qué es mejor hacerla por hardware o por software, según su criterio

2. Deben incluirse consideraciones de tamaño y forma (características físicas, en general) para las secciones relevantes de su equipo (si éste tiene, por ejemplo, que caber en un espacio predeterminado)

3. Costo: Deben establecerse los límites impuestos desde afuera del proyecto

4. Cuando haya alternativas, deben determinarse las clases de Materiales que van a emplearse (tecnología de semiconductores; materiales especiales para la caja, pinturas, soportes, etc.)

5. Debe detallar los Sistemas de comunicación disponibles o que deben usarse: Email, FTP, RS-232 normal, infrarrojo, radio: Tecnologías WiFi, Zigbee, si fuera el caso

6. Alternativas de interconexión (entre equipos; con las interfaces; con la maestra: Lazo de corriente, fibra óptica) Cada unidad que deba interactuar con otras requiere una explicación y un razonamiento de por qué se va a establecer de una manera y no de otra

7. Análisis de las dificultades de la instalación: Ver cómo impacta el ambiente de trabajo (en el que se va a operar el sistema) en sus decisiones de diseño: Caja, soportes, aislamientos térmicos, pinturas, gabinetes presurizados, etc.

8. Tiempos de ejecución de las actividades: Comparándolos con los parámetros externos puede establecerse la cantidad de recursos que habrá que asignarse (ejemplo: Si determina que la programación va a tomar 3 meses, y usted tiene un solo mes para entregarla, necesitará 3 o más programadores. [Nota: En el caso del software, la relación NO es lineal])

9. Se realizarán los experimentos y las pruebas necesarios para definir aspectos indeterminados en la especificación. Ejemplos: √ Establecer cómo interfieren diversas unidades, como los conversores A/D, los temporizadores y las comunicaciones; √ Determinar longitudes de onda de los dispositivos infrarrojos y

ultrasónicos; √ Asegurarse de cuáles son con exactitud las diferentes frecuencias de modulación. √ Caracterizar los componentes que sean novedosos; apoyarse en sus hojas de especificaciones técnicas. √ Las potencias y frecuencias a las que se trabajará, ¿son seguras tanto para el usuario como para el ambiente?

10. El RESULTADO de los puntos considerados es el ***Documento de Diseño Conceptual*** del sistema, producto o servicio, que según su criterio resuelva el problema planteado en la Especificación. En la vida real, hay que obtener por escrito la aceptación del cliente: Éste es la oportunidad para resolver discrepancias, ***NO en el momento de la entrega***, meses después

11. Se genera el ***Cronograma de Trabajo***. Como toda proyección de lo que sucederá en el futuro, habrá que *adivinar*, pero mientras más *experiencia* tenga más precisos resultarán sus pronósticos. ¿Sabe usted ya, cuántas instrucciones por hora realiza? ¿Cómo puede, si no lo sabe, determinar el tiempo se le va a dedicar a un programa?)

DISEÑO FUNCIONAL: HARDWARE Y SOFTWARE

12. Descomposición de su diseño en Módulos (Funciones *específicas*). Incluir Diagramas que muestren las interrelaciones entre las diferentes partes

13. Puede ***refinarse*** el Cronograma de Trabajo original para incluir mayor lujo de detalles, ahora que ya se tiene la concepción funcional. Usted ***difícilmente*** puede *modificar* el cronograma. Cómo pudo saber antes, cuánto tiempo dedicarle a cada actividad, sin tener el detalle del proyecto? La experiencia y el buen juicio se lo dirá. Note que si se equivoca y necesita más tiempo del presupuestado, usted pierde dinero [usted paga los retrasos]; si ofrece cronogramas más largos de lo necesario, para cubrirse en salud, resultarán más altos sus costos y el proyecto se lo adjudicarán a la competencia. (This game is not so easy after all)

DISEÑO FUNCIONAL: HARDWARE

14. Definición completa, precisa, exacta, detallada, cuantificada, sin vaguedades, de:

15. CPU (características de velocidad, longitud de palabra, familia, cantidad de CPU's y función de cada uno). Analizar VARIAS alternativas y justificar su elección.

16. Si hacen falta, incluir memoria externa; definir cantidad y tipo.

17. Especificar el almacenamiento secundario (tipo: Escoger entre la diversidad de alternativas: Magnética, semiconductora, portátil, enchufable; identificar el tamaño. Justificación de todo)

18. Interfaces varias: Seriales, paralelas; cantidad y tipo (teclados, displays, sonido, impresoras…)

19. Módulos de conversión A/D: Número de bits requeridos (justificar), velocidad, necesidades especiales de filtraje; voltajes o corrientes que van a manejarse, necesidades de dispositivos de Sample&Hold; justificación por qué si, o por qué no, etc. Si se sobrepasan las características del micro seleccionado, y se requieren componentes externos, explicar por qué, y cómo se van a interconectar (I^2C, SPI, otros buses)

20. Módulos de entrada digital (cantidad, niveles de tensión o corriente de entrada que van a manejarse, filtrajes necesarios, aislamientos. Si se requieren componentes externos, explicar por qué, y cómo se van a interconectar)

21. Unidades acondicionadoras de analógicos (AIM): Filtros de entrada, amplificadores, aisladores analógicos, desacoples, mecanismos de conexión con los dispositivos externos; si se llegara a requerir independizar el sistema de cómputo (micro) de las señales del campo (externas), indicar qué mecanismos de interposición va a utilizar.

22. Unidades acondicionadoras de digitales (DIM): Filtros de entrada, aisladores ópticos, desacoples, mecanismos de conexión con los dispositivos externos; si llegara a necesitar independizar el campo y el sistema de cómputo (micro), indique qué mecanismos de interposición va a utilizar.

23. Unidades de interfaz con las señales analógicas provenientes del campo (AFI): Transductores, transmisores, etc. Clase y cantidad. Ver cómo se van a "muestrear" las señales, si se requiere que sean sincrónicas las muestras, y cómo se las va a adquirir en caso de que sean mucho más que las que puede manejar directamente el microcontrolador (este es siempre el caso en la vida real).

24. Unidades de interfaz con las señales digitales provenientes del campo (DFI): Tipo de dispositivos digitales externos (interruptores, semiconductores), ubicación (distancia), método de transmisión, cantidades (ver cómo se va a multiplexar una gran cantidad de interruptores externos, en caso de que haya que adquirir grandes cantidades de "status" (este también es casi siempre el caso en la vida real).

25. Determinar qué Aisladores galvánicos va a emplear para señales analógicas (ISOs) si su sistema lo requiere.

26. Optoaisladores para señales digitales (éstos son muy sencillos y económicos, comparados con los analógicos). Establecer con claridad la forma de alimentación que permita en realidad que unos dispositivos estén completamente aislados de otros. Recuerde que no puede haber nada en común; mucho menos la tierra!

27. Definición de Modems y otros mecanismos de interconexión, como radios, etc.

28. Otros optoacopladores (de potencia, para controlar señales externas; en las líneas

de comunicación, como los que se usan en los PC del Lab. C para evitar que se quemen las tarjetas madre). Cada punto que requiere, o no requiere, consideraciones de aislamiento, debe ser razonado. Por qué sí o por qué no, y de qué tipo.

29. Relés (capacidades de los contactos, tiempos de conmutación, "duty cycle", maneras de "mojar" los contactos)

30. Gabinetes (tamaño; norma NEMA, si aplica; capacidad de: Entrada, salida y control)

31. Previsiones de alambrado de señales de control, de comunicaciones, etc.

32. Señalar restricciones que limiten la agrupación de unas funciones con otras, en una misma tarjeta, o "backplane". Por ejemplo: Podría no convenir que se conectaran en la misma tarjeta los conversores analógicos con el control de potencia. O los radios con el micro, etc.

33. Indicar si pueden emplearse elementos suplidos por otros *(no todo hay que fabricarlo. A lo mejor fuentes, Modems, Computadoras, Tarjetas con Microprocesadores, etc., hay que comprarlos como subsistemas)* o si pueden emplear dichos elementos cuando han sido diseñados *previamente* en la compañía. Si fuera necesario modificar algunos elementos empleados por la compañía anteriormente para adaptarlos al proyecto actual, deberán señalarse aquí las diferencias y mejoras que hay que introducirles.

34. RESULTADO: ***Documento de Diseño Funcional del Hardware***.

DISEÑO FUNCIONAL: SOFTWARE

Debe incluir la descripción de las funciones de alto nivel (las que "ve" el usuario; estas deben ser aparentes de la especificación original):

35. Protocolos de comunicaciones (cómo va a transmitirse la información entre las diversas unidades que interactuarán entre sí; si los protocolos serán de la industria, o propios. *En este caso hay que incluir simulaciones que garanticen la integridad del protocolo, y la recuperación y control de situaciones anómalas, al estilo de los errores de comunicación, las reposiciones inesperadas del hardware (reboots), interferencias, etc. No es nada trivial diseñar un protocolo...*

36. Cantidad de información que va a transmitirse y su velocidad. Estos parámetros tienen que **cuantificarse**. En caso de que haya alternativas, explicar la selección (por qué)

37. Categorización de las informaciones; por ejemplo: Si hay "analógicos rápidos", o señales analógicas que necesiten ser adquiridas a alta velocidad y con tiempos muy precisos entre muestras (sincrónicas); si hay analógicos lentos, que puedan leerse a conveniencia del micro, si además del "scan normal", en donde a lo mejor se procesan solo valores que hayan cambiado significativamente, también amerite

incluir "scans de seguridad", que lean informaciones viejas a pesar de no haberse detectado cambios significativos en ellas; con qué frecuencia. *La estrategia precede a la funcionalidad*

38. Capacidades de control: Qué señales deben manipularse. Características de dichas señales. Si son SCRs, relés, señales optoacopladas. A qué velocidad; cuánto debe durar cada pulso de control. Cómo será la secuencia para encender o activar un punto en el campo, y cuál para cerrarlo. *Algo que no puede tolerarse es que una falla en su dispositivo produzca* ***activaciones indeseadas*** *en las señales de control. Garantizar esto es de crucial importancia, porque lo que sí está claro es que ¡ su sistema fallará ! (todos lo hacen). Así que en una industria, el manejo de las señales de control no es tan sencillo como colocar activadores en el micro (relés, SCR's), y listo. Nada de eso. Se necesita garantizar, mediante procesos dinámicos, que una falla en su equipo no activará inadvertidamente las señales que él controla*

39. Funciones de diagnóstico: Dentro del ciclo infinito del microcontrolador pueden y deben incluirse lecturas que permitan saber si las unidades externas están funcionando: Si una fuente se cayó, si algún dispositivo de comunicaciones falló, o la computadora externa, etc. Es importante establecer la ***estrategia***: Qué va a hacer su sistema en caso de que alguna de estas cosas suceda. Si no se le ocurre qué hacer, es preferible no programar nada! Pero normalmente un diseño tiene que prever qué acciones tomar cuando algo falla; al menos las cosas previsibles. Y la alternativa tiene que ser inteligente, y razonada, para cada cosa que se prevea que puede fallar

40. Interfaz con el usuario (MMI): Cómo interactuará el usuario con su sistema: Teclado, voz, pantalla, bombillos o leds, otros mecanismos (controles remotos), etc. Qué funciones debe incluir en su sistema para manejar la interacción con el usuario (lectura de caracteres, por interrupciones o vía "encuesta" [polling]; determinación de buffers circulares, tamaño, previsión de overflow y acciones consecuentes, diversas rutinas de conversión; manejo de CRCs u otros mecanismos de control de error)

41. Interfaz de Mantenimiento. Hay funciones, diferentes a las normales, que le permiten al encargado del servicio realizar las labores de mantenimiento. Hay que establecer cuáles son esas funciones, y cómo operarán

42. Funciones de respaldo, en caso de que haya necesidad. La información adquirida, y no transmitida, puede ser preciosa. Por ejemplo, si el sistema fuera un contador de personas para el Metro de Caracas, y al final de la tarde hubiera un apagón, no por eso usted puede darse el lujo de "perder" la cuenta de cuántos usuarios habían entrado a los trenes

43. Funciones de cálculo específicas (FFT, etc). Hay que establecer todas las que necesite; identificarlas, e incluirlas dentro de las actividades que hay que realizar.

44. Necesidades de multitarea, spooling, comunicación en back-ground, etc. Debe incluir un pequeño sistema operativo? Debe comprarlo, o hacerlo como parte del

proyecto?

45. Diagramas de despliegues. Si algo tiene que mostrarse en la pantalla, aquí hay que incluir un diagrama, o foto, de dicha pantalla

46. Editores, menús, reportes. Si algo va a imprimirse, aquí tiene que mostrarse un facsímile de lo que va a imprimirse. Obsérvese que los facsímiles se incluyen ANTES de haber generado una sola línea de código para producirlos

47. Identificar programas que van a ser modificados (en caso de que ya la compañía, o usted, haya hecho algunos similares con anterioridad), o creados nuevos.

Funciones y actividades que el usuario no ve, pero que se incluirán en esta etapa son:

48. Rutinas de autodiagnóstico:

49. Sistema de Detección de fallas; definir qué estrategia deberá seguirse y qué acciones se tomarán en caso de detectarse fallas: Reportes, señales, alarmas, etc.

50. Cada posible falla detectable deberá tener su especificación correspondiente. Como de costumbre, no tiene sentido incluir una rutina de diagnóstico, si no está claramente especificada qué acción inteligente y apropiada deberá adoptarse en caso de detectar una falla. Normalmente, ¡ ***apagar el equipo NO es lo indicado*** !

51. Indicar si hay funciones que han de operar en diferentes CPU's (Pentium, 8031, HC08, etc.). Proponer qué estrategia se va a emplear en su elaboración (diseño independiente para c/u; una misma copia del programa, que se compile y funcione en c/u, etc)

52. Indicar si pueden emplearse rutinas suplidas por otros (modificándolas o no): Spoolers de impresión, rutinas estándar de comunicación Xmodem, FFT, etc. Tener en cuenta, no el desarrollo, pero sí su incorporación e integración al resto de las rutinas y programas.

53. Indicar si pueden emplearse rutinas codificadas previamente, modificándolas (señalar las diferencias que hay que introducirles) o sin modificación.

54. Identificar qué funciones van a realizarse como multitareas (como impresión de reportes, adquisición de información, procesamiento y transmisión de datos, *simultáneamente*). Las decisiones varían si no se tiene necesidad de tiempo real para la adquisición y almacenamiento de la información, o si la actividad primordial es la de adquisición de información en tiempo real; en este caso, las demás actividades deben supeditársele.

55. Evaluar si habrá cambios fundamentales en el software estándar de la compañía (por ejemplo, en las rutinas de tiempo real, que las empresas suelen desarrollar,

probar y comercializar en gran cantidad de productos y proyectos); en tal caso, debe seguirse un proceso muy riguroso (*si están funcionando y probadas, RESÍSTASE A MODIFICARLAS*)

56. RESULTADO: ***Documento de Diseño Funcional del Software***.

DISEÑO DE DETALLE

Hasta aquí NO HEMOS ESCRITO NI UNA LÍNEA DE CÓDIGO, NI HEMOS ARMADO CIRCUITO ALGUNO...

DISEÑO DE DETALLE: HARDWARE

*Debe incluir un **Diseño de Implementación**, que muestre en el Hardware:*

57. Las relaciones entre las diversas tarjetas y módulos

58. La definición de las Interfaces entre ellos (cómo se van a conectar unos con otros)

59. Debe mostrarse el funcionamiento de tarjetas y módulos mediante **Diagramas Electrónicos**.

60. Hay que identificar todos los componentes, sus proveedores principales y los secundarios (*second sources*), poniendo especial cuidado en no olvidar los indispensables *Condensadores de Desacople para los ICs: De qué tipo, de qué valor; ubicación*

61. Se seleccionarán los Condensadores de Desacople para las Fuentes y el sistema de baterías.

62. Se detallará el diseño y disposición de los diferentes Blindajes y el Cableado; se hará la determinación y ubicación de los Planos de Tierra, para la apropiada anulación del Campo Eléctrico. Se explicará cómo se va a cablear, aprovechando los Planos de Tierra y el Chasis para eliminar la radiación entrante y saliente.

63. Se identificará qué cables serán entorchados, apantallados, etc.

64. Se identificarán los Circuitos impresos de múltiples capas, indicando en cada caso cuántas capas serán, y el propósito. Se señalarán dónde se manufacturará cada tarjeta (y los costos).

65. Definir dónde irán cables planos; su clase, el tipo de conector que llevarán, etc.

66. Se identificarán las diversas impedancias características, en los sitios que lo ameriten y se indicarán las apropiadas terminaciones. Deben anotarse aquellos sitios en que no sea necesario terminar o acoplar las impedancias.

67. Identificar con absoluta exactitud y claridad, para aquellos pines del microprocesador que deben reprogramarse como ***salidas***, cómo va a garantizar

que todo funcione perfectamente ***entre el momento en que se alimenta el micro y el instante en que las señales pasan de ser entradas a ser salidas.*** Qué ocurre si el micro se "guinda" después de alimentarlo, pero antes de programar perfectamente las salidas? No puede permitirse que haya cortos circuitos entre componentes externos, ni siquiera durante intervalos muy cortos, porque si en el ínterin llegara a congelarse el programa, el corto circuito destruirá los componentes

68. RESULTADO: ***Diseño de Detalle de cada tarjeta***, que permite la generación de los PCB (printed circuit boards) y la manufactura completa de las mismas.

69. **Anexo A**: **Pruebas para verificar la operabilidad de cada tarjeta**. Deben ser completas y describirse en profundidad, para permitir su empleo en otras etapas, como control de calidad, pruebas de aceptación (FAT), etc. Debe incluirse la descripción de los Equipos de Prueba necesarios; especificar si serán alquilados o comprados; incluir también los "Programas de ***Calibración*** y Prueba", que servirán luego en la etapa de producción. En su defecto, hay que anexar el procedimiento manual, completo, que hay que seguir para las pruebas. Debe definir cuáles (y cuántos) serán dicho programas para el Diagnóstico del Hardware (Hardware Diagnostics): El micro, sus bloques funcionales, las diversas entradas a cada dispositivo, los motores, los transmisores y receptores, los acopladores. Casi todo el hardware que vaya a incluir necesitará programas de diagnóstico. Definir también cuáles piezas del equipo NO requerirán estos programas, por qué, y en ese caso cómo se probarán.

    Indicar también qué Programas de **Calibración** va a necesitar (en los conversores, temporizadores, transmisores y receptores, sensores) Deberá indicarlas qué piezas no necesitarán estos programas, con su justificación (se calibrarán a mano?, por qué? ¿no necesitan ajuste?; etc)

    La estrategia de Depuración (debugging) del proyecto: Hardware Y Software hay que establecerla con antelación; *si no sabe cómo lo va a probar, probablemente no vale la pena ni comenzar a programar, o armar.*

70. **Anexo B**: Se identificarán las Indicaciones de Error (leds encendidos, sirenas, etc.) que sean producidos por cada tarjeta cuando se detecte una falla en ellas. La descripción debe ser completa y detallada; por ejemplo, si hay luces que parpadean a diferentes velocidades, o hacen beeps de diversas formas, etc.

DISEÑO DE DETALLE: SOFTWARE

Incluye un Diseño de Implementación, que muestra en el Software:

71. Las relaciones entre las diversas rutinas, mediante un diagrama apropiado (En proyectos de tamaño considerable, puede estudiarse la necesidad de hacer una

descripción empleando UML: Universal Modeling Language)

72. Definición de variables (locales, estáticas, externas; nombres, significados, uso; "include files")

73. Selección del lenguaje en que se hará el desarrollo (C, assembler, Java, etc.)

74. Diagrama de flujo y posiblemente listados de pseudo código

75. RESULTADO: ***Diseño de Detalle de cada función***, que permita realizar la codificación de cada una, y cada subrutina, módulo y programa de los subsistemas.

76. **Anexo A**: Pruebas para verificar la operabilidad de cada función. Deben ser completas, describirse en profundidad y deben anexarse los Programas de Prueba que permitan constatar la operación de cada función, si fueran necesarios o, en su defecto, el procedimiento completo que hay que seguir para las pruebas.

77. **Anexo B**: Incluir todos los mensajes de error y todas las otras señalizaciones de error (leds encendidos, marcas en la pantalla, etc.) que sean producidos por cada función de software ante la detección de una falla interna.

VARIOS (PELIGROSOS) EN LA ETAPA FINAL

78. Llegada la etapa de producir el dispositivo (probablemente un prototipo), si se llegan a encontrar problemas en el Hardware (ruidos, imprecisiones, inestabilidades), que pueden ser corregidos por Software (promediación de señales, filtrajes, etc. *la vida real es así*), tienen que quedar explícitas esas correcciones en la documentación del proyecto (bitácora) y ser aprobados por el Gerente de Proyectos (o por el profesor, en caso de nuestros cursos)

79. Si se llegan a encontrar errores en el Software, y usted piensa que se pueden corregir por Hardware (haciendo, por ejemplo, "reboots" mediante Watch Dog Timers, etc.) tienen que quedar documentados de manera explícita en la bitácora del proyecto y ser aprobados por el Gerente de Proyectos de la compañía (o el profesor del curso)

80. La puesta en marcha de equipos **sin** la inclusión de funciones originales (por ejemplo, sin activar las protecciones, o el "watch-dog timer", sin las rutinas de diagnóstico, sin algún componente mecánico), tiene que ser igualmente aprobada.

PRUEBAS DE HARDWARE Y SOFTWARE: PROTOTIPO

81. Planes detallados y completos. Se pretende evitar problemas (rutinas de interrupción defectuosas) y demostrar la funcionalidad (ver anexo a continuación, pruebas tomadas del IEEE Standard for Software Test Documentation (Std 829-

1998)

PRUEBAS DE HARDWARE

82. Evitar que en producción se produzcan parches, techitos, "piggy-back's", cablecitos, producción de circuitos impresos nuevos, etc., medidas que a ese nivel hay que realizárselas a toda la producción

83. De ninguna manera la verificación del diseño se realizará en el sitio del cliente: Se hará en el laboratorio.

PRUEBAS DE SOFTWARE

84. Evitar "parches".

PRUEBAS DEL SISTEMA

85. Los Sistemas nuevos deben demostrarse trabajando como un todo. Esto incluye la verificación de la operatividad bajo las condiciones de operación extremas previsibles. Pruebas especiales solicitadas por el cliente, tales como Surge Withstand, etc., deben realizarse a nivel de prototipo, antes de pasar a la etapa de Producción.

86. HAY QUE DUPLICAR EL AMBIENTE DE TRABAJO REAL o, en su defecto, llevar un prototipo ("piloto") al sitio de trabajo final. **TOMEN NOTA:** Sistemas de control a lazo cerrado tienen que probarse ANTES de incorporarlos a una planta o fábrica; de lo contrario, los errores pueden ser devastadores. Pero muy probablemente haya que hacerse un dispositivo que *simule* la planta (lo cual puede ser un proyecto en sí mismo, hasta más complejo que el proyecto original). *Está descartado que un ingeniero incorpore un sistema de control a lazo cerrado en una planta, sin haberlo probado antes, simulando el proceso.* Debe definir con exactitud cómo será su System Exerciser

Tomado del: IEEE Standard for Software Test Documentation (Std 829-1998).

TESTING

- ***Component Testing***: Conducted to verify the implementation of the design for one software element (e.g., unit, module) or a collection of software elements. Sometimes called unit testing. The purpose of component testing is to ensure that the program logic is complete and correct and ensuring that the component works ***as designed***.

- ***Integration Testing (System Testing)***: Testing conducted in which software, hardware or both elements are combined and tested until the entire system has been integrated. The purpose here is to ensure that ***design objectives are met*** and that the software, as a complete entity, complies with operational requirements

- **Conversion Testing**: To ensure that all data elements and historical data are converted from an old system format to the new system format.

- ***Job Stream Testing***: Testing to ensure that the application ***operates*** in the production environment.

- ***Interface Testing***: Testing done to ensure that the application operates efficiently and effectively outside the application boundary with all interface systems.

- ***Security Testing***: Testing done to ensure that the application systems control and auditability features of the application are functional.

- ***Recovery***: Testing done to ensure that application ***restart*** and backup and recovery facilities operate as designed.

- ***Performance***: Testing done to ensure that the application performs to customer expectations (response time, availability, portability, and scalability)

- ***Regression*** Testing done to ensure that applied changes to the application have not adversely affected previously tested functionality

- ***Acceptance Testing***

  Testing conducted to determine whether or not a system satisfies the acceptance criteria and to enable the customer to determine whether or not to accept the system. Acceptance testing ensures that customer ***requirements' objectives are met and that all components are correctly included*** in a customer package.

- ***Beta*** Testing done by the customer, using a pre-release version of the product to verify and validate that the system meets business ***functional requirements***. Detect application faults, failures, and defects.

PASS / FAIL CRITERIA

Specify the criteria to be used to determine whether each item has passed or failed testing.

- **Suspension Criteria**

  Specify criteria to suspend all or a portion of the testing activity on test items associated with the plan.
- **Resumption Criteria**

  Specify the conditions that need to be met to resume testing activities after suspension. Specify the test items that must be repeated when testing is resumed
- **Approval Criteria**

  Specify the conditions that need to be met to approve test results. Define the formal testing approval process.

ERRORES COMUNES

*Verifique **al menos** los siguientes aspectos:*

1. LEDs sin resistencias
2. Código de resistencias (valores de condensadores)
3. Capucha Punta Osciloscopio (¡ no usar la punta para "clavarla" en el protoboard !)
4. Osciloscopio mal manejado: √Punta atenuadora; √descalibración
5. Armar el circuito y encenderlo de una (*porsia...*) Así van a quemar el micro...
6. Síndrome del "*empty main()*": ¡ Todo en sus programas está "*pegado*" de las interrupciones !

CUANDO TODO FALLA

1. Tenga a mano diagramas: Bloques, lógico y circuital; esté seguros de la ubicación
2. Asegúrese de haber hecho la simulación
3. Fundamental probar la circuitería (protoboard) ANTES de conectar el micro. Generar a mano las señales que provendrían del micro; así saben si los LEDs encienden o no. Cuando el circuito externo esté bien, puede conectarse el micro
4. En el peor caso: Quite TODO y comience paso a paso: Sólo el micro, con un programa viejo (¡ que seguro funcione !). Añadir luego una entrada y verificar que la leamos bien. Si ni siquiera eso funciona, estamos MUY MAL, pero es fácil verificar qué falla: La fuente está apagada o desconectada; hay resistencias *en serie* con la alimentación del chip; el osciloscopio tiene la entrada apagada...
5. Método Incremental: *Un pequeño paso; prueba, y verificación... repita*
6. "Perder los archivos" ¡ NO ES EXCUSA PARA RETARDAR EL PROYECTO !
7. "No puedo leer el floppy, o el Pendrive, con mi proyecto": TAMPOCO
8. "Un virus dañó el disco duro, con todo y proyecto": MENOS
9. Perdí el Pendrive; lo mojaron en la lavandería ¡!
10. El cuidado con la información hace referencia a los programas, y a los circuitos impresos (Orcad), diseños metalmecánicos (Autocad), toda la documentación (cliente e interna: MSWord, Excel, MSProject), notas de entrega, facturas, etc.

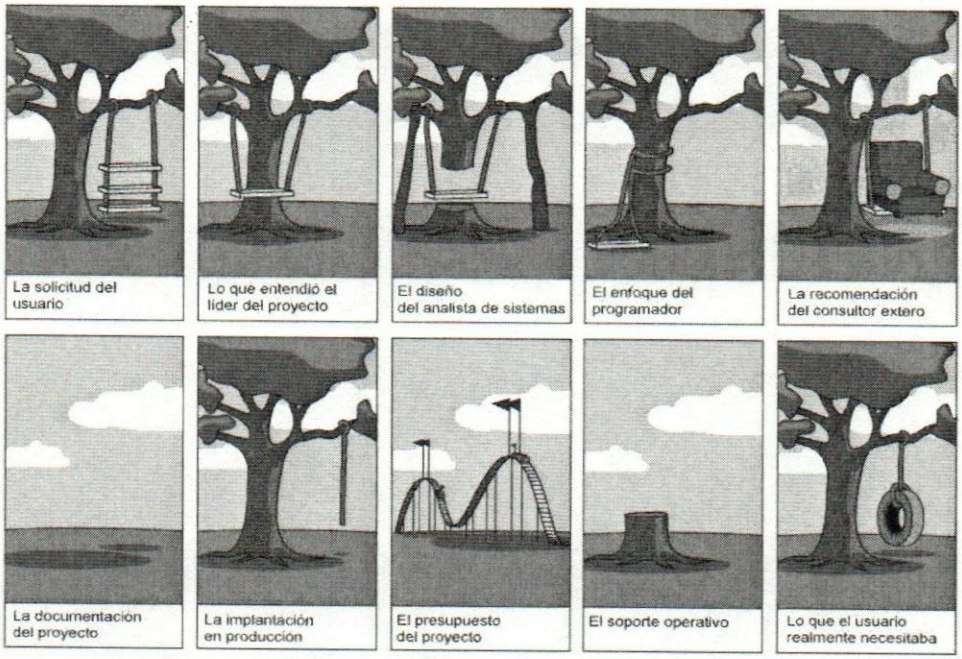
ESTÁNDARES

*IEEE (están **todos** en la biblioteca de la USB):*

- AnsiC37.90.1-2002(IEEE-472,1974).pdf: Surge Withstand Capability (SWC; hardware)
- AnsiC37.90.2-2004(IEEE-472,1974).pdf: Surge Withstand Capability (SWC; hardware)
- IEEE1008(SoftwareTesting).pdf
- IEEE12207(Software).pdf: Software life cycle processes
- Normas Nema (Nema 12: Interiores; Nema 4X:
- **Cada aspecto del diseño electrónico tiene un estándar que lo cubre, procedimientos internacionales que hay que cumplir. Usted debe estar atento de ellos: IEEE, ANSI, IEC, UL...**
- Y cada país tiene los suyos. La normalización en Venezuela es **Covenin**.
- Además, ISO-9000... para producción. Nada entra a Europa sin ISO-9000. Mi opinión en relación a ese estándar, me la reservo por el momento.

COMUNICACIÓN

El problema más difícil en una relación humana, y los proyectos no escapan a él, es la **comunicación**.



El cliente cree que dijo con toda claridad lo que quería, el analista entiende algo muy distinto, el programa difiere bastante de lo que se pide; en fin: Aquí solo se puede rozar el tema, pues ni está resuelto, ni hay espacio, pero esta gráfica es una manera ilustrativa y muy simpática de representarlo.

FLUJO DE CAJA: ZERO CASHFLOW

Una técnica muy empleada para hacer la **Programación Financiera** es la del "**Zero Cash Flow**", que consiste en lograr que quien desarrolla el proyecto no desembolse **NADA** de su propio dinero a lo largo de todo el trabajo. Esto es ***fundamental***, sobre todo en compañías pequeñas que no tienen suficiente capacidad financiera, o que no cuentan con un historial crediticio. Para lograr un "Zero Cash Flow" se debe establecer, en cada ítem importante del proyecto (***milestone***), el momento en que el cliente va haciendo pagos para cubrir cada etapa. Por ejemplo: Se da **comienzo** al proyecto cuando se ha recibido la **Cuota Inicial** ("down payment"). Llegado el momento de ordenar la compra de insumos (computadoras que van a instalarse, periféricos, etc.), se debe hacer otra aportación, y así sucesivamente. Los clientes corporativos (sobre todo si son oficiales) suelen no entregar dinero en anticipo sin una ***garantía***; aparecen entonces las ***Fianzas*** (Bid bond, o fianza de licitación; están también las de fiel cumplimiento, la de anticipo, fianza de garantía, etc.), que son ***avales*** que, por ejemplo, las Compañías de Seguros otorgan al cliente, a quien para cobrarse o resarcirse (recuperar su anticipo) sólo le basta con indicarle a la aseguradora que quiere ***Ejecutar La Fianza***; no tiene que hacer grandes justificaciones de por qué. (La aseguradora con toda certeza procederá a ***embargar*** al suplidor para a su vez recuperar su dinero, probablemente ejecutando algún documento que le habrá firmado previamente para ello: *Hipoteca, giros, letras de cambio, "**side letters**"*, etc.)

En cada etapa se vence una fianza y hay que entregar una nueva para continuar. Una fianza de licitación garantiza la seriedad de una oferta (que si le otorgan el proyecto a cierto suplidor, éste estará ahí para ejecutarlo; se hace porque el Cliente no desea perder tiempo evaluando propuestas que no sean "en firme"). Con la inicial va la correspondiente Fianza de Anticipo. Hay fianzas de Fiel Cumplimiento, de Garantía y más.

Es importante notar dos cosas: Si no puede convenirse con el cliente una estructura de pagos del tipo "Zero Cash Flow" (en algún momento se prevé que faltará dinero), ***hay que garantizar el financiamiento externo!!!***, probablemente con un banco. Es **importantísimo** considerar los intereses como "costos" del proyecto, que hay que cargárselos al cliente; de lo contrario, será el banco quien haga las ganancias y, el Suplidor... sólo el trabajo.

Proporciones: En relación con este tema financiero, es importante observar las proporciones que a veces ocurren con algunos proyectos o trabajos. Recuerdo haber emprendido varios viajes a Puerto Ordaz, por ciertos trabajos de asesoría, y caer en cuenta de cuánto estaba costando el pasaje en avión (Bs. 100.000 en su época), el hotel (Bs. 160.000 al día), alquiler de vehículo (Bs. 60.000 diarios), restaurantes, etc. ¿Y cuánto estaba cobrando yo por mi labor? Quienes realmente ganaban con mi trabajo eran ***Avensa (en esa época), el Intercontinental y demás compañías "accesorias"!!!***. Y a pesar de que el cliente estaba dispuesto a reconocer esos gastos, mi propia ganancia quedaba ***eclipsada*** ante los demás desembolsos. Hay que ver si eso tiene sentido, o no...

FAST PROTOTYPING

Como acabamos de ilustrar, uno de los principales y más graves inconvenientes en los proyectos (y en la vida, en general) son los problemas de comunicación. Una cosa creemos nosotros que el cliente (***el profesor***) pidió, y otra ***muy distinta*** está pensando él. Así, ¿cuántas veces no vi en mi vida profesional, la culminación de un proyecto, para encontrar que en el **acto de entrega** el cliente indicaba que ***no era eso lo que había solicitado***?

Una forma de lidiar con este problema en Proyectos de Desarrollo que pueden durar varios años, consiste en realizar a la mayor brevedad posible un "**prototipo**" del mismo, que permita al cliente ver y ***aprobar*** el funcionamiento del sistema. Se requiere validar tanto el Hardware como el Software. El cliente podrá ver cómo serán las pantallas con las que se interactuará con los programas; cuáles los campos de información que hay que llenar, en qué orden; cuánto se ayudará con el mouse y cuánto con el teclado, etc. En relación al Hardware, podrá verse también cuánto espacio se requerirá, cómo se instalará, si cabrá en el sitio previsto, o si las puertas y demás lugares de acceso están donde sean más convenientes. Un prototipo presentado con la mayor antelación ***no garantiza*** que no se encontrarán problemas y malentendidos después, **pero** es realmente una ayuda invaluable.

Como parte del ejercicio puede acordarse con el cliente que acepte: Reportes, pantallas, procedimientos..., firmando un acta y los ***mímicos*** de los reportes, etc., de tal manera que al finalizar el proyecto (y cuando probablemente la persona que lo vaya a recibir ***sea otra***, porque la que contrató el proyecto acaba de renunciar, o fue ascendido...), se tenga *al menos* respaldo escrito, aval del trabajo que se ha hecho.

Son innumerables las oportunidades en que encontré este problema: Una persona recibiendo (o ***desaprobando***, mejor dicho) un proyecto que, al aplicar su criterio de cómo debía funcionar algo, o qué debería habérsele entregado..., no coincidía con lo que se había producido, tras largos, arduos y costosos años de trabajo...

QUE FUNCIONE

Desafortunadamente para el ingeniero de diseño, tanto el cliente como sus jefes tienen una visión sesgada del estado del proyecto de usted (y sus profesores también); ellos ven su desarrollo de la siguiente manera: Suponga que usted ha resuelto perfectamente el problema de la alimentación eléctrica; la parte del aislamiento de las señales, la amplificación, el filtro de orden "n", la conversión de la señal; la RAM funciona impecablemente; usted probó en forma independiente el protocolo de comunicaciones; los interruptores para realizar la interfaz con el usuario en su equipo están operativos... Pero si el cliente, nuestros jefes o profesores, **no** ven que ***algunos*** datos del campo van y vienen entre el equipo y el PC, **ni** observan que se ***presente información*** en la pantalla, aun cuando ésta sea un poco maltrecha; si ***no hay movimiento*** de luces en el equipo..., su percepción será que al proyecto le falta una eternidad, ***que usted se encuentra estancado***... Su calificación ***instantánea*** en el curso será mínima.

Si, por el contrario, el ingeniero emplea la ***fuente del laboratorio*** en vez de su propio sistema de alimentación de energía; ***la señal aún no tiene aislamiento***, ***ni*** conformación ***ni*** amplificación (se inyectan valores de 0 a 5 voltios, provenientes –no de termocuplas– sino del generador que tenemos a nuestra disposición); si la comunicación ***aún no sigue el protocolo***, sino que es un vaciado a ciegas de la información; si ***la velocidad*** de transferencias es de 9600 bps porque usted ***no*** ha logrado vencer la barrera del sonido; si ***no se verifican*** ni paridades ni CRCs..., **la percepción de todo el mundo será que el proyecto va bien encaminado** ("***e pur se muove***", como decía Galileo), y que sólo faltan los "accesorios". ¡ Su calificación instantánea en la asignatura estará por encima del mínimo ! y posiblemente a usted realmente le falte mucho más que al personaje del ejemplo anterior. ¡ No importa ! Viendo tanta actividad, el cliente y su jefe estarán tentados a flexibilizar su cronograma.

Esta es una particularización del "**Fast Prototyping**" del que hablé con anterioridad. No lo olvide, aunque, probablemente, sus profesores del Laboratorio no flexibilicen nada...

En todo caso, una buena recomendación para todo proyecto consiste en que Usted siga estas tres fases:

- Primero, **que funcione** (aunque cojee, tal como acabo de describir)
- Luego, que funcione **bien** (que la información sea real, las comunicaciones también, etc.)
- Finalmente, que funcione **de manera óptima** (que no se usen las fuentes del laboratorio, que los filtros sean de verdad y que el proyecto cumpla con todo lo que se pidió)

Es harto difícil demostrar que filtros y amplificadores, optoacopladores y micros están todos bien, y que lo único que falta es integrarlos.

Una integración progresiva calma los nervios.

KISS

Edsger W. Dijkstra, famosísimo computista contemporáneo de apellido impronunciable, aconsejaba a su séquito en relación a su método preferido para lograr que inmensos proyectos de programación culminaran a satisfacción; el método **KISS**: "**K**eep **I**t **S**imple, **S**tupid!".

En la vida real, **todos** se oponen al principio **Kiss**. El cliente cree que quiere cosas complicadas; el gerente de ventas se empeña en agregarle características ("*features*") al proyecto, que ya casi no cabe en el micro; y a muchos de mis ingenieros (usted mismo) no le da nota hacer cosas sencillas. Un proyecto simple se convierte en un gran desarrollo, por el placer de complicar. Luego, obviamente no funciona, y hay que comenzar la dolorosa depuración (debugging).

Hoy en día en mis laboratorios de proyectos siempre pido que cada grupo agregue al esquema básico que propongo, algo que denominamos "la aplicación". Y en cuántas me veo para lograr que pongan los pies en la tierra. Todos quieren llegar a la luna, en el lapso de un curso...

TEMAS VARIOS QUE SON "PURO ORO"

CONCEPTOS BÁSICOS DE ELECTRÓNICA.

ARMADO DE CIRCUITOS.

Debo decir que siempre detecto fallas fundamentales en aspectos técnicos BÁSICOS relacionados con el ARMADO de los circuitos, que PUEDEN ser los culpables del funcionamiento incorrecto en sus anteriores proyectos, y lo serán también en el futuro, si no los subsanan.

Más o menos la mitad de los alumnos de digitales y demás proyectos de electrónica manifestaron no haber ni oído sobre la necesidad y el uso de los Condensadores de Desacople! No tiene mucho sentido conceptualizar un proyecto, por ejemplo, de digitales, perfectamente ceñido a los cánones lógicos y de programación, y luego encontrarse con que el prototipo presenta fallas aleatorias, que nunca terminan por individualizarse.

No tiene mucha gracia hacer bien un diseño y luego omitir el uso de condensadores de desacople al armarlo, o traer la alimentación desde la fuente hasta el Protoboard mediante un famélico cablecito #22, y pretender que todos los integrados reciban un apropiado Vcc!!!

Denle una mirada con detenimiento a la sección que llamé "**Lab0** ", hoy y después... Esa es "La Biblia". Aunque no fue redactado específicamente para nuestros Laboratorios de digitales y de proyectos, y por tanto existen muchas lagunas en él en aspectos que no considera, como por ejemplo en la parte analógica..., por lo menos puede servirles parcialmente como lista de verificación de algunos conceptos que necesitarían aprender.

Hago el comentario de que a lo mejor a su profesor de digitales se le pasó explicarles a tiempo la importancia de aspectos tales como el de los Condensadores de Desacople, pero tampoco son ustedes muy proactivos; les hubiera bastado con abrir cualquier libro del área, como el Wakerly, que ha sido el texto designado desde hace ya un buen tiempo, y habrían encontrado qué son, para qué se aplican, cuál es su importancia, etc.

En la vida profesional a usted no lo van a evaluar por lo que los profesores dejaron de enseñarle, ni eso le servirá de atenuante cuando desconozca un concepto fundamental.

Usted siempre será el responsable directo, único y exclusivo de lo que ignora.

BLINDAJE Y CABLEADO

Revisen también los conceptos sobre:

- Plano de Tierra,
- Cargas Virtuales,
- Anulación de Campo Eléctrico mediante Plano de Tierra,
- Anulación de efectos del Campo Magnético mediante Plano de Tierra,
- Modo de cablear aprovechando un Plano de Tierra o el Chasis,
- Uso de cables entorchados, etc.,

que son aspectos en los que, según he podido observar, suelen tener lagunas serias nuestros estudiantes. Espero que apliquen estas técnicas en el armado de sus proyectos.

Si necesitan alguna ayuda en relación a estos temas, consulten con sus profesores de electrónica.

Les quedan muchas otras cosas por considerar, tales como:

- circuitos impresos de múltiples capas,
- cables planos,
- impedancias características y terminación, etc.

EXPERIMENTACIÓN

Cuando ya se está haciendo un proyecto, EL INGENIERO NO PUEDE IGNORAR ningún aspecto técnico fundamental en relación a él. Pero, antes de comenzar, a lo mejor uno desconoce muchas cosas.

Cuando algo no se sabe, lo importante es IDENTIFICARLO, y APRENDERLO.

Una parte importantísima del aprendizaje se obtiene de la EXPERIMENTACIÓN. Suponga que usted quiere determinar, por ejemplo, si un cierto protocolo de comunicación interferirá con la forma como ha decidido tomar las muestras del conversor análogo a digital. Sin dejar del todo el análisis concienzudo del problema, lo mejor es que se haga un pequeño experimento ¡ y salga de una vez de dudas !

Recomiendo que hagan UNA CANTIDAD APRECIABLE DE EXPERIMENTOS, **ANTES** de hacer la propuesta de sus proyectos, para permitirles conocer con certeza aquellos aspectos que ahora puedan considerar oscuros.

QUÉ CHIPS USAR

Me ha sorprendido que no tengan acceso electrónico a *algún* manual APROPIADO de circuitos digitales. Ya que es así, déjenme decirles que TODOS fabricante importante (National Semiconductors, Fairchild, Analog Devices, Burr Brown, Texas Instruments, etc.) tienen publicados manuales (PDF) que son joyas, realmente. Además, NO VENDEN su mercancía si no tienen un manual así, y NO VENDEN tampoco si no tienen NOTAS de Aplicaciones (Application Notes), en donde llevan de la mano al ingeniero para que use sus chips.

Les recomiendo, en particular, el "LOGIC Pocket Data Book" de la Texas Instruments, que es un compendio de *TODOS* los chips digitales que se usan actualmente; sus especificaciones técnicas, la distribución de los terminales, etc.

Les recomiendo bajarlo (6.4 Megabytes) para su uso personal. Les será muy útil.

http://www-s.ti.com/sc/psheets/scyd013/scyd013b.pdf

NOTA: De tanto en tanto parece ser que el webmaster de la Texas se queda sin coger oficio y, por ocioso, no se le ocurre mejor idea que la de mover ligeramente de sitio los documentos de su 'site', lo que hace que los links queden obsoletos. Sin embargo, siempre termino localizando el documento haciendo una búsqueda por la palabra "**pocket**".

En segundo lugar, valdría la pena que obtuvieran el listado de componentes del almacén, ya que allí aparecen también los chips más empleados.

Por último, no quiero ni preguntarles si saben lo que es Open Colector, Tri-State, Schmitt Trigger, el 555. ¡Espero que lo sepan! Todos estos temas tienen que dominarlos, para su vida profesional, y para uno que otro Quiz de Laboratorio.

DOS LINKS FABULOSOS

La suscripción a EDN es GRATUITA: http://www.reed-electronics.com/ednmag/subscribe.asp

El MIT, Massachusetts Institute of Technology, ha publicado:

"Every lecture, every handout, every quiz. All online. For free. Meet the global geeks getting an MIT education, open source-style."

TODAS SUS CLASES, MATERIAL DE LOS PROFESORES, TODOS LOS QUICES, EN LÍNEA Y GRATIS. El trabajo de publicación comenzó en septiembre del 2000, con un costo de $10 millones por año durante 10 años (en el 2010 estará disponible todo el material)

Ustedes pueden explorar lo que hay publicado en: http://ocw.mit.edu/index.html

Imagino que saben calibrar qué clase de institución es el MIT. Es el mejor instituto tecnológico del mundo, y ofrece tan valiosa información, gratis y al alcance de la mano del planeta entero.

No dejen de mirar qué se dicta, por ejemplo, en:

http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/indexa.htm

"C"

Si ustedes son como otros grupos, es muy probable que la mayoría maneje el lenguaje "C" ¡ como si fuera **Basic** !

Si no entienden claramente:

- expresiones que involucren apuntadores,
- el operador coma,
- el operador ternario,
- los manejos de bits (tal como 1<<0, etc.),
- las estructuras
- las Uniones...
- los bit fields

se les va a hacer muy difícil comprender los programas que a veces les envío, y mucho más, hacer los suyos propios para el laboratorio.

El C es un lenguaje que permite manejar los recursos de Hardware, tanto a nivel del PC como del Freescale (antiguo Motorola) excepto que quieran hacerlos en Assembler, lo cual debería estar prohibido. Un problema de comunicaciones puede llevarnos 10 líneas en C, y 5 páginas en Assembler; una hora en C y un mes en Assembler...

El siguiente es un pequeño COMPENDIO que escribí para ilustrar 'union, struct y typedef', y unos ejemplos de uso común. Donde aparece la forma estática xxx.yyy, también puede emplearse un pointer: ppp->yyy (dinámico)

Revisen, en todo caso, los manuales de C.

Si aun así permanecen en la penumbra, recuerden que no debe insistir en trabajar en assembler sólo porque desconoce el C. **APRENDA C.**

También hay un breve "Compendio de BIT FIELDS" que escribí para que vean su definición y manejo.

En primer lugar, va una estructura definiendo los bits de Status del registro LINE_STATUS del puerto de comunicaciones en el PC.

Luego va una manera de almacenar los 31 bits de la dirección _Y_ el bit de "valid", en la misma dirección de memoria, sin tener que ocuparse de andar haciendo máscaras para extraer ese bit y separarlo de la dirección...

```
// COMPENDIO: ESTRUCTURAS Y UNIONES Luis G. Uribe C C11F4 L24A6

struct s_one {          // Define el formato de la estructura
   char data[32];       // ... pero no define ninguna variable
   int  tag[32];        // ... con ese formato... 's_one' es el
};         // ... "tag" o idlentificación de esta estructura.
```

```
struct  {               // Define el formato de la estructura,
   char data[32]; // ... SIN tag. Por tanto, no puede utilizarse
   int  tag[32];        // ... después, para definir variables.
} Two;             // Seguramente que "Two" es la única estructura
                        // ... con este formato!
```

```
struct s_one one;    // Ahora se define aquí una variable (se le
                // ... concede ESPACIO), con el formato 's_one'
```

```
struct  s_tree {     // Esta define AMBAS cosas de una sola vez:
   char data[32];// ... el formato s_tree y una variable 'tree'
   int  tag[32];
} tree;
```

```
typedef struct{// Define UN TIPO con el formato de la estructura
   char data[32];       // ... pero no define ninguna variable
   int  tag[32];        // ... con ese formato... 'tfour' es el
} tfour;                // ... tipo de esta estructura.

tfour four;     // Ahora se define una variable 'four' del tipo
                // ... 'tfour'
```

```
union {        // Union SIN nombre. Solo podrá declararse 'cache'
   tfour four2;      // ... como variable de este tipo de union
   char x[2];
} cache;
```

```
union ufive{       // Union CON nombre. No reserva espacio; solo
   tfour four2;    // ... define el formato
   char x[2];
};
```

```
union ufive five; // Ahora define five como variable consistente
                        // ... de la union ufive
```

```
union {          // Igual a 'cache' (arriba), excepto que se usa
  struct s_tree tree2;// ESTA LINEA con tree2 que AQUI se define
  char x[2];
} cache2;
```

```
void main( void )
{                          // USOS:

   Two.data[3] = 'a';
   one.tag[4] = 5;
   tree.data[5] = 'b';
   four.tag[2] = 6;
   cache.four2.tag[2] = 7;
   cache.x[3] = 7;
   five.x[2] = '7';
   cache2.tree2.data[4] = 'c';
   cache2.x[1] = 'c';

}
```

```
// Embeded.c, Luis G. Uribe C., C26Y4
// Example: Transfer Serial received chars to Parallel Port A

// ------------------------------------------------------------
// REVIEW NEXT CODE...

// Define Register Addresses

volatile char *RBuf =(char *)0xFFFFFFE0;// Input Serial I/O Port
volatile char *SStat =(char *)0xFFFFFFE2;//Serial I/O Stat. Port
char *PAOut        =(char *)0xFFFFFFF1; //Parallel Output A Port
char *PADir        =(char *)0xFFFFFFF2; // A Port Address

#define SReady ( 1 << 0 )   // Bit 0 is ready on SStat Register

void main( void )
{
   // Init Paralel Port
   *PADir = 0xFF;      // Configure Port A for Output

   // Transfer Characters
   for(;;) {                    // For Ever
       while( ( *SStat & SReady ) == 0 ) //Wait for Ready status
            ;    /* empty while */
       *PAOut = *RBuf;
   }
}

// ------------------------------------------------------------
// ... AND COMPARE WITH THE FOLLOWING PROGRAM:
```

```
// Define Register Addresses

#define RBuf (volatile char *)0xFFFFFFE0;//Input Serial I/O Port
#define SStat (volatile char *)0xFFFFFFE2;//Serial I/O Stat Port
#define PAOut (char *)0xFFFFFFF1; // Parallel Output A Port
#define PADir (char *)0xFFFFFFF2; // A Port Address

#define SReady ( 1 << 0 ) // Bit 0 is ready on SStat Register

void main( void )
{
   /* Init Paralel Port */
   *PADir = 0xFF;        /* Configure Port A for Output */

   /* Transfer Characters */
   for(;;) {                        // For Ever
       while( ( *SStat & SReady ) == 0 )//Wait for Ready status
           ;     /* empty while */
       *PAOut = *RBuf;
   }
}
```

```
// COMPENDIO DE BIT FIELDS
// Luis G. Uribe C., V13F4

/* OJO CON LAS ALINEACIONES Y LOS 'HUECOS' CUANDO LA INFORMACION
   ALMACENADA NO MIDE EXACTAMENTE UNA PALABRA (int, long... ) */

#include <stdio.h>

struct LINE_STATUS {
   unsigned int dta_rdy : 1;   // bit0
   unsigned int ovr_run : 1;   // bit1
   unsigned int par_err : 1;   // ...
   unsigned int frm_err : 1;
   unsigned int brk_det : 1;
   unsigned int xmt_rdy : 1;
   unsigned int xmt_empty : 1;
   unsigned int no_resp : 1;   // bit7
} ls;            // variable ls, de tipo struct LINE_STATUS

struct {
   unsigned long address : 31; // bit0 a bit30: TAG
   unsigned long valid   : 1;  // bit31
} tag;
```

```
void main( void )
{                          // USOS:

unsigned int rcv_errs = ls.ovr_run | ls.par_err | ls.frm_err |
                        ls.brk_det | ls.no_resp;

unsigned long address = 0x7FFFFFFFL; // comienza con 0111...
tag.address = address;
```

```
// Observe que no hay necesidad de hacer máscaras...

tag.valid   = 0;
printf( "tag bit is %u\n", tag.valid );

tag.valid = ~tag.valid;
printf( "tag bit is %u\n", tag.valid );

printf( "Address is %08lX\n", tag.address );

tag.address = 0xF0000000L;   // comienza con        1111 0000...
                             // Observe que imprime 0111 0000
printf( "Address is %08lX\n", tag.address );

}
```

Otras muchas cosas ignoraban del C mis exalumnos (¿Y ustedes?):

```
void rutina( void ) {
   static int i;          // <<< static: Desconocido por ustedes
...
}
```

Qué significa "**static**"?

Una variable definida fuera de cualquier función (main, rutina, etc.,) es VISIBLE por todas las rutinas de un mismo archivo (Para hacer visible una variable en más de un archivo, hay que categorizarla como 'extern')

Dentro de una rutina, las variables definidas son visibles solamente ahí. Sin embargo, las que se definen automáticamente (variables "automáticas"), además de ser visibles sólo dentro de la rutina que las define, también son volátiles, es decir, que desaparecen al retornar la rutina.

¿Qué hacer cuando se requiere que ciertos valores permanezcan de llamada en llamada de una subrutina?

Podría declararse una variable global, pero eso no es lo que normalmente uno quiere, ya que se trata de mantener la mayor privacidad posible en las variables (¡ para no parecerse al BASIC !), de tal manera de disminuir la posibilidad de equivocar el uso de unas variable por otras!

La solución es declarar la variable DENTRO de una rutina, pero con el atributo "static".

Este tipo de variables se usan en rutinas recursivas.

El calificativo "static" se emplea también así:

```
static int rutina( void ) {
 ...
}
```

Ahora se le ha aplicado a la definición de una subrutina. El resultado es ESCONDER el nombre de esa rutina, de tal manera que sea vista SOLO dentro del archivo en el que se la define. Así, podríamos tener rutinas con el mismo nombre, en otros archivos, sin que hubiera colisiones.

Un error que trajeron a mi atención en un curso pasado: `c = c++;`

¡ No funciona ! (Es decir, si c comienza en cero, al terminar la instrucción ¡ c CONTINUA en cero !)

Pero SÍ FUNCIONA. Lo malo es que programaron la instrucción equivocada. En primer lugar, porque para incrementar una variable bastaba con hacer: `c++`; NO se necesita la asignación!!!

Pero la asignación le está diciendo al compilador que haga lo siguiente:

Tome el valor actual de la expresión `c++` (**0**). Guárdelo momentáneamente en una variable temporal (0). Incremente C (**1** para `c++`).

Tome el valor temporal (**0**) y colóquelo en `c` `(c = c++, o sea: c = 0)`

Y FIN. (C termina valiendo **0** otra vez) ¿Ven?

Por último. Al programar el microcontroladores, NO OLVIDEN que son MICROs, **no Mainframes**. En especial, su RAM es minúscula y ustedes tienen que optimizar su empleo!

PROGRAMAR LA FLASH

Si se quiere guardar información en la Flash (por ejemplo, la correspondiente a la Configuración de su equipo...): Hay ejemplos en los manuales de los microcontroladores. Es SENCILLÍSIMO.

A ustedes les toca, sin embargo, averiguar cuántas veces puede programarse en su vida útil, el micro. Es un número FINITO, del orden de los cientos de miles. Está en los manuales.

Supongan que le contrataron un contador de las personas que viajan en el metro. Usted diseña y construye los equipos, los instala en las entradas de todas las estaciones, y el sistema comienza muy bien. Es más, el primer día ya pudo entregar la cifra de los pasajeros transportados. Pero el siguiente día se va la energía eléctrica al final de la jornada, y todos sus contadores se borraron.

Eso es inaceptable. Lo que corresponde es que usted grabe los resultados en FLASH (EEPROM).

Ahora, si entran cien mil personas al día en una estación, en un solo día, o en dos, usted habrá excedido la especificación del número de veces máximo que puede grabarse la Flash… Necesita **otra** solución.

INTEN, INTDSB; TRAZAS

Puede ser útil que ENSAYEN (experimenten) a hacer interrupt enable y disable desde C.

Una de las recomendaciones que siempre doy consiste en hacerles ver que, un proyecto que no puede TRAZARSE, no es factible debuggearlo. Y así no puede ponerse a punto.

Cuando veo proyectos SIN leds que parpadeen anunciando el paso por los diversos estados, o que NO envían información de STATUS hacia el PC, desde donde, luego, se pueda hacer un análisis (si algo está funcionando mal, lo que ES SIEMPRE EL CASO), o que no pueden detener y continuar a voluntad... veo proyectos que no van a funcionar!

Les propongo un lema: Si no lo puede probar, ¡ NI SI QUIERA LO PROGRAME !

Generarse una estrategia de DEBUG es más importante que generar la estrategia de operación del sistema (bueno; a veces exagero...) Y no debe comenzar a pensar en cómo se va a probar su sistema, una vez que éste no opera.

A FAVOR DE "C"

Los electrónicos suelen aprender *un poco* de Assembler (Intel MASM, Motorola WinIDE, PIC MPLAB...) como parte de algún curso de arquitectura en donde se quieren realzar aspectos tales como: Conjunto de Instrucciones, Modos de Direccionamiento, lenguaje de máquina, proceso ensamblador, etc. Luego, al pasar a proyectos complejos, quieren mantenerse trabajando en ASSEMBLER, con la excusa de "ya manejan la herramienta", y porque no quieren o no pueden estudiar una nueva.

Miren: Si el Assembler diera beneficios, programas y sistemas tan complejos como Linux, o Windows, se escribirían en Assembler. ¡ Y SE ESCRIBEN EN **C** !

Sólo el hecho de que usted termine su proyecto, y que el cliente decida cambiar la plataforma (de Motorola a Intel, por ejemplo), y que a su software, en C, le baste una recompilada (y algunos ajustes menores), en tanto que un proyecto escrito en assembler HAYA QUE BOTARLO A LA BASURA Y COMENZARLO DE NUEVO, sería suficiente argumento para JAMÁS programar en assembler.

Eso sí: Para trabajar en "C", HAY QUE SABER **"C"**.

TEST: Si Ud. no entiende TODOS los componentes de c/línea, debe estudiar más **C**...

```
1)
#define defq( in, n ) char q##in[ n ]; que in
#define initq( in )   in.base = in.put = in.get = q##in;   \
                      in.n = 0; in.size = sizeof( q##in ); \
                      in.limit = in.base + in.size

#define enq( in, c ) ( in.n >= in.size ? EOF :             \
                         ( *in.put ++ = (uchar) c,         \
                           ( in.put = in.put >= in.limit ? \
                                      in.base : in.put     \
                           ),                              \
                           in.n ++, c                      \
                         )                                 \
                     )
```

```
2)
#ifdef COM2
 #define BASE    ( 0x2F8 ) /* 0x2F8 para COM2 */
#else // COM1 (default)
 #define BASE    ( 0x3F8 ) /* 0x3F8 para COM1 */
#endif

#define RCV_DATA        ( BASE + 0 )
#define XMT_DATA        ( BASE + 0 )

/* ------------------------------------------------------------ */
/*      Interrupt Enable Register and Bits      */

#define INT_ENABLE       ( BASE + 1 )
#define RCV_INT  ( 1 << 0 )
#define XMT_INT  ( 1 << 1 )
#define STAT_INT         ( 1 << 2 )
#define MDM_INT ( 1 << 3 )

#define pic_inten(vec) ( outp( MASK_PORT, inp( MASK_PORT )       \
                       & ~ ( 1 << vec ) ) )
```

"LA PESADILLA ¿HARDWARE O SOFTWARE?"

¿Saben qué es una pesadilla?

Es poner a funcionar un programa sobre una plataforma de hardware de la cual ni siquiera se sabe si está operativa, o si está intermitente... Cuando algo falla, usted nunca sabrá si es un problema con el programa, o con el hardware.

Los proyectos del estilo de un Scada, o una Estación Meteorológica, suelen realizarse al menos por dos grupos de trabajo diferentes en las compañías: El de Hardware y el de Software. (Pueden haber muchos más grupos: Partes metalmecánicas, gabinetes, cableado, instalación, pruebas, integración de hardware, diseño mecánico, pinturas; software básico, sistema operativo, programas de aplicaciones, integración de software... integración de sistemas, "commissioning"... son solo algunos ejemplos)

El grupo de Hardware tiene que diseñar una tarjeta con determinadas características, y que obedezca ciertas restricciones.

El grupo de Software, parte de las especificaciones del Hardware y va produciendo los programas que adquieren información, la filtran, la manipulan, obtienen valores diversos, la almacenan y la transmiten.

En algún momento el primer grupo anuncia que el Hardware está listo, y se lo entrega a la gente de programación para la integración final. La gente de Software no ha podido probar sus programas, porque no tenían el hardware, hasta ahora... Sólo los ha simulado.

Muchas veces la gente de Hardware da por concluida su actividad ("tiene un producto") en el momento en que puede encender el equipo sin que se salten los fusibles, algunos leds parpadean y a lo mejor pueden hacerse algunas conversiones de análogo a digital. Aun cuando el procedimiento de aceptación de Hardware sea mucho más riguroso que lo que aquí esbozo, y se incluyan programas de prueba y calibración, puede no ser suficiente...

Ningún departamento de Software suele aceptar por buena, la palabra del departamento de Hardware cuando éste dice que la tarjeta recién diseñada funciona a la perfección. **SERÍA SU RUINA**. El grupo de Hardware tiene que aplicar una serie de pruebas que garanticen que 1) la tarjeta está bien diseñada y 2) que está bien manufacturada (operativa)

El procedimiento varía de un caso al otro, pero siempre la gente de Hardware tiene que disponer de ***Programas de Diagnóstico de Hardware*** para sus productos. Estos programas tienen que ser lo suficientemente sencillos, como para garantizar que están buenos mediante simple inspección. (Si son muy complicados y fallan, volvemos a lo mismo: ¿Fue el hardware o el programa de diagnóstico? Cuando trabajé en DEC, no era ***nada*** raro encontrar que muchos diagnósticos fallaban)

Hay comúnmente dos clases de Programas de Diagnóstico de Hardware: Programas individuales que prueban cada pieza del hardware y determina si está operativa o no, según especificaciones. Normalmente un programa por cada pieza (uno para el conversor A/D, otro para las comunicaciones, uno para la memoria, etc.). Cuando el subsistema es muy complejo, puede haber varios programas para la misma unidad: El de la verificación del almacenamiento, el de aseguramiento de las lecturas, etc.

La segunda clase de Diagnósticos son los ***System Exercisers***, que verifican la integridad del sistema cuando se le añaden diversas unidades de hardware. Ellos garantizan que, unidades probadas individualmente como operativas, también trabajan armónicamente en conjunto (muchas veces no ocurre así, y al querer hacer funcionar ciertas partes del conversor A/D, interfiere con la memoria, etc.)

Recién cuando se ha probado la operatividad de cada pieza de hardware y se está conforme, y se ha verificado que las interrelaciones entre subsistemas son armónicas y no se entorpecen unas unidades con otras, es cuando pueden comenzar las pruebas del Software propiamente dicho.

Ustedes que, además, trabajan en un soporte delicado (Protoboard, baquelita), en donde siempre están bajo la posibilidad de que algo se suelte y el hardware no trabaje, tienen que planificar una batería de programas de Diagnóstico que les permitan **RÁPIDAMENTE** estar seguros de que su hardware opera perfectamente.

PREMISA DE URIBE

No puede "debuggearse" el hardware y el software al mismo tiempo...

Un tercer tipo de programas usados en manufactura, además de los ejercitadores y los de diagnóstico, son por ejemplo los de ***Calibración***. Ellos permiten fácilmente ajustar los amplificadores y otros dispositivos que requieran, caso por caso, determinados toques a fin de cubrir el rango completo, o que ciertos valores de entrada produzcan unas bien conocidas cifras de salida, etc. Son "**utilitarios**".

Dentro de su proyecto usted debe identificar, además de las cosas obvias en que ya había pensado, las herramientas de diagnóstico que le ayudarán a ponerlo a punto. No hacerlo es garantía de que su sistema no funcionará adecuadamente.

Les recuerdo:

Un proyecto que no puede TRAZARSE, no puede "debuggearse".
Y así no puede ponerse a punto.

Generar una estrategia de DEBUGGING es TAN importante como generar la de operación del sistema. Y no debe comenzar a pensar en cómo se va a probar su sistema, una vez que éste no funcione.

"PERDER LOS ARCHIVOS"

Las excusas más socorridas de los últimos tiempos, en los proyectos, son:

- "No puedo leer el floppy/Pendrive con mi proyecto"
- "Un virus dañó el disco duro, con todo y proyecto"

Ni en el laboratorio, y menos en su práctica profesional, pueden usar como excusa que se les dañó la PC y que por lo tanto los meses de esfuerzo dedicados al proyecto se fueron por el desaguadero... Si se le presenta una situación así, solo demostrará IRRESPONSABILIDAD de su parte para con el trabajo.

En esta época, es su obligación tener acumuladas copias de respaldo INCREMENTAL, que les permitan SIEMPRE devolverse a una situación en la cual Uds. (y la compañía) pierden a lo sumo unas cuantas horas de trabajo.

Recuerden: INCREMENTAL BACKUP.

En la época del PKZIP/WinZip/WinRAR y similares, y del correo electrónico bueno, bonito y barato (GMail), en donde Uds. pueden enviarse a sí mismos correos periódicos con las copia incrementales de respaldo, NO HAY DERECHO a que se pierda un proyecto, o la calificación de un laboratorio, porque se les dañó el disco...

Hay también sitios en el Internet donde le ***regalan*** espacio en disco (que comprimido alcanza cada uno para guardar muchos archivos)

Cuando ha tomado precauciones, ¡ hasta desde un Cibercafé puede recuperar su información !

NO LO OLVIDE:

Una Estrategia de Respaldo es fundamental en el desarrollo de cualquier proyecto

Están advertidos: Excusas de tal naturaleza son inaceptables, ahora y luego. En la universidad, proyecto borrado es proyecto reprobado. En la vida como ingeniero implica, al menos, un ***despido***.

Además, en la actividad profesional esta advertencia ¡ no se limita a los programas ! Todo lo que esté en medio magnético debe respaldarse, como: ➢ los circuitos impresos (Orcad), ➢ diseños metalmecánicos (Autocad), ➢ toda la documentación del cliente y la interna (MSWord, Excel, MSProject)

PROYECTOS II

Algunos ejemplos de Proyectos en el Laboratorio

DOCUMENTACIÓN

VOY a presentar como ejemplo, dos desarrollos típicos del Laboratorio de Proyectos. Los requisitos van más adelante; incluiré por el momento la identificación de la información que los estudiantes deben entregar como parte del proyecto. También es indispensable, en el laboratorio o en su vida profesional, llevar una Bitácora.

IDEAS PARA MANUALES DEL PROYECTO "ESTACIÓN METEOROLÓGICA DE ALTAS PRESTACIONES"

- Se deben construir dos escritos, uno a manera de manual del producto y otro un informe técnico del mismo.
- Ambos tipos de escritos deben venir en papel impreso

MANUAL DE USO E INSTALACIÓN DEL PROTOTIPO

- Título.
- Uso o descripción del mismo en forma breve.
- Descripción de cómo trabaja de manera directa y corta.
- Listado con las especificaciones y estándares que cumple.
- Descripción de la instalación: Requerimientos de potencia y el entorno físico.
- Seguridad industrial, del equipo y las personas.
- Lo anterior puede realizar se a manera de tríptico o folletín.

MANUAL TÉCNICO

- Carátula con título del proyecto particular desarrollado.
- Índice del manual incluyendo la referencia a los gráficos y anexos.
- Breve descripción del proyecto.
- Sobre el proyecto desarrollado:
- Diagrama de flujo
- Diagrama en bloques
- Diagrama eléctrico
- Descripción de lo que hace (cómo trabaja cada etapa)
- Normas que cumple, en caso de que aplique alguna.

EXPLICACIÓN

- Describa cada bloque de manera breve e independiente. Coloque los circuitos desarrollados, descríbalos de manera concisa y directa. En el caso de los programas, solo coloque el diagrama de flujo del algoritmo y coméntelo. Los detalles de programación y listados del programa deben ir en un anexo claramente indicado y referenciado desde aquí.
- Comente los problemas encontrados durante el desarrollo y la solución adoptada. Qué fallas aun persisten y como cree puedan corregirse.

CONCLUSIÓN

- Experticias desarrolladas durante el transcurso del laboratorio.
- Mejoras posibles al mismo.
- Costos finales gastados en el hardware y el número de horas/hombre que realmente le dedicó.

BIBLIOGRAFÍA CONSULTADA Y SITIOS WEB EN LOS QUE SE SOPORTÓ

ANEXOS

- Estos deben clasificarse e indicarse en forma clara.

- Coloque aquí de manera detallada los circuitos empleados, que deben estar claramente identificados y brevemente comentados en cuanto a su funcionamiento.

- Los flujos de los programas en Code Warrior y LabView deben estar a manera de bloques y su código identificado con el bloque respectivo y comentado adecuadamente.

EXTRAS

Deben añadir lo indicado en el llamado "Avance Inteligente":

- ¿Usará fuente-UPS-Batería Solar o pilas?
- Diseño de la maqueta virtual (indicar el tamaño)
- Fusibles: ¿Qué clase y de qué valor?
- A futuro, ¿pensó en la facilidad del ensamblaje? Comentar.
- Cómo solucionará la tolerancia de los componentes por unidad - por lote?
- ¿Requiere indicación de "low-battery"?
- ¿Tiene recuperación automática en caso de fallas? (watch-dog)
- ¿Qué harán con la basura de la producción?
- ¿Existe reciclaje (producción)?
- ¿Cómo arreglarán los daños, a nivel de componente o cambiando tarjetas? Comente.
- ¿Qué tiempo de garantía darán? ¿En qué modalidad? (¿deben traer el equipo a su taller; o ustedes irán al sitio a repararlo?
- "Isla de trabajo" mínima para <u>fabricar</u>. "Isla de trabajo" mínima para <u>instalar</u>.
- Expansión a futuro: ¿Pueden realizar otras mediciones/actividades?

SCADA

INTRODUCCIÓN

SCADA

Un "Supervisory Control and Data Acquisition (SCADA) SYSTEM" está destinado a permitir al personal de operación, la vigilancia y el gobierno de localidades industriales lejanas, que pueden ser de muy variada naturaleza, como edificios, viviendas, subestaciones eléctricas, instalaciones industriales y manufactureras, pozos petroleros y similares.

A diferencia de otros sistemas, un SCADA funciona primordialmente "a lazo abierto", siendo indispensable la actividad de un operador humano para emitir las órdenes que se ejecutan, finalmente, en el proceso. Lo anterior, sin demérito de algunas actividades minoritarias que pueden operar a "lazo cerrado", tanto localmente en las instalaciones remotas (PLC, Programmable Local Controller), como a través de Estaciones de Aplicaciones (PCs). Pero la responsabilidad de las acciones y decisiones reposa, eminentemente, en una persona.

CONSTITUCIÓN

Un SCADA se conforma mediante 3 subsistemas: La Estación Maestra, las Unidades Terminales Remotas o RTU y el sistema de comunicaciones que los interconecta. Las RTU, a su vez, podrían ser Maestras de otras unidades que jerárquicamente se encuentren debajo, llamadas RTU Esclavas.

La Estación Maestra tiene como función hacer un barrido en el tiempo, de los valores que adquieren en el campo las RTU; mantiene archivos históricos con toda esa información; la evalúa y valida contra diversos límites que impone el usuario y, en caso de violaciones, genera alarmas, auditivas y visuales, para alertar de condiciones anómalas al operador. Permite al encargado tomar acciones para controlar las instalaciones distantes, cerrando o abriendo, a lo lejos, válvulas y solenoides, emitiendo consignas a los PLC remotos, etc.

La RTU se diseña para estar atenta a las solicitudes periódicas de información que hace la Maestra, y cuando se lo ordenan, envía los valores analógicos leídos del campo, que con mucha probabilidad provienen de "transductores", esos dispositivos que convierten a información electrónica, parámetros físicos tales como temperaturas, flujos, presiones, posiciones, etc. Es posible, de igual manera, que la RTU produzca valores "calculados"; este es el caso, por ejemplo, cuando la misma remota lee señales eléctricas instantáneas de corriente alterna y produce valores RMS para enviarlos al "Despacho" o Centro de Control (CC). Las RTU también leen valores digitales, procedentes de interruptores o contactos, cerrados o abiertos, que representan posiciones tales como puerta cerrada o abierta, dispositivo operativo o inoperativo, etc.

Por último, las RTU permiten del operador accionar equipos a distancia, para abrir interruptores, o cerrarlos, activar o desactivar unidades en la planta, etc. Algunas RTU pueden a veces entregar valores analógicos a unidades que en el campo pudieran necesitarlo; tal es el caso de valores "consigna" que se proporcionan a los PLC y otros subsistemas de control local instalados en el campo.

El medio de comunicaciones que permite la interacción entre Maestra y Remotas puede ser organizado alrededor de una variedad de topologías, como líneas telefónicas, dedicadas o discadas, fibra óptica, radio, Internet, etc. Uno o varios "protocolos", o disciplinas de intercambio, encapsulan la información que se transfiere de un lugar a otro, garantizando que llegue al destino correcto sin errores. Muchos protocolos existen o persisten en este tipo de sistemas que van cruzando ya los 40 años desde sus orígenes, siendo algunos de los más renombrados el Modbus (Modicon), DNP (Harris), Asea (ABB), Sinaut (Siemens), Fieldbus, y los venerables Honeywel 7024 y Conitel 2020, entre una multitud.

ESPECIFICACIONES DEL PROYECTO

Diseñar un pequeño SCADA conformado por dos (2) RTUs operando en el microprocesador Intel x86 (PC) bajo protocolo Modbus, y una (1) Estación Maestra en un PC, interconectados los tres mediante los puertos seriales de comunicación.

Uno de los dos integrantes de cada grupo será el encargado del diseño y puesta en marcha de la Maestra, en tanto que el segundo participante asumirá iguales responsabilidades con las Remotas. La evaluación será individual. En caso de que la Maestra, o la RTU, no funcionen, se hará la entrega empleando software comercial.

ESTACIÓN MAESTRA

El software de la Estación Maestra correrá en el PC y deberá tener como mínimo los siguientes subsistemas:

- Adquisición de Datos
- Presentación de Información
- Generación y Procesamiento de Alarmas
- Control Remoto
- Comunicación con las RTU del campo

Quedan excluidas muchas funciones comunes, como el manejo histórico de información, generación de reportes, operación redundante (hot stand-by), estadísticas de comunicaciones, editor de base de datos, configurador del sistema, generador de despliegues, y aplicaciones tales como control mediante "recetas", control estadístico, control automático, generación de secuencias coordinadas, etc., las que se mencionan solo para que tengan idea de lo que se ofrece en este mercado.

SUBSISTEMA DE ADQUISICIÓN DE DATOS

Debe recabar la información del campo, de manera secuencial, empleando al máximo la funcionalidad del protocolo Modbus, que se verá reducida a los valores que la RTU puede manejar y que serán definidos a continuación. El tiempo entre barrido y barrido lo define el diseñador, pero deberá tener una frecuencia suficientemente alta, para permitir evaluar prontamente las modificaciones que se produzcan en los valores del campo. Si por alguna razón la RTU deja de responder, se deberá esperar

un tiempo máximo antes de indicar visualmente la situación de "Falla de Comunicaciones", caso en el cual el Subsistema de Presentación de Información debería proporcionar algún valor convenido, como el promedio histórico, o el último valor leído, identificándolo claramente ante los ojos del operador para que no confunda valores reales con cantidades supuestas.

SUBSISTEMA DE PRESENTACIÓN DE INFORMACIÓN

Debe mantener actualizados en la pantalla los valores que se van leyendo. Cuando se trata de variables analógicas hay normalmente por lo menos dos clases de valores: "raw" o crudos, que son simplemente los datos numéricos que se transmiten; en nuestro caso serían números de 0 a 255 (conversor analógico a digital de 8 bits), correspondiendo a valores de entre 0 y 5 voltios; y los valores de Ingeniería, que representan unidades físicas como temperaturas, presiones o voltajes, y que se obtienen manipulando matemáticamente las cantidades "raw". Para nuestro caso será preferible presentar valores de Ingeniería, pero las conversiones serán muy simples, tales como Val_Ing = 'Val_0 + raw*K'.

La información digital debe presentarse adecuadamente, de manera que pueda distinguirse con rapidez el estado de cada interruptor del campo. Además de un símbolo apropiado (O y X, para Abierto y Cerrado), deberá agregársele color. En la industria eléctrica se estila que Rojo sea para un circuito energizado, y Verde para abierto (esto es todo lo contrario a la convención de los ingenieros electrónicos, que suelen indicar con Verde una señal activa, correcta, y con Rojo los errores!)

Se excluyen en nuestro proyecto funciones tales como la presentación por pantalla de gráficos de tendencia, o de secciones de valores históricos (los que se mencionan solo para que tengan idea de lo que se ofrece en el mercado).

GENERACIÓN Y PROCESAMIENTO DE ALARMAS

Cada valor presentado debe compararse con límites superiores e inferiores, y en caso de excederlos se alertará al operador mediante la corneta del PC y empleando en la pantalla algún sistema de mensaje, por ejemplo escribiendo una línea en la parte inferior de la pantalla, y señalando de alguna forma el valor problemático (con colores, asteriscos, parpadeos, etc.)

Se utilizarán para las señales continuas (AI, Analog Inputs) los indicadores Hi, Lo y HiHi y LoLo (alto, alto alto, bajo y bajo bajo), con lo cual se definen la región normal, una zona de alerta y una de alarma. El diseñador del sistema definirá los valores de las alertas (Hi y Lo) y de las alarmas (HiHi y LoLo).

CONTROL REMOTO

Debe existir la capacidad para que el operador ordene el cierre o apertura de un relé a través de una salida digital (DO, Digital Output) activada por la RTU, y que se simulará convenientemente en la RTU.

Se excluyen en nuestro proyecto funciones tales como la de tener un contacto para ordenar la apertura (trip) y otro distinto para el cierre (close), lo que es la práctica más común (Se menciona para que tengan idea de lo que se ofrece en el mercado).

COMUNICACIÓN CON LAS RTU DEL CAMPO

Se empleará como capa física (Physical Layer) el sistema de comunicación serial del PC, y como Capa de Red (Data Link Layer), un subconjunto apropiado del protocolo Modbus RTU (funciones de lectura de analógicos, lectura de "coils" o digitales, y activación y desactivación de "coils" para el control). Debe incluirse la funcionalidad de CRC (Cyclic Redundancy Check) o LRC (Longitudinal Redundancy Check), definidos por el protocolo según sea el caso (Modbus–RTU o Modbus–ASCII respectivamente). La velocidad no debe ser motivo de preocupación en el desarrollo de este sistema, por lo cual bien puede trabajarse a 1200 bps, lo que además resulta corriente en la realidad.

El Modbus tiene dos modalidades de operación, conocidas como Modbus–RTU, y el Modbus–ASCII. La primera requiere el empleo de una función polinómica para el cálculo del CRC (Cyclic Redundancy Check). La segunda precisa una simple verificación (LRC: Longitudinal Redundancy Check) que corresponde a los ExOr de cada uno de los bytes del mensaje. El método más empleado es el Modbus Binario y su procedimiento de verificación es el CRC. (Para ambos sistemas, CRC y LRC, refiérase a la documentación oficial del Modbus, publicada en Modulo7)

Se excluyen en este proyecto las demás funciones definidas por el protocolo (que ustedes deberán revisar, para informarse)

Hay que tener especial precaución en el diseño para no caer en situaciones de descoordinación, en las cuales alguno de los dos extremos pierde un símbolo o byte, y ahora nunca jamás se sincronicen (incluir Watch-dogs, Timers...)

RTU

El software de la RTU correrá en el microprocesador Intel x86 (PC) y deberá tener los siguientes subsistemas:

Adquisición de Datos Analógicos (2 AI)

Adquisición de Datos Digitales (8 DI)

Control (1 DO)

Comunicación con la Estación Maestra

Quedan excluidas en nuestro proyecto muchas funciones comunes, como el envío espontáneo de información a la Maestra SCADA, ya que no está soportado por el Modbus. De esta manera, la RTU es una unidad verdaderamente esclava o supeditada. Tampoco se elaboran valores calculados; simplemente se envían los leídos (no se está simulando, por ejemplo, la conversión de señales sinusoidales a RMS). Como tampoco se pide que la RTU tenga ninguna funcionalidad de control local, tipo PLC, resulta innecesario leer tanto valores digitales como analógicos "antes" de que llegue la solicitud respectiva por parte de la Maestra. Así que el funcionamiento es bastante secuencial: Llega una solicitud, se la analiza, se procesa dicha solicitud (leyendo, por ejemplo, los valores analógicos), se elabora la trama de transmisión y se envía de vuelta la información correspondiente.

Cada RTU dentro del sistema SCADA debe poseer una dirección única. Utilice el número del mesón como la dirección.

ADQUISICIÓN DE DATOS ANALÓGICOS (AI)

Los valores considerados serán simplemente dos (2) analógicos de 0 a 5 voltios, y las señales que en la vida real provendrían de los transductores se simularán mediante el teclado, para permitir la prueba de concepto. El tipo de señales simuladas varía muy lentamente, como lo haría por ejemplo la temperatura de una planta, y a este tipo de señales se las conoce como señales DC (aunque en realidad sí varían).

Se excluye la adquisición de señales AC, que requerirían que la RTU sí estuviera trabajando permanentemente en el cálculo de, por ejemplo, los valores RMS, o los componentes armónicos, etc.

ADQUISICIÓN DE DATOS DIGITALES (DI)

Los valores considerados serán solo 8 bits de un puerto elegido por el diseñador, y las señales de prueba se alimentarán mediante 7 teclas para permitir la prueba del concepto (para el octavo bit, vea el próximo párrafo).

Uno de los 8 bits digitales de entrada siempre se conectar de alguna manera a los relés de control, de tal manera que la Unidad Maestra tenga una realimentación en cuanto al estado de esos dispositivos. Así, por ejemplo, si se manda a abrir un relé, la siguiente lectura de digitales enviará a la Maestra el valor del contacto, indicándole al operador si en realidad se abrió o nov el interruptor. Igualmente ocurre al ordenarse un cierre. De lo contrario el operador trabajaría a ciegas, sin saber si sus órdenes se están en realidad realizando o no.

Se excluye en nuestro proyecto la adquisición de señales digitales con procesamiento, como por ejemplo el que se requeriría para "marcarlas" con una etiqueta que indicara el instante en el que ocurrió una activación (cierre o apertura: Secuenciadores de Eventos, SOE). Se excluyen así mismo los Digitales con Memoria, que recuerdan si un contacto cambió entre barridos ("scans") de la maestra; (esto funcionaría así: Si un contacto está reportado como cerrado en la Estación Maestra, y ocurre que se abre, pero antes de llegar la solicitud de su lectura, vuelve a cerrarse –caso, por ejemplo, de una puerta, cuando la Maestra requiere la información de los digitales el valor enviado será cerrado, que es como está ahora, y en el Centro de Control no se habrán enterado de que en el ínterin hubo algún cambio. Esto se menciona para que sepan lo que se ofrece en este mercado).

CONTROL (DO)

Debe incluirse la simulación de al menos un relé mediante el cual pueda materializarse una función de control remoto, por parte del operador.

En un equipo real debe diseñarse muy bien el circuito de manejo del relé, a fin de evitar que los picos inversos producidos por su bobina dañen permanentemente el resto de la electrónica.

Se excluyen en nuestro proyecto funciones tales como la de autoretención, que permitirían mantener el estado (abierto o cerrado) de un relé en caso de falla de la alimentación eléctrica para la RTU (lo que se ofrece por lo común en este mercado).

COMUNICACIÓN CON LA ESTACIÓN MAESTRA

Se empleará como capa física (Physical Layer) el sistema de comunicación serial del PC y como Capa de Red (Data Link Layer), un subconjunto apropiado del protocolo Modbus RTU (funciones de lectura de analógicos, lectura de "coils" o digitales, y activación y desactivación de "coils" para el control). Debe incluirse la funcionalidad de CRC (Cyclic Redundancy Check) o LRC (Longitudinal Redundancy Check), definidos por el protocolo según sea el caso (Modbus–RTU o Modbus–Ascii respectivamente). Puede trabajarse a 1200 bps.

El Modbus debe ser RTU o ASCII, dependiendo de lo que se seleccionó para la Unidad Maestra y el cálculo del campo de verificación debe ser el que corresponda: CRC o LRC.

Se excluyen en nuestro proyecto las demás funciones definidas por el protocolo (que deberán revisar, para informarse)

Hay que tener especial precaución para no caer en situaciones de descoordinación, en las cuales alguno de los dos extremos pierde un símbolo o byte, y ahora nunca jamás se sincronicen (Watch-dogs, Timers...).

ANTEPROYECTO

El Anteproyecto debe incluir todas las definiciones que a cada uno correspondan y todas las funcionalidades, según se ha indicado en este documento, y que son, entre otras:

- Qué modalidad del Modbus se escogió (RTU, ASCII)
- Qué señales están simulando, y qué rangos tendrán sus valores
- Ecuaciones que usarán para convertir a unidades de ingeniería
- Tiempo entre barridos
- Qué funciones (3) del Modbus estarán usando
- Qué acciones tomará para evitar el "congelamiento" en las comunicaciones
- Qué valores se proporcionarán en caso de perder la telemedida ("supuestos")
- Cómo diferenciará en la pantalla los valores leídos de los valores "supuestos"
- De qué manera indicará los valores analógicos (barras horizontales, verticales, relojes, etc.)
- Cómo mostrará las zonas normal, alerta (baja y alta), alarma (baja y alta) (Se sugieren colores Verde, Amarillo y Rojo)
- Cómo se identificarán en la corneta del PC las alertas y las alarmas
- Cómo se "reconoce" (o acepta) una alarma, para que cesen los parpadeos y termine el sonido por la corneta
- Cómo serán los despliegues de la información digital; cómo representará los ceros y cómo los unos (qué símbolos y qué colores empleará)
- Cómo mostrará la simulación del Control
- Asegúrese de incluir la realimentación automática del control, sobre uno de los 8 bits de entrada digital

El Proyecto debe incluir todas las funcionalidades y definiciones hechas en el Anteproyecto (itemizadas arriba).

El Software y la documentación deben incluir, y en las fuentes quedar resaltadas, todas las funcionalidades y definiciones hechas en el Anteproyecto. Además, debe suministrarse en un floppy el listado de los programas, comentados, modulares, en donde se destaquen:

- La rutina principal
- Los manejos de colas
- Los manejos de interrupciones
- Definición de variables
- Inicializaciones
- Manejo de error
- Cálculos de CRC/LRC
- Subsistema de transmisión y recepción
- Simulaciones
- Presentación de información
- Generación y procesamiento de alarmas
- Control remoto
- etc.

DATA LOGGER

TÍTULO DEL PROYECTO

Diseño y construcción de un "Sistema de adquisición de datos, portátil, con capacidad de comunicación serial y almacenamiento local".

CONTRATANTE DEL PROYECTO

La empresa contratante desea el diseño de prototipos de desarrollo, para posteriormente construir uno industrial. En este contrato no se llegara a la implantación del prototipo industrial. Se desea el diseño y entrega del primero, para analizarlo en el laboratorio.

OBJETIVOS DEL PROYECTO

Los objetivos del proyecto a realizar por los Ingenieros de Desarrollo son:

- Diseño y montaje de un Sistema de Adquisición de Datos Portátil con capacidades de comunicación con una estación central.
- Los subsistemas de Adquisición, Acondicionamiento, Alimentación y Aplicación Final, deben contar con las características funcionales mínimas indicadas en el siguiente apartado.
- Deberá entregar la lista de componentes del prototipo: Número de parte, descripción (incluye: Si es montaje convencional o de superficie, características especiales como temperatura de operación, voltaje, etc.), proveedor, "third party", cantidad requerida, mínimo lote por pedir, precio unitario, precio total, tiempo de entrega, comentarios (back log, descontinuado, si hay reemplazo), así como todos los esquemáticos con sus informes explicativos.
- Deberá entregar una Bitácora de Trabajo donde recoja todas y cada una de las ideas, documentación, copias de manuales empleados, problemas encontrados con sus soluciones, planteamientos efectuados en el Desarrollo del Sistema de Adquisición, ideas desechadas, etc. La Bitácora es su diario de trabajo.
- Deberá entregar manuales técnicos de operación, así como un informe detallado de explicación de los esquemáticos.

DESCRIPCIÓN DE LOS EQUIPOS A SER DESARROLLADOS

CARACTERÍSTICAS FUNCIONALES MÍNIMAS DEL SISTEMA

El subsistema de Adquisición de datos debe contar con las siguientes características mínimas:

- Debe ser un sistema basado en microprocesadores.
- Debe entregarse el diseño del impreso en EAGLE, ORCAD, etc.
- El subsistema o tarjeta de adquisición podrá usarse en tres modos principales:
  - ✓ Modo de adquisición "Independiente", en el cual el sistema funciona parecido a como lo hace un "Data Logger". Adquiere información por un período de tiempo fijo y a una tasa de muestreo fija, esos datos son guardados para su posterior transmisión a un Sistema Central (basado en PC podría ser). PROTOCOLO: La comunicación con el PC se establecerá mediante un protocolo simple, del tipo de la familia XModem (XModem, YModem, ZModem), o un subconjunto apropiado del MODBUS.
  - ✓ Modo de adquisición "En – línea", en el cual se adquiere un dato y se transmite a un PC, donde el dato es visualizado. PROTOCOLO: La comunicación con el PC se establecerá sin protocolo; el "Sistema de adquisición de datos portátil simplemente enviará, de manera permanente, los datos adquiridos, codificados en ASCII, terminados en caracteres CR, LF. Es responsabilidad del PC leer convenientemente la información recibida.
  - ✓ Modo de adquisición "Por ráfagas", en el cual el sistema adquiere datos por un período de tiempo finito y a tasa definida, luego bajo petición de un sistema central transmite lo adquirido y espera una nueva orden de adquisición. PROTOCOLO: La comunicación con el PC se establecerá mediante el mismo protocolo empleado en la opción 1) arriba (familia XModem).

Nota: La escogencia de uno de los 3 modos principales se hará mediante un selector localizado en el panel frontal del instrumento (no será una operación de comando remoto, originada desde el PC).

- El sistema deberá ser capaz de adquirir, mínimo, dos señales analógicas de ±5 Voltios. Las señales deben ser leídas en modo diferencial o en modo unipolar. Indique la precisión de su sistema de adquisición, siempre que se aproveche todo el rango dinámico de la entrada.
- El diseño debe incluir la definición de la tasa máxima de muestreo que Usted puede ofrecer en cada uno de los distintos modos de operación; deberá tomar en cuenta la existencia de memoria adicional, que Usted deberá incluir en su diseño. Deberá indicarse también el tiempo máximo de adquisición.
- El subsistema de adquisición también debe ser capaz de manejar 8 señales digitales, programables como entradas o salidas digitales.

- El subsistema de adquisición deberá incorporar el manejo de memoria adicional a la interna del microprocesador, que será usada para el almacenamiento temporal de la data adquirida.

- El subsistema de adquisición debe contar con un sistema de comunicación serial con una estación central, basada en PC. La comunicación debe poder realizarse hasta 115.2 kbps.

El subsistema de Acondicionamiento consistirá en la electrónica adecuada para el manejo de dos señales analógicas, de las cuales Usted debe definir sus características. Usted puede proponer dos tipos de señal que desee procesar, o puede elegirlas de la lista indicada más abajo. El diseño del módulo de acondicionamiento tomará en cuenta:

- La salida típica de los sensores o características de las señales a medir (por ejemplo, lazos de corriente, voltajes DC ó AC, etc.)

- La entrada máxima de la etapa de adquisición, sin despreciar la resolución del sistema.

- Medidas de anti – aliasing.

Usted puede proponer cualquier par de señales a ser adquiridas, pero sólo a modo ilustrativo indicamos una lista de posibles tipos de señales a ser procesadas:

- ✓ Temperatura.
- ✓ Lazos de corriente (0 – 20 mA).
- ✓ Micrófono.
- ✓ Nivel de líquidos.
- ✓ Voltajes DC (Salidas de fuentes DC con sus características: Rizado, ruido, etc.).
- ✓ Voltajes AC.
- ✓ Corriente DC.
- ✓ Corriente AC.
- ✓ Strain gauges (señales provenientes de puentes resistivos).
- ✓ Potencia eléctrica.
- ✓ Factor de potencia y armónicos.
- ✓ Caracterización de una fuente DC.
- ✓ Frecuencia.
- ✓ Presión.
- ✓ Caracterización y pruebas de transformadores.

Debe entregarse el impreso de esta tarjeta diseñado en EAGLE, ORCAD, etc.

SUBSISTEMA DE APLICACIÓN FINAL

Es un sistema totalmente libre que Usted debe diseñar y proponer. Es un requisito de la empresa que la aplicación final sea desarrollada bajo LABVIEW. Por supuesto su aplicación dependerá en parte de la señal que desee procesar. Sólo a modo ilustrativo exponemos algunos sistemas posibles, que de ninguna forma deben coartar su libertad de proponer aplicaciones diferentes.

Algunos ejemplos de aplicaciones podrían ser:

- Analizador de respuesta al impulso.
- Analizador de espectro.
- Espectrogramas.
- Analizador de Bode.
- Sistema supervisorio sencillo.
- Analizador de voz.
- Sistema de pruebas de transformadores.
- Sistema de supervisión de calidad del servicio eléctrico.
- Osciloscopio Digital

ASPECTOS EXTRAS QUE DEBEN CONSIDERAR

- Calidad del diseño del sistema de alimentación y carga de baterías junto con la calidad del diseño del sistema usando componentes de bajo consumo.
- Diseño de caja (sólo planos y modelo virtual). El sellamiento será de tipo NEMA 4X
- Consideraciones de aislamiento de las señales analógicas.
- Diseño de un protocolo de pruebas para el sistema y los respectivos sub – sistemas.
- Control y programación de ganancias y offset en la etapa de acondicionamiento.
- Generación de señales que actúen sobre circuitería que genere, a su vez, la señal a ser procesada.
- Buena implementación del Modo "En – línea", evitando la pérdida de datos (implementación de sistemas CRC, u otros, para supervisar la entrega de los datos al PC).

FAT, SAT: FORMULARIOS DE EJEMPLO

FAT: DOCUMENTO DE ACEPTACIÓN EN FÁBRICA (PARA EL "DATA LOGGER")

Grupo #____, Integrantes: _____,
_____

CARACTERÍSTICAS GENERALES

√	
	1. Alambre tipo telefónico, de un solo conductor (usualmente de tamaño #22)
	2. Todos los chips insertados con la misma orientación
	3. Cables con la longitud adecuada. removida ½ in. en los extremos. Cortados en ángulo de 45°
	4. Verificar que no pasan cables por encima de los ICs.
	5. Observar si los cables van aplanados contra el protoboard y ORDENADOS.
	6. Ver si están codificadas mediante colores en los cables las categorías de las señales que se distribuyen: Rojo: 5V, Negro: Tierra, Morado: Control, Naranja: Datos, Café: Direcciones, y similares.
	7. Ver que los componentes no se calienten fuera de lo normal.
	8. Verificar que "switches" de reposición y de corredera, "push buttons", contactos de relés, han sido apropiadamente conectados, con resistencias de pull-up o pull-down, eliminando el rebote (hardware o software)
	9. Verificar que se ha conservado el fan out de cada salida, en particular señales de amplia distribución, como relojes, reset, buses de direcciones y de datos. Constatar que se revisó apropiadamente el consumo de las entradas que se conectan a cada salida (En algunos circuitos ciertas señales como el Reloj tienen FAN-IN de 2, o más)
	10. Verificar que los niveles de tensión y corriente, y el fan out, están bien en las interconexiones entre familias CMOS y TTL, TTL-LS, y cualquier otra diferente que se haya empleado. Comprobar el apropiado uso de "buffers" TTL o "drivers"
	11. Ver si se han dejado entradas AL AIRE o si se ha empleado directamente la fuente de alimentación como un "1", sin la apropiada resistencia limitadora de corriente.
	12. Si alguna entrada se ha colocado a tierra a través de una resistencia, revisar por qué, y verificar que el valor del pull-down sea apropiado.
	13. Determinar si se han manipulado apropiadamente los CMOS: Si se han usado bolsas conductoras especiales para transportarlas; si el estudiante toca un terminal que esté conectado a tierra antes de manipular estos ICs con la mano, o con un cautín o un instrumento.
	14. Verificar la existencia correcta de condensadores de desacoples: Entre el terminal de Vcc y el de tierra de cada IC, lo más cercanamente posible al IC.

	15. Verificar que el alumno emplea correctamente el generador de señales, el multímetro y el osciloscopio, las "puntas lógicas" y los pulsadores (si los hay).
	16. Comprobar que todos los LEDs tienen su resistencia limitadora de corriente
	17. Si algunos LED se han conectado con el ánodo a la salida y el cátodo a tierra, verificar que se ha garantizado que los voltajes y corrientes de esa salida se mantienen en el rango de operación del circuito que lo maneja (sobre todo si esa señal es una salida directa de un biestable, o si esa señal se encuentra conectada a otras entradas lógicas)
	18. Constatar que si hay compuertas que multiplican en and lógico, el pulso de reloj por una combinación de salidas de Flip-Flops (FF), lo hacen con el borde correcto del reloj.
	19. Comprobar que la inicialización de toda la circuitería se realiza apropiadamente.
	20. Garantizar que todas las señales asíncronas han sido sincronizadas.
	21. Constatar la existencia de una infraestructura apropiada para efectuar la puesta en marcha: Corrimiento de secuencias –paso a paso o a baja velocidad–, existencia de señales indicativas del estado del circuito o del flujo de información, "test points", y de un mecanismo de inyección de señales externas que garanticen el resultado.
	22. Validar que la conexión de las fuentes de alimentación es la apropiada, que se han hecho las estimaciones de carga del circuito, que se tienen los desacoples indispensables a la entrada, y las demás consideraciones comunes.
	23. Revisar la calidad de las soldaduras (si se ha hecho alguna).
	24. Tener a mano, aunque sea en borrador, el diagrama en bloques del sistema en el que se destaquen los subsistemas y sus interrelaciones. Tener una descripción del Algoritmo basada en el Diagrama en Bloques; explicar claramente la forma en que su circuito opera.
	25. Tener a mano, aunque sea en borrador, el diagrama lógico, que muestre con detalle los componentes (contadores, registros, sumadores) y sus conexiones, mediante compuertas AND/OR, o NAND.
	26. Tener a mano, aunque sea en borrador, el diagrama eléctrico y de interconexión, identificando cada componente (74LSxx o similar), sus pines, cuáles pines están conectados o desconectados.
	27. Mostrar que cada etapa del diseño ha sido realizada empleando las técnicas apropiadas enseñadas en la teoría (que ustedes no se basan en concepciones empíricas). Recuerde que a usted no se le va a calificar sólo por el resultado, SINO TAMBIÉN POR LA METODOLOGÍA. Usted tiene que demostrar que sabe. El resultado es importante, pero es la metodología la que garantiza el resultado (con un poco de ingenio...)
	28. Validar las justificaciones para las diferentes decisiones que se tomaron.
	29. Indique EL TIEMPO que emplearon en cada parte del diseño (análisis, diseño, fabricación, pruebas)
	30. Entregar borrador con la lista de componentes: Número de parte, descripción, precio estimado.

ESPECIFICACIONES PARTICULARES

	1. Debe ser un sistema basado en microprocesadores. Se usará el HC908.
	2. Recibir el Modo de Adquisición "En – línea", en el cual se adquiere un dato y se transmite al PC, donde es visualizado. PROTOCOLO: La comunicación con el PC se establecerá sin protocolo; el "Sistema de adquisición de datos portátil simplemente enviará, de manera permanente, los datos adquiridos, codificados en ASCII, terminados en caracteres CR, LF.
	3. Verificar que el PC lee convenientemente la información recibida.
	4. Verificar que el sistema deberá ser capaz de adquirir, dos señales analógicas de ±5 Voltios.
	5. Verificar que las señales deben poder ser leídas en modo diferencial o en modo simple.
	6. Indique y demuestre la precisión de su sistema de adquisición, que debe aprovechar todo el rango dinámico de la entrada.
	7. Indique y verifique la tasa máxima de muestreo que Usted puede ofrecer.
	8. Verifique que cumple con el tiempo máximo de adquisición.
	9. Muestre las 8 señales digitales, y que sean programables como entradas o salidas.
	10. Verifique que su sistema incorpora memoria externa, y que ésta funciona.
	11. Verifique el comportamiento del sistema de comunicación serial con el PC (¿velocidad?)
	12. Verifique que el subsistema de Acondicionamiento tiene la electrónica adecuada para el manejo de dos señales analógicas, según su especificación de sus características.
	13. Constate que su subsistema de Acondicionamiento opera correctamente con lo que sería la salida típica de los sensores, y con las características de las señales que usted va a medir (por ejemplo, lazos de corriente, voltajes DC ó AC, etc. Especifique cuáles son)
	14. Muestre que el rango dinámico de la entrada de la etapa de adquisición se ajusta a la resolución del sistema.
	15. Indique qué medidas tomó para evitar el aliasing; verifique el rango de frecuencias de funcionamiento.
	16. OPCIONAL: Muestre si incluyó un control y programación de ganancias y offset en la etapa de acondicionamiento.
	17. OPCIONAL: Muestre si incluyó la generación de salidas para actuar sobre circuitería que genere a su vez la señal a ser procesada (actuadores sobre circuitería externa para implantar la aplicación final).

Se debe suministrar impresa, cualquier documentación relacionada, como la Especificación de Características de sus señales de entrada.

SAT: DOCUMENTO DE ACEPTACIÓN EN SITIO (PARA EL "DATA LOGGER")

Grupo #____, Integrantes: _____,
_____

CARACTERÍSTICAS GENERALES

	1. El día de la entrega deberá tener su equipo listo para ser evaluado, a la 1:30 pm en punto. El profesor decidirá sobre la marcha el orden de la recepción. Ese día no es momento de armar, verificar, montar, soldar o probar... Es solo día de entrega.
	2. Es obligación de los estudiantes tener lista la infraestructura necesaria para verificar cada cosa: Programa para probar la funcionalidad de la memoria, método para probar las comunicaciones, etc., al menos para los ítems descritos a continuación.
	3. El equipo debe cumplir con todas las características que se probaron en la primera entrega (FAT)

ESPECIFICACIONES PARTICULARES

	1. Demostrar el modo de configuración. ➢Al menos debe poder escoger y programar cada uno de los 3 siguientes modos de operación, y ➢el número de bits de entrada y salida.
	2. Demostrar el modo de Adquisición "En – línea", en el cual su equipo enviará, de manera permanente, los datos adquiridos, codificados en ASCII, terminados en caracteres CR, LF. En el PC debe verse un listado eterno con el siguiente formato:
	3. Demostrar el modo de adquisición "Independiente", en el cual el sistema funciona parecido a un "Data Logger". Adquiere datos por un período de tiempo fijo y a una tasa de muestreo fija, esos datos son guardados para su posterior transmisión a un Sistema Central (basado en PC). La comunicación con el PC se establecerá mediante el protocolo XModem.
	4. Modo de adquisición "Por ráfagas", en el cual el sistema adquiere datos por un período de tiempo finito y a tasa definida, luego bajo petición de un sistema central transmite (XModem) lo adquirido y espera una nueva orden de adquisición.
	5. Demostrar la funcionalidad del subsistema de Aplicación Final que Usted diseñó.
	6. OPCIONAL: Muestre si incluyó un control y programación de ganancias y offset en la etapa de acondicionamiento.
	7. OPCIONAL: Muestre si incluyó la generación de señales para actuar sobre circuitería que genere a su vez la señal a ser procesada (señales de actuación sobre circuitería externa para implantar la aplicación final).
	8. OPCIONAL: Muestre si incluyó mecanismos extra (crc) para evitar problemas de comunicaciones.

DOCUMENTACIÓN DEL SISTEMA

	1. Deberá entregar la lista de componentes del prototipo: ➢Número de parte, ➢descripción (incluye: Si es montaje convencional o de superficie, características especiales como temperatura de operación, voltaje, etc.), ➢proveedor, ➢"third party", ➢cantidad requerida, ➢mínimo lote por pedir, ➢precio unitario y precio total, ➢tiempo de entrega, ➢comentarios (back log, descontinuado, si hay reemplazo).
	2. Deberá entregar todos los esquemáticos con sus informes explicativos detallados.
	3. Deberá entregar la Bitácora de Trabajo donde recopiló todas y cada una de las ideas, documentación, copias de manuales empleados, problemas encontrados con sus soluciones, planteamientos efectuados en el Desarrollo del Sistema de Adquisición, ideas desechadas, etc. La Bitácora es su diario de trabajo.
	4. Deberá entregar Manuales Técnicos de Operación.
	5. Debe entregarse el arte (layout) del impreso de esta tarjeta diseñado en EAGLE, ORCAD, etc.
	6. Debe suministrar las características completas para el par de señales que emplearon en su aplicación indicando (sin limitarse a): Frecuencia de operación, temperatura, rangos, impedancias, voltajes, corrientes, fuerzas, etc.
	7. Suministrar la especificación completa del subsistema de Aplicación Final que Usted diseñó.
	8.
	9.
	10.
	11.
	12.
	13.

Se debe suministrar impresa, cualquier documentación relacionada, como la Especificación de Características de sus señales de entrada.

BIBLIOGRAFÍA

Además de la reseñada *in situ* dentro del texto:

Basic concepts of real-time operating systems.mht:
http://linuxdevices.com/articles/AT4627965573.html

D_ Kalinsky Associates - Whitepaper Architecture of Device I-O Drivers.mht
http://www.kalinskyassociates.com/Wpaper4.html

STATEMENT OF WORK, Marc A. Brauer, rfsow.pdf

Varios:
http://phoenix.nmt.edu/~rison/ee321_fall02/labs/

http://www.epanorama.net/ (incluye bar codes)

Buscar pelotas de golf:
http://phoenix.nmt.edu/~rison/ee382_spr03/ee382.html

Todos los cursos: analógicos, digitales...
http://phoenix.nmt.edu/~rison/ee321_fall02/homepage.html

Capítulo 3

CIRCUITOS DIGITALES

Motivación y ambientación

INTRODUCCIÓN

A PARTIR del segundo tercio del siglo 19 se conocía ya el electroimán; con él se fabricó el primer dispositivo de código Morse, que evolucionó hacia el telégrafo y luego al teléfono. A comienzos del siglo 20 ya se empleaban complejos circuitos hechos con relés, sobre todo centrales de conmutación telefónica. A pesar de que Boole desarrolló su álgebra a mediados del siglo 19, no fue hasta 1938 que Claude Shannon planteó aplicarla a los Sistemas Digitales. Quine primero, Karnaugh y McCluskey después, consolidaron el estado actual del basamento del diseño digital.

En un lapso tan amplio es muy probable que ya se haya escrito todo lo que puede decirse como introducción al tema. Sin embargo, es muy importante y resulta siempre conveniente poner en **Perspectiva** la Ingeniería Digital para quienes van a estudiarla; tal es el propósito de la siguiente presentación, como introducción y ambientación para nuestros lectores y alumnos que se adentran en el torbellino actual de las Técnicas Digitales.

REALIDAD Y ELECTRÓNICA ANALÓGICA

LOS fenómenos físicos con los que interactuamos en el mundo real son **mecánicos**, y los percibimos como si fueran continuos aunque casi todos, al estilo de la emisión de energía, sean cuánticos, o discretos. Antes del advenimiento de la electrónica, la manipulación de estos procesos físicos: Su amplificación, transmisión, medición y control..., se hacía por medios mecánicos, hidráulicos o neumáticos. Por ejemplo:

- La veleta servía para controlar la posición de un molino de viento;
- Una palanca operaba como timón de una embarcación;
- Un cuerno o bocina amplificaba sonidos.
- La transmisión de la fuerza era hecha mediante ejes, correas o tubos.
- Un dispositivo para suavizar los movimientos bruscos (filtro) se armaba con amortiguadores (muelles, resortes, cámaras con aceite), y
- Un gobernador de Watt controlaba la velocidad, aprovechando el giro de unas esferas.

¿POR QUÉ EL NOMBRE DE "ANALÓGICO"?

Estos sistemas vinieron con la electrónica, llamados así porque operan sobre voltajes, corrientes o impedancias, que son ***similares*** –o ***análogos***– a los fenómenos físicos que representan. Estas señales electrónicas, análogas a las variables reales, tienen su ***misma forma***, y son obtenidas directamente del campo, o del proceso, mediante transductores (transducers, Xducers).

Por ejemplo, observemos el fenómeno que ocurre cuando hablamos: El sonido de nuestra voz es, en realidad, un conjunto de alteraciones en la presión del aire que nos rodea, que obligan a la masa de gases a moverse con desplazamientos ondulatorios de vaivén, cuyas elongaciones están en proporción directa a la intensidad sonora o ***volumen*** de nuestras expresiones (gritos, susurros). Las diferentes frecuencias con que varía la presión corresponden a la ***altura*** de los sonidos que emitimos: Mayores vibraciones para los agudos, y menores en las voces graves.

EJEMPLO DE TRANSDUCCIÓN

- La posición –lineal o angular–,
- El desplazamiento,
- Las fuerzas,
- Temperaturas,
- Vibraciones,
- Presiones neumáticas,
- Flujos (de fluidos),
- Radiaciones,
- Niveles (de líquidos),
- PH (químico),
- Luz, etc.

son ejemplo de cantidades que se convierten, o traducen, a variables electrónicas que son directamente proporcionales a los fenómenos mecánicos externos que modelan. (NB: En comunicaciones, las antorchas y banderas de señales, y el telégrafo, fueron digitales desde el principio...)

ELECTRÓNICA DIGITAL: CARACTERÍSTICAS IMPORTANTES

1) REPRESENTACIÓN NUMÉRICA

Las técnicas ***Digitales*** son una rama de la electrónica, que toman su nombre porque no representan los fenómenos como los sistemas analógicos, sino que lo hacen expresándolos mediante una ***serie de números***, o cifras digitales (**dígitos**). La primera ventaja de representar así los fenómenos externos estriba en que el ***procesamiento es eminentemente aritmético***. Por ejemplo, para amplificar una señal al doble de su valor se requiere... simplemente multiplicarla por 2. Para filtrarla y eliminarle componentes indeseadas basta con aplicarle un algoritmo matemático a la representación numérica, o digital, de la señal de entrada. Un proceso de control se convierte en un procedimiento aritmético, que analiza los números que representan la señal de entrada en diversos instantes de tiempo y toman decisiones. Por ejemplo, un control digital de temperatura es un dispositivo que lee las cifras que representan cada valor de la variable externa, y decide si hay que encender el calefactor, o apagarlo, al sobrepasar los límites inferior o superior impuestos al artefacto.

Es una gran ventaja poder sumar, multiplicar, integrar... o efectuar cualquier otro procesamiento numérico sobre las variables. Y es muy importante también que todas ellas compartan la misma representación numérica, y que los algoritmos dependan sólo de este factor, y no de que unos números representen, por ejemplo, temperaturas mientras que otros simbolicen flujos.

2) VARIABLES DISCRETAS, INMUNIDAD AL RUIDO

La segunda gran característica de los sistemas digitales es que son ***discretos***, como contraposición a los analógicos, que son ***continuos***. Esto quiere decir que solamente son significativos ciertos valores, y como no se necesita distinguir entre todos ellos, los componentes que los procesan son sencillos de elaborar. En el caso de ***la variable más elemental***, se requiere que represente sólo dos posiciones: Abierto o cerrado; saturación o en corte; cinco voltios o cero. Nada pasa en el intermedio, así que, por ejemplo, si una señal debe estar en un nivel alto o bajo, y tal situación va a ser representada por 5 ó 0 voltios, y si en un momento determinado dicha señal se encuentra en 4 voltios, el discriminador puede asumir que el estado es "alto". Y si vale 3 V también. En cambio, si la señal mide 2 V el dispositivo indicará que el estado es "bajo", y hará lo mismo si el valor es de 1 V. (Estas cifras se presentan como ejemplo, no son reales...)

Se denomina ***inmunidad al ruido*** a la diferencia entre aquellos dos rangos que sean claramente discernibles como un valor o el otro. Los sistemas digitales tienden a ser altamente inmunes al ruido. Señales espurias de importante magnitud pueden superponerse a las variables sin que ocasionen la interpretación errónea de las mismas.

Está claro entonces que los sistemas digitales discriminan el ruido por niveles: De cierto valor hacia arriba, se supone la variable en el estado "alto", y disminuyendo desde allí, que está en "bajo". Eliminar el ruido en los sistemas analógicos requiere filtros, que operan dejando pasar la banda de frecuencias de la señal y eliminan las demás, que pertenecen al ruido. Pero si una interferencia, aunque sea pequeña, tiene el mismo componente armónico que la señal, puede resultar imposible de eliminar.

3) EXACTITUD, PRECISIÓN Y RESOLUCIÓN

- **Exactitud** ("*accuracy*") es un indicativo de **qué tan de cerca** representa, la señal medida, el valor real de la variable. Usualmente se especifica de dos maneras: **a)** como un porcentaje de la escala total (**full scale**) o **b)** como un porcentaje **de la lectura** actual.

  Por ejemplo, una señal analógica puede especificarse con una exactitud de ***0.1% de la escala total***, lo que significa que para cualquier valor leído el error absoluto será el mismo que para la escala completa. Esto implica que una medición que se ubique en el 10% de la escala, tendrá un error del **1%**.

  Instrumentos con especificaciones de error de ***1% de la lectura***, son iguales, o mejores, que aquellos con exactitudes del 1% de la escala completa.

  La exactitud refleja los inconvenientes introducidos por todas las posibles fuentes de error; por tanto depende de factores tales como ***resolución, linealidad, repetibilidad, ganancia, estabilidad y rango dinámico***.

- La **Precisión** ("*precision*") se refiere al **número de dígitos** significativos con los que puede expresarse una variable.

- **Resolución** ("*resolution*") indica el **menor cambio discernible** con un instrumento.

A veces precisión y resolución significan lo mismo. Nótese que precisión **no** implica exactitud; es posible que pueda discernirse un cambio con una **resolución** de 1 parte en 10.000, y que la **exactitud** sea menor: Una en 1.000, por ejemplo. Al revés esto no es posible: ***¡ no puede tenerse una exactitud que sea superior a la precisión o a la resolución !***

La tercera gran característica de los sistemas digitales es que se puede incrementar su ***resolución*** simplemente aumentando el número de dígitos elementales (**bits**) que se usan en la representación: 8, 10, 12, 16 y 24 bits son comunes para codificar música en un CD de audio. Es factible aumentar la exactitud con que se manipulan los datos ***adentro*** de los sistemas digitales, a cualquier valor deseado. Y las informaciones pueden ***adquirirse*** desde el exterior con niveles impresionantes también de exactitud y resolución.

Ejemplo: Los sistemas de seguimiento de satélites, con **láser** y codificadores de posición angular (**shaft encoders**), obtienen con 18 bits una resolución mejor que 5s de arco (2^18 = 262.144 partes de circunferencia).

Ejemplo: Un termómetro digital de cuarzo tiene grandes niveles de exactitud y resolución. Miden la frecuencia, dependiente de la temperatura, de un oscilador de cuarzo. Tiene resoluciones de 0.0001°C, y exactitudes absolutas de 0.1°C.

En el eje vertical de la Figura 1 se indica una cantidad de **referencia**. De ella se hacen varias medidas, que se centran alrededor de ***otro*** valor. La distancia que separa estas dos (2) cifras es la **exactitud** ("***accuracy***"). Mientras más exacto es un sistema, el promedio de las medidas tomadas se acercará más al valor de referencia. Ahora, la mayor o menor **precisión** se representa por lo ancho o estrecho de la campana de Gauss. Mientras **más compacta** sea la gráfica habrá **mayor precisión** en las medidas, porque una curva es apretada si casi todas las aproximaciones indican el mismo valor, y es ancha si las estimaciones difieren bastante unas de otras (menor precisión).

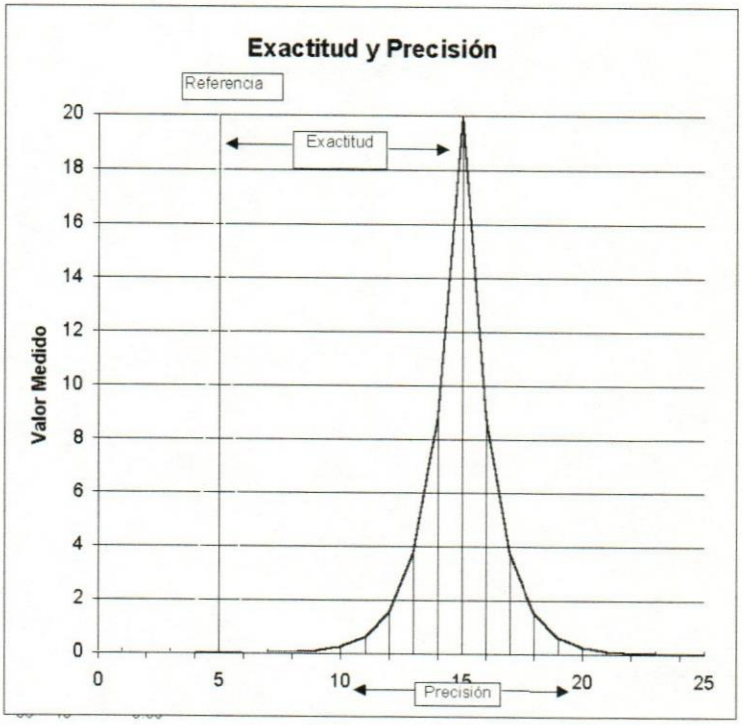


Fig. 1: Exactitud, Precisión, Resolución

Un sistema ideal tiene buena exactitud y una gran precisión. Se puede conseguir aceptable precisión y pobre exactitud. En la figura esto indicaría que todas las medidas se centran, muy estrechamente, alrededor de un valor... ***¡ equivocado !***

¡ Lo que sí **no** puede tenerse es una buena exactitud con mala precisión !

La **resolución** ("***resolution***") indica también la **granularidad** con que pueden diferenciarse unas medidas de otras. Si la resolución fuera mala, el gráfico de la Fig. 1 quedaría formado por pequeños segmentos de líneas rectas horizontales. Mientras peor sea la resolución, más "discreto" será el gráfico de las medidas (como una escalera). Si la resolución aumenta, los valores asumen mayor cantidad de posiciones diferentes, y el gráfico se verá más suave o "continuo".

4) CONTROL DE ERRORES: DETECCIÓN Y CORRECCIÓN

Además de lo ya dicho en relación a la inmunidad al ruido para cada variable elemental (bit), y que mayores resoluciones se logran agregando bits, también pueden concebirse ***códigos*** que controlen errores que, aunque difíciles de ocurrir, podrían presentarse en la lectura de bits individuales, y que los ***detecten*** e incluso los ***corrijan***. Un ejemplo en telecomunicaciones consiste en añadirle un bit (de paridad) a cada conjunto de 8 (byte), de tal manera que siempre el número de "unos" sea par o impar. Si se presentara un error en un bit de uno de los símbolos (cambio de un cero por un uno, o viceversa), podría fácilmente detectarse ese error contando en el receptor el número de bits, y viendo si cumple con

la paridad establecida. Si hay un error puede indicársele al transmisor que repita la comunicación (este mecanismo se conoce como ***ARQ: Automatic Repeat Request***)

Otra forma de ***control de error*** consiste en que, una vez puesto a punto un sistema **digital**, basta con duplicarlo para obtener copias que funcionen exactamente igual que el prototipo, sin necesidad de ajustes ni calibraciones, propios de los sistemas analógicos, que usan potenciómetros, condensadores variables e inductores de núcleo ajustable. Los de esta tecnología, una vez que uno los reproduce, necesitan pasar por las manos de personal técnico especializado que los entone; de lo contrario no funcionan. Pero uno digital no precisa ajustes (al menos, en teoría...)

5) ALMACENAMIENTO DE INFORMACIÓN

Los sistemas digitales graban datos de maneras que perduran en el tiempo mucho más allá que los sistemas análogos, que emplean dispositivos tales como condensadores en los que almacenan ciertos voltajes, pero que luego se van descargando y perdiendo paulatinamente la información. Esto es cierto bien sea que los valores en los dispositivos digitales estén representados por huecos, magnetizaciones de elementos en discos, o posiciones de interruptores.

6) PROCESAMIENTO DE LA VARIABLE TIEMPO

Un sistema digital que toma muestras a determinados intervalos puede luego procesarlas a velocidades diferentes, de forma repetida si se quiere (para efectuar promedios y eliminar el ruido), o, por ejemplo, para incluir dentro de una evaluación, una copia de sí misma, a lo mejor desplazada en el tiempo (si lo que se quiere es obtener la convolución). Estas modalidades son infinitamente más fáciles de realizar en un sistema digital que en su contrapartida analógica.

El procesamiento en el dominio de la frecuencia, de señales dependientes del tiempo, es muy simple de manera digital, aplicándoles por ejemplo transformadas del tipo FFT (Fast Fourier Transform)

Otro aspecto notable relacionado con la manipulación de la variable tiempo lo constituyen los sistemas que programan o **secuencian** la operación de plantas o procesos, y que son fundamentales para la automatización industrial, lo mismo que las "Máquinas de control numérico", que leen sus instrucciones desde una cinta o un disco externo y producen, en serie, piezas metal-mecánicas trabajadas con taladros, esmeriles y otra diversidad de utensilios para el maquinado.

7) ESTANDARIZACIÓN

Una gran ventaja de los sistemas digitales es la existencia de multitud de ***componentes estándar***, como compuertas, memorias, microcontroladores y microprocesadores, y de subsistemas completos similares a impresoras, monitores, módems, computadoras, interfaces de redes, motores de paso, codificadores, teclados y sistemas para el despliegue de información. La cantidad de transistores en los circuitos integrados se duplica cada 18 meses (ley de Moore) y sus precios bajan irremisiblemente.

Después del **hardware** programable vino el **software**, que lo potencia de una manera exponencial, haciéndolo reusable. Antes, un televisor era sólo eso. Ahora, un PC puede ser un TV, o un pasacintas (VHS, CD, DVDs), o un sistema de composición musical, o uno para la autoría de textos, etc.

CONVERSIÓN ENTRE BASES

ESTE tema se trata con suficiente propiedad en los libros de texto; sin embargo, por lo general los estudiantes no logran entenderlo al punto de no necesitar calculadora para las operaciones. Los siguientes ejemplos se enfocan solo en números enteros; los procedimientos pueden extrapolarse de manera sencilla a la parte fraccional, pero esta representación se emplea mucho menos en el día a día.

1) CONVERSIÓN DE DECIMAL A BINARIO O A HEXADECIMAL

MÉTODO CONVENCIONAL

El procedimiento común que se emplea para la conversión de decimal a binario consiste en ir dividiendo el número decimal por 2 hasta que ya no pueda continuarse; luego se toman los residuos de cada operación, ***en sentido inverso***. Por ejemplo: Convertir `843` a binario:

843 ÷ 2 = 421, R=1	421 ÷ 2 = 210, R=1	210 ÷ 2 = 105, R=0	105 ÷ 2 = 52, R=1
52 ÷ 2 = 26, R=0	26 ÷ 2 = 13, R=0	13 ÷ 2 = 6, R=1	6 ÷ 2 = 3, R=0
3 ÷ 2 = 1, R=1	1 ÷ 2 = 0, R=1		

La tabla se lee de izquierda a derecha, y de arriba a abajo. Los residuos se toman comenzando por el último, que TODAS LAS VECES es "1". De hecho, la división final siempre es la misma (uno dividido por 2: Cociente 0 y residuo 1), no importa de cuál número, mayor que cero, estemos hablando.

Entonces, `843` en binario es: `1101001011`.

MÉTODO RECOMENDADO

Para acelerar el proceso hay que realizar menos divisiones, lo que se logra convirtiendo el número a una base auxiliar intermedia, más grande, siendo la más empleada la base 16 (hexadecimal). Una vez hecha esta conversión, obtener el resultado en binario es inmediato, puesto que las bases que son potencias de dos, como la octal y la hexadecimal, se convierten a binario por simple inspección.

Sin embargo, dividir por 16 no siempre se le facilita al alumno, que se ayuda a veces con la calculadora. ***El método recomendado es el de convertir a octal (base 8)***, pues las divisiones por 8 sí son muy sencillas, y de ahí a binario o a hexadecimal, según se solicite. El ejemplo anterior sería:

843 ÷ 8 = 105, R=3	105 ÷ 8 = 13, R=1	13 ÷ 8 = 1, R=5	1 ÷ 8 = 0, R=1

`843` en octal es: `1513`, y en binario (directo del octal): `1 101 001 011`, idéntico al ejemplo original.

Si se pide en hexadecimal, el resultado puede extraerse del número binario en forma trivial e inmediata:

`11 0100 1011` (reacomodado en grupos de a cuatro, de derecha a izquierda): `34B` (hex).

2) CONVERSIÓN DE BINARIO A DECIMAL

MÉTODO CONVENCIONAL

El método que más se emplea es el de obtener el valor en decimal de cada "1", de acuerdo a la notación posicional. Ejemplo: Convertir a decimal el número: `1011 0100`: (Observe que TODAS las cantidades binarias, menos el cero, comienzan con "1" a la izquierda. Sin embargo, a veces se agregan ceros, no significativos, para hacer que el último grupo de la izquierda tenga 3 o 4 bits y así acomodar con propiedad los números octales o hexadecimales).

```
128 64 32 16   8 4 2 1
 1   0  1  1   0 1 0 0

128 + 32 + 16 + 4 = 180 decimal.
```

MÉTODO RECOMENDADO

Se denomina "***doble y sume***", y consiste en procesar los números binarios, de izquierda a derecha, con la siguiente aproximación: Tome el primero de la izquierda (que debe ser un "1"; los ceros no significativos se pueden ignorar); si no hay más columnas ya ha obtenido el resultado; si hay una más, multiplique por **2** la cifra que lleva (***doble***) y ***súmele*** el valor que se encuentre en esa columna de la derecha.

Para nuestro ejemplo el resultado es así:

`1 0 1 1  0 1 0 0`. Tomamos el primer "1". Como hay al menos una columna a la diestra, multiplicamos nuestro "1" por **2**, y le sumamos lo que hay en ella, que en este caso es un "0". El resultado hasta ahora es "2".

Como aún hay otra más a la diestra, ***doblamos*** el valor que tenemos y ***sumamos*** el "1" que hay en esta columna: `2*2 + 1 = 5`.

Para la siguiente columna: `5*2 + 1 = 11`.

Siguiente: `11*2 + 0 = 22`.

Próxima: `22*2 + 1 = 45`.

Penúltima: `45*2 + 0 = 90`.

Última: `90*2 + 0 = 180`, que era el valor original.

3) CONVERSIÓN DE HEXADECIMAL A DECIMAL

MÉTODO CONVENCIONAL

El método que más se emplea es el de obtener el valor en decimal de cada columna, de acuerdo a la notación posicional.

Ejemplo: Convertir a decimal el número: **9AB**:

```
256 16  1
 9   A  B
```

9*256 + 10*16 + 11*1 = 2475 decimal. (Recuerde que en hex, A es 10 y B es 11)

MÉTODO RECOMENDADO

Es el equivalente al "doble y sume", y consiste en procesar los números hexadecimales, de izquierda a derecha, de la siguiente manera: Se toma el primero de la izquierda, si ya no hay más columnas ya se ha obtenido el resultado; si hay una columna más, se multiplica por **16** la cifra que llevemos y se le suma el valor que se encuentre en esa columna de la derecha.

Para nuestro ejemplo el resultado es así:

9 A B. Tomamos el "9". Como hay una columna a la derecha (al menos), multiplicamos nuestro "9" por 16, y le sumamos lo que hay a la derecha, que en este caso es un "10". El resultado hasta ahora es **9*16 + 10 = 154**.

Como hay una columna a la derecha, multiplicamos por 16 el valor que tenemos y sumamos el "11" que hay en esta columna: **154*16 + 11 = 2475**.

4) CONVERSIÓN DE OCTAL A DECIMAL

Es similar al procedimiento convenido para convertir de hexadecimal a decimal, sólo que se multiplica ahora por 8, que es la base.

LAB0

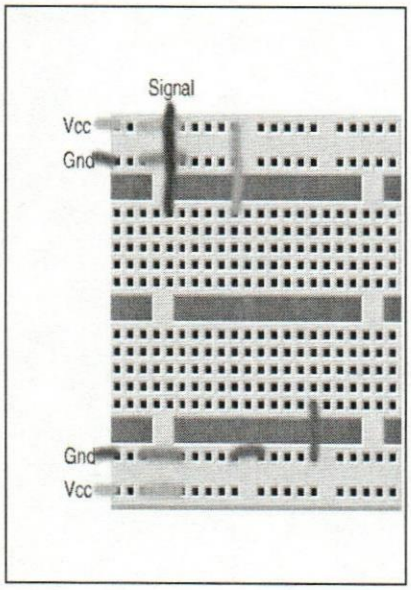
ALGUNOS estudiantes llegan a sus prácticas de circuitos o electrónica digital con deficiencias que les dificultan alcanzar el mayor provecho posible. En estas breves notas se pretende subsanar varias carencias, y advertir sobre los procedimientos que deben seguirse en la elaboración de sus proyectos, las dificultades con las que pueden encontrarse, y una que otra recomendación importante.

Los temas que se repasan son los siguientes:

- Uso del Laboratorio
- inserción y extracción de circuitos integrados en el Protoboard.
- Alimentación de los componentes.
- Consideraciones con los interruptores.
- Carga máxima de las señales (Fan out).
- Entradas no usadas.
- Cuidados con el manejo de los CMOS.
- Uso obligatorio de condensadores de desacople.
- Leds, Indicadores 7 Segmentos y equipos de prueba.
- ¡ Qué va a fallar en sus proyectos de laboratorio !

PROTOBOARD

Cada tira del "protoboard" suele tener un canal central. A lado y lado hay 64 filas de 5 columnas de terminales para Circuitos (huecos), de **0.1 in.**, correspondientes al espaciado estándar de los pines en los Circuitos Integrados. Los 5 huecos en cada columna están interconectados eléctricamente entre sí. El espacio en el canal central es de **0.3 in.**, el mismo que existe entre las líneas de terminales de los circuitos estándar de 14, 16 y 18 pines en los dispositivos DIP (**D**ual **I**n-line **P**ackage). Al introducir uno de estos circuitos quedan 4 huecos sin usar por c/terminal del chip, que se emplean para interconexiones punto a punto con otros circuitos, las alimentaciones, los LEDS de salidas y los interruptores de entrada.



Circuitos más grandes, como los de 24 terminales, tienen una separación entre líneas de **0.6 in.**, por lo que dejan menos agujeros libres para las interconexiones. También pueden incluirse componentes discretos como *resistencias*, *condensadores* de paso o desacople y *transistores*. (Nota: Resistencias de más de ¼ de vatio, y otros componentes como condensadores grandes, tienen terminales muy gruesos que ¡ **dañarán el conector del "protoboard" si se los fuerza** !)

Normalmente hay también dos columnas a lado y lado, con terminales interconectados, que sirven para distribuir alimentación, tierra y señales comunes como relojes, resets, etc.

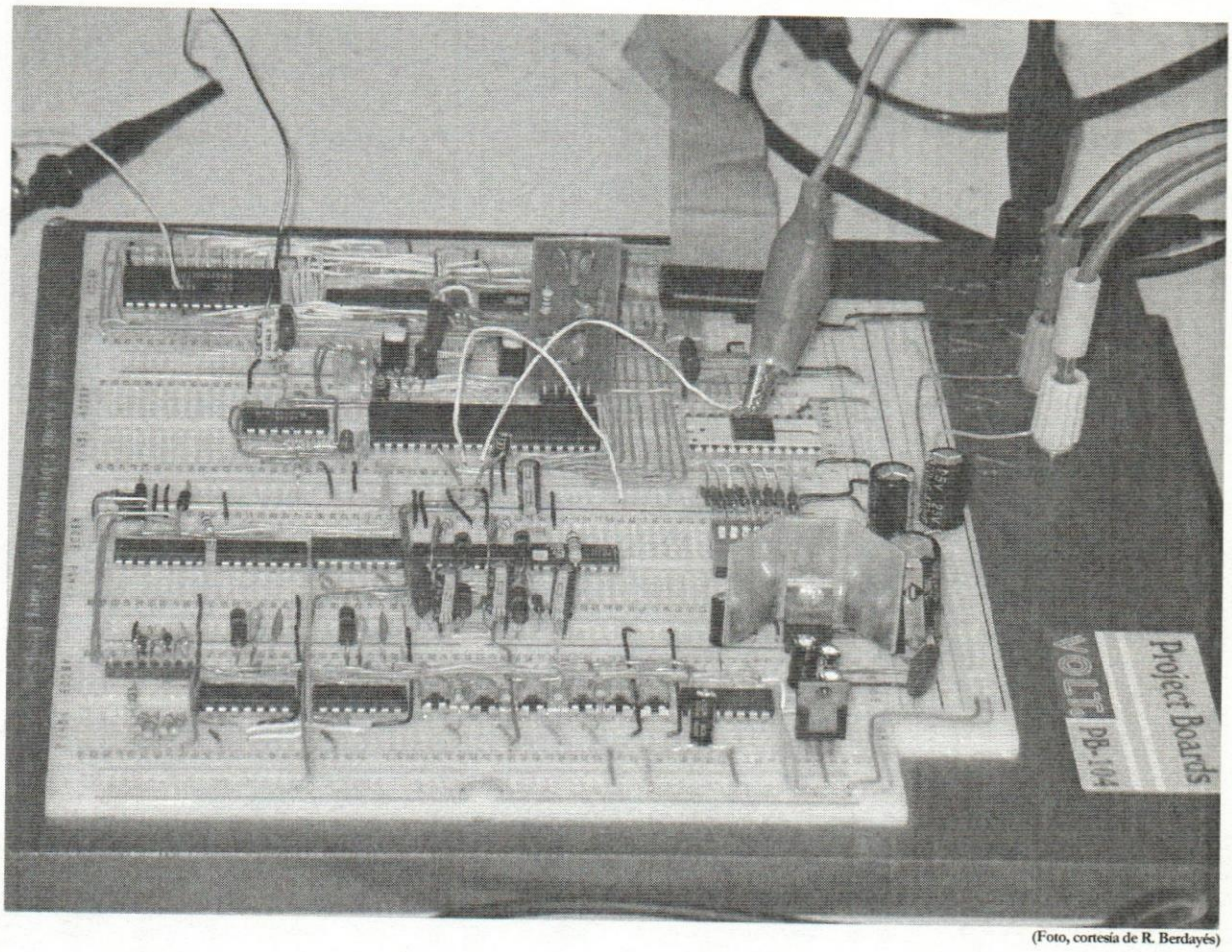
El cableado debe hacerse con alambre tipo telefónico, de un solo conductor (usualmente de tamaño **#22**). En ningún caso debe emplearse alambre de múltiples hilos o "pelos". (Si no lo sabe ya, debe averiguar por su cuenta cómo se numeran los conductores, y tener una idea exacta de lo que es un conductor telefónico, #22)

CIRCUITOS INTEGRADOS Y ALAMBRES

Los chips son sencillos de insertar en el protoboard, una vez que se los acondiciona, porque no suelen venir con los terminales verticales, sino un poco inclinados hacia fuera, para que las máquinas de inserción automática puedan manejarlos con facilidad. Así que a los ICs nuevos, hay que retornarles sus terminales a la vertical, de manera que su espaciamiento sea exactamente 0.3 in. Esto puede hacerse o con los dedos de la mano, o con unas pinzas largas.

También pueden insertarse "sockets" en el protoboard, que luego alojarán ICs que resultan costosos. De esta manera, nunca se tienen que mover estos ICs de sus sockets.

Esta es una distribución aceptable de componentes en un protoboard.



(Foto, cortesía de R. Berdayés)

Para remover los chips **tiene** que usar una herramienta especial de extracción, porque **quitarlos a mano los destroza** (daña sus terminales ¡ y los dedos !). Es una especie de "U" metálica, cubierta de material plástico aislante, muy sencilla y económica de conseguir. **¡ No vaya al laboratorio sin ella !**

Todos los chips deben insertarse con **la misma orientación** para facilitar el alambrado y el "debugging". Normalmente los ICs tienen una marca o muesca que señala dónde está el terminal #1. La mayoría de los ICs tienen la tierra abajo a la extrema derecha (pin #7) y la alimentación arriba en el terminal situado más hacia la izquierda (pin #14), y también los protoboards exhiben normalmente la alimentación con esa misma orientación. Recuerde verificar doblemente la localización exacta de los terminales de energía eléctrica en sus chips. Colocar mal la fuente es un ***error*** casi siempre ***destructivo*** para el IC (porque queda, por ejemplo, el Vcc a la salida de una puerta).

Los cables se insertan y se remueven mejor con la ayuda de unas **pinzas**. Se necesita también un **pelacables** que permita cortarlos con la longitud adecuada y removerles ½ in. aproximadamente en los extremos. Los cables que se inserten deben ser ***nuevos***, porque así son más fáciles de introducir. Los cables muy usados son difíciles de insertar, y si se logra finalmente colocarlos, *probablemente dañen el hueco del protoboard.* Si se quiere reusar un cable que ya no entra con facilidad, corte de nuevo el extremo y vuélvalo a pelar. La punta del cable debe quedar biselada en ángulo de 45°.

No pase cables por encima de los ICs. ¡Rodéelos! Esto facilita el debugging y el cambio de componentes, en caso de ser necesario. Los cables deben estar lo más aplanados contra el protoboard que sea posible. Y deben estar ORDENADOS. Trate de mantener codificadas, mediante colores en los cables, las categorías de las señales que se distribuyen. Por ejemplo:

Rojo: 5V **Negro**: Tierra **Morado**: Control **Naranja**: Datos **Café**: Direcciones

En un cableado mal hecho, remover una conexión puede tener consecuencias impredecibles en el comportamiento del circuito, pues pueden extraerse, sin pensar, hilos que no se deseaba mover.

En un cableado ordenado, el profesor puede, al menos, tratar de ayudar cuando algo falle.

FUENTES DE ALIMENTACIÓN

Normalmente las fuentes de energía eléctrica vienen protegidas de manera que un corto a su salida lo único que hace es desactivarla hasta que se lo elimina, pero lo que sí resulta seguramente **catastrófico** es, por ejemplo, conectar invertida la polaridad de Vcc y Tierra!!! Casi siempre que se conecta invertido un IC es **garantía** de que el chip se quemará.

SIEMPRE ALGUIEN INSERTA AL REVÉS EL MICRO EN EL PROGRAMADOR

Aunque **una única salida** de un IC TTL puede cortocircuitarse a tierra sin que el chip se queme por disipación, lo que sí es cierto es que conectarle ***el Vcc a la salida de la compuerta*** **la quemará**, si ésta va a saturación. A veces este tipo de corto circuito no daña un IC CMOS, pero sí le reduce su vida útil, o termina por quemarlo si dicha situación se prolonga en el tiempo por varios minutos.

Entonces, cuando alimente un proyecto, **lo primero** que hay que revisar es que ninguno de los componentes eche **HUMO**. Y luego, tocarlos para ver que ninguno se caliente fuera de lo normal (con precaución, porque los micros suelen calentarse bastante; no sea que se queme la mano).

Los IC CMOS tienen un problema particular: Si se aplica señal en sus entradas, ANTES de que se le haya conectado la alimentación (Vcc y tierra), puede dañarse la estructura interna del IC, o pueden activarse SCRs parásitos, que hacen que si a continuación se aplica la alimentación de energía, el IC se ponga en corto (Vcc a tierra)

INTERRUPTORES

Los "switches" de reposición, los de corredera, los "push buttons" y los contactos de relés, pueden servir como entradas alimentando 0 y 1 al circuito. Normalmente estos interruptores presentan lo que se conoce como "***bouncing***", que son múltiples aperturas y cierres, que pueden durar entre 10 y 40 ms (menos de 20 ms típico). Hay que tener cuidado de conectar este tipo de interruptores a través de latches SR u otros dispositivos similares que garanticen que cada conmutación se refleje como una sola transición porque, de lo contrario, si se alimenta, por ejemplo, un contador, éste avanzará muchos pasos cuando debería contar sólo uno.

FAN OUT

Es la capacidad que tienen las salidas de unos dispositivos de manejar las entradas de otras compuertas. Normalmente se mide su número; por ejemplo, si la salida de una compuerta TTL puede absorber **16 mA** en su nivel bajo, y una entrada de ese mismo tipo de compuerta entrega **-1,6 mA** cuando se la lleva al mismo nivel, puede decirse que el fan out es **10** para voltajes bajos. Si, además, ese terminal puede entregar **–400 uA** en nivel alto, y la respectiva entrada consume **40 uA**, el fan out es de **10** también en el voltaje alto. Y el fan out, en general, es el valor más pequeño escogido de entre los dos anteriores. En nuestro ejemplo, el **fan out es de 10**.

En general los CMOS tienen un fan out de 20. (***Verificarlo*** con cada línea de productos en particular).

Uno suele escoger una familia de ICs para trabajar, y emplear TODOS los ICs de allí. En ocasiones especiales, hay entradas que presentan más de una carga unitaria (sucede a veces con los terminales de Clock de ciertos ICs...). Hay que tener cuidado, porque entonces ya no son 10 circuitos los que una salida puede manejar. Pero, en general, es fácil tener en cuenta esta condición cuando sólo se emplean chips de una misma familia.

Si hay que emplear mezcla de familias, por ejemplo CMOS y TTL, la **regla de oro** es que una salida CMOS sólo puede cargarse con UNA entrada TTL. Lo que se suele hacer, entonces, es que para conectar una sección hecha con CMOS con otra realizada con TTL, se coloca en la mitad (a modo de ***interfaz***) unos circuitos **"buffers" TTL o "drivers"**, que hay que seleccionar del manual TTL. En todo caso, hay que **verificar** que la salida CMOS en realidad sí pueda manejar el buffer escogido.

ENTRADAS NO USADAS

Lo que **NUNCA** HAY QUE HACER es dejar **AL AIRE** las entradas que no se usan; es decir, sin conectarlas. Esto es fuente de dolores de cabeza con su diseño. Si es una entrada de una compuerta con funcionalidad AND, la no usada debe llevarse a un "1".

NOTE: La fuente de alimentación *NO ES UN "1"*. Si se la aplica directamente a entradas no usadas, hay una alta probabilidad de que un "overshoot" al aplicar la energía, dañe permanentemente los IC.

Recuerde, por ejemplo, que una compuerta TTL tiene que alimentarse con 5V; 6V es el máximo absoluto, y 7V quemarán con toda seguridad el IC. Los CMOS tienen un rango más grande, pero también hay que respetar los límites.

Entonces, lo que se hace para conectar las entradas no usadas a "1" es:

Una compuerta no usada se la coloca de tal manera que su salida esté en "1", y de allí pueden alimentarse hasta 10 entradas al aire. (Si la compuerta es NAND, basta con conectar sus entradas a "0" para que su salida esté en "1")

ó:

Se colocan a ***Vcc, a través de una (1 K***, por ejemplo), que limita la corriente que pueda generarse en caso de overshoot. Hay que tener en cuenta, de todas maneras, de no conectar una cantidad muy grande de entradas, tal que el voltaje baje de los 3.5 V.

Si la entrada al aire está en una compuerta con funcionalidad OR, hay que conectarla a "0". Normalmente se conectan directamente al bus de tierra del circuito. Aquí no hay problemas con overshoot. ***Casi*** nunca las entradas se colocan a tierra a través de una resistencia.

Note: Las resistencias que se conectan a Vcc se llaman de "**pull up**", y las que pudieran conectarse eventualmente a tierra serían de "**pull down**". (***¿Para qué podría usted querer conectar algo así? !PIENSE!***)

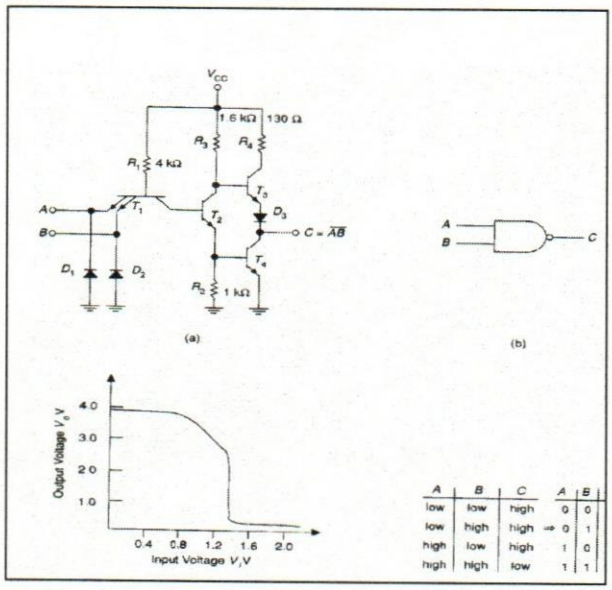
CUIDADOS CON LOS CMOS

NUNCA TOQUE UNA COMPUERTA CMOS. **JAMÁS**. Tocarla es **garantía de dañarla** si usted ha estado un rato en un asiento de cuero, o sobre una alfombra, y si la humedad relativa del ambiente es baja, usted se carga con electricidad estática; como las compuertas CMOS tienen impedancias altísimas en sus entradas, unos pocos electrones desprendidos al tocar el chip, significarán muchos voltios en los terminales del CMOS, **quemándolo**.

Los ICs deben transportarse dentro de bolsas conductoras especiales, que hacen que la posible electricidad estática se reparta, o no se produzca. Permanentemente usted debería conectarse a un terminal que esté conectado a tierra, antes de tocar estos ICs con la mano, o con un cautín o un instrumento. Si toma el micro con la mano y lo guardar en el bolsillo de la camisa..., usted se está

buscando que se le queme. Después no pida prórroga en su proyecto. ¡ Y si es un Pentium ($1,000), usted está despedido del trabajo y quebrado !

CONDENSADORES DE DESACOPLE



Tanto los IC del tipo TTL, con salida "totem pole", como los CMOS, tienen una estructura a la salida compuesta por un transistor encima y otro debajo. Cuando el de arriba está cortado, el de abajo se satura, y viceversa. Que el transistor inferior tenga como carga otro transistor (una carga no resistiva, sino activa), es lo que hace que la capacitancia (parásita) que tiene que manejar, se cargue y se descargue muy rápidamente, lográndose con ello tiempos de conmutación muy pequeños, ya que la resistencia del transistor de arriba es muy pequeña.

Sin embargo, esta configuración presenta un inconveniente muy grave. En el tránsito de un estado a otro, hay un momento en que ambos transistores están conduciendo (región activa). En un TTL, si asumiéramos que durante este breve instante los dos transistores están en saturación (realmente andan por región activa, pero de esta manera simplifico el análisis), más o menos puede decirse que hay un pico de corriente aproximadamente igual a 5V / 130 Ohms (que es el valor de la carga del transistor de arriba, para TTL. He despreciado los voltajes de saturación y el del diodo). Esto es aproximadamente igual a 40 mA, POR CADA SALIDA del chip. Los IC están diseñados para que el calor generado por este pico no los dañe. Note que si hay más de una salida conmutándose dentro del mismo IC, el problema empeora.

Además, los circuitos complejos suelen ser sincrónicos, lo que significa que cambian todas las señales AL MISMO TIEMPO, bajo el comando de una señal de reloj. Entonces, si tenemos **10 IC**, c/u con 40 mA u 80 mA o más, podemos tener **picos** de más de medio amperio, y hasta de **más de uno**, en la fuente!!! Debido a la impedancia de salida de la alimentación, y a la del **cableado** de distribución de la alimentación, estos picos pueden hacer que la fuente tenga una disminución en su valor, de la misma duración y forma. Estos "**glitches**" en el suministro de energía pueden lograr, con mucha probabilidad, que circuitos realimentados, como contadores y registros, cambien de estado inopinadamente.

Es necesario resolver la situación, para lo cual debe conectarse, **entre el terminal de Vcc y el de tierra *de cada IC***, y lo más cerca posible al IC, **un condensador de desacople** ("decoupling capacitor") que absorba, en el sitio, el pico de corriente, y no se lo pase a la fuente de alimentación. El valor de este condensador suele ser entre 0.01 uF y 0.1 uF, de cerámica. **Un circuito que NO tenga este condensador, está mal diseñado, NO IMPORTA SI FUNCIONA...**

El mismo problema se presenta, y más agravado, en los CMOS, ya que éstos ni siquiera tienen resistencias de carga: Sólo los dos transistores P y N a la salida.

EQUIPO Y SALIDAS

Solemos contar con un generador de pulsos al que le cambiamos el período y el "dutty cycle" de la onda para ajustarlo a nuestro diseño.

Para ver los valores a veces se usan el multímetro y el osciloscopio, y también unas "puntas lógicas" que despliegan de manera luminosa el estado de encendido o apagado de cada línea. También suele haber unos pulsadores, que permiten inyectar señales inclusive en terminales que están conectados...

Las salidas suelen ser LEDs (diodos luminosos), que consumen como norma entre 10 y 16 mA, y el voltaje en sus terminales, cuando conducen casi siempre es de unos 1.2 V (estas características hay que verificarlas caso por caso con el manual de especificaciones; cambian dependiendo, por ejemplo, del **color** del LED!). Si se va a alimentar un LED desde una compuerta, éste tiene que tener una **resistencia limitadora de corriente, en serie**, que garantice que los parámetros antes mencionados se cumplan; de lo contrario se quemará el LED (y posiblemente la compuerta?)

Cuando el dispositivo de salida es un "7 segmentos" o similar, hay que escoger el circuito que lo maneje, de acuerdo a su característica, pues hay de dos tipos: Ánodo común y de cátodo común.

QUÉ COSAS VAN A FALLAR EN SUS PROYECTOS DE LABORATORIO!!!

Los conceptos de esta sección no necesariamente los va a comprender el primer día de clase. Guarde el documento hasta que la necesidad se presente más adelante. **Recomiendo** imprimir estas hojas, y tenerlas como un "check list".

1) FAN OUT

Si no lleva una contabilidad estricta de cuántas cargas ha colocado en las diferentes salidas de las compuertas y flip-flops (FF) que las manejan, el circuito:

simulará bien (en EWB, Simulink o PSpice)..., **pero no funcionará en el protoboard.**

Una señal que va por todas partes es, por ejemplo: El reloj; otra, el Reset...

Verifique cuál es el Fan Out de la familia de chips que está usando. Úselos TODOS de la misma familia. Por ejemplo, que todos sean 74LSxxx. Cuando mezcla familias, la regla normalizada no funciona, y usted debe hacer cálculos de voltajes y corrientes **individualmente**, para saber cómo conectar las salidas de una familia de Circuitos Integrados, con otras... (No es que no se pueda hacer; sólo que ahora no es tan fácil como contar del 1 al 10)

El Fan Out suele ser 10 normalizado, para TTL, y 20 para CMOS (VERIFIQUE LA VALIDEZ DE ESTA PREMISA CON LA FAMILIA DE CHIPS QUE VA A EMPLEAR); es decir, que cada salida puede manejar hasta un máximo de 10/20 terminales de entrada (de componentes de la misma familia). Si hay más cargas que 10/20, hay que ***replicar*** las señales. Para ésto debe tratar que las replicadas tengan

el mismo retraso; es decir, que todas pasen por igual número de negadores/buffers/drivers, etc. Esto TIENE QUE HACERSE ASÍ SIEMPRE, pero sobre todo si la señal ES LA DEL RELOJ. Todas las señales que lógicamente sean "el reloj", pero que no sean eléctricamente las mismas (relojes replicados), deben subir y bajar lo más sincronizadamente posible las unas a las otras; de lo contrario, usted no tendrá UN reloj, sino varios...; ¡ su diseño no será SINCRÓNICO! (¡y no funcionará !)

Debe cerciorase de que todas las entradas tengan FAN-IN de 1 (En algunos circuitos antiguos ciertas señales, como el Reloj, tenían un FAN-IN de 2, o más... Entonces, esas no cuentan como 1, sino que son 2... Nada se pierde con verificar)

2) MULTIPLICACIÓN (AND) POR EL PULSO DE RELOJ

Si hay por lo menos una compuerta que está multiplicando (and) el pulso de reloj por una combinación de salidas de Flip-Flops (FF), la forma en que todo funcione a la perfección es que el reloj ejecute las transiciones CON EL BORDE DE BAJADA. La lógica detrás de esto es la siguiente:

Sea: $Z = f(Q1...Qn) * Cp$

Mientras el reloj está en cero, la compuerta que multiplica el pulso de reloj por una combinación de salidas de FF (f(Q1...Qn) en la ecuación de Z), está entonces en cero. El reloj sube ahora a 1. Si f(Q1...Qn) vale 1, Z comienza a valer "Cp". Note que, ahora, NADA en "f(Q1...Qn)" se mueve, ya que TODO cambio en Q1...Qn se hará SOLO con el borde de bajada del reloj (esa es la premisa de diseño de redes sincrónicas).

Luego de permanecer un cierto tiempo en 1, el reloj baja a cero. INMEDIATAMENTE la señal Z vale CERO. Ahora, sí es cierto que "f(Q1...Qn)" se mueve, y pueden producirse picos, riesgos estáticos y dinámicos, multiplicaciones de variables afirmadas por variables negadas (spikes), etc.

Esto ¡ AHORA NO IMPORTA !, ya que Cp vale CERO y cierra el paso de todos esos inconvenientes, hacia Z.

Note que si el reloj está haciendo cambios *con el flanco de subida*, usted TENDRÁ PROBLEMAS CON LA ECUACIÓN Z, que se reflejarán en que Z no valdrá UN "Cp", sino que podrá tener picos. Usted seguramente necesitará muchas de estas señales, que irán a contadores y a otras unidades que tienen memoria. Si son contadores, éstos se adelantarán.

Todo iría muy bien si usara un reloj que opere con el borde de bajada. Solo hay un problema: La gran mayoría de los Flip-Flops y Contadores se activa con el borde de ¡¡ Subida !!.

La solución consiste en tomar la fuente de reloj, NEGARLA para llevarla a los Flip-Flops y Contadores que se activen con el borde de subida, y en los sitios en donde haya que hacer funciones del tipo $Z = f(Q1...Qn) * Cp$, continuar empleando la señal AFIRMADA. (cfr. referencias tales como el Wakerly)

También se deben ecualizar las trayectorias del reloj afirmado y negado, para evitar, en lo posible, retrasos. Y si tienen que mezclar circuitos que operan con el borde de subida, con los que funcionan con el de bajada, HAY QUE HACERLO CON EXTREMO CUIDADO.

3) ESTADO INICIAL

En el laboratorio solemos hacerla la inicialización a mano, empleando un botón de MASTER RESET. Si su circuito de inicialización es simple (sólo el botón y sus drivers), PUEDE existir el problema de que la señal de reset se desactive (vaya a 1) muy cerca del momento en que el reloj está cambiando también. Esto PUEDE (o no) producir una carrera, y dar al traste con el funcionamiento del circuito (Lo ideal sería que NO hubiera pulsos de reloj hasta un rato después de terminado el lapso de inicialización)

> La solución se logra haciendo que el estado inicial esté representado por **un sólo flip-flop**, que se pone en 1 cuando todos los demás FF (y contadores, etc.) se ponen en cero. Luego, con el primer pulso de reloj, ***y sin que medie ninguna otra señal***, se pasa al estado siguiente, que es *REALMENTE* el primero de su circuito. De esta manera, si hay algún problema como el mencionado, lo más que puede ocurrir es que haya un retraso de un pulso de reloj **antes** de comenzar todo a funcionar como estaba previsto... *(cfr.* ***Hágase la Luz*** *en este mismo capítulo)*

4) SEÑALES NO SINCRONIZADAS

En nuestras redes secuenciales, TODAS las señales tienen que estar SINCRONIZADAS. Las entradas manuales o provenientes de otros subsistemas, están desincronizadas. Hay que sincronizarlas (además de eliminarles el REBOTE).

Una señal debe sincronizarse EN UN SOLO PUNTO (un sólo FF), cuya salida ES la señal que se usa. Por ningún motivo puede sincronizarse una señal EN VARIAS PARTES.

5) SEÑALES MÁS RÁPIDAS QUE EL PULSO DE RELOJ

Suponga que diseñó un control de semáforos, con un reloj de 30 segundos. Si su circuito es sincrónico, cuando una persona llegue al botón para activar el paso de peatones y lo oprima, tiene que activarse el FF de sincronización. El problema es que el pulso de reloj puede demorar un promedio de 15 segundos antes de activarse (30 segundos máximo) ¡ El peatón no tiene por qué estar pendiente de este detalle ! Entonces, es necesario almacenar la acción de ese botón hasta que llegue el pulso de reloj. Se necesita un FF SIN reloj (el primer S-R [*Eccles-Jordan*] que se estudia en clase, con 2 compuertas Nand realimentadas). En una entrada estará el botón del peatón, y por la otra su circuito deberá devolver ese FF a cero, cuando haya procesado esa entrada.

6) FALTA DE SOPORTE PARA "DEBUGGING"

Si usted no puede verificar la operación de su circuito, Pulso a Pulso o a baja velocidad, Instrucción por Instrucción, paso a paso... ***¡ está bien lejos de ponerlo a funcionar !*** Al concebir su diseño tiene que pensar en poder ver por qué no funciona. (***"Si no sabe cómo lo va a probar, ni siquiera lo arme"***; recomendación de Uribe)

7) SWITCHES CON REBOTE

Este problema tiene una solución clara y específica, que se aprende en la teoría, tanto en redes combinatorias como si emplea microcontroladores. ¡ NO INVENTE !

8) LED. CARGAS NO TTL

Los LED en circuitos TTL tienen que estar manejados por un buffer. Si los coloca a tierra DIRECTAMENTE a la salida de un FF TTL, **¡ éste no funcionará !**. Si los coloca a tierra a la salida de compuertas TTL que luego va a emplear en otras secciones, **no funcionarán**. Un LED a tierra hace que la salida TTL del circuito al cual se conectó, YA NO SEA UNA SEÑAL TTL APROPIADA PARA FUNCIONAR en el resto del circuito digital (los valores de voltaje y la capacidad de manejo de corriente de la compuerta, seguramente estarán muy afectados). ***Si son CMOS, puede ser otra cosa...***

9) ALIMENTACIÓN DEL PROTOBOARD

Algunos vienen con su propia fuente (5V @ 700 mA). Deben notar que, al poblarlos con circuitos TTL de la serie LS, los Protoboards probablemente se están sobrepasando de los 700 mA... (uno no imaginaría que componentes tan pequeños consumieran TANTO...); esto hace que la fuente salga de su punto de trabajo, con lo que se consigue:

- Que el voltaje baje de 5V, violando posiblemente el nivel de trabajo satisfactorio garantizado para los chips

- Que la fuente salga de regulación, por lo cual el voltaje DC comienza a tener "ripple", que es una ondulación (AC) superpuesta a la menguada señal DC.

El resultado neto es que el circuito no funciona, lo hace intermitentemente, u opera hasta que lo vaya a presentar... EFECTO ENTREGA --corolario de Uribe sobre la ley de Murphy--:

"Todo proyecto de Digitales funcionará hasta el momento en que lo vaya a presentar; ahí dejará de hacerlo.
Los profesores que lo vieron trabajar la tarde anterior no podrán asistir a la entrega definitiva,
a atestiguar fehacientemente que no es sólo bla-bla lo que se va a decir..."

10) SOLDADURA!!!

Es cierto que puede estar prohibido realizar soldaduras en el recinto de los laboratorios, pues algunos han quemado con un cautín los valiosos y flamantes instrumentos nuevos... Pero eso NO significa que los interruptores no tengan qué soldarse a los cables que van a servir para conectarlos al circuito. Los conductores pelados, entorchados en los terminales de los interruptores, **no** hacen buen contacto (tanto que se preocupan del rebote, para luego venir a dejar flojos los terminales...) HAY QUE SOLDARLOS. Pueden hacerlo en sus casas, en el corredor, PERO HÁGANLO...

11) CONDENSADORES DE DESACOPLE

Ya lo dijimos con anterioridad: Durante las transiciones hay un pico de corriente de aproximadamente 40 mA, POR CADA SALIDA del chip. Si hay varias señales conmutándose dentro del mismo IC (que es lo más común...) el problema empeora. Y como los circuitos complejos suelen ser sincrónicos, todas las señales cambian AL MISMO TIEMPO. **¡ PEOR !** Si tenemos 20 IC, cada uno con 80 mA o más, podemos tener picos de más de UN (1) amperio en la fuente de alimentación!!!

Debido a la impedancia de salida de la fuente (empeorada por la resistividad del Protoboard!!! y la del cableado de distribución de la alimentación, y si usan cablecitos MUY DELGADOS para conectar el

Protoboard a la fuente), estos picos se suman y pueden hacer que la alimentación tenga una disminución en su valor. Estos "glitches" en la alimentación pueden hacer que circuitos realimentados (contadores y registros), cambien de estado inopinadamente.

Es necesario resolver esta situación, para lo cual **DEBE CONECTARSE**, entre el terminal de Vcc y el de tierra ***de cada*** IC, y lo más cercanamente posible al IC (para evitar introducir INDUCTANCIAS parásitas, que empeoran el asunto), un condensador de desacople (**decoupling capacitor**) que absorba el pico de corriente, y no se lo pase a la fuente de alimentación. El valor de este condensador suele ser entre 0.01 uF y 0.1 uF. Son condensadores cerámicos.

¡UN CIRCUITO SIN ESTOS CONDENSADORES, ESTÁ **MAL** DISEÑADO,
AÚN SI FUNCIONA!

A veces (en trabajos que no van a producción), nos las arreglamos con UN condensador de desacople por cada 10 Circuitos Integrados... PERO HAY QUE COLOCARLOS. NO LO OLVIDE.

NOTA: En alguna oportunidad alguien conectó los condensadores de desacople, ***en serie*** *entre la fuente de alimentación y el Protoboard! Eso no es lo que esta nota señala... y el circuito de ninguna manera funcionará así, ¡ como es apenas lógico esperar ! De hecho, el alumno, a quien no le funcionaba el circuito, me llamó y me dijo que ya lo había arreglado, al retirarle todos los condensadores de desacople...*

12) DIAGRAMAS

La mayoría tiende a usar como documentación principal de diseño, el diagrama circuital extraído de EWB, Simulink o PSpice. Esto es ABSOLUTAMENTE INCONVENIENTE. Cuando se está en la etapa de diseño, lo importante es establecer la funcionalidad de los diversos subsistemas (contadores, multiplexores, sumadores...) y las relaciones que existen entre ellos (el sumador se conecta a la salida de XXX; el contador #1 se alimenta de la entrada YYY en el momento o estado ZZZ, etc.)

Es FUNDAMENTAL generar primero un ① DIAGRAMA EN BLOQUES, en el que se destaquen dichos subsistemas y sus interrelaciones (***cfr. Wakerly***, distintos tipos de diagramas y el momento apropiado de su APLICACIÓN).

Sobre ese Diagrama en Bloques es que se diseña el Algoritmo, que es la forma en que su circuito operará. Usted. puede describir su algoritmo de manera verbal, o empleando un lenguaje de programación cualquiera, o alguno específico (ABEL, VHDL, Verilog) o, inclusive, un Diagrama de Estados. Lo importante es que estén conscientes de que su sistema NECESITA un algoritmo, que es el que va a producir como resultado que su circuito funcione.

Después de lo anterior, hay que hacer un ② DIAGRAMA LÓGICO, que muestre con un mayor nivel de detalle los componentes (contadores, registros, sumadores) y sus conexiones, mediante compuertas AND/OR, o NAND.

Ambos diagramas, el de Bloques y el Lógico, sirven para explicar al profesor –y a usted mismo– el funcionamiento de su circuito. Nótese que aquí **no** aparece por ningún lado cuáles son los pines de tal o cual chip, ni cómo se alimenta el Vcc, ni cuáles pines están conectados o desconectados. Ese nivel de

pedido que le ayudara a diseñar, por ejemplo, un juego de Ping-Pong. Seguramente que, con su ingenio, ¿algo habría resuelto... ? Esta cualidad es fundamental en los ingenieros, ¡ PERO NO ES SUFICIENTE ! Por eso se va a la universidad.

Algo parecido ocurriría si usted encontrara una persona accidentada en la carretera y la ayudara, devolviéndole las vísceras al estómago y amordazándolo estrechamente con sábanas mientras lo llevan al hospital. Muy comedido de su parte, muchas gracias; pero no podrá decirse por eso que usted es un médico o un doctor.

Un algoritmo es un procedimiento. Su circuito va a seguir –si es que ha de funcionar– un procedimiento. La descripción del algoritmo puede ser verbal, o realizarse mediante un Diagrama de Flujo, como hacen los programadores, o puede ser descrito mediante un Diagrama de Estados (todos los diagramas resultan más o menos equivalentes; usted selecciona el que mejor le calza). O puede ser presentado empleando algún lenguaje de programación, que se escoge convenientemente. (Si un programa necesita o no de un diagrama de flujo previo, puede ser discutible).

Tener el algoritmo expresado mediante un lenguaje de programación le permitiría, de paso, simularlo, y así poder garantizar que su algoritmo realmente funciona. (Además, podría después emplear técnicas automatizadas que generarían el circuito final, a partir de su programa: VHDL, ABEL, Verilog, etc. ***cfr.*** el código de varios problemas resueltos aquí, como el del Juego de Ping-Pong)

Recuerde que a usted no se le va a calificar sólo por el resultado, SINO TAMBIÉN POR LA METODOLOGÍA. Usted tiene que demostrar que sabe. El resultado es importante, pero es la metodología la que garantiza el resultado (ayudada con un poco de ingenio...)

!CUIDADO CON LA IMPROVISACIÓN!

Otro tema en el que inventamos es en el de la programación. Resulta que usted ha tomado sus cursos de electrónica y digitales; sabe elaborar su diseño; emplea condensadores de desacople; lleva el control del fan-out..., pero al final, hay que hacer un programa para el micro. Y entonces usted se enfrenta a conceptos tales como colas, procesos, semáforos, interrupciones, drivers, expropiación y apropiación de recursos, todo lo cual se estudia en Sistemas o en una licenciatura de Computación. Usted, sin embargo, no tiene cursos apropiados y decide "improvisar".

Tenga mucho cuidado. Esa no es una actitud profesional.

Su diseño puede padecer defectos ocultos, que para un profesional serían evidentes, pero no así a los ojos del improvisado. Y usted tiene una ***responsabilidad social***. Su proyecto puede ocasionar toda clase de inconvenientes, algunos de los cuales pueden resultar, inclusive, mortales (piense en una falla en el control de un tomógrafo. ***cfr.*** capítulo 1, en Tomógrafos Computarizados, y la falla del Ariane 5).

14) MAL USO DE LOS INSTRUMENTOS

Un grupo quemó su microcomputador haciendo lo siguiente: Empleó el osciloscopio para ajustar la fuente de 5 V. DC, pero olvidó revisar si la punta estaba atenuada por 10 o no. Sí, tenía activo el divisor;

y cuando midieron 5 V, la fuente estaba entregando **50** V. Además, en un arranque inverosímil, le pidieron prestado el micro a sus compañeros (no sé qué pensaban verificar) y se los dañaron también!

Otros frieron su microcomputador siguiendo el mismo procedimiento, pero ahora la punta NO estaba atenuada... Sólo que las ***perillas de calibración*** estaban al otro extremo, fuera de su posición convencional. Y cuando midieron 5 V la fuente estaba entregando 12V. Si un cirujano blandiera su bisturí con tan desacertada maestría, lo demandarían e iría preso. Ustedes también son profesionales...

15) LA JUSTIFICACIÓN

En algún proyecto los alumnos se decidieron a usar 16 LEDs. Cuando pregunté por qué no 4, 5, 7 o 12, me dijeron que el demultiplexor que consiguieron era de 4 a 16... ¡ Esa no es realmente ninguna razón ! Sería como si a usted le hubieran recetado 4 pastillas para curarse de alguna infección, y se hubiera tomado 20, porque 20 eran las que traía el frasco que el farmaceuta le dio.

Usted se está dejando atrapar por el entorno. Note que, pensándolo un poco, hubiera podido reducir los 16 LEDs a 8, sólo con no usar todas las salidas DCBA del contador (¡ que también lo atrapó !), sino solamente CBA. O podría haber hecho que el contador no llegara a 16 sino hasta el número que creyera conveniente para su diseño. En todo caso, esto demuestra que les suele faltar el primer paso del diseño: LA ESPECIFICACIÓN TÉCNICA.

Éste es el documento mediante el cual, el cliente solicita la funcionalidad que desea en el proyecto que encomendó. Si el cliente quiere 7 LEDs, una de dos: O el ingeniero lo convence de que son mejores 8 o 4, o **le tiene que entregar 7 LEDs**. No le puede salir con 16 dizque porque su decodificador era de 4 a 16!!! Usted le va a hacer comprar a su cliente (¡ su papá !) LEDs que no necesita, más todo el soporte asociado: Resistencias de 330 Ohms, posiblemente drivers, espacio en el protoboard (*"Real Estate"*; necesitará uno más grande que si hubiera usado 7 LEDs), y habrá perdido confiabilidad (*"Reliability"*), que es inversamente proporcional a la cantidad de componentes empleados.

En resumen: Sus proyectos tienen que incluir:

- La Especificación Técnica de su diseño, que debe con precisión contener, como mínimo:
- Qué va a tener su circuito (cuantos LEDs, interruptores y demás elementos de interacción con los usuarios).
- Cómo va a funcionar: Cuándo gana un jugador; cómo se representa el evento ganancioso. Si su proyecto es, por ejemplo, un frecuencímetro: Cómo mide frecuencia, período, etc.; cuántos dígitos se van a representar; cómo va a eliminar el 10% de error ocasionado por el +- 1 conteo, que produce un error de 1 en 10)
- El Diagrama en Bloques, tal como mencionamos arriba, acompañado de la descripción exacta del algoritmo. Yo prefiero que me lo den en algún lenguaje de programación (BASIC, C, JAVA), y que me muestran la simulación.
- El Diagrama Lógico, tal como mencionamos arriba.

- El Diagrama Circuital.
- La simulación en el EWB, el Simulink o el PSpice
- El protoboard con el circuito, según el Diagrama Circuital (o el impreso, si corresponde).
- Es fundamental que anote EL TIEMPO empleado *en cada parte* del diseño, para que lo reporte en el informe final, en el cual deberá indicar los costos de análisis, diseño, fabricación, pruebas, componentes, etc. Lleve siempre su *bitácora*!
- Si es su primer diseño digital, es importante que se vaya acostumbrando a pensar que hay un mundo real, más allá de lo académico, en el cual no sólo es importante que funcionen las cosas, sino que deben ser pautadas mediante cronogramas de trabajo, especificaciones, etapas de diseño, presupuestos y costos, e informes.

Finalmente, no dejen que los exámenes le resten impulso a su actividad.

La elaboración de un proyecto **no** *es una etapa de descanso que se da en el laboratorio para que ustedes tengan facilidad de presentar sus exámenes de teoría.*

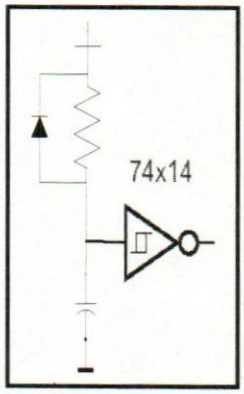
Hay que cumplir con todo. "*C'est la vie*".

HÁGASE LA LUZ

EL diseñador digital novel suele entender bastante bien la operación de sus circuitos "en estado estacionario", es decir, una vez que llevan algún tiempo conectados a la alimentación eléctrica (5 V DC típica). Los autores de libros de texto tienen su temática depurada y hasta uniforme, y casi cualquier tomo moderno cubre ciertos recovecos teóricos como la pulcra sincronización de las señales de entrada –indispensable premisa para el diseño de redes secuenciales sincrónicas–; la apropiada captura de información externa con duración menor a la de uno de los sempiternos pulsos del reloj maestro; la eliminación de rebotes; los condensadores de desacople, la incorporación de resistencias limitadoras de corriente en las entradas no usadas y demás aspectos técnicos relacionados.

Pero hay una zona oscura que casi nunca se toca, en la que el ingeniero digital navega a tientas, a pesar de ser esencial para el funcionamiento de sus proyectos: La adecuada materialización automática del debido "***estado inicial***", al momento de encender su equipo.

1) RECONECTADOR



A este respecto hay que indicar que el voltaje de alimentación debe aplicarse de tal manera que se satisfagan los requisitos de estabilidad en el oscilador maestro, del cual se deriva el mecanismo de reloj; es común que haya que esperar un tiempo considerable, comparado con los valores de trabajo del equipo, a fin de que la oscilación obtenga el voltaje y la frecuencia deseados, máxime si en su diseño se incluyen técnicas de PLL (Phase Lock Loop), comunes en los microcontroladores (por ejemplo, Freescale). No resulta conveniente usar la señal de reloj hasta que no haya alcanzado su valor estacionario de diseño. Lo usual es inhabilitar sobre todo los flip-flops, lo que se logra generando una señal de "***Clear***" que dura entre 100 y 500 ms, con un circuito RC simple y una compuerta del tipo Schmitt Trigger (74xx14 o similar). Este dispositivo se conoce como el "***Reconectador***".

Al aplicarle la alimentación al circuito de la figura, el condensador tarda en cargarse un tiempo determinado por su constante RC, que debe especificarse según sea necesario para lograr la estabilidad de los osciladores. La salida del Schmitt Trigger se lleva a todos los puntos de "Estado Inicial", probablemente las señales de "Clear" de los flip-flops. Si éstas se activan en un nivel bajo, lo que es típico, hace falta añadirle un negador al circuito de la figura, además de las consideraciones referentes al fan-out o carga de esta señal, pues se puede exceder con facilidad la capacidad normal de las compuertas, implicando la adición de buffers/drivers. El diodo inverso, en paralelo con la resistencia, sirve para descargar con rapidez el condensador y reinicializar todo el proceso de nuevo, en caso de que la fuente baje a cero, por un apagón, o un "glitch".

Si se necesitara un nivel mayor de control sobre el voltaje, porque pudieran aparecer en la fuente de alimentación niveles intermedios, menores de 4.5 V –lo que es previsible en dispositivos con pilas– que ameritaran desconectar el circuito, podría incluirse un "***Supervisor de Voltaje***", al estilo del TL7705.

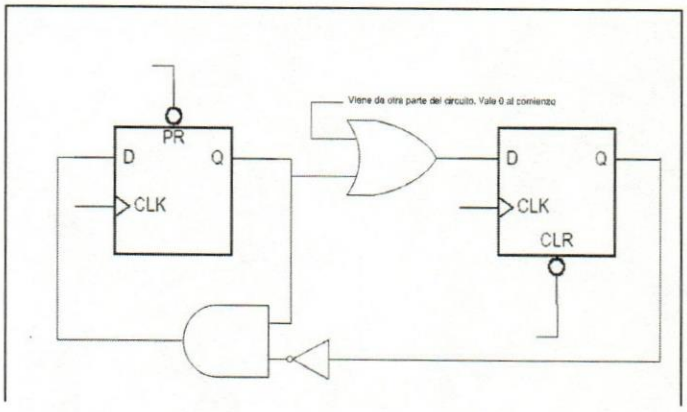
2) COMPORTAMIENTO ASINCRÓNICO

Ahora bien; ni el supervisor ni el más simple reconectador de la figura resuelven a priori un ***problema de sincronización, sutil pero vital***, que consiste en que si no se toman mayores medidas, el reloj comienza a aparecer en todas partes –no influyendo en el comportamiento del circuito, gracias a que la señal de "Clear" la mantiene activa el reconectador– pero cuando el Schmitt Trigger cambia de estado, ***lo hace de manera asincrónica respecto al reloj***. Terrible inconveniente que viola la norma que exige garantía de que todas las señales, incluyendo el "Clear", tengan un valor estable cierto tiempo **antes** de la llegada del reloj. Si no se sigue ese precepto el resultado es que algunos flip-flops pueden comenzar a "***ver***" el pulso de reloj, y otros no... y así, el comportamiento del circuito se vuelve indeterminado.

Cuándo y cuánto afecta este problema a un dispositivo depende de la probabilidad de que el fenómeno aleatorio produzca una ventana de tiempo inapropiada. Puede ser que el circuito funcione bien una y otra vez, pero si la probabilidad de que falle no es nula *el diseño tiene un error, que es* ***imperativo corregir***.

3) "ARRANCADOR" (STARTER)

El siguiente circuito garantiza el estado inicial previsto por el diseñador, a pesar de la incertidumbre mencionada en los párrafos anteriores. Por facilidad, el ejemplo hace referencia a aquellas máquinas que se programan con ***un flip-flop por estado***, pero el concepto puede ampliarse a diseños con contadores comerciales, o *ad-hoc*, que necesiten inicialización sincrónica o a asincrónica.



Se muestran sólo dos FFs, representando cada uno un estado. El FF de la izquierda es el único de todos que tiene activación inicial vía ***Preset***, los demás lo hacen mediante ***Clear***, así que el primer flip-flop asume el valor 1 y los demás, 0: **10**.

Tal como se indicó, la señal de Estado Inicial (Power-up) se encuentra conectada a los Preset y Clear apropiados y se activa un largo tiempo (100 a 500 ms), durante el cual el reloj comienza a entrar de manera estable a todos los FFs, sin efecto alguno pues la señal de Estado Inicial está activa. Ahora, cuando se desactivan los Clear y Presets, liberando los FFs para que comience la secuencia de estados, al compás del Reloj, se ve que el estado futuro del *primer* FF es **1**, producido por la compuerta **and**, su propia **Q** en 1, y la Q' del segundo flip-flop, que aún está en **0** (Q en 0; Q' en 1). Si el primer pulso en llegar no fuera visto por el FF de la *derecha*, la situación se mantendría exactamente así: **10**, hasta la llegada del próximo pulso. Nótese que el FF de la izquierda mantiene su estado en 1, vea –o no– el pulso de reloj.

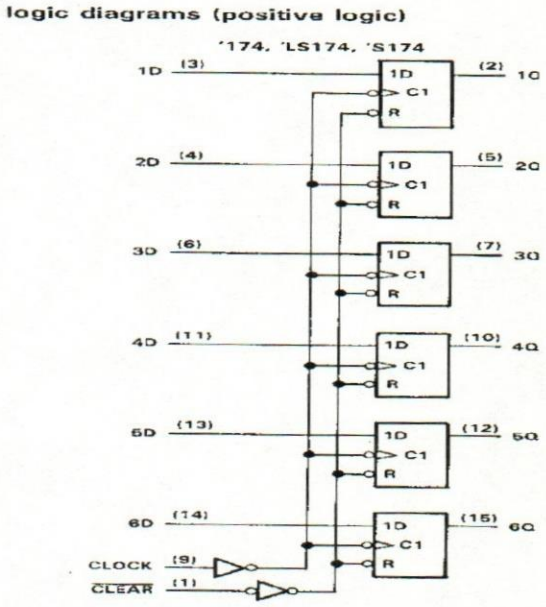
Cuando finalmente el segundo flip-flop cambia de estado (**11**), ahora sí es verdad que todos están en sincronismo (*las señales de la red cambian sólo con el borde de subida del Reloj*); el estado es: Ambos flip-flops en **1** (**11**) y, luego, el hecho que el flip-flop de la derecha asuma el valor de **1** hará que el FF de la izquierda vaya finalmente a **0** (**01**), de donde no saldrá jamás, salvo que se reanude la secuencia de energización al volver a encender el circuito, o mediante una señal externa de Reset (no representada en la figura).

Siendo una red con un FF por estado, el hecho de que ambos flip-flops de la figura se encuentren alguna vez en 1 imposibilita la utilización de la Q del flip-flop izquierdo para algún otro propósito en el circuito; es decir, ese biestable sirve únicamente para arrancar la secuencia (***starter***).

4) CONVERTIR FLIP-FLOPS CON "CLEAR", EN BIESTABLES CON "PRESET"

Este es un problema práctico que los principiantes no saben resolver. Espero que mi solución resulte provocativa, y les llame la atención el ver que basta con ***cambiar el nombre*** de las entradas y las salidas para resolverlo. Quién iba a creerlo...

Hay alumnos que cuando necesitan un flip-flop con Preset y los que están a la mano solo tienen Clear, salen de compras. Intento mostrar cómo hacer que flip-flops como los 74xx73, 74xx107 o similares, para flip-flops **JK**; y los 74xx**174** (*ver figura*), 74xx175, etc., para tipo **D**, ***que sólo tienen Clear***, se comporten como si **lo que tuvieran fuera Preset**. Analice a conciencia el caso particular del **74174**, hex D-type flip-flops with clear. Son seis biestables tipo D, con una sola entrada de Clear común a los 6, y de cada uno sale exclusivamente su Q; la Q' (negada) no tiene un terminal asignado en el 174.

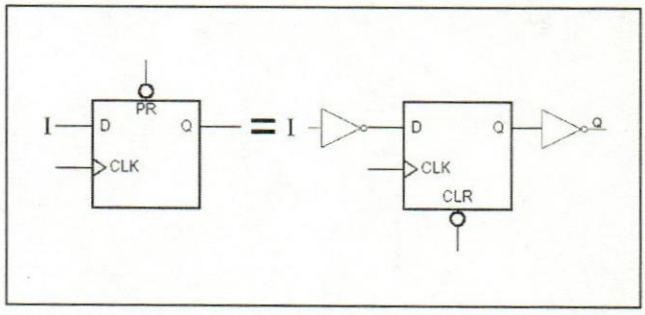


En los flip-flops tipo **JK** la conversión es fácil, porque son absolutamente simétricos: La trayectoria de la **J** va por las compuertas superiores y llega a la Q, y la K entra por las inferiores y progresa hasta la Q'. El **Clear** va también por la línea inferior, la correspondiente a la **K**. Ahora, si ***renombramos*** los terminales (**sí**; las denominaciones son una convención, ¿o no?), y llamamos **J** a la anterior **K**, **K** a la vieja **J**; lo que conocíamos como **Q** lo nombramos Q' y viceversa, la Q' es la Q..., todo seguirá siendo igual para la nueva denominación de los terminales, sólo que ahora ¡ la señal de **Clear** se ha convertido en **Preset** !

El flip-flop tipo **T** también es absolutamente simétrico, puesto que no es otra cosa que un JK con las entradas unidas, por lo que el método se aplica igual.

Como los flip-flops que más se usan, sobre todo a nivel de PLD (FPGA, PAL, PLA, etc.) son los tipo D (con Clear), resulta de la mayor importancia descubrir cómo convertirlos para que el Clear opere como Preset...

Si recordamos que para hacer un tipo **D** con JK, la D se conecta a la J, y a la K va la D', podemos deducir que, ***para convertir un flip-flop tipo D, con Clear, en uno con Preset, bastará con incluir un negador en la entrada D, y otro a la salida Q***, como muestra la figura de la derecha:



EJERCICIOS

MOTIVACIÓN

Los estudiantes suelen quejarse de que la literatura que introduce el tema de Circuitos o Electrónica Digital ofrece problemas muy simples, divorciados de las aplicaciones reales, y omiten establecer con claridad una metodología de solución de problemas de cierta magnitud. Por eso se incluye el planteamiento de los siguientes problemas.

Lo primero que hay que hacer en cada ejercicio de Circuitos Digitales es un Programa en lenguaje **C**, en el que las entradas se simulan leyendo el teclado (getchar), y las salidas mediante impresiones en la pantalla del PC (putchar; printf). Esto se debe, en primer lugar, a que los alumnos han de saber programar como requisito de estas asignaturas; en segundo término, porque un programa es la manera idónea de formular una solución (algoritmo), y no hay mejor simulación que hacerlo correr. En tercer lugar, si se codifica con especial atención puede ejecutarse directamente en un microcontrolador, que son los dispositivos que se utilizarán el **99.99100%** de las veces en la vida real para resolver los problemas, pues esta es la era de los dispositivos electrónicos programables. Y, por último, si no pudiera, por algún motivo, implantarse el programa directamente desde C, siempre es un paso ***importantísimo*** para su codificación en el Assembler propio de cada microcontrolador.

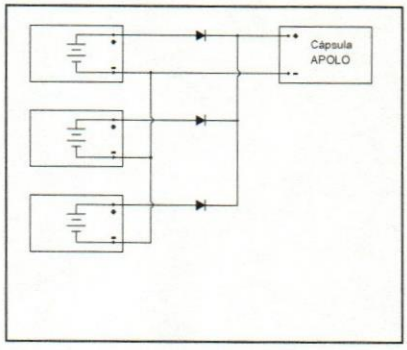
Otros lenguajes aplican también, en especial Java, C++, Basic, Visualbasic (***cfr***. ejercicio de Ping-Pong más adelante)

Nota: *Observe con detenimiento los ejercicios combinatorios 3, 4, 5 y 6:* ***Son el mismo problema con diferente fraseo***.

REDES COMBINATORIAS

1) DISPOSITIVO DE ALARMA PARA LAS CÁPSULAS APOLO

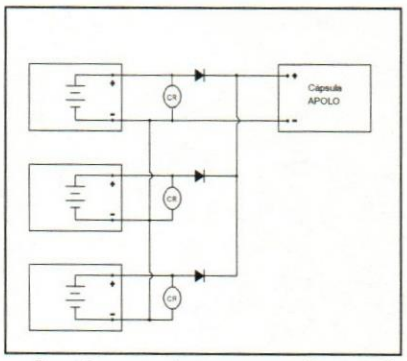
A) PRIMERA PROPOSICIÓN



El sistema de alimentación eléctrica de las naves Apolo es tan crítico, que recibe suministro redundante de energía DC, de tres (3) fuentes de alimentación independientes; cada una se conecta a través de un diodo para evitar que un eventual daño (por ejemplo, un corto circuito a la salida de una fuente) afecte a las demás.

El sistema cumple muy bien su trabajo suministrando fluido eléctrico confiable en forma continua, pero tiene el inconveniente de que, precisamente porque todo opera con normalidad, incluso cuando fallan uno o dos dispositivos externos, los astronautas podrían no enterarse de cuándo están en situación de riesgo. Por eso hay que

diseñar un sistema ***visual y audible*** que les indique el estado de funcionamiento de las tres baterías solares, de la siguiente forma: Se instalarán a la salida de cada batería (antes del diodo) unos sensores en paralelo, que pueden ser, por ejemplo, ***relés*** o dispositivos electrónico apropiados (CR en la siguiente figura), que suministren un "1" lógico cuando se encuentren en operación normal y "0" si el voltaje ha disminuido por debajo de un cierto límite preestablecido y prudencial.



Habrá una señal verde (**V**) que, de estar encendida, indicará que las tres baterías se encuentran operando correctamente; una luz amarilla (**Am**) se alumbrará cuando una y sólo una de las baterías haya salido de funcionamiento (el amarillo indica precaución: Ha habido un defecto que obligará a los astronautas a desconectar aquella parte del equipo no indispensable, pero el viaje puede continuar); por último, estará el indicativo rojo (**R**) que encenderá cuando dos baterías cualesquiera, o las tres (?), se hayan dañado (el roja es señal de peligro: Es menester economizar energía al máximo y utilizar los acumuladores auxiliares de abordo para la maniobra de regreso inmediato). Las situaciones de alarma amarilla y roja activarán además una sirena, para llamar la atención de los tripulantes, que tienen que mirar el tablero para ver qué ha ocurrido. El indicador verde no va acompañado de sonidos. Diseñar el sistema **lógico** (compuertas, contactos, relés, switches; no incluya **por ahora** otros circuitos electrónicos o de potencia necesarios, y que sí van en la vida real...)

B) MANTENIMIENTO

Este ejercicio es, además, ejemplo de cómo van evolucionando los requerimientos de los clientes..., que nunca piensan en todos los escenarios la primera vez, y obligan luego al departamento de Ingeniería a hacer rediseños (***parches***) en forma permanente... Pero el cliente y los empleados son el capital más importante que una empresa puede tener, y el más volátil también.

El circuito que usted diseñó en el paso anterior ¡ ***funcionó a la perfección*** ! pero cuando los astronautas tenían que retirar una fuente de alimentación solar para hacerle Mantenimiento Preventivo (PM) de rutina (limpieza, verificación de su estado de operatividad, medición de los voltajes generados y demás parámetros de diseño) se producía una alarma de Alerta Amarilla enloquecedora. Se decidió que cuando fuera a trabajarse en el equipo se desactivaría la corneta por completo, agregando una llave (**K**) que el astronauta debía insertar y girar para indicarle al sistema que se estaba por comenzar el mantenimiento. Modifique su diseño para incluir la nueva entrada y altere en concordancia el comportamiento del circuito.

C) SOLUCIÓN PEOR QUE LA ENFERMEDAD

Habiéndose implementado la propuesta del punto anterior ocurrió que uno de los astronautas ***olvidó retirar la llave*** cuando finalizó el trabajo, con tan mala suerte que al día siguiente se dañó una batería, y el sistema de alarmas no advirtió a la tripulación sobre el acontecimiento... ¡ DURANTE SEMANAS ! El cliente quiere, *de nuevo*, que usted modifique su diseño a fin de que si se daña una batería mientras está introducida la llave **K**, opere la Alarma Roja y por esa vía, la corneta. Decida usted qué hacer en otros casos no contemplados; por ejemplo, si se dañan dos (2) baterías cuando está insertada la llave **K** (¿Ulular? ¿Flashing o parpadeo?) ¿Puede eliminar el efecto de la llave después de un tiempo prudencial?

D) VERIFICABILIDAD DE SU CIRCUITO

Como el sistema de alimentación DC es vital, la paranoia cunde entre los astronautas y personal de tierra en relación a poder garantizar que si las alarmas no se activan, en realidad las baterías están bien, ¡ no porque se haya dañado alguna sección del circuito detector ! Por eso le piden una vez más que rediseñe el dispositivo para añadirle uno o dos botones externos que permitan ejercitar los leds y la sirena momentáneamente (y así asegurarse de que encienden y apagan bien los unos, y suena como corresponde la otra). Además, debe incluirse un mecanismo que permita simular, a mano, el comportamiento de las baterías y certificar así que el circuito opera satisfactoriamente. También le piden avanzar el diseño para que si, por ejemplo, la señal Verde no está encendida (indicando que una o más fuentes de alimentación se encuentran fuera de servicio), y tampoco se enciende la amarilla, se active inmediatamente la roja, con independencia de ninguna otra circunstancia.

NOTA: Es ideal que durante la etapa de análisis del problema, que antecede al diseño, se establezcan con la mayor exactitud todos los posibles escenarios, a fin de producir el diseño definitivo con el menor número de iteraciones. Sin embargo, para hacer su trabajo, ***¡ es preferible descomponerlo todo otra vez en partes !*** Imagine cómo podría diseñar este sistema considerando **todas** las entradas y salidas a la vez... (Baterías B1, B2, B3, K, Lamp-On, Lamp-Off, Sirena-On, Dummy para batería D1, D2, D3, Botón de TEST...): ¡ A mano, sería prácticamente IMPOSIBLE !

2) LA SOCIEDAD Y LOS ACCIONISTAS

A) PLANTEAMENTO INICIAL

Los 4 dueños de una sociedad poseen 20, 22, 27 y 31 acciones. Las decisiones se toman por mayoría. Diseñe un circuito que determine si se aprueba o se deniega un acuerdo; incluya una luz que represente "aprobado" y otra para "denegado". Defina las entradas y especifique cómo debe operarlas el votante. Suministre:

a) Tabla de Verdad
b) Solución con ROM
c) Implementación con "multiplexores"
d) Minimización con MK
e) Implementación de **d)** con compuertas AND, OR, NOT
f) Implementación de **d)** con compuertas NAND
g) Implementación con "decodificadores"

B) SEGURIDAD

Comente si usted cree que hasta ahora hace falta o no, agregar una capacidad de "lamp test" para verificar si el sistema de despliegue de información trabaja bien (por qué sí, o por qué no).

C) CONFORT

Dependiendo de cómo van votando los accionistas, las luces pueden pasar de "denegado" a "aprobado" y viceversa. Estos resultados espurios exasperan bastante a los electores; para impedir su

aparición, añada una señal más: "Output Enable", que le permita al Presidente habilitar la salida sólo cuando la totalidad de los accionistas manifiesten que ya votaron; añada **una** (1) tercera luz que muestre esa condición y se encienda cuando "todos votaron". (¿Hará falta ahora la funcionalidad del "lamp test"? Diga por qué sí o por qué no)

D) ALTERNATIVAS

Diseñe (Tabla de Verdad, MK, compuertas **nand**) las siguientes ***dos*** variaciones, que eliminan la entrada extra del Presidente (punto **C**), diferenciando si un accionista "no votó" o "votó **0**" (que no es lo mismo...) Active ***en forma automática*** las salidas: "aprobado", "denegado" y "todos votaron", sólo cuando el total de los accionistas lo haya hecho. Variaciones: **1**) cada elector cuenta con **un** botón que representa sí y no, y **otro** para votó, o no. **2**) cada socio tiene **un** interruptor indicativo de **sí,** y con **otro** señala **no**; si activa ambos debe hacérsele una advertencia local (el ***parpadeo*** de una luz al lado de sus botones), y declarar que no ha votado. (Comente sobre incluir ahora, o no, la funcionalidad del "lamp test" ¿Por qué sí o por qué no?)

E) CAMBIO DE ACCIONES

La composición accionaria varía a 4 grupos de 24, 26, 24 y 26 acciones respectivamente. Haga un rediseño del punto **D** para desplegar el hecho de que ahora puede haber **empates**; incluya una nueva luz que lo indique, y evite activar las salidas de "aprobado" y "denegado" en este caso. Agregue una luz de "aún no han votado todos" y diga ¿qué opina ahora sobre incluir o no, el "lamp test"?

3) CONTROL AUTOMÁTICO DE VELOCIDAD

Diseñar el sistema **CAV** para un automóvil. El conductor tiene acceso, en el volante, a un interruptor que manipula con el dedo pulgar; cuando quiere que su vehículo se desplace controlado automáticamente, acelera el carro en forma convencional y al llegar a la velocidad deseada mueve el botón de "latch" a su posición "activa" y lo deja allí. El circuito debe almacenar el valor de velocidad que hay en ese momento, para lo cual se cuenta con un dispositivo de "Memoria" al que le entra por un lado el número binario que corresponde a la velocidad instantánea y que, al activarse el botón de "latch", almacena el número que en ese preciso momento había en sus entradas (Usted puede ignorar cómo funciona el dispositivo de memoria y emplearlo como una "caja negra" [black box]). La salida permanente de la "Memoria" es el valor "consigna", almacenado. El vehículo tiene un artefacto que mide la velocidad y que consta de un pequeño engranaje en la caja de velocidades, al cual se le acopla un cable de acero que gira a razón de un cierto número de veces por segundo, de manera proporcional a la velocidad del carro. Se posee también un "Tacómetro Electrónico" que convierte las revoluciones de la "guaya" en un número binario de 7 bits que indica en todo momento la velocidad a la que se desplaza el automóvil, en Km/hora (entre 0 y 127 Km/h). De allí precisamente se toma la lectura que se almacena en la Memoria al presionarse el "latch".

Su sistema CAV debe producir una señal digital "**A**" que active automáticamente el acelerador (sin intervención del chofer) cuando se dictamine que la velocidad se ha reducido por debajo del valor deseado, y debe también producir una señal digital "**F**" que active el sistema de freno hidráulico cuando

la velocidad haya subido por encima del límite establecido. Tanto el mecanismo de aceleración como el de frenado operan de manera suave, sin producir grandes cambios ni sobresaltos, a fin de que la travesía sea lo más confortable posible; usted no tiene qué pensar en cómo hacer esa parte, pues ésta ya viene incluida dentro de los controles del vehículo; basta con que su circuito active las señales de Acelerar y de Frenar, que del resto se encargará el automóvil. Lo que sí debe incluir es un ciclo de ***histéresis*** que evite pasar de Aceleración a Frenado en forma muy seguida; para esto, su sistema sólo debe acelerar cuando la velocidad sea inferior al valor "consigna" menos 12.5%, y frenar al sobrepasarla en un 12.5%.

Para deshabilitar el Control Automático de Velocidad el conductor regresa el "latch" a su posición de reposo, en la cual se desactivan las señales de aceleración y de frenado.

Presente 1) un Diagrama de Bloques y una descripción verbal que indique, a nivel funcional, los diversos componentes que va a emplear en su diseño y sus interconexiones y, 2) un Diagrama Lógico para cada bloque funcional (comparadores, sumadores, etc.), con el diseño realizado con compuertas Nand.

4) FRENAR LOS COHETES

Diseñar el "Controlador de Frenado para la Ejecución de la Maniobra de Regreso a Tierra" de cierto vehículo espacial, que emplea para detenerse 3 motores independientes, de combustible sólido. Una vez emprendido el reingreso, si el piloto precisa una operación lenta (V0 en la tabla): ★ No se activa ningún motor si el transbordador aún está "Detenido" (0); ★ enciende uno (1) si su velocidad es "Lenta" o "Normal", ★ y dos (2) en caso de que venga "Sobreacelerado". Si se necesita realizar la maniobra de frenado más fuerte se emplea la posición V1; de nuevo: ★ no se enciende ningún motor (0) si el transporte está "Detenido"; ★ se prende uno (1) si su velocidad es "Lenta", ★ dos (2) si es "Normal" y ★ tres (3) si está "Sobreacelerado". Para V2 y V3 se indican las posibilidades en la tabla. El sistema dispondrá de una palanca de velocidades, lineal como la de un carro automático, que usted debe incluir en su diseño para especificar el frenado, de manera creciente, como: V0, V1, V2 y V3. También cuenta con un sistema externo (que usted ***no*** tiene que diseñar) que mide la velocidad actual de la nave y se la suministra a su Controlador, mediante dos (2) dígitos codificados en BCD (de 00 a 99 Km/minuto). El vehículo se considera "Detenido" por debajo de los 39 Km/m; en movimiento "Lento" hasta 59 Km/m, y 89 Km/m son el límite de velocidad "Normal". De 90 Km/m en adelante está "Sobreacelerado".

La NASA resume la operación así:

Sobreacelerado	2	3	3	3
Normal	1	2	3	3
Lento	1	1	2	3
Detenido	0	0	0	0
	V0	**V1**	**V2**	**V3**

Los números indican ***cuántos*** motores deben encenderse para cada condición. Usted debe decidir ***cuáles***.

a) Apoyándose en un Diagrama en Bloques explique un dispositivo que materialice su diseño, identificando ➢ las señales de entrada que ha planeado emplear y ➢ cómo serán físicamente (★ si son switches, de qué clase; si necesitan resistencias de pull-up o pull-down; ★ si son señales

lógicas) ➢ Debe dividir el problema en tres (3) etapas por lo menos, a fin de hacerlo manejable, y explicar en detalle su criterio para esa separación. ➢ En particular debe indicar: ★ de dónde obtiene todas las entradas para cada etapa; ★ cómo las codifica; ★ de qué manera produce los resultados; ★ cuáles salidas de una sección sirven de entradas a las siguientes, etc. (AYUDA: Los motores SON salidas: ¡ Usted va a controlarlos !)

b) Además de la función principal de su controlador (manejar el reingreso a tierra) está la de Supervisión; añada despliegues (Leds) que indiquen: ➢ el rango de velocidad en que se encuentra la nave (una luz para cada uno: De Detenido a Sobreacelerado); ➢ qué comando ha seleccionado el operador (de V0 a V3); ➢ una luz intermitente y una alarma audible si está "Sobreacelerado". Suponga que los actuadores que necesita operan con señales TTL.

c) Diseñe cada bloque de su diagrama y construya Diagramas Lógicos (compuertas, exors, etc.) Las secciones que pueda elaborar empleando circuitos integrados (CI) estándar serán mucho mejor evaluadas que si las tiene que construir usted mismo. Emplee la mayor integración que pueda (LSI, MSI, SSI); es decir, si necesita, por ejemplo, circuitos sumadores, es preferible usar "full adders" en lugar de hacerlos con compuertas o Exors, y así para cualquier otra función que haya incluido, como decodificadores, normales y de prioridad, etc. NO tiene que identificar los chips con su número (ej.: 7400), pero las funciones empleadas TIENEN que existir en el mercado. No puede inventarse chips... Si no las conoce mejor diseñarlas a su medida.

5) MONTACARGAS

Se trata de diseñar un dispositivo para supervisar y controlar un montacargas de émbolo hidráulico que se desplaza hasta una altura de 12 m. Cuando la carga está ya lista en la plataforma, abajo, el operario oprime el botón de **Subir**, que inicia el movimiento suave de la plataforma arrancando una (1) bomba hidráulica (**B1**) que inyecta aceite al pistón, hasta que se alcanza el primer sensor de posición (**S1**), que produce un "1" cuando el pistón pasa por ahí; esto hace que se encienda la segunda bomba (**B2**) para obtener la máxima fuerza de ascensión; el operador puede soltar ahora, a discreción, el botón de **Subir**, y el montacargas continuará su rápida elevación. El siguiente sensor (**S2**), a 11 m. del piso, hace que se detenga la primera bomba (**B1**) para desacelerar el movimiento; el sensor de "Fin_de_Carrera" (**FdC**) se encuentra en el punto preciso para detener **TODO** el movimiento. Las dos (2) bombas hidráulicas, que hay que controlar, se activan, para fortuna suya, con solo suministrarles un "1". Deben encenderse cinco (5) **LEDs**: "**LSubir**", mientras la plataforma esté ascendiendo; "**LB1**" y "**LB2**", en tanto que estén siendo energizadas las respectivas bombas B1 y B2; "**LDetenidoAbajo**" mientras la plataforma se encuentra abajo en reposo; y "**LDetenidoArriba**", cuando la plataforma active el "Fin_de_Carrera" (**FdC**) (usted debe indicar, con toda precisión, cómo intenta manejar los LEDs)

- El LED de "LDetenidoArriba" debe **parpadear** con período de 0.6 segundos (0.4 s on, 0.2 s off)
- Ninguna bomba encenderá, ***bajo ninguna circunstancia***, si está activo el interruptor **FdC** de "Fin_de_Carrera" (quiere decir que **ningún** punto ES condición de indiferencia, si FdC vale 1).
- Ninguna bomba encenderá hasta que no hayan pasado al menos 10 minutos desde que se la apagó.

Resuelva el ejercicio sólo para hacer **subir** la plataforma; no se preocupe luego de cómo hacerla bajar; su problema comienza abajo y termina arriba.

6) CALDERA

Diseñe un dispositivo con compuertas TTL (Ojo: "**NO**" son CMOS), que supervise y controle una caldera para producir el vapor que mueve un generador eléctrico de 400 Kva. Se poseen 3 grupos independientes de quemadores de gasoil. Si se desea una operación lenta, el operador enciende sólo el primer grupo, y va agregándole el segundo y el tercero si ve que necesita acelerar el calentamiento de la caldera. El sistema calefactor dispondrá de comandos (switches) que usted debe incluir en su diseño para efectuar las operaciones: Apagada, **M**oderada, **A**celerada y **U**ltra acelerada. También se cuenta con un sistema que mide la temperatura de la caldera y la suministra, para suerte suya, en 3 dígitos codificados en BCD (000 a 999 ºC). La caldera se considera "Fría" por debajo de 300 ºC; luego, "Tibia" hasta 599 ºC, y después, "Caliente" hasta 899 ºC. De los 900 ºC hacia arriba es "Sobrecalentada", lo que constituye una situación de riesgo que usted debe considerar y apagar inmediatamente TODOS los quemadores, SIN QUE **NINGUNA** OTRA CONDICIÓN SE INTERPONGA. [Produzca estas señales con comparadores, desde el termómetro BCD]

El cliente sumariza la operación así:

Sobreca-lentada	0	0	0	0
Caliente	0	1	2	3
Tibia	0	2	3	3
Fria	0	3	3	3
	Apa-gada	Mode-rada	Acele-rada	Ultra-Acel.

Los números indican cuántos quemadores deben encenderse para cada condición.

a) Explique con ***LUJO DE DETALLES***, apoyándose en un **Diagrama en Bloques**, un dispositivo que materialice su diseño, incluyendo cuáles son las señales de entrada que ha planeado emplear y cómo serán físicamente (switches, de qué clase; señales lógicas). Debe dividir el problema en etapas, a fin de hacerlo manejable, y explicar en detalle su criterio para escoger dichas etapas. Además de la función principal de su controlador (manejar el calentamiento de la caldera) está la de Supervisión; añada despliegues que indiquen: –el rango de temperatura en que se encuentra la caldera (cuatro luces, una para cada rango); qué comando ha seleccionado el operador; una luz **intermitente** y una alarma audible si se ha llegado al estado de Sobrecalentada. En todos los casos suponga que los actuadores que necesite operan con señales compatibles con TTL.

b) Diseñe cada bloque de su diagrama y construya un **Diagrama Lógico** (compuertas, exors, etc.) Use al máximo circuitos integrados estándar. Si necesita por ejemplo, sumadores, use chips sumadores en vez de hacerlos con compuertas o Exors, y así para cualesquiera otras funciones.

7) CALCULADORA BRAILLE

Se está preparando una calculadora para personas ciegas, en la que cada número debe representarse mediante un dispositivo de salida, consistente en un pequeño recuadro con 6 micro-solenoides (ver figura), que cuando se activan o energizan hacen que sobresalga levemente un pequeño punzón, conformando dinámicamente las marcas que codifican información en código braille. Diseñar el dispositivo con circuitos AND, OR y NOT, sólo para la posición numérica de las unidades (las decenas, centenas, etc. serán circuitos similares, agregados a la izquierda de las unidades), de acuerdo a la codificación de los números en Braille, indicada en la figura. Las entradas se representan en forma Binaria Natural, del 0 al 9, identificándose los bits de entrada como I3, I2, I1, I0. Los mapas y las tablas debe hacerlas **conservando este mismo orden**. Las salidas se identifican de la "a" a la "f" (ver figura).

Out ID	1	2	3	4	5	6	7	8	9	0
a b	● ·	● ·	● ●	● ●	● ·	● ●	● ●	● ·	· ●	· ●
c d	· ·	● ·	· ·	· ●	· ●	● ·	● ●	● ●	● ·	● ●
e f	· ·	· ·	· ·	· ·	· ·	· ·	· ·	· ·	· ·	· ·

REDES SECUENCIALES

1) JUEGO DE PING-PONG (BASIC) [Solución]

Este entretenimiento se presenta a vía de ejemplo reducido, pues sólo usa 4 Leds, en vez de los 12 propuestos en el laboratorio, y gana el primero que llegue a 5 puntos. La idea es hacer notar que el lenguaje es sencillo; no se necesita más si lo que se desea es realizar una implementación con compuertas y flip-flops, y tiene la ventaja de ser "interpretado", lo que significa que pueden hacerse cambios y verificarlos sin pasar por el proceso de compilarlo. Para jugar se necesita el "QBASIC.EXE", que puede encontrarse en el web, o algún otro interpretador de Basic como Powerbasic.

Obsérvese que cada instrucción numerada corresponde a un estado de la FSM (finite state machine), y que si se ha planeado implementarla con un flip-flop por estado, la correspondencia entre instrucciones numeradas y biestables es uno a uno (UN FLIP-FLOP de control por cada instrucción numerada)

El juego de Ping-Pong digital, estará conformado por dos secciones: Una de Simulación del juego y una de Anotación.

SIMULACIÓN DEL JUEGO

Esta etapa estará compuesta por:

- 12 emisores luminosos (LED) que simularán el recorrido de la pelota.
- 2 pulsadores que simularán los jugadores.

ANOTACIÓN

Se conformará esta etapa por un conjunto de 2 elementos luminosos de 7 segmentos, uno para cada jugador, para la presentación del conteo de los puntos obtenidos durante el juego.

REGLAS DEL JUEGO

- Cada jugador debe retornar la pelota cuando se encienda el primer led correspondiente a su zona, de lo contrario el jugador contrario anotará un punto.
- El inicio del juego no deberá estar asignado a un jugador específico, es decir, se debe poder iniciar un juego en cualquier lado de la cancha.
- Ganará un juego el primer jugador que marque 5 puntos, a no ser que ambos jugadores marquen 4 puntos; en este caso ganará el juego el primer jugador que posteriormente marque dos puntos más que el jugador contrario.

```
'PingPong.bas; Luis G. Uribe C., D15J3 V20J3 L23J3
'Para jugar: "QBASIC.EXE pinpon2b.bas" y Run-Start
'..(en el Menú)
'Juego REDUCIDO: Usa sólo 4 Leds, y gana el primero
'..que llegue a 5
'C/Instrucción numerada es UN FLIP-FLOP de control
'-------------------------------------------------------

'Init. Se llega aquí sólo por "pup": Power Up signal.
   C1 = 0: C2 = 0:   'Init. asíncronamente los contadores

10  'SLoop:              'Aquí comienzan los pulsos de reloj
    INPUT "P1-10 ", P1
    IF P1 = 1 THEN GOTO 30 : 'P1-Saca

20  INPUT "P2-10 ", P2
    IF P2 = 1 THEN GOTO 70 : 'P2-Saca
    GOTO 10 : 'GOTO SLoop
'------------------------------------- Ruptura de Secuencia

30  'P1-Saca, 'P2Responde
    INPUT "P1-Saca, P2Responde ", P1
    IF P1 = 1 THEN GOTO 110 : 'P1Error; P1 must return to 0
    L1 = 1: INPUT "P2-Responde ", P2
    IF P2=1 THEN GOTO 140 :  'P2ERR
40  L2 = 1: INPUT "P2-Responde ", P2: IF P2=1 THEN GOTO 140
50  L3 = 1: INPUT "P2-Responde ", P2: IF P2=1 THEN GOTO 140
60  L4 = 1: INPUT "P2:Responde: ",P2
    IF P2<>1 THEN GOTO 140 ELSE GOTO 70 : 'P1Respond
'------------------- Ruptura de Secuencia (No necesaria...)

70  'P2-Saca, 'P1Responde
    INPUT "P2-Saca, P1Responde ", P2
    IF P2 = 1 THEN GOTO 140
'---   140 es P2Error; P2 must return to 0
    L4 = 1: INPUT "P1-Responde ", P1
    IF P1 = 1 THEN GOTO 110 :  'P1ERR
80  L3 = 1: INPUT "P1-Responde ", P1
    IF P1 = 1 THEN GOTO 110
90  L2 = 1: INPUT "P1-Responde ", P1
    IF P1 = 1 THEN GOTO 110
100 L1 = 1: INPUT "P1:Responde: ", P1
    IF P1 <> 1 THEN GOTO 110 ELSE GOTO 30 : 'P2Responde
'------------------------------------- Ruptura de Secuencia

110 'P1ERR
    PRINT "P1ERR"   : 'This instruction ONLY for Simulation
    C2 = C2 + 1: 'Inc Count2
120 IF C2 = 5 THEN GOTO 180 : 'GOTO C2Win

130 'P1toPlay
    INPUT "P1-Saca  ", P1
    IF P1 = 1 THEN GOTO 30 ELSE GOTO 130
'---30: P2Responde; else P1toPlay (130)
'------------------------------------- Ruptura de Secuencia

140 'P2ERR
    PRINT "P2ERR"   : 'This instruction ONLY for Simulation
    C1 = C1 + 1: 'Inc Count1
150 IF C1 = 5 THEN GOTO 170 : 'GOTO C1Win

160 'P2toPlay
    INPUT "P2-Saca ", P2
    IF P2 = 1 THEN GOTO 70 ELSE GOTO 160
'---70: P1Responde; else P2toPlay (160)
'------------------------------------- Ruptura de Secuencia

170 PRINT "C1Win": 'C1Win
'---Equivalente al FLASHING del resultado
    END

180 PRINT "C2Win": 'C2Win
'---Equivalente al FLASHING del resultado
    END
```

2) CERRADURA DE COMBINACIÓN

Se posee un teclado de 10 botones individuales de reposición (push-buttons abierto/cerrado, ***SPST*** [*Single Pole Single Through*]), numerados del 0 al 9 y colocados en fila sobre una botonera. Diseñe con ella una llave electrónica que, si se sigue cierta secuencia de **N** valores, accionará durante 2 segundos una válvula solenoide que abre la cerradura. Activar más de una tecla al tiempo inhabilita el sistema por una (1) hora. Su programa debe detectar cuándo se ha oprimido un botón y eliminar rebotes; *en ningún caso* debe su sistema dar ***ni*** indicios que sirvan para adivinar la clave. Dos segundos es el tiempo máximo que separa las entradas; si no se cumple con él, su circuito deshabilitará el sistema durante 5 minutos. El usuario podrá **reiniciar** (Reset) el equipo si queda inhabilitado, tocando consecutivamente las teclas 0, 9, 1, 8, 2, 7, 3, 6, 4, 5 (visualice bien la secuencia, sobre la botonera, y verá que es fácil de recordar). Una vez hecho el programa, definir una nueva clave sólo deberá requerir del usuario, la introducción de una tabla tipo "DT" en su PIC 16F84A, con la secuencia de números que la conforman, *¡ y nada más !* (Cambiar una tabla con constantes en el Freescale). La determinación de la cantidad **N**, etc., debe hacerla automáticamente su programa (se sugiere que cuente cuántos elementos hay en la tabla, sin que el usuario tenga que hacerlo; para eso está el microcontrolador). Defina usted el tamaño máximo de la secuencia, que no será menor a 10 números, y la longitud mínima, que deberá ser superior a 4 dígitos.

3) UN ASCENSOR INTERESANTE

Un elevador cuya puerta permite sólo el paso de una persona tiene un sensor que genera un pulso muy estrecho cada vez que el peso aumenta (10 Kg. o más), y un sistema óptico en la entrada, que genera una señal que se activa mientras alguien intercepta la luz, al pasar por la puerta. Diseñe (diagrama lógico COMPLETO) un circuito que permita determinar el número de personas que hay dentro del ascensor, de manera que si éste es superior a 7 no arranque, y genere una señal de "exceso de personas y/o peso". Además, debe producir una salida que pueda luego emplearse para impedir que la puerta se cierre, antes de 10 segundos después que pasó el último usuario.

Recuerde: a) Garantizar que su circuito leerá las señales externas aunque éstas sean muy estrechas en relación al propio pulso de reloj de usted, y b) Sincronizar TODAS las entradas externas para que su circuito sea absolutamente sincrónico. Asegúrese de incluir en su diseño el diagrama de estados y el período de su señal de reloj. c) Genere su estado inicial de la manera correcta. d) Indique cómo es su circuito de "power-on reset", e incluya todas aquellas cosas que además crea convenientes para el esclarecimiento de su diseño.

4) REPRESENTACIÓN NUMÉRICA

Diseñar una red Secuencial Sincrónica con una línea de entrada **I**, serial, por donde se introducen cadenas de bits que representan información numérica codificada en **complemento a 2**, ***de longitud indeterminada***, que se inyectan desde el exterior comenzando por los bits menos significativos. La señal de control "**complemente**", **C**, opera así: Un cambio en **C**, bien sea a 0 o a 1, indica que por la entrada **I** llega el **bit de las unidades** de un nuevo número. El valor que toma C hace que la salida I sea como sigue: Si C = 0, la salida **Z(t)** debe ser igual a **I(t)**; Si C = 1, **Z(t)** debe tomar el valor del ***complemento a 2*** de **I(t)**. Produzca la salida Z del sistema, usando FF-D.

Repita, asumiendo que la información está representada en **complemento a 1**.

BIBLIOGRAFÍA

APPLICATION NOTE 287, Switch Bounce and Other Dirty Little Secrets:
http://www.maxim-ic.com/legal

A Guide to Debouncing, April, 2006, Jack G. Ganssle
Por suscripción, a jack@ganssle.com

Capítulo 4

PROGRAMACIÓN GENERAL

Todo el mundo debería aprender a programar computadoras; hoy es una necesidad equivalente a saber leer y escribir.

INTRODUCCIÓN

EL tema de la programación se reserva, no un capítulo ni un libro; abarca toda una profesión y hasta la vida entera. Es una actividad profundamente gratificadora, ampliamente creativa y una forma sin parangón de materializar ideas. ***Yo creo que programar es una manera de Arte***. A diferencia de mis hijos, mi primera experiencia en programación fue tardía, en mis cursos de Fortran de la universidad, pero el pináculo del éxtasis y del deleite lo alcancé al realizar el CODIDAC, primer **CO**mputador **DI**gital colombiano, y probablemente también de Latinoamérica, diseñado y construido con propósitos di**DÁ**Cticos entre 1969 y 1971 en la Universidad Javeriana de Bogotá. Nunca antes fue tan palpable para mí, como entonces –y tan sobrecogedora– la sensación de presenciar la ***concreción del curso del pensamiento***. Fue embelesante y arrobador ver cómo ***casi discurría*** aquel artefacto, de materia moldeada con nuestras manos; cómo tomaba decisiones, ejecutaba operaciones, hablaba, escribía, leía y reaccionaba... tanto el hardware, primero, como los programas y juegos que para él desarrollé después.

En este capítulo se pretende mostrar temas interesantes y útiles, rara vez encontrados en libros especializados en el tema, y que aplican tanto a la programación de sistemas a nivel de estaciones de trabajo y servidores, como también a los dispositivos hechos con microcontroladores "embebidos".

DÍGITOS DE VERIFICACIÓN

Sobre el tema genérico de la detección y corrección de errores y los programas asociados se encuentra bastante información: Manejo de bits de paridad, cálculos de códigos LRC (tipo "check-sum"); evaluación de polinomios CRC, bit a bit y mediante tablas; código Hamming para un bit y para ráfagas de errores... El Internet y las telecomunicaciones abundan en este tema.

Existe, sin embargo, para aplicaciones que deben operar con códigos introducidos a mano por el usuario o por un operador, la necesidad de tomar medidas para asegurar que no se trastocan los dígito, o se omite alguna cifra, lo que pudiera generar otro identificador, también válido, pero indeseado.

Cuando las personas usan códigos numéricos como los que identifican cuentas bancarias, tarjetas de crédito, artículos de una tienda por departamentos, etc., bien sea que se dactilen en un teclado de computadora o en algún cajero automático, se digiten empleando el teléfono, o se los lea en voz alta para los demás, existe la tendencia a cometer ciertos errores con mucha más frecuencia que otros.

De acuerdo a **Hamming**, padre de los códigos de detección y corrección de errores, las anomalías más comunes que comenten las personas en estas circunstancias son el intercambio de dos dígitos adyacentes de un número (por ejemplo, el 67 se escribe como 76) y doblar el dígito equivocado de un trío (el 667 se convierte en 677)

Verhoeff profundizó aún más en el tema, con los siguientes resultados:

- Errores sencillos: ***a*** se convierte en ***b*** (60% a 95% de todos los errores)
- Omitir o añadir un dígito (10% a 20%; corregible en códigos de longitud fija)
- Transposiciones adyacentes: ***ab*** se torna en ***ba*** (10% a 20%)
- Errores gemelos: ***aa*** se troca en ***bb*** (0.5% a 1.5%)
- Transposiciones de salto: ***acb*** se vuelve ***bca*** (0.5% a 1.5%)
- Errores gemelos de salto: ***aca*** se cambia por ***bcb*** (menos del 1%)
- Errores fonéticos (en inglés): ***a0*** se escucha como ***1a*** (por ejemplo, *thirty* y *thirteen* pueden confundirse; 0.5% a 1.5%)

En forma similar al CRC o LRC, se le añade a esta clase de códigos un **dígito de verificación** (**check digit**) con el propósito de detectar tales errores. No se emplean CRCs, a pesar de ser más robustos, porque éstos generan demasiados dígitos y los usuarios detestan oprimir tantas teclas.

Hay una variedad considerable de algoritmos para la generación del dígito de verificación, cada uno con fortalezas y debilidades; entre los más destacados están:

- IBM check (módulo 9) usado en la mayoría de las tarjetas de crédito (visa, mastercard, etc.)
- ISBN módulo 11, que forma parte de los códigos de identificación de libros (International Standard Book Number)
- Electronic Funds Transfer Routing Number check, que se emplea aplicado a los códigos con que se transfieren depósitos electrónicos entre bancos (ABA: American Bank Association)
- UPC check, para productos identificados con códigos de barras (Universal Product Code)
- Dual Mod 10
- Arithmetic Mod 10
- Arithmetic Mod 11
- Geometric Mod 11
- Geometric Mod 10
- Verhoeff's dihedral group $\mathbf{D}_5$

Los diversos "módulos" mencionados arriba implican que como parte del algoritmo se realiza una división (por 9, 10 u 11, en los ejemplos mencionados) y se toma el residuo. Hay que observar que si la división es por 11, los posibles residuos van del 0 al 10, sobrepasándose en uno (1) la cantidad de dígitos decimales (0-9), por lo que códigos como el ISBN emplean en este caso un décimo símbolo, una "**X**", lo que resulta incómodo de manipular en teclados numéricos convencionales, como los telefónicos.

Sobre las clasificaciones mencionadas se consiguen algoritmos y programas con profusión; sin embargo, el método más completo es el de **Verhoeff**, que detecta:

- **todas** las oportunidades en que se comete un (1) solo error,
- **todas** las transposiciones adyacentes,
- más del **95%** de los "**twin errors jump transpositions** and **jump twin errors**"
- y la mayoría de los errores fonéticos en inglés.

Su solución incluye la realización de multiplicaciones entre los diversos dígitos del código, considerados como integrantes de un "***dihedral group D5***", que **no** es conmutativa, lo que significa que a*b no es siempre igual a b*a). La tabla de multiplicar en estas condiciones es la siguiente (hay ***4 cuadrantes***):

	0	1	2	3	4	5	6	7	8	9
0	**0**	1	2	3	4	**5**	**6**	**7**	**8**	**9**
1	**1**	2	3	4	**0**	**6**	**7**	**8**	**9**	**5**
2	**2**	3	4	0	**1**	**7**	**8**	**9**	**5**	**6**
3	**3**	4	0	1	**2**	**8**	**9**	**5**	**6**	**7**
4	4	0	1	2	**3**	**9**	**5**	**6**	**7**	**8**
5	**5**	**9**	**8**	**7**	**6**	**0**	**4**	**3**	**2**	**1**
6	**6**	**5**	**9**	**8**	**7**	**1**	**0**	**4**	**3**	**2**
7	**7**	**6**	**5**	**9**	**8**	**2**	**1**	**0**	**4**	**3**
8	**8**	**7**	**6**	**5**	**9**	**3**	**2**	**1**	**0**	**4**
9	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

La ecuación de **Verhoeff** genera el dígito de verificación, y para determinar si hay error o no, se lo incluye, y un nuevo recálculo sobre el código extendido debe dar "cero" si el ingreso es correcto (como en el caso de ***algunos*** CRCs, que cuando se le agregan al mensaje original y se repite el algoritmo sobre el mensaje ampliado, producen un valor global de "cero"). Obsérvese que el dígito

de verificación de Verhoeff toma ***la primera posición en el código***; como lo usual es añadirlo al final, la rutina que se suministra incluye funciones para reversar una y otra vez los dígitos de la cifra original.

Ciertos errores pasan **sin ser detectados** por este algoritmo como, por ejemplo, la *mayoría* de los "jump twin errors" que involucran ***dígitos con diferencias de 5***, como 050 y 505, 161 y 616, 272 y 727, 494 y 949), pero sí captura, por ejemplo, el 383 y 838.

Además, para eliminar los casos en que se añade o pierde algún dígito, es **recomendable emplear códigos de tamaño fijo**, que se verifican totalizando el número de cifras introducidas.

La descripción matemática del algoritmo de Verhoeff y el "dihedral group $\mathbf{D}_5$" sobrepasan los alcances de esta presentación, pero sí pueden encontrarse a continuación las rutinas en **C**, y un programa de prueba. Van **dos listados**; el primero, genérico, para uso en estaciones de trabajo o servidores, el segundo para microcontroladores embebidos, que optimiza el uso de la memoria **empaquetando dos (2) matrices en un solo arreglo de caracteres**, aprovechando el hecho de que la información que las conforman es módulo 10 (del 0 al 9); por tanto cada dígito necesita 4 bits, y caben 2 dígitos en cada byte ("char").

Se recomienda leer antes el primer programa, que es más claro, para no abrumarse con la presentación del segundo. Desde luego, este último puede emplearse también en estaciones de trabajo…, no está vedado.

Los siguientes valores sirven como testigos para verificar la operatividad del programa:

- 1
- 1**5**
- 15**0**
- 150**4**
- 1504**9**
- 15049**3**
- 150493**0**
- 1504930**6**
- 15049306**8**
- 150493068**6**
- 1504930686**1**
- 15049306861**3**
- 150493068613**3**
- 1504930686133**6**
- 15049306861336**6**
- 150493068613366**1**
- 1504930686133661**3**
- 15049306861336613**1**
- 150493068613366131**3**
- 1504930686133661313**7**
- 15049306861336613137**1**
- 150493068613366131371**1**
- 1504930686133661313711**9**
- 15049306861336613137119**4**
- 150493068613366131371194**2**
- 1504930686133661313711942**4**
- 15049306861336613137119424**9**
- 150493068613366131371194249**4**
- 1504930686133661313711942494**9**
- 15049306861336613137119424949**8**
- 150493068613366131371194249498**3**
- 1504930686133661313711942494983**6**

Por último, comenzando con 1, 2, 3..., 0, y repitiendo el ciclo hasta completar las 127 posiciones de la línea de entrada, el código de verificación que se produce es el **6**.

```
// ChkDigit.c, Luis G. Uribe C. M04S7
//
//          VERHOEFF'S DIHEDRAL GROUP D5 DIGITS CHECK
//
// SUMMARY: When keying numbers (codes, amounts) people tend
// ..to make certain kinds of mistakes more often than others.
// ..J. Verhoeff (Error Detecting Decimal Codes, Mathematical
// ..Centre Tract 29, Amsterdam, 1969) categorizes:
//
// .>Single errors: a becomes b (60%--95% of all errors)
// .>Omitting or adding a digit (10%--20%) [detected counting
// ..number of digits in fixed format codes: credit cards, etc]
// .>Adjacent transpositions: ab becomes ba (10%--20%)
// .>Twin errors: aa becomes bb (0.5%--1.5%)
// .>Jump transpositions: acb becomes bca (0.5%--1.5%)
// .>Jump twin errors: aca becomes bcb (below 1%)
// .>Phonetic errors: a0 becomes 1a [since the two have similar
// ..pronunciations in English: thirty & thirteen] (.5%--1.5%)
//
// SOLUTION: Add ONE redundant digit. More used:
// ISBN mod 11 (awkward: use 0..9 and X, difficult to key)
// IBM (used by most credit card companies)
// Electronic Funds Transfer (ABA EFT) used between banks
// Universal Product Code (UPC bar code; similar to IBM)
// Verhoeff's Dihedral Group D5 (multiplication in this group is
// ..NOT commutative i.e., a*b is not always equal to b*a).
// See program bellow. Consult references for further analysis.

/* ---------------------------------------------------------- */
// Include Files & DEFINES
#include <stdio.h>
#include <string.h>

#define LINESIZE   130 // would be smaller for a ATM or similar

#define compute(l)      Verhoeff( (l), 1 )
#define check(l)        Verhoeff( (l), 0 )

/* ---------------------------------------------------------- */
// GLOBAL VARIABLES
static
int F[8][10] =   { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, },
                   { 1, 5, 7, 6, 2, 8, 3, 0, 9, 4, },
                   { 5, 8, 0, 3, 7, 9, 6, 1, 4, 2, },
                   { 8, 9, 1, 6, 0, 4, 3, 5, 2, 7, },
                   { 9, 4, 5, 3, 1, 2, 6, 8, 7, 0, },
                   { 4, 2, 8, 6, 5, 7, 3, 9, 0, 1, },
                   { 2, 7, 9, 3, 8, 0, 6, 4, 1, 5, },
                   { 7, 0, 4, 6, 9, 1, 3, 2, 5, 8, }, };
static
char Inv[10] =     { 0, 4, 3, 2, 1, 5, 6, 7, 8, 9, };
static
int Op[10][10] = { { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, },
                   { 1, 2, 3, 4, 0, 6, 7, 8, 9, 5, },
                   { 2, 3, 4, 0, 1, 7, 8, 9, 5, 6, },
                   { 3, 4, 0, 1, 2, 8, 9, 5, 6, 7, },
                   { 4, 0, 1, 2, 3, 9, 5, 6, 7, 8, },
                   { 5, 9, 8, 7, 6, 0, 4, 3, 2, 1, },
                   { 6, 5, 9, 8, 7, 1, 0, 4, 3, 2, },
                   { 7, 6, 5, 9, 8, 2, 1, 0, 4, 3, },
                   { 8, 7, 6, 5, 9, 3, 2, 1, 0, 4, },
                   { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, }, };

/* ---------------------------------------------------------- */
char Verhoeff ( char *line, char compute )                /*()*/
{  char check;
   unsigned int i;
   char *tmp = "0";

   strrev( line );          // strrev: reverse a string on place
   check = 0;
   for( i = 0; i < strlen( line ); i++ )
       check=Op[ check ][ F[(i+compute)%8][line[ i ] - '0'] ];

   strrev( line );
   if( compute ) {
       tmp[ 0 ] = Inv[ check ] + '0';
       strcat( line, tmp );
       return Inv[ check ];
   }
   return check;

}

/* ********************************************************** */
void main ( void )                                        /*()*/
{   static char line[ LINESIZE ];

   for( ;; ) {
       fputs( "Initial Number: ", stdout );
       gets( line );
       compute( line );
       puts( line );

       fputs( "Verificar: ", stdout );
       gets( line );
       if( check( line ) == 0 )     // check()==0 mean ok
           puts( "OK" );
       else                     // May use a while(check) to wait
           puts( "Try again" );// ..until key in correct code!
   }
}
```

```
// ChkDig4.c, Luis G. Uribe C. M04S7, for embedded
//            VERHOEFF'S DIHEDRAL GROUP D5 DIGITS CHECK
//
// Because all numbers are module 10 (0..9), we may store BOTH
// ..arrays, F[] and Op[], in one single char array, FOp[].
// ..We use 3 macros to assist us: s(a,b) to store both numbers
// ..in one byte over FOp[], F(a,b) to retrieve the F[] array
// ..and Op(a,b) to retrieve the Op[] array, from FOp[].
/* ---------------------------------------------------------- */
// Include Files & DEFINES
#include <stdio.h>
#include <string.h>
#define LINESIZE   130 // would be smaller for a ATM and phones
/* ---------------------------------------------------------- */
static char Verhoeff ( char *line, char compute )          /*()*/
{
#define compute(b) Verhoeff( (b), 1 )  // fake routines for bet-
#define check(b)   Verhoeff( (b), 0 )  // ..ter understanding
// Ancillary Macros
#define s(F,Op)    ((F)<<4|(Op))           // store  F,Op in Fop[]
#define F(a,b)      (FOp[(a)][(b)]>>4)     // retrieve F  from FOp
#define Op(a,b)    ((FOp[(a)][(b)])&0x0F) // retrieve Op from FOp

static unsigned char Inv[10] = { 0, 4, 3, 2, 1, 5, 6, 7, 8, 9};

static unsigned      // Store F at left of FOp, Op at right
char FOp[10][10]= { s(0,0), s(1,1), s(2,2), s(3,3), s(4,4),
                    s(5,5), s(6,6), s(7,7), s(8,8), s(9,9)},
                  { s(1,1), s(5,2), s(7,3), s(6,4), s(2,0),
                    s(8,6), s(3,7), s(0,8), s(9,9), s(4,5)},
                  { s(5,2), s(8,3), s(0,4), s(3,0), s(7,1),
                    s(9,7), s(6,8), s(1,9), s(4,5), s(2,6)},
                  { s(8,3), s(9,4), s(1,0), s(6,1), s(0,2),
                    s(4,8), s(3,9), s(5,5), s(2,6), s(7,7)},
                  { s(9,4), s(4,0), s(5,1), s(3,2), s(1,3),
                    s(2,9), s(6,5), s(8,6), s(7,7), s(0,8)},
                  { s(4,5), s(2,9), s(8,8), s(6,7), s(5,6),
                    s(7,0), s(3,4), s(9,3), s(0,2), s(1,1)},
                  { s(2,6), s(7,5), s(9,9), s(3,8), s(8,7),
                    s(0,1), s(6,0), s(4,4), s(1,3), s(5,2)},
                  { s(7,7), s(0,6), s(4,5), s(6,9), s(9,8),
                    s(1,2), s(3,1), s(2,0), s(5,4), s(8,3)},
                  {     8,      7,      6,      5,      9,
                        3,      2,      1,      0,      4,},
                  {     9,      8,      7,      6,      5,
                        4,      3,      2,      1,      0 }};
```

```
   char check;
   unsigned int i;
   char *tmp = "0";

   strrev( line );          // strrev: reverse a string on place

   check = 0;

   for( i = 0; i < strlen( line ); i++ )
       check=Op(check,F(((i + compute) % 8),(line[ i ] - '0')));

   strrev( line );

   if( compute ) {
       tmp[ 0 ] = Inv[ check ] + '0';
       strcat( line, tmp );
       return Inv[ check ];
   }

   return check;

}



/* ********************************************************** */
void main ( void )                                          /*()*/
{
   static char line[ LINESIZE ];

   for( ;; ) {
       fputs( "Initial Number: ", stdout );
       gets( line );
       compute( line );
       puts( line );

       fputs( "Verificar: ", stdout );
       gets( line );
       if( check( line ) == 0 )
           puts( "OK" );
       else
           puts( "Try again" );
   }
}
```

PARECE SIMPLE

Este ejemplo lo presento porque muchos estudiantes fallan al codificar problemas similares en **C**:

"Escriba un programa que genere una sola vez los números del 0 al 255 inclusive, empleando sólo variables de tipo "char".

Se le ha enseñado al buen programador que ***todos*** los ciclos que escriba deben tener ***siempre***, una forma o estructura muy bien definida, siempre la misma, para evitar errores del tipo "más menos uno" (***off by one error***), que hacen que los "loops" a veces ejecuten una pasada de más, o de menos. La forma suele ser ésta:

```
char i;
for( i = 0; i < LIMIT; i ++ )
```

Se garantiza que un "for" así siempre realizará `LIMIT` pasadas, con la variable de control variando de `0` a `LIMIT-1`, siempre que en el ciclo no se altere `i`. Por tanto se estará tentado a codificar la solución así:

```
for( i = 0; i < 256; i ++ )
```

lo que –supone quien programa– cubriría precisamente el rango solicitado. Claro que 256 está por encima de lo que puede representarse en un byte, que es un requisito… Si `i` ha de ser un byte, lo mejor será escribir el for así:

```
for( i = 0; i <= 255; i ++ )
```

Todo arreglado: El rango es el acordado, las variables son "char" y las constantes caben todas en un byte…

!Equivocación!

Ese código nunca concluye; es un "loop" infinito. La razón es la siguiente: Cuando `i` llega a `255` la condición `i <= 255` aún se cumple, por lo que se repite el lazo para las acciones concernientes a `i == 255` (imprimir i, por ejemplo); al llegar a la finalización del for se incrementa primero `i` y se regresa a la comparación (`i <= 255`) a ver si se concluyó el trabajo; sin embargo, al incrementar `i`, ésta pasa a valer **cero**, pues en variables restringidas a un byte, incrementar `255`, máxima representación, lo convierte en cero ("**rollover**"); cero es una cantidad válida para ese rango, y el ciclo continúa **sin final**.

Entonces nuestro nervioso codificador estrella (¿o estrellado?) opta por proponer el siguiente cambio:

```
char i;
for( i = 0; i < 255; i ++ ){    printf( "i vale… /* ejemplo incompleto */ }
printf( "i vale… /* ejemplo incompleto */
```

Ahora sí es verdad. Cuando `i` vaya por `254` se entra en el loop, al final `i` se incrementa y vale `255`; ahora `i` **no** es menor que `255` y el ciclo termina. Como faltó una pasada, se repite el código una vez más al

finalizar el "for", lo cual no resulta bonito en extremo, pero lo principal es que funcione. (El ***printf*** del ejemplo está incompleto; es sólo para indicar que ahí ha de ejecutarse algo para cada valor de `i`).

!PEOR!

Ese ciclo hace mucho rato que está mal, porque "char" es "**signed char**" en la implementación estándar ANSI del C; así que después de `127`, `i` toma valores ***negativos***, que **siempre** van a ser **inferiores** a `255`; el ciclo jamás termina pues `i` siempre resulta válida para todos los valores que asume.

Y por qué al `255` no lo trata el compilador como `-128`? En el texto de referencia por excelencia, "The C Programming Language", por Kernighan y Ritchie (éste último es el inventor del lenguaje...), en la Pág. 171 dice a la letra:

"The type of an integer constant depends on its form, value and suffix. If it is unsuffixed and decimal, it has the first of these types in which its value can be represented:

`int, long int, unsigned long int.`

Así que `255` se representa como esa cantidad, en el espacio reservado para un entero, y siempre será positivo, y por tanto mayor que todos los números entre `-128` y `127`.

HAY VARIAS ALTERNATIVAS DE SOLUCIÓN. UNA CONSISTE EN LO SIGUIENTE:

```
#include <stdio.h>
unsigned char i;

void main ( void )
{  unsigned char i;     // ahora sí es "unsigned"

   for( i = 0; i < 255; i ++ ) {
       printf( "%3u ", i );         // note el empleo de %3u (unsigned)
   }
   printf( "%3u ", i );
}
```

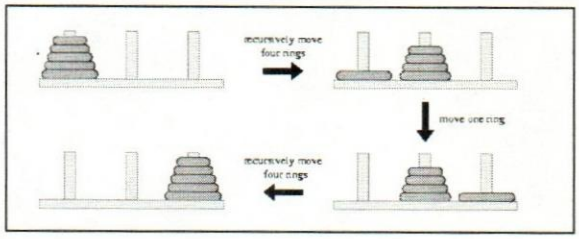
EL "FOR" REALIZA EL TRABAJO DE 0 A 254 INCLUSIVE, AL LLEGAR A 255 TERMINA Y SE EJECUTA LA ACCIÓN, UNA VEZ MÁS, POR FUERA DEL CICLO (COMO YA DIJE, NADA EXCEPCIONALMENTE HERMOSO, PERO FUNCIONAL).

Similares problemas ocurrirían si se pidiera **llegar** a alguna de las fronteras representadas por enteros [**u**int: `65535`] o enteros largos [**u**long: 4.294,967.295]; el problema solo tiene que ver con la **finalización del ciclo**. En el caso del "**u**long" es peor, porque a veces uno puede colocar como variable de control, una cuya representación tiene más tamaño del que se pide en el problema; por ejemplo, `int` para llegar a `256` y `long` para ir hasta `65535`, pero no es fácil conseguir algo más largo que `long` . . .

Una nota aclaratoria: Algunos compiladores para sistemas embebidos toman como representación por defecto el "**unsigned char**", lo cual ocasiona toda clase de inconvenientes pues se apartan del estándar ANSI; como ejemplo está el ***Codewarrior*** para microcontroladores Freescale (antes Motorola).

RECURSIVIDAD

Hay aproximaciones a la programación que sorprenden y a mí me alelan. Uno de los ejercicios clásicos que más deslumbran se llama Las Torres de Hanoi, inventado por el matemático francés Édouard Lucas en 1883. Se tienen tres postes y una serie de discos que reposan, de mayor a menor, en el primero. El ejercicio consiste en ir moviendo uno a uno los discos, hasta trasladar la pila original a la tercera torre, la de la derecha, respetando una sola regla: Nunca un anillo grande puede colocarse encima de uno más pequeño.



Varias consejas adornan el juego: Dicen que algunos monjes están en la tarea de resolver uno de estos ejercicios, con 64 discos de oro; cuando terminen sobrevendrá el fin del mundo, pero podemos estar tranquilos porque se ha calculado que el número de jugadas será de 2^64-1; si les lleva un segundo cada movimiento, tardarán unos 584.542 millones de años, y se estima en la actualidad que la creación tiene como 13.700 millones de años, así que los monjes tardarán en terminar su juego unas 43 veces más que la edad actual del universo. (*Note que el problemita de premiar al inventor del juego de Ajedrez con un granito de trigo por el primer cuadro del tablero, dos por el segundo, y así sucesivamente, doblando siempre en cada posición la cantidad de granos del cuadro anterior, tiene la misma respuesta numérica:* ***2^64-1****. Si cada grano de trigo pesa 1/5 de gramo, la producción de trigo necesaria para pagar el premio llegaría al equivalente de 460 toneladas* ***por cada uno*** *de los 8 mil millones de personas que habitan la tierra, cuando en realidad la producción promedio anual de trigo en la actualidad sólo asciende a 592 millones de toneladas, por lo que habría que acopiar* ***la producción completa de 6,232 millones de años****, a las cifras actuales, para conseguir el "puñado" de trigo que, según el rey, fue lo que le pidió aquel anónimo genio. La mitad de la edad del universo produciendo trigo para pagarle…*

Si piensan en jugar con unas 12 fichas, verán que el enfoque tradicional (ir tanteando...) no resulta tan sencillo. Pero puede formularse el problema de una manera recursiva, y entonces verán cómo la solución es en realidad muy simple y elegante.

La formulación recursiva es la siguiente: Para pasar N discos desde la torre **a** (izquierda) hasta la **b** (derecha), usando la del centro, **t**, en forma **t**emporal, basta con trasladar N-1 discos desde **a** hasta **t**, ubicar el disco restante (el más grande de la pila) en **b**, y luego colocarle encima los N-1 discos que estaban en **t**.

Para escribir un programa resulta más conveniente numerar las torres como 1, 2 y 3, en vez de identificarlas con letras. Así, para mover discos desde la 1 hasta la 3, se usa la 2 como temporal, y en general, para mover algo desde la torre **a** hasta la **b** (ahora **a** y **b** representan valores numéricos), se usa la torre "**6-a-b**" como **t**emporal. Si no lo ve a la primera, escriba las posibles combinaciones y verá cómo funciona esa cifra mágica.

Este programa lo denomino Hanoi mínimo, porque luego se le pueden poner muchos adornos a fin de que se vean discos de colores moviéndose lentamente en la pantalla. Pero la esencia del algoritmo es la que aquí presento. Al correr, el programa irá escribiendo de dónde a donde hay que mover una ficha. A

continuación del código va un ejemplo de una corrida. Si recortan unas fichas de papel y van siguiendo los movimientos que indica el programa, podrán hacerse una idea de cómo funciona el algoritmo.

```
// HANOImin.C, Luis G. Uribe C.: J23M6 J08F7
// The power of recursive calling: To move n disks from tower 1 to 3,
// .. move n-1 disks to tower 2 (tmp), then move disk n to tower 3,
// .. finally move the n-1 disks from tower 2 to 3. THAT'S IT !

const int N = 3;   // define here NUMBER OF DISKS; compile and go

void hanoi( int n, int a, int b );    // function prototype

/* ************************************************************** */
void main ( void )                    /* HANOImin.C */        /*()*/
{
   hanoi( N, 1, 3 );   // MOVE n disks from tower 1 to 3. That's all

}  /* main() */

/* ------------------------------------------------------------- */
void hanoi ( int n, int a, int b )                              /*()*/
{
   int t;              // temporarily tower to move n-1 disks

   if( n == 1 ) {      // Got only 1 disk? It's easy! JUST MOVE IT ON
       putchar( a + '0' );  putchar( '>' );  putchar( b + '0' );
       while ( getchar() == 0 );   // Press 'Enter' to continue...
       return;
   }

   t  =  6 - a - b;        // Now, THIS is a magic number...
   hanoi( n - 1, a, t );   // move all n - 1 disks from tower a to t
   hanoi( 1    , a, b );   // move the remaining 1 disk from a to b
   hanoi( n - 1, t, b );   // then move the n-1 disks from t to b !!

}  /* hanoi() */
```

Para 3 fichas:

1>3 1>2 3>2 1>3 2>1 2>3 1>3 (ficha de la torre 1 a la 3, de la 1 a la 2, 3 a 2, etc.) Observen que las reglas se siguen, y se necesitan 7 movimientos: 2^3-1

Para 4 fichas, 15 movimientos:

1>2 1>3 2>3 1>2 3>1 3>2 1>2 1>3 2>3 2>1 3>1 2>3 1>2 1>3 2>3

ROMANOS Y ARÁBIGOS

Este ejercicio es un abrebocas a los alardes de creatividad. Se trata de una rutina clásica de conversión de números, esta vez entre romanos y arábigos, que funciona en ambos sentidos, tomando la información de la línea de comandos: `romans n...n [1..3999] MCM...`

Como siempre, ***hay que validar la información suministrada por el usuario***, para verificar que corresponda a cifras bien formadas, tanto si son arábigas como si son números romanos. Para validar los Arábigos basta con determinar si están dentro del rango, que hemos definido según la usanza, como: Mayor que cero y Menor que 4000. Pero validar el número romano es complicado, porque a veces la información está compuesta por dos y hasta tres símbolos repetidos. Pero a veces el pensamiento lineal no es la mejor solución a un problema; miremos una alternativa:

```
char *_id = "romans.c,  Luis G. Uribe C, J13S2007\n"
"Conversion between Arabic and Roman numbers (both ways)\n"
"08F1991 M13S4 D20S8 .\n\n"
"    USAGE: romans n...n [1..3999] MCM... (may be mixed)\n";

/* Include Files */

#include <ctype.h>      // isdigit()
#include <stdio.h>      // printf, puts
#include <stdlib.h>     // exit(), atoi()
#include <string.h>     // str*()

/* Typedefs */

typedef struct {
   int   order;   /* Número que rompe la secuencia   */
   char *symbol;  /* Símbolo que define 'order'      */
}  roman;

typedef enum { FALSE, TRUE } bool;

/* Function Prototypes */
int  toArabic( char * s );
void toRoman( int i, bool prnt );

/* External Variables */

char   buf[ 60 ];
roman romans[] =  {
       1000, "M",     900, "CM",     500, "D",      400, "CD",
        100, "C",      90, "XC",      50, "L",       40, "XL",
         10, "X",       9, "IX",       5, "V",        4, "IV",
          1, "I"
       };

/* Defines */
#define SIZE    ( sizeof( romans ) / sizeof( roman ) )
#define PRINT       TRUE
#define NO_PRINT    FALSE

/* ******************************************************** */
void main ( int argc, char ** argv )                        /*()*/
{  /* romans.c */

   int v;

   if( argc == 1 ) {    // cmnd line need 1 parameter at least
       puts( _id );
       exit( 1 );
   }

   while ( * ++ argv ) {    // may give several (mixed) inputs
       if( isdigit( ** argv ) ) {  // 0-9 means input Arabic
           printf( "Arabic  '%4d'\t\tis romanic: '",
                   v = atoi( * argv ) );
           toRoman( v, PRINT );
           puts( "'" );
       }else {                     // else input is Roman
           strupr( * argv );
           printf("Roman '%s'\tis arbic:    ", * argv );
           if((v = toArabic( *argv )) > 0) { // error if v<=0
               printf( "'%d'\n", v );
           }
       }
    }
    exit( 0 );

}  /* main() */

void toRoman ( int i, bool print )                          /*()*/
{  /* toRoman(): Arabic to Roman conversion */

   int   j;
```

```
    roman *rp;                      /* rp: romans pointer */

    if( i >= 4000 || i <= 0 ) { // Validate arabic input
        strcpy( buf, "Error" );
        if( print ) {
            printf( "\tRomans.c Error. Range: 1 .. 3999" );
        }
        return;
    }

    for(*buf ='\0',j=0, rp = &romans[ 0 ]; j < SIZE; j++, rp++){
        for( ; i >= rp -> order; i -= rp -> order ) {
            strcat( buf, rp->symbol );
        }
    }
    if( print ) {
        printf( "%s", buf );
    }

}  /* toRoman() */
```

```
int toArabic ( char * s )                                   /*()*/
{  /* toArabic(): Roman to Arabic conversion */

    bool    err;
    char   *str = s;
    int     j, n = 0, q;
    roman *rp;                      /* rp: romans pointer */

    while( *s ) {
        err = TRUE;
        for(j = 0, rp = &romans[ 0 ]; j < SIZE; j ++, rp ++) {
            if(strncmp( rp->symbol, s, q = (j % 2)+1 ) == 0) {
                n += rp -> order;
                s += q;
                err = FALSE;
            }
        }
        if( err ) {
          error:
            puts( "\tRomans.c Error. Bad Roman Number" );
            return -1;
        }
    }

    // Validate Romanic input
    // CLEVER: because validating romans is complicated, use
    // ..this approach: Calculate Arabic (this, we know how to);
    // ..silent reconvert it to Romanic (also, we know how to
    // ..do this), and then: BOTH strings must compare 'equals'
    // ..for Roman input to be valid!

    toRoman( n, NO_PRINT );        // Silent reconvert n to roman
    if( strcmp( buf, str ) == 0 ) { // reconverted Arabic ==
        return n;              // ..original Romanic, return Arabic
    }
    goto error;                    // ..else, signal error

}  /* toArabic() */
```

```
#if 0
Arabic   '2007'  is roman: 'MMVII'
Roman 'MCMXCVII' is arbic:  '1997'
#endif
```

Cuando vamos a convertir de romano a arábigo, hay que validar la entrada en romanos, que es lo difícil. Como ya sabemos convertir de arábigo a romanos, y es fácil, lo que se hace es convertir primero el número (antes de validarlo) de romano a arábigo, reconvertirlo de arábigo a romano, y comparar las dos cadenas de caracteres; si son iguales, la entrada era válida. Por ejemplo, si se introduce el número 4 como una serie de cuatro unos (**IIII**, lo que es inapropiado para Romanos), la rutina los convierte a 4 en arábigo; lo reconvierte a romanos, lo que produce el valor correcto: **IV**, y como las dos secuencias son diferentes se advierte al usuario que ha habido una entrada inapropiada. Pero si la entrada es correcta, como en el ejemplo (`Roman 'MCMXCVII' is arbic: '1997'`), se convierte el `'MCMXCVII'`, lo que produce `'1997'`, se lo reconvierte a romanos y vuelve a dar: `'MCMXCVII'`, por lo que se sabe que la entrada era apropiada, y se imprime el resultado.

PI

3.141592_653589_793_238462_643383_279502_884197_169399_375105_820974_944592_307816_406286_208998_628034_825342_11

A veces vemos métodos ingeniosos para obtener ciertos resultados, como éste, pensado para aproximar el ***valor de PI*** ***lanzando números al azar***... ***¿Qué tienen que ver los números al azar... con PI***?

```
// pi.c (by random), Luis G. Uribe C., V14S7

// OBSERVACIÓN: El área del círculo unitario (radio 1) es PI. El cuadrado
// ..en el que se encuentra inscrito dicho círculo, tiene lado 2 y área 4.
// ..La relación entre las dos áreas (círculo/cuadrado) es PI/4. De una
// ..cantidad TOT, muy grande (100 millones) de puntos tomados al azar dentro
// ..del cuadrado, algunos estarán también dentro del círculo (InCircle).
// ..La relación entre esas dos cifras se aproxima a PI/4, y por tanto PI será:
// ..   "4 * InCircle / TOT"

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define  STEP 1000000L                    // one million

void main( void )
{
   double x, y;              // coordenadas de un punto en el espacio
   double InCircle = 0;      // puntos tomados por azar, en el círculo
   double tot      = 0;      // total de puntos, dentro del cuadrado
   double range    = 0;      // var. aux, para impresión

   srand( (unsigned)time( NULL ) );     // randomize
   putchar( '\n' );

   printf( "Valor al que debe aproximarse PI:\n%s\n\n", // Referencia!
" 3.141592653589793,238,462,643,383,279,502,884,197,169,399,375,105\n"
"                   820,974,944,592,307,816,406,286,208,998,628,034,"
          "825,342,11" );

   for(;;) {                            // choose x,y by random
       x=(double)rand() / RAND_MAX;     // RAND_MAX is 2,147,483,647
       y=(double)rand() / RAND_MAX;     // range is 0..1
       if( x*x + y*y <= 1.0 )           // x,y inside circle?
           InCircle++;
       if( ++tot > range ) {        // inc tot, and conditional print
           printf( "\r%- 18.15f, % 19.0f",4.0 *InCircle / tot, tot );
           range += STEP;               // get new range to print
       }
   }
}
```

Se imprime uno de cada millón de resultados, para no *ralentizar* el procesador; `printf( "\r` sobrescribe los valores.

POLÍTICA Y MECANISMOS

"Politics and Mechanisms" es un concepto ***clave*** y fundamental en la programación moderna. Aunado al "Top-Down design", es la base de la actual Orientación a Objetos, y los conceptos fundamentales que nos enseñan: Encapsulación, ocultamiento, independencia y desacople entre módulos.

En programación, la Política significa "**lo que hay que hacer**", y los Mecanismos "**cómo se va a hacer**". Si se logran escribir los programas indicando primero la serie de actividades que hay que hacer, con independencia del cómo, y al final se implementan dichas actividades o políticas, y si la separación es absoluta, se tendrá un sistema al que podrá cambiársele el sustrato (hardware, configuraciones) con sólo modificar las partes que se relacionan; el programa principal no habrá que tocarlo. Esto es fundamental cuando queremos que nuestras aplicaciones corran con libertad en variedad de configuraciones de hardware, como es el caso en los sistemas embebidos.

Para ilustrar el tema *acerca de* Política y Mecanismos, lo *que* voy a comenzar *yo* por hacer es un programa de animación que muestre en la pantalla del PC una película, cuyas secuencias estarán en texto común y corriente (no son ***bit maps***) *porque* así se mantiene la simpleza del programa y enfocamos los conceptos.

¿*Con qué* información se alimentará el programa? Un archivo con la serie de secuencias (frames), cuyo formato será muy simple; cada frame consistirá en una secuencia de caracteres ASCII imprimibles, que representarán las escenas en la pantalla del PC. La primera línea contiene un número, a la izquierda, que indica cuántas veces hay que desplegar esa figura antes de pasar a la otra. Como ejemplo, dos escenas tomadas de La Guerra de las Galaxias, por Simon Jansen (ver referencias); se muestra la princesa Leia colocando la grabación en R2D2. El dibujo está formado por 13 líneas de 67 caracteres máximo.

```
11
                    ,===
                   (@o o@
                    \ -/
                  //~  ~~\
                 /  (   ) \          ___
                /_/\    /\_|        / ()\
                 \\ \\ /| ||      _|_____|_
                   @| | | @      | | === | |
                    | | |         |_|  o  |_|
                    | | |          ||  o  ||
                    | | |          ||__*__||
                   /  |  \        |~ \___/ ~|
                   ~~~~~~~        /=\     /=\
_____________________(_)(__\_____[_]_____[_]_____________________
8

                        ,===
                       (@o o@
                      / \_-/
                     /| |) )        ___
                    |  \ \/__      /() \
                    |   \____@=|  |_____|_
                    |   |        |_|  o  |_|
                    | | |         ||  o  ||
                    | | |         ||__*__||
                   /  |  \       |~ \___/ ~|
                   ~~~~~~~       /=\     /=\
_____________________(_)(__\_____[_]_____[_]_____________________
```

La pantalla tendrá información adicional a la de la película, con el siguiente formato:

```
// ==================== Screen usage (80Colsx24Rows):=======================
// 0123456789d123456789v123456789t123456789c123456789L123456789s123456789F123456789
//00
//01
//02
//03
//04     +-------------------------------------------------------------+
//..     |                    P.I.C.T.U.R.E                            |
//..     |                                                             |
//22 Input field number or new frame time                              |
//23Foot G Uribe C V03G7 Art: Simon Jansen Inp. # to JUMP; '=#': Time. 32109|
```

Aquí se indica el lugar que ocupará la proyección (picture, líneas 5 a 21, columnas 6 a 73), y dos líneas debajo de todo, donde se agrega información como el número del frame (32109), y algún recordatorio de cómo puede el usuario situarse en un frame en particular (jump) o cambiar el tiempo de exposición de los frames. Por defecto se tomará un tiempo de 67 milisegundos por frame, equivalentes a 15 frames/segundo. (Las películas en televisión tienen 24 frames por segundo, y en el cine son de 30)

A continuación la representación de las **Políticas** de este programa (lo que hay que hacer); *mírenlo bien*:

```
// ascmat2.c, AsciiMation, Luis G. Uribe C., L30L7
// ...Politics & Mechanisms Separation
// ==================================================================
// This file has ONLY ***POLITICS*** (WHAT will be done);
// ..There are  NO  implied Mechanisms (or How things will be done).
// In some places this is called pseudo code, but we have arranged to
// ..show you one 'C' implementation, using Include Files and MACROS

#include "ascmat2.h"   // Defines and implements almost ALL Semantics
                       // ..except for second and deeper levels inside
// ******************************************************************
// Main Level Politics:

DefineGlobals();

void main( int argc, char **argv )
{
   DefineLocals();

   OpenInputDataFile();
   InitTimers();
   PrintStaticFooter();

Restart:
   InitProcess( xy, frame );
   DoUntilEOF() {
       GetRepeatFactor();
       FillFrameBuffer();
       DrawFrames( WithRepeatFactor );
       PrintVariableFooter();
       CatchNew( Time, Frame, RestartOnError );
   }
}
```

Primero figura: Identificación (`ascmat2.c`), autor (`Luis G Uribe`), fecha (`L30L7`), tema, comentarios (Políticas de primer nivel, ningún Mecanismo). A veces se llama *pseudo código*; pero **es código en C**.

Luego está `#include "ascmat2.h"` donde se implementa casi toda la semántica (se le da "significado" al pseudo código; no es "toda" porque la parte de graficación y temporización se tratan aparte)

Después va el primer nivel de Política. Define variables globales (`DefineGlobals();`); todo programa puede tenerlas; son visibles por varias rutinas, como los buffers; aquí se ordena su definición; obsérvese, sin embargo, que ésta, propiamente dicha, no aparece aún, pues forma parte de los Mecanismos…

A continuación comienza main (`void main( int argc, char **argv ) {`). Se definen variables locales (`DefineLocals();`); es más que probable que todo programa tenga variables locales, invisibles para las demás rutinas, como contadores internos, por ejemplo, y se indica que aquí han de definirse; sin embargo, la materialización en realidad no aparece aún, pues forma también parte de los Mecanismos…

Después de los prolegómenos, que pueden ser idénticos en la mayoría de los programas, vienen las instrucciones más específicas del programa, comenzando por la instrucción de que se abra el archivo de datos de entrada (`OpenInputDataFile();`). Al profundizar un poco más veremos que esta sección trata de localizar un archivo de nombre determinado (`asciimat.txt`) y en caso de no encontrarlo, pregunta al usuario por otro nombre.

Luego se inicializa el subsistema de temporización, que habrá de marcar el paso de 67 milisegundos por frame (`InitTimers();`). De más está recalcar que aquí no tenemos ni idea de cómo vamos a medir este tiempo, ni nos interesa; esto forma parte de los Mecanismos.

Al final se imprime la parte estática de la pantalla (`PrintStaticFooter();`), según se definió antes.

Ahora sí viene el proceso iterativo. Fíjense que hasta aquí, nadie pensaría que esto es **C**, ¡pero sí lo es! Algún indicio de que así es lo tenemos con main y la llave (`void main(int argc, char **argv) {`), los puntos y comas, los paréntesis…

Lo que va a hacerse a continuación es: Inicializar los procesos (`InitProcess( xy, frame );`); por ejemplo, debe colocarse el cursor en la posición de la línea de entrada donde el usuario puede escribir para escoger un frame en especial, o cambiar el tiempo de exposición de los cuadros, y hasta que se termine el archivo (`DoUntilEOF() {`) se va a repetir el proceso de: Leer cuántas veces hay que mostrar el siguiente frame (`GetRepeatFactor();`), leer la información del mismo (`FillFrameBuffer();`), dibujarlo y esperar según el factor de repetición (`DrawFrames( WithRepeatFactor );`), actualizar la variable del pie de la pantalla (`PrintVariableFooter();`) y ver si el usuario desea realizar algún cambio por teclado (`CatchNew( Time, Frame, RestartOnError );`). Note que en caso de que se ordene mostrar un frame inexistente, la política consistirá en rearrancar el proceso (`Restart:`). Errores de temporización (tiempos muy pequeños, por ejemplo), se tratarán automáticamente, sin molestar al usuario, escogiendo valores por defecto para subsanar esas situaciones. **¿Vieron qué fácil es?**

No solo parece sencillo, sino que es muy conveniente. Supongamos que escogemos una cierta forma de medir el tiempo, como por ejemplo, ejecutar inmensos loops de instrucciones, a fin de perder los milisegundos necesarios. Si luego se desea cambiar este mecanismo, porque se va a correr el programa en máquinas más rápidas y los viejos loops ya no satisfacen los retrasos, el programa sigue inmutable. Mientras la interfaz con la parte de Mecanismos (las rutinas que definen el API: Application Program Interface, sus parámetros) mantengan la consistencia, nada hay que cambiarle al código.

Analicemos ahora el `#include "ascmat2.h"`:

```
// ascmat2.h, AsciiMation, Luis G. Uribe C., L30L7 V03G7 C05S7
// ...Politics & Mechanisms Separation
// This file has ONLY **MECHANISMS** (HOW Politics will be done)
// ..(Politics is What will be done)
/* ------------------------------------------------------------ */
/* Include Project Specific Files */
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "rtc.h"
extern volatile unsigned int timers[ 8 ];

#include "video.h"

// ==============================================================
// Parameter definitions taken from Jansen's site
#define NLINES      13      // 13 Lines per frame
#define NCOLUMNS    74      // 74 Columns per line in frame
#define TIME        67      // 67 mseconds per frame (15 frm/sec)

// --------------------------------------------------------------
// Other constants
#define BUFLINES    25      // 25 Lines per frame
#define BUFCOLUMNS  80      // 80 Columns per line
#define LINESIZE    80      // 80 chars, Line Size

#define MAXREPEAT   69      // 69, as seen inside input file...
#define BEGINLINE    4      //  4
#define BEGINCOL     6      //  6

#define INPUTLINE   22      // 22
#define INPUTCOL     0      //  0
#define FOOTERLINE  23      // 23

// --------------------------------------------------------------
// MACROS definitions (ancillary)
#define DoUntilEOF()        while( 1 )
#define TryNextFrame()      continue

#define AdjustEOL()                                              \
   buffLen = strlen( buff[ i ] );                                \
   if( buff[ i ][ buffLen ? buffLen - 1 : 0 ] ==  '\n' ) {       \
       buff[ i ][ buffLen ? buffLen - 1 : 0 ] =   '\0';          \
   }                                                             \
   if( buff[ i ][ buffLen ? buffLen - 1 : 0 ] ==  '\r' ) {       \
       buff[ i ][ buffLen ? buffLen - 1 : 0 ]  = '\0';           \
   }                                                             \
   buffLen = strlen( buff[ i ] );                                \


// ------------------------------
// Second Level Politics:
```

```
// Init timers, change attribute used by cls(), and do cls()

#define    InitTimers()                                          \
                                                                 \
   IniTimers();                                                  \
   clsatt = FG( BOLD | BROWN ) | BG( BLUE ); /*cls attribute*/\
   cls();                                                        \

// ==============================================================
// Global Variables definitions
#define DefineGlobals()                                          \
                                                                 \
char buff[BUFLINES ][ BUFCOLUMNS]; /* screen buf. Uses 17,74*/\
char line[LINESIZE];   /* buffer to get frame number */         \
extern int clsatt; /* att. used by cls(); user modifiable d*/ \
// --------------------------------------------------------------
// Local Variables definitions
#define DefineLocals()                                           \
                                                                 \
FILE *fp;                /* input file pointer          */       \
int  i;                  /* variable used in loops      */       \
int  buffLen;            /* Buffer Lenght               */       \
int  repeat;             /* number to repeat each frame*/       \
int  n;                  /* times repeated each frame   */       \
int  r, c;               /* (r)ow and (c)olumn          */       \
long f;                  /* frames                      */       \
long frame = 0;          /* frame counter               */       \
long nframe;             /* new frame to jump           */       \
int  time  = TIME;                                               \
char *file = "asciimat.txt";   /* default input file name */  \
// ------------------------------------------------------------
// Open input data file
#define    OpenInputDataFile()                                   \
                                                                 \
   if( *argv[ 1 ] ) {                                            \
       file = argv[ 1 ];                                         \
   }                                                             \
   if( ! ( fp = fopen( file, "r" ) ) ) {                         \
       printf("Oooops... Input file '%s' not present\n",file);\
       exit(0);                                                  \
   }                                                             \
// ------------------------------------------------------------
#define PrintStaticFooter() /*line 23, diff. cols on screen */\
                                                                 \
  PrintSC(23, 0,"Prog: ", BOLD | GREEN | BG(BLUE) );             \
  PrintSC(23, 6,"G Uribe C V03G7 ", BOLD | BROWN | BG(BLUE) );\
  PrintSC(23,22,"Art:", BOLD | GREEN | BG(BLUE) );               \
  PrintSC(23,26,"Simon Jansen ", BOLD | WHITE | BG(BLUE) );      \
  PrintSC(23,39,"Inp. # to JUMP; '=#': Time. ",                  \
          WHITE|BG(BLUE));                                       \
// ------------------------------------------------------------
#define InitProcess(xy, frame)                                   \
                                                                 \
   gotoxy(INPUTLINE,INPUTCOL );     /* bios function */         \
```

```
   frame = 0;                          /* init frame counter */  \
// ---------------------------------------------------------------
#define GetRepeatFactor()                                          \
                                                                   \
   if( ! fgets( buff[ 0 ], BUFCOLUMNS, fp ) ) {                    \
       exit( 0 );          /* EOF. exit() triggers clean-up */    \
   }                       /* ..See rtc_int_dsb() in rtc.c */     \
   repeat = atoi( buff[ 0 ] );                                     \
   if( repeat <= 0 || repeat > MAXREPEAT ) { /* error */           \
       TryNextFrame();         /* ...skip */                       \
   }                                                               \
// ---------------------------------------------------------------
#define FillFrameBuffer()                                          \
                                                                   \
   for( i = 0; i < NLINES; i ++ ) {                                \
       if( ! fgets( buff[ i ], BUFCOLUMNS, fp ) ) {                \
           exit( 0 );    /* EOF; exit() triggers clean-up */      \
       }                                                           \
       AdjustEOL();                                                \
                                                                   \
      /* ------------------------------------------------- */      \
      /* fill end of buffer with blanks   */                       \
       for( c = buffLen; c < BUFCOLUMNS; c ++ ) {                  \
           buff[ i ][ c ] = ' ';                                   \
       }                                                           \
   }                                                               \
// ---------------------------------------------------------------
#define    DrawFrames( WithRepeatFactor )                          \
                                                                   \
   for( n = 0; n < repeat; n ++ ) {                                \
       SetimerMS( 0, time );                                       \
                                                                   \
       /* ------------------------------------------------- */     \
       /*Draw ONE frame */                                         \
       /*First, move picture BEGINLINE (4) lines down */           \
                                                                   \
       for( i = 0; i < BEGINLINE; i ++ ) {                         \
           LineFill( i, ' ', BOLD | BROWN | BG( BLUE ) );          \
       }                                                           \
       /*------------------------------------------------- */      \
       /*Draw now */                                               \
                                                                   \
       for( r = 0; r < NLINES; r ++ ) {                            \
           /*---------------------------------------- */           \
           /*1st move picture BEGINCOL (6) spaces from left*/      \
           for( c = 0; c < BEGINCOL; c ++ ) {                      \
               CharFillXY(r,c, ' ',BOLD | BROWN | BG(BLUE));       \

           }                                                       \
           for( c = 0; c < BUFCOLUMNS; c ++ ) {/*draw frame*/      \
               CharFillXY(r+BEGINLINE,c+BEGINCOL,buff[r][c],       \
                   BOLD | BROWN | BG( BLUE ) );                    \
           }                                                       \
       }                                                           \
```

```
       /* ------------------------------------------------ */      \
       /* dinamic footer */                                        \
                                                                   \
        WTimer( 0 );                                               \
                                                                   \
   }   /* Endfor( n = 0; n < repeat; n ++ ) */                     \
// ---------------------------------------------------------------
#define    PrintVariableFooter()                                   \
                                                                   \
sprintf( line, "Frame:%7ld", ++frame );                            \
PrintSC( FOOTERLINE, 67, line, WHITE | BG( BLUE ) );               \
// ---------------------------------------------------------------
#define    CatchNew( Time, Frame, RestartOnError )                 \
                                                                   \
   /*jump to any frame; write the frame number and <ENTER>*/       \
   /* NOTE: Frames NOT adjusted by repeating factor to KISS*/      \
                                                                   \
   if( kbhit() ) {                                                 \
       gets( line );              /* get new frame number */       \
       PrintSC( INPUTLINE, INPUTCOL,                               \
              "              ", /*erase input data*/               \
           BOLD | BROWN | BG( BLUE ) );                            \
       if(! isdigit(line[0] ) ) {  /* time begins with '=' */      \
           time = atoi( &line[ 1 ] );  /* get new time */          \
       } else {                                                    \
           nframe = atoi( line ); /* get frame nmbr */             \
           if( nframe < frame ) {                                  \
               fseek( fp, 0, SEEK_SET ); /* reposition file*/      \
               frame = 0;                                          \
           }                                                       \
           for( f = frame; f < nframe; f ++ ) {/*read new */       \
               for( i = 0; i < NLINES; i ++ ) {/* ..frame */        \
                   if(! fgets(buff[i], BUFCOLUMNS, fp ) ) {        \
                       fseek(fp,0,SEEK_SET);/*restart on EOF*/\
                       goto Restart; /*sorry for using goto */\
                   }                                               \
                   AdjustEOL();                                    \
               }                                                   \
           }                                                       \
           frame = nframe;                                         \
       }   /* EndElse if( ! isdigit( line[ 0 ] ) )  */             \
       gotoxy( INPUTLINE, INPUTCOL );                              \
   } /* Endif( kbhit() )  */                                       \
```

De nuevo, el archivo comienza con una sección de identificación (ascmat2.h, AsciiMation), autor (Luis G. Uribe C.), fecha (C05S7) y propósito (Politics & Mechanisms Separation), include Project Specific Files, entre las que resaltan: #include "rtc.h" y "video.h", que definen parte de las funcionalidades para los temporizadores y el manejador de la pantalla.

A continuación vienen algunas definiciones de parámetros para caracterizar las dimensiones de la proyección en la pantalla, y algunas otras para establecer el tamaño del buffer de pantalla (char buff[BUFLINES][BUFCOLUMNS];), cuál es el máximo valor esperado para el factor de repetición de un mismo frame (#define MAXREPEAT 69), en qué parte comienza la sección útil de la pantalla (#define BEGINLINE 4, BEGINCOL 6), dónde va a estar la línea que el usuario emplea para dialogar con el programa (#define INPUTLINE 22, INPUTCOL 0), y la línea donde se presentará la información del pié de pantalla (#define FOOTERLINE 23).

Luego está la sección que define las primeras partes de los Mecanismos; comienza con MACROS definitions, en donde están las principales, identificadas de forma que su nombre indique con precisión la funcionalidad: DoUntilEOF(), TryNextFrame(), AdjustEOL() (adecúa el End Of Line. Note que esta AdjustEOL() sirve para manipular *sólo* un buffer de nombre "buff"; podría rescribirse de manera genérica, para cualquier buffer).

Después viene una sección en donde se implementan la inicialización de los InitTimers(), que a su vez llama la función IniTimers();, la implementación de DefineGlobals(), donde destacan el buffer de pantalla (char buff[BUFLINES][BUFCOLUMNS]) y el de línea (char line[LINESIZE]).

A continuación las variables locales de main (DefineLocals()), con el indicador de archivos (FILE *fp;), una variable utilitaria para el control de lazos de repetición (int i;), la longitud del buffer (int buffLen;), el factor de repetición del frame (int repeat;), las veces que ya lleva repetido un frame (int n;), las variables para memorizar la fila y columna (int r, c;), la cantidad de frames (long frame;), el número del frame que se desea comandar por teclado (long nframe;), una variable auxiliar para recorrer los frames (long f;), cuánto es el tiempo para el frame (int time = TIME;), inicializado en TIME para comenzar y el nombre del archivo que se toma por defecto (char *file = "asciimat.txt";).

Luego se incluyen las materializaciones de OpenInputDataFile(), que leer el nombre del archivo desde la línea de comandos (file = argv[1];), en caso de que el usuario quiera emplear uno diferente, y también el manejo de error por si no se encuentre el archivo indicado (printf(...); y exit(0);)

La rutina de PrintStaticFooter() se basa en funciones primitivas (PrintSC()) definidas en #include "video.h" arriba; luego vienen InitProcess(xy,frame), FillFrameBuffer() y GetRepeatFactor(); este último incluye TryNextFrame(), que como ven: #define TryNextFrame() continue, es un alias por "continue"; es decir, presupone que éste código estará encerrado dentro de un while() o for(). Podría rescribirse para que devuelva un valor, según el cual pueda llamarse en forma condicionada, el "continue", a fin de hacerlo explícito. Sin embargo se escribió así, tratando de sobreenfatizar el tema de la separación de Políticas y Mecanismos, para que fueran lo menos visibles dentro de la Política decisiones como ésta. Ciertamente que la reusabilidad de TryNextFrame() es cuestionable...

`DrawFrames( WithRepeatFactor )` invoca primitivas del subsistema temporizador:
(`SetimerMS( 0, time );` y `WTimer( 0 );`),
y también las del subsistema graficador:
(`LineFill( i, ' ', BOLD | BROWN | BG( BLUE ) );` y
`CharFillXY(r,c, ' ',BOLD | BROWN | BG(BLUE));`).

Vienen de último `PrintVariableFooter()`, y `CatchNew( Time, Frame, RestartOnError )`.

Revise con cuidado el código que viene a continuación; verá que no es difícil. Algunas realizaciones no se emplean por lo común; están colocadas aquí para **resaltar** al máximo la separación entre Políticas y Mecanismos, pero en C no resultan siempre tan nítidas de escribir como en otros lenguajes, probablemente orientados a objeto como el C++, C# o Java.

A continuación analizaremos los timers y el subsistema de despliegue, `"rtc"` y `"video"`.

Varias técnicas son indispensables para el trabajo en tiempo real. De ellas, el manejo de temporizadores, que le permiten al programa llevar varios flujos de secuencias en el tiempo, son de las más importantes.

La primera aproximación tiene como objetivo la sencillez; se basa en la función **_bios_timeofday**, que devuelve el número de **ticks** ocurridos desde que se encendió la máquina o desde que se reinicializó la función del BIOS. Un **tick** es un número de 32 bits (long) que representa cuentas de 54.92 ms/tick (milisegundos/tick). La siguiente es una secuencia típica:

```
long t2;
_bios_timeofday( _TIME_GETCLOCK, &t2 );
```

La variable t2, cuya dirección se pasa en la llamada (&t2), recibe el número de ticks. Normalmente uno se construye una **batería de temporizadores** por software, que consisten en *variables* en las cuales usted registrará el tiempo futuro en el que debe ocurrir algún evento. Una cantidad común es de 8 temporizadores, que son un arreglo de "**long**s", algo así como:

```
volatile long timersMS[ 8 ];// timers en MS (miliseconds)
```

pero usted puede seleccionar la cantidad que necesite.

Se programan las siguientes funciones de Soporte: **SetimerMS, Timeout, WTimer y WaitMS_on**.

SetimerMS: Recibe como parámetros, el número del temporizador que usted quiere activar (del 0 al 7) y la cantidad de **milisegundos (MS)** que a futuro deberán indicarle a usted que ha expirado ese lapso. Note que el valor que debe almacenar corresponde al tiempo actual (devuelto por la llamada al BIOS), **más** el número de milisegundos deseados. Tenga la precaución de convertir a milisegundos el número de ticks que devuelve el BIOS (sabiendo que son 54.92 ms/tick)

Timeout: Retorna verdadero o falso ("**1**" o "**0**"), dependiendo de si el tiempo actual (que se lo proporciona la función del BIOS) ha sobrepasado (**1**), o no (**0**), el tiempo registrado dentro del

temporizador que se le pasa como parámetro, del 0 al 7. No olvide que para comparar el valor de los ticks con el del timer, tiene que convertir los ticks a milisegundos, que es lo que está almacenado en los timers.

Wtimer: Es una función en la que usted especifica el número del temporizador deseado, y ésta espera en un lazo, sin devolver el control al programa principal, hasta que **Timout** le indica que ya se venció el lapso registrado en el temporizador. Este tipo de funciones, en las que el procesador se bloquea esperando por la ocurrencia de un evento, desperdician el tiempo de la máquina pero, en ***ocasiones***, en realidad el programa no puede continuar haciendo nada hasta que ese evento suceda...

WaitMS_on: Esta es una función compuesta, que recibe como parámetros el número del temporizador sobre el cual usted tiene que esperar sin que el programa continue más allá, y el número de milisegundos que desea. Esta es una llamada a: SetimerMS y Wtimer, la que a su vez llama a Timeout.

Miremos el código, que está dividido en dos archivos, `rtc.h` y `rtc.h`:

```
/* ************************************************************ */
// rtc.h, bios time usage Luis G. Uribe C. L13M6 L1606  C05S7

/* ------------------------------------------------------------ */
// Include Files

#include <bios.h>
#include <conio.h>

#define NTIMERS    8   // This MUST be a power of 2: 2, 4, 8, 16...

/* ------------------------------------------------------------ */
// Function Prototypes

void IniTimers( void );
void SetimerMS(  int timer, int timeMS );
int  Timeout( int timer );
void WTimer( int timer );
void WaitMS_on(  int timer, int timeMS );
```

```
/* ************************************************************ */
// rtc.c, bios time usage Luis G. Uribe C. L13M6 L1606

/* ------------------------------------------------------------ */
// Include Files

#include "rtc.h"

/* ------------------------------------------------------------ */
// Global Variables


volatile long timersMS[ NTIMERS ];

/* ============================================================ */
void SetimerMS (int timer, int timeMS )                    /*()*/
{

   long t2;

   _bios_timeofday( _TIME_GETCLOCK, &t2 );

   timersMS[ timer & ( NTIMERS - 1 ) ] =
      (( t2 * 5492 ) / 100) + (long)timeMS;   // 54.92 mSec/tick

}  /* SetimerMS() */

/* ------------------------------------------------------------ */
int Timeout (int timer)                                    /*()*/
{

   long t2;

   kbhit();                          // Enable Ctrl-C testing!
   _bios_timeofday( _TIME_GETCLOCK, &t2 );
   if( timersMS[timer & (NTIMERS - 1) ] >= (( t2 * 5492 ) / 100 ))
       return( 0 );
   return( 1 );

}  /* Timeout() */

/* ------------------------------------------------------------ */
void WTimer (int timer)                                    /*()*/
{

   while( ! Timeout( timer ) )
       ;

}  /* WTimer() */


/* ------------------------------------------------------------ */
void WaitMS_on (int timer, int timeMS)                     /*()*/
{

   SetimerMS( timer, timeMS );
   WTimer( timer );

}  /* WaitMS_on() */

/* ------------------------------------------------------------ */
void IniTimers (void)                                      /*()*/
{
       /* Do nothing in this implementation of timers */

}  /* IniTimers() */
```

De acuerdo a lo anterior, el subsistema temporizador ha sido diseñado con las siguientes funcionalidades: ➤ Debe poder inicializar los distintos timers (`void IniTimers( void );`), ➤ posicionar un cierto timer (tomado de un grupo de `#define NTIMERS 8`) en cualquier valor, en milisegundos (`void SetimerMS( int timer, int timeMS );`), ➤ debe poder investigar si cierto timer ya expiró (`int Timeout( int timer );`), ➤ suspender la operación del programa hasta que un timer expire (`void WTimer( int timer );`) ➤ y una operación compuesta, que inicializa un timer y suspende la operación hasta que éste termine (`void WaitMS_on( int timer, int timeMS );`).

Los 8 timers están representados por: `volatile long timersMS[NTIMERS];` esta forma de exponer las funcionalidades establece una interfaz con el susbsistema (**API**), del cuál no se ven mayores interioridades; es decir, son las Políticas; luego vendrán los Mecanismos, susceptibles de cambiar.

Como ya se advirtió, se ha escogido para comenzar una implementación muy elemental; por ejemplo, el `SetimerMS(int timer, inttimeMS)` funciona de la siguiente manera: Se lee la hora actual (`_bios_timeofday( _TIME_GETCLOCK, &t2 );`), se la convierte a milisegundos aplicando la fórmula `((t2*5492)/100)`, se le suma el número de milisegundos estipulado (`+ (long)timeMS;`), y se lo almacena en el temporizador requerido: `timersMS[ timer & ( NTIMERS - 1 ) ] =`.

Verificar si un temporizador expiró (`int Timeout (int timer)`) consiste simplemente en comparar la hora actual con la hora almacenada para el timer, todo convertido a milisegundos: `if( timersMS[ timer & (NTIMERS-1)] >= ((t2 * 5492)/100))`; retorna 1 si expiró, 0 de lo contrario.

Esperar hasta que un temporizador expire (`void WTimer ( int timer )`) es muy sencillo:

```
while( ! Timeout( timer ) ) /* empty while */ ;
```

Y la funcionalidad compuesta: `void WaitMS_on ( int timer, int timeMS )`, consiste en `SetimerMS( timer, timeMS ); WTimer( timer );`

En esta implementación, `void IniTimers ( void )` no desempeña ningún papel.

Resulta que, si bien es muy sencillo el subsistema de temporizadores tal como aquí se muestra, tiene algunos compromisos: El tiempo mínimo, en un PC, está alrededor de 54.92 mSec/tick, por lo que podría resultar conveniente reemplazar el Mecanismo por uno basado en interrupciones del Real Time Clock (**rtc**), con lo que se puede llevar la granularidad a un (1) milisegundo. Otras arquitecturas de estaciones de trabajo, como el Macintosh, u otros sistemas operativos, como Linux, no tienen exactamente ésta misma funcionalidad del `_bios_timeofday( _TIME_GETCLOCK, &t2 );` del DOS, por lo que habrá que modificar los *Mecanismos* para adecuarlos, en caso de necesidad.

Incluyo ***sólo para ilustrar el tema***, las no tan simples rutinas del Real Time Clock por interrupciones en el PC, con exactamente la misma interfaz anterior, pero con un **Mecanismo muy diferente**; maneja las interrupciones directamente (en las rutinas anteriores, el **bios** manejaba las interrupciones, pero la granularidad no es tan buena, como ya se dijo). Verá que el código es mucho más complejo, con mayor nivel de detalle, de acuerdo al **chip** que se maneja. Ensaye reemplazar los anteriores `rtc.h` y `rtc.c` por éstos nuevos; **verá cómo todo funciona a la perfección, ¡sin tener que hacer ningún otro cambio!**

```
/* rtc.h: Luis G. Uribe C., S29N2003 v22O4 J27E5 M12D6 C05S7    */

/* External Variables                                            */

void    far  *old_rtc_vec;
volatile unsigned long ticks;
volatile unsigned int timers[ 8 ];

/* Macros definitions                                            */
#define  WTimer(t)      { while( timers[ t ] ) kbhit() ; }
#define  Timeout(t)     ( timers[ t ] ? 0 : !0 )

/* Function Prototypes                                           */
void IniTimers( void );
void WaitMS_on( unsigned int t, unsigned int time );
void SetimerMS( unsigned int t, unsigned int time );

void interrupt far rtc_int( void ),  rtc_int_dsb( int signal );
void rtc_int_dsb2( void );
void fin ( int sig );

/* Include Files */
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

/*  RTC Addresses, Vector & Registers A, B and C:               */
#define RTC_add     0X70
#define RTC_data    0X71
#define RTC_vect    0X70

/* Status Register A;   cfr. prof. Luis Alvarado: RTC.ASM, 1999 */
#define RTC_SRA 0x0A          /* Status Register A (read/write) */
#define UIP     ( 1<<7 ) /* Update in Progress. ouputs undefined */
#define Khz_8  0x03 /* 0011b: 122.070 uSec (2^15/2^2 = 2^13 Hz) */
#define Khz_def 0x06 /* 0110b: 976.562 uSec (2^15/2^5 = 2^10 Hz)*/

// Bit 6,5,4-22 stage divider. 010b - 32.768 Khz time base (default)
//  Bit 3-0 - Rate selection bits for interrupt.
//            0000b - none
//            0001b -
//            0010b -
//  Khz_8:    0011b -    122.070 uSec (2^15/2^ 2 = 2^13 Hz)
//            0100b -    244.140 uSec (2^15/2^ 3 = 2^12 Hz)
//            0101b -    488.281 uSec (2^15/2^ 4 = 2^11 Hz)
//  Khz_def:  0110b -    976.562 uSec (2^15/2^ 5 = 2^10 Hz) default
//            0111b -   1953.123 uSec (2^15/2^ 6 = 2^ 9 Hz)
//            1000b -   3906.247 uSec (2^15/2^ 7 = 2^ 8 Hz)
//            1001b -   7812.495 uSec (2^15/2^ 8 = 2^ 7 Hz)
//            1010b -  15625     uSec (2^15/2^ 9 = 2^ 6 Hz)
//            1011b -  31250     uSec (2^15/2^10 = 2^ 5 Hz)
//            1100b -  62500     uSec (2^15/2^11 = 2^ 4 Hz)
//            1101b - 125000     uSec (2^15/2^12 = 2^ 3 Hz)
//            1110b - 250000     uSec (2^15/2^13 = 2^ 2 Hz)
//            1111b - 500000     uSec (2^15/2^14 = 2^ 1 Hz)

/* Status Register B                                             */
#define RTC_SRB 0x0B          /* Status Register B (read/write) */
#define RTC_ALL_INTS    0x8F  /* all possible interrupt sources */
#define RTC_PERIODIC_INT   (1<<6)

//  Bit 7 - 1 aborts update cycle, 0 enables
//  Bit 6 - 1 enables periodic interrupt
//  Bit 5 - 1 enables alarm interrupt
//  Bit 4 - 1 enables update-ended interrupt
//  Bit 3 - 1 enables square wave output
//  Bit 2 -   Data Mode                - 0: BCD, 1: Binary
//  Bit 1 -   24/12 hour selection     - 1 enables 24 hour mode
//  Bit 0 -   Daylight Savings Enable - 1 enables

/* Status Register C                                             */
#define RTC_SRC 0x0c          /* Status Register C (Read only) */

// 0Ch Status Register C (Read only)
//  Bit 7 - Interrupt request flag-1 when any or all of bits 6-4 are
//          1 and appropriate enables (Register B) are set to 1.
//          Generates IRQ 8 when triggered.
//  Bit 6 - Periodic Interrupt flag
//  Bit 5 - Alarm Interrupt flag
//  Bit 4 - Update-Ended Interrupt Flag
//  Bit 3-0 ???

/*  PIC Addresses and RTC related interrupt enable bits          */
#define RTC_PIC_INTEN  ( ~ (1<<0) )   /* enable mask in PIC2 */
#define RTC2_ENABLE    ( ~ (1<<2) )   /* enable PIC2 in PIC1 */

#define EOI_PORT1  0x20
#define EOI_PORT2  0xA0
#define EOI        0x20

#define PIC1_Mask  0x21
#define PIC2_Mask  0xA1

#define sti    _enable()
#define cli    _disable()

#pragma intrinsic( _enable, _disable )
#pragma intrinsic( inp, outp )

/* Include Files */
#include <signal.h>
/* rtc.c: Luis G. Uribe C., S29N2003 v22O4 J27E5 C13D6 C05S7   */

/* ------------------------------------------------------------ */
/* Include Files */
```

```
#include "rtc.h"

/* ============================================================ */
void IniTimers ( void )                                    /*()*/
{

   volatile unsigned int   tmp;
   int i;

   /* ---------------------------------------------------------- */
   // RTC Setup.El ORDEN de estas operaciones hay que mantenerlo!

   old_rtc_vec = _dos_getvect( RTC_vect );      // SAVE OLD VEC
   signal ( SIGINT, fin );                      /* CONTROL C */
   atexit( rtc_int_dsb2 );

   /* ---------------------------------------------------------- */
   // Programar nueva velocidad

   cli;                                 // RTC *needs* CLI
   outp( RTC_add, RTC_SRA );
   tmp = ( inp( RTC_data ) & 0xF0 ) | Khz_8;

   outp( RTC_add, RTC_SRA );
   outp( RTC_data, tmp );

   /* ---------------------------------------------------------- */
   // Colocar nuevo versión, con nuestra rutina de interrupciones

   _dos_setvect( RTC_vect, rtc_int );  // Set new RTC Vector
   cli;                                 // DOS makes STI. Redo CLI

   /* ---------------------------------------------------------- */
   // Programar Interrupt periódico en RTC

   outp( RTC_add, RTC_SRB );
   tmp = ( inp( RTC_data ) & RTC_ALL_INTS ) | RTC_PERIODIC_INT;

   outp( RTC_add, RTC_SRB );
   outp( RTC_data, tmp );

   /* ---------------------------------------------------------- */
   // Habilitar que el RTC interrumpa por el segundo PIC;
   // ..habilitar que el segundo PIC entre al primer PIC

   outp( PIC2_Mask, inp( PIC2_Mask ) & RTC_PIC_INTEN );
   outp( PIC1_Mask, inp( PIC1_Mask ) & RTC2_ENABLE );
   /* ---------------------------------------------------------- */
   // Limpiar cualquier posible interrupt pendiente

   outp( RTC_add, RTC_SRC );
   inp( RTC_data );            /* dummy read, clear any pending */

   /* ---------------------------------------------------------- */
   // Inicializar todos los timers en 0

   ticks = 0L;                         /* init values */

   for( i = 0; i < 8; i ++ )
       timers[ i ] = 0;                /* init values */

   /* ---------------------------------------------------------- */
   // Habilitar interrupciones globales del CPU

   sti;

}  /* IniTimers() */

/* ============================================================ */
void interrupt far rtc_int ( void )                        /*()*/
{      /* rtc_int() */      // Nueva rutina de interrupcion
   int i;

   /* ---------------------------------------------------------- */
   // Reconocer (borrar) el interrupt que acaba de generarse en RTC

   outp( RTC_add, RTC_SRC );
   inp( RTC_data );              /* clear any pending interrupts */

   /* ---------------------------------------------------------- */
   // Incrementar 'ticks' (LONG)

   ticks ++;
   /* ---------------------------------------------------------- */
   // Procesar 8 siguientes timers (INT). Si ya están en cero, no
   // ..se hace nada con ellos; de lo contrario, se los decrementa

   for( i = 0; i < 8; i ++ )
       if( timers[ i ] )             // skip if timer is zero already
           timers[ i ] --;
   /* ---------------------------------------------------------- */
   // Reconocer el interrupt en los dos PICs. El orden de estas
   // ..operaciones es siempre importante

   outp( EOI_PORT2, EOI );
   outp( EOI_PORT1, EOI );

}  /* rtc_int() */ // RTI: Retorno de interrupt: CLEAR CPU IREQ
```

```
/* ---------------------------------------------------------- */
void rtc_int_dsb ( int signal )                          /*()*/
{    /* com_int_dis() */     // invoked by control-c, and at end

   volatile unsigned int   tmp;
   static done = 0;

   if( done )  return;

   /* ---------------------------------------------------------- */
   // RTC necesita CLI para su programación. Se deja la velocidad
   // ..estandar: Khz_def

   cli;
   outp( RTC_add, RTC_SRA );
   tmp = ( inp( RTC_data ) & 0xF0 ) | Khz_def;

   outp( RTC_add, RTC_SRA );
   outp( RTC_data, tmp );

   /* ---------------------------------------------------------- */
   // Se recupera el vector original. CLI porque DOS retorna con STI

   _dos_setvect( RTC_vect, old_rtc_vec );
   cli;

   /* ---------------------------------------------------------- */
   // Se deshabilitan las interrupciones del RTC en el PIC2
   // ..Sería preferible haber guardado el valor original de los
   // ..PICs al empezar, y restaurar sus valores aquí

   outp( PIC2_Mask, inp( PIC2_Mask ) & ~ RTC_PIC_INTEN );
   sti;


   /* ---------------------------------------------------------- */
   // Se imprimen los ticks finales y se hace call exit

   getchar();
   fprintf( stderr, "\n\nFinal TICKs: %lu\n", ticks );
   done++;
   if( signal == SIGINT )
       exit( signal );

}   /* rtc_int_dis() */

/* ---------------------------------------------------------- */
void rtc_int_dsb2 ( void )                                 /*()*/
{      /* com_int_dis2() */   // Ancillary routine, invoked atexit
                              // ..cause cannot have any parameters
   rtc_int_dsb( 0 );

}   /* rtc_int_dis2() */

/* ---------------------------------------------------------- */
// WaitMS_on() implements a delay using RTC.C

void WaitMS_on (unsigned int t, unsigned int time) //milliS/*()*/
{

   SetimerMS( t, time );
   WTimer( t );

}   /* WaitMS_on() */

/* ---------------------------------------------------------- */
// SetimerMS() implements a delay using RTC.C

void SetimerMS (unsigned int t,unsigned int time ) //milliS/*()*/
{

   timers[t] = (int)( ((long)time * 1000) / 122);// 122 uSec/tick

}   /* Setimer() */

/* ---------------------------------------------------------- */
void fin ( int sig )                                       /*()*/
{

    exit( 0 );

}   /* fin() */
```

Por último, revisemos el subsistema de despliegue de información:

```
/* video.h, Luis G. Uribe C., J04Y06, video defines              */

/* ------------------------------------------------------------ */
/* Include Files */

#include <conio.h>      // for getch()

/* ------------------------------------------------------------ */
/* Macros and Typedefs */

#if !defined uchar
   typedef unsigned char uchar;
#endif
#if !defined uint
   typedef unsigned int  uint;
#endif

#define FG( c )     (  (c) & 0x0F )              // ForeGround
#define BG( c )     ( ((c) & 0x0F ) << 4 )       // BackGround

// -----------------
// Fill one possition with 'ch' char and 'att' attributte,
// .. at 'poss' possition (See how to build and use a Macro)
// Note: CharFill receives poss, uint far *; cast it to uchar

#define CharFill(pos,ch,att ) *((uchar far *)(pos))      = (ch), \
                              *(((uchar far *)(pos))+1) = (att);\
                               kbhit()

#define CharGet( pos )        *((uint far *)(pos))

/* ------------------------------------------------------------ */
/* Function Prototypes */

void CharFillXY( int row, int column, int ch, int att );
void cls( void );           // Clear Screen
void LineFill( int row, int ch, int att );
void PrintLines( int Att );
void PrintSC(  int row, int column, char * string, int att );
void PrintSCr( int row, int column, char * string, int att );
void ScreenFill( int FillCh, int Att );
```

```
void ScreenFill( int ch, int att );

int CharGetXY( int row, int col );

void gotoxy( int x, int y);

/* ------------------------------------------------------------ */
/* Some Defines and Enums */

#define VIDEO  (int far *)0xB8000000 // Base Address for Video VGA

// Standard Mode (3): 80 columns * 25 lines

#define NCOLS  80        // Number of Columns (chars/line) 0..79
#define NROWS  24 //25   // Number of Rows (Lines) 0..24

// The screen is seen as a vector of integers (2 bytes)
// .. The first (even) byte is the Character to be displayed
// .. The second (odd) byte is the Attribute of the character
// Attribute Bits 2, 1, 0: Foreground color
// Attribute Bits 6, 5, 4: Background color,
// .. with same meaning as Foreground Colors

enum colors {        // both for Foreground & B_ackground
 // normal colors:
   BLACK,  BLUE,  GREEN,  CYAN,  RED,   MAGENTA,   BROWN,   WHITE,
 // BOLD   colors:
   B_BLACK,B_BLUE,B_GREEN,B_CYAN,B_RED, B_MAGENTA, B_BROWN, B_WHITE,
 // Special color ID:
   YELLOW = B_BROWN,
};

// Attribute's Bit 3 (decimal 8) is BOLD, so adding '8' to colors
// .. redefine them as BOLD chars (i.e., BOLD BROWN is Yellow)

#define BOLD  ( 1 << 3 )   // Bold: Attribute Bit 3 (decimal 128)

// Attribute's Bit 7 (decimal 128) is Foreground Blink

#define BLINK ( 1 << 7 )   // Attribute Bit 7 is Foreground Blink
```

```
/* video.c, Luis G. Uribe C., J04Y06, C13D6 video functions    */

/* ------------------------------------------------------------ */
/* Include Files */
#include "video.h"
/* ------------------------------------------------------------ */
void CharFillXY ( int row, int col, int cht, int atr )     /*()*/
```

```
{
   int far * pvideo = VIDEO;
   int pos = ( row * NCOLS ) + col;  // video RAM lineal position

   if( ( pos >= 0) && ( pos < NROWS * NCOLS ) ) { // OK
       pvideo += pos;           // else: print at begining of RAM
   }
   CharFill( pvideo, cht, atr );
}

/* ------------------------------------------------------------ */
void LineFill ( int row, int ch, int att )                    /*()*/
{
   int col;

   for(col = 0; col < NCOLS; col ++ ) {
       CharFillXY( row, col, ch, att );
   }
}

/* ------------------------------------------------------------ */
void ScreenFill( int ch, int att )                            /*()*/
{
   int row;

   for( row = 0; row < NROWS; row ++ ) {
       LineFill( row, ch, att );
   }
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int clschar = ' ';          // global variables for cls character
int clsatt  = FG( WHITE ); // ..and attribute

/* ------------------------------------------------------------ */
void cls (void)                                               /*()*/
{
   ScreenFill( clschar, clsatt );
}

/* ------------------------------------------------------------ */
void PrintSC ( int row, int col, char * string, int att )  /*()*/
{
   while( *string ) {
       CharFillXY( row, col, *string, att );
       string ++;
       col ++;
   }

}
/* ------------------------------------------------------------ */
void PrintSCr ( int row, int col, char * string, int att ) /*()*/
{
   while( *string ) {
       CharFillXY( row, col, *string, att );
       string ++;
       col --;
   }

}

/* ============================================================ */
int CharGetXY ( int row, int col )                            /*()*/
{
   int pos = ( row * NCOLS ) + col; // video RAM lineal position
   int far * pvideo = VIDEO + pos;

   if( ( pos >= 0) && ( pos < NROWS * NCOLS ) ) { // OK
       return CharGet( pvideo );
   }
   return '\0';

}

/* ------------------------------------------------------------ */
#include <dos.h>

void   gotoxy( int x, int y )                                 /*()*/
{
   typedef union REGS Regs;

   Regs regs;

   regs.h.dl = (unsigned char)y;        // BIOS video driver
   regs.h.dh = (unsigned char)x;        // .. int 10h, funt. 02h
   regs.h.bh = 0;                       // .. positions the cursor
   regs.h.ah = 0x2;
   int86( 0x10, &regs, &regs );

}      /* gotoxy() */
```

El manejo de la pantalla de video sirve bien para estudiar la clase denominada "**memory-mapped I/O**" en su forma más elemental (sin handshaking). Además, toda presentación de información que emplee atributos como colores de las letras y del fondo, parpadeo, sobre intensidad, etc., necesitan la aplicación de los procedimientos que aquí se presentan.

Se define primero el módulo elemental: **CharFill(poss, ch, att)**, macro que escribe un símbolo determinado (**ch**) en una posición lineal, entera, (**poss**) y con los atributos definidos en **att**:

```
#define CharFill(poss,ch,att ) {*((char far *)(poss))     = (ch),    \
                                *(((char far *)(poss))+1)=(att);}
```

Se añaden a continuación las definiciones de soporte necesarias, tales como la dirección base de la memoria de video, los nombres para los 8 colores, en base a estas definiciones principales, se genera las funciones de Soporte:

CharFillXY (row, column, ch, att) que presente la ilusión de que la pantalla está estructurada en filas y columnas, lo que es más cómodo de manejar que CharFill, dependiendo del caso. Se incluyen decisiones inteligentes que confinan el almacenamiento al área de memoria asignado para el video, aún si los parámetros de entrada, row y column, tienen valores ilegítimos (números negativos, o que produzcan direcciones de memoria más allá del área de video).

LineFill() llena con una letra y un atributo, escogidos por el usuario, una línea determinada.

ScreenFill(), con parámetros apropiados que usted deberá señalar, explicando sus razones. El propósito aquí es el de llenar la pantalla completa con un símbolo y su atributo, escogidos por el usuario.

cls() (clear screen), que debe borrar la pantalla, rellenándola de espacios, con características de letra blanca en fondo negro, que es lo convencional. Esta rutina no lleva parámetros, pero usted debe definirla de tal manera que el usuario pueda cambiar el símbolo y los atributos que **cls()** emplea. Indique cual será su aproximación para ofrecer esta facilidad.

PrintSC (row, column, string, att) imprime una cadena de caracteres "string" a partir de la posición definida por row y column, aplicando a todos los caracteres el mismo atributo att. Esta función maneja bien el posible cambio de fila, en caso de que el texto sobresalga de la línea actual en la pantalla.

En el código (**video.h**) se encuentran macros auxiliares para manipular el ForeGround `(#define FG(c) ((c)&0x0F))` y el BackGround `(#define BG(c) (((c)&0x0F)<<4))`. A continuación vienen las macros de más bajo nivel, `CharFill(pos,ch,att)` para llenar una posición con un carácter y su atributo, y `CharGet(pos)` para *leer* de la memoria de video una cierta posición. Estas dos macros direccionan las posiciones en la memoria de video, considerándolas como compuestas por dos (2) bytes: Uno para la letra y otro para el atributo.

Después vienen los prototipos de las diversas funciones, la posición de la memoria de video (`#define VIDEO (int far *)0xB8000000`), el tamaño de la pantalla, en columnas y filas (en este caso, algunos

dispositivos tienen 24 líneas, entre tanto que otros tienen 25; se emplean 24 para ser compatibles con ambos)

Se definen los colores asignando símbolos a los números que entiende el hardware, y dos símbolos auxiliares, `BOLD (1<<3)` y `BLINK (1<<7)`.

En `video.c` se incorporan las rutinas de mayor nivel, soportadas por las de más baja jerarquía, ya referidas. `void CharFillXY(int row,int col,int cht,int atr)`, recibe el número de una fila, una columna, el símbolo que se quiere representar y su atributo; convierte fila y columna a la posición lineal entera que la macro `CharFill()` entiende, y la llama.

`LineFill(int row,int ch,int att)` llama repetidas veces la rutina anterior (`CharFillXY()`); `ScreenFill(int ch,int att)` llama a `CharFillXY()`; `void cls()` se basa en `ScreenFill()`, y `void PrintSC(int row,int col,char *string,int att)` requiere a `CharFillXY()`.

Una nota final para advertir que la implementación de un sistema no siempre se hace Top-Down; a veces se comienza, como en el caso de video.c, desde las primitivas más elementales, y se va construyendo el sistema hacia arriba (Down-Top)

Recuerden que la magia que sirve para reemplazar un sistema tan sencillo como el de los temporizadores originales, por el que maneja directamente, por interrupciones, el chip del timer del PC, estriba en la **Separación de Políticas y Mecanismos, y la definición y aseguramiento de la Interfaz (API)**.

REALLY SIMPLY HASH

En la programación de sistemas embebidos, pero también en las estaciones de trabajo y servidores, es recurrente la necesidad de almacenar y recuperar (store and retrieval) información mediante algún tipo de clave. Ya hemos hablado de la necesidad de proteger estos códigos cuando deben ser manipulados a mano por el usuario, mediante dígitos de verificación. Hay claves seriales, que se forman en secuencia numérica, como los números que se asignan para manejar los turnos en algunos sistemas de colas. Otras claves se componen mediante dígitos (y a veces letras) que agregan información al código; así, los primeros dos números pueden identificar el suplidor, los otros dos el modelo, uno más la ubicación, etc., según las necesidades del usuario.

Lo cierto es que muchas veces sucede que el número de claves es muy superior a la cantidad de artículos que en realidad hay que identificar. Por ejemplo, una de mis cuentas de ahorro tiene 20 dígitos, con lo cual se podrían asignar 100 millones de millones de millones de cuentas (100 trillones). Aún si descartáramos un dígito de verificación y 4 o 5 que pueden ser internacionales, cien billones de posibilidades en la asignación de cuentas bancarias parecen exceder cualquier escenario comercial.

Otro ejemplo corriente es cuando se usa el nombre para buscar datos asociados en una tabla. Mi hijo mayor, Alejandro Uribe (sin incluir más nombres ni el segundo apellido), alcanza las 13 letras. Si se emplearan todas las 26 letras en la conformación de nombres, podrían identificarse 2.5 trillones de personas. Aún si las combinaciones son mucho menores, la cantidad es muy elevada.

Parece evidente entonces que en tales casos el número de identificación no puede usarse directamente como índice para tener acceso a la información. La gente de computación trabaja este tema bajo el nombre de table loockup; el algoritmo se llama una búsqueda por "hash", en donde el nombre o el código se convierten en un número entero pequeño, que es el que en realidad se emplea como índice.

El siguiente es una modificación del recomendado por K&R, válido para cadenas de caracteres; la denomino **RSHash** (**R**eally **S**imple Hash) por la moda actual de simplificar viejos protocolos de red, y bautizarlos así, como el RSS: Really Simple Sindication):

```
/* ------------------------------------------------------------ */
unsigned int hash( char *s )                                /*()*/
{
   unsigned int hashval;

   for( hashval = 0; *s != '\0'; s++ )
       hashval = (*s - 'A' ) + (31 * hashval);
   return hashval % HASHSIZE;
}
```

(K&R usan: `hashval = *s + 31 * hashval;`)

```
                                                                #if 0
hash.c, Luis G. Uribe C., L20J2005. Prueba de función hash M24L7
                                                                #endif
/* ------------------------------------------------------------ */
// Include files

#include <stdio.h>

/* ------------------------------------------------------------ */
// Function defintions

unsigned int hash( char *s );

#define HASHSIZE 101

/* ------------------------------------------------------------ */
// Global Variables. May change table contens w/different strings

char *tabla[] = {
   "Abboud",        "Almansor",      "Alzamora",      "Anda",
   "Anzola",        "Azuaje",        "Benadivas",     "Bentolila",
   "Castillo",      "Chang",         "Correia",       "De Majo",
   "De Sousa",      "Garcia",        "Gonzalez",      "Issa",
   "Jauregui",      "Jimenez",       "Liendo",        "Lopez",
   "Ma",            "Machado",       "Machado",       "Marquez",
   "Mata",          "Mencia",        "Mendez",        "Perez",
   "Pignatari",     "Rangel",        "Rosario",       "Tremont",
   "Wasylkowski",   "Zambrano",
};


#define CHECKSIZE 256  // HASHSIZE 101, may grow up to 256 (SAssert)
char check[ CHECKSIZE ];   // global: linker init to 0's


/* ============================================================ */
void main( void )                                          /*()*/
{
   int i, j, TotalSample, Doubles, decimals, integers, fraction;
   unsigned int HIndex;

   TotalSample = sizeof( tabla ) / sizeof( char * );

   // ------------------------------------------------------------
   // load table using hash function as an index

   for( i = 0; i < TotalSample; i++ ) {
       HIndex = hash( tabla[ i ] );
       check[ HIndex ] ++ ;
   }

   // ------------------------------------------------------------
   // print table

   printf( "Total sample: %d; HashSize: %d\n\n",
       TotalSample, HASHSIZE );

   for( Doubles = i = 0; i < HASHSIZE; i ++ ) {
       // don't verify all CHECKSIZE elements; just HASHSIZE
       if( check[ i ] ) {
           if( check[ i ] > 1 ) {
               Doubles ++;
           }
           printf( "%3d ", i );
           for( j = 0; j < check[ i ]; j ++ ) {
               putchar( '*' );
           }
           putchar( '\n' );
       }
   }

   // ------------------------------
   // calculate % with one decimal, not using floating point

   // ---------------
   // Ancillary variables
   decimals = Doubles * 100 * 10 / TotalSample;
   integers = decimals / 10;

   fraction = decimals - integers * 10;
   printf( "\nUsed %d, Doubles %d, Merit Figure "
           "(Used/TotalSample%; should aproach 100%%): %d.%1d%%\n",
       TotalSample - Doubles /*=Used*/, Doubles,
       (TotalSample- Doubles) * 100 / TotalSample, fraction);

}


/* ************************************************************ */
unsigned int hash( char *s )                               /*()*/
{
   unsigned int hashval;

   for( hashval = 0; *s != '\0'; s++ ) {
       hashval = (*s - 'A' ) + (31 * hashval);
   }
   return hashval % HASHSIZE;
}
```

La estadística del programa anterior arroja los siguientes resultados con los datos allí presentes:

```
Used 28, Doubles 6, Merit Figure (Used/TotalSample; should aproach 100%): 82.6%
```

Los "dobles" indican que hubo 6 localidades de memoria por la que compitieron dos nombres; en ese caso hay que establecer una rutina de búsqueda secuencial (cfr. K&R). Pude haber "triples" y más.

Es conveniente que HASHSIZE (101 en el ejemplo) sea un número primo.

SASSERT

Es muy importante verificar ciertas suposiciones en los programas; el C provee desde siempre un mecanismo, la macro assert(), para pruebas dinámicas, es decir, cuando el programa está ejecutando. Se encuentra definido en **<assert.h>**, y puede tener el siguiente aspecto:

```
/* assert.h: assert macro      */

#if !defined(NDEBUG)
  #define assert(p) if(!(p)){fprintf(stderr,     \
      "Assertion failed: %s, file %s, line %d\n",\
      #p, __FILE__, __LINE__); abort(); }
#else
  #define assert(p)
#endif
```

La ventaja del mecanismo es que uno compila su programa con las verificaciones activas y cuando ya lo ha depurado no hay que reeditarlo para quitárselas; basta con incluir definir el identificador NDEBUG. Recuerde que esto puede hacerlo mediante un #define, o también incluyendo un "/D" en la línea de comando al invocar al compilador (**cc /D<u>NDEBUG</u>**), así que es todo muy simple.

Lo que quisiéramos es poder extender este tipo de verificaciones también al momento de compilar (**Static asserts**), para verificar si ciertas condiciones se cumplen, por ejemplo, que algunas constantes tengan determinados valores (tamaños de buffers, por ejemplo), o que sean potencias de 2 (a veces se aprovecha la expresión K-1 para obtener una máscara llena de 1's, la cual se aplica a los apuntadores o subíndices que operan ciertos buffers, para lograr, por ejemplo, el efecto de **<u>redondeo</u>**: **Wrap around**. Pero ésto solo funciona si K es potencia de 2).

Igualmente se puede verificar que la máquina para la que se compila el programa tenga enteros de 2 bytes, o de 4; que sus chars sean signed, en fin, una cadena de suposiciones que, de llegar a fallar, harían que nuestro programa no corriera.

Se presenta a continuación una forma excelente pero desconocida de lograr este propósito, simplemente incluyendo un #define con **Sassert** (Static assert). Se incluye un programa de ejemplo:

```
#define  Sassert(e)  do{ enum { Sassert__ = 1/(e) }; } while (0)

// if Sassert fails print: "error C2057: expected constant expression"
// Sassert (static assert) macro does NOT generate run time code

void main( void )
{
   getchar();          // filling instructions

   // ---------------------------------------------------------------
   // EXAMPLE: Ensure 'char' is a signed type
   //
   // Note that this Static Assert, Sassert, will run fine on MSVC
   // ..(Microsoft Visual C), but will fail miserably on CODEWARRIOR
   // ..This is due to the fact that CodeWarrior uses UNSIGNED chars
   // ..by default. The last ANSI standar C enforecs chars to be
   // ..SIGNED by default but people on Metrowerks did not pay attention.

   Sassert( (char)-1 == -1 );

   puts( "Hola" );     // filling instructions


}
```

Al correr este programa en MSVC, compila bien; bajo Codewarrior compila mal, produciendo el error anunciado: **"expected constant expression"**.

El mensaje no es exactamente lo que uno quisiera, pero esto es lo más que puede estirarse este truco.

ROUNDS UP TO A POWER OF 2

A veces hay algoritmos para manejar por ejemplo buffers circulares, que se simplifican sobremanera si sus tamaños son potencias de dos. Miremos una hipotética rutina de encolar: A la izquierda, genérica con buffers que pueden tener **cualquier** tamaño; a la derecha para tamaños de buffer **potencias de 2**:

```
#define BUFSIZE 13 //NOT a 2's power


   if( n >= size ) return EOF;
   buf[put++] = (uchar)c; //put incr
   put = put >= BUFSIZE ?
      0 : put;  //wrap around
   n ++;
   return c & 0xFF;
```

```
#define BUFSIZE   16  // 2's power
#define BUFMASK   (BUFSIZE-1)

   if( n >= size ) return EOF;
   buf[put++] = (uchar)c; //put incr
   put &= BUFMASK; //wrap around

   n ++;
   return c & 0xFF;
```

En la rutina del lado izquierdo, si hay espacio se encola el símbolo **y se actualiza (incrementa) el pointer**: `*q->put ++ = (uchar) c;`, si este incremento sobrepasa el final del buffer, se le ajusta el valor al pointer, con la dirección de la base del buffer (wrap around), de lo contrario se lo deja así: `q->put = q->put >= q->limit ? q->base : q->put;`. Esta operación requiere una comparación, y si es el caso, un movimiento de datos.

Ahora miremos al lado derecho, cómo sería la misma rutina si el tamaño del buffer fuera potencia de 2. Las partes en que **difieren** las dos rutinas se muestra en **negrita**. Se define el tamaño del buffer, `#define BUFSIZE 16`, que tiene que ser potencia de 2. Se define la máscara, que contiene unos (1's) en todos los bits del tamaño del BUFZISE (4 bits): `#define BUFMASK (BUFSIZE-1)`; como `BUFSIZE es 16`, `BUFMASK`, que vale: `BUFSIZE-1` será 15, que son 4 unos: `1111`.

Ahora, corregir el apuntador se convirtió en: **`q->put &= BUFMASK;`**, es decir, se cambia el apuntador, por su valor en AND con `1111 (0xF)`. Así, mientras **put** toma los valores del 0 al 15, esos mismos, en AND con **`0xF`**, siguen produciendo valores del 0 al 15; cuando **put** llegue a 16, el AND lo convierte en 0, y se repite la secuencia de 0 a 15. El AND con `0xF` confina el resultado al rango entre 0 y 15. (NOTA: Hay un caso aún más particular, cuando la potencia de dos forma frontera con un byte [256] o con un entero [32768]; en ese caso, si las operaciones se realizan en char o int, el "wrap around" es automático al incrementar de la última posición [255 o 32767], pues el resultado de esa operación es cero]

Esto está perfecto, si se usan tamaños de buffers (en nuestro ejemplo) que sean potencias de dos puede emplearse rutinas especialmente ajustadas para que resulten muy eficientes, pero suele ocurrir que quien escribe las rutinas (de encolar y desencolar en nuestro caso y, en general, las rutinas que empleen este último método para acelerar el proceso de actualizar los índices de acceso a los arreglos), no es el mismo usuario o programador del resto del código. ¿Cómo garantiza el programador del paquete de colas, que el usuario de sus rutinas no escogerá un número `BUFSIZE` que **no** sea potencia de 2? Porque es claro que su método sólo funciona para potencias de dos.

Emplear el **Sassert** de la sección anterior sería correcto pero ¿cuál expresión, en C, indica si un número es, o no, potencia de dos? Ese es el propósito del siguiente desarrollo.

Comenzaremos viendo una forma dinámica, por programa, que se calcula en "run time", a diferencia de la forma estática, por macros, que se calcula en "compile time", y que hacia donde luego iremos. Sabemos que 'nbits', el número de bits de un 'number', es (parteEntera)log2(number)+1; supondremos también que el **cero** necesita al menos un bit; de esta manera, 0 bits puede usarse para marcar **error**. Así, **'number'** (global) comienza en uno. 'number' también es global (variable de entrada)

```
void calculo ( int i )                                    /*()*/
{
   if( i >= number )   return;
   nbits ++;
   i <<= 1;
   i ++;
   calculo( i );
}
```

Un ejemplo para ver cómo funciona el algoritmo:

```c
// q#2-R.c, Luis G. Uribe C. J30G7
// Answer to Quiz #2, Arquitectura USB, Jun2007
// This algorithm gets Log2N. Log2N+1 is the number of bits
// ..needed to represent the number (assume '0' needs 1 bit)

/* ------------------------------------------------------------ */
// Include Files

#include <stdio.h>

/* ------------------------------------------------------------ */
// Function Prototypes and typedefs

typedef unsigned long  ulong;
typedef unsigned char  uchar;

char *xtobinf( ulong d, int nbits, ulong mark );
void calculo( int i );

/* ------------------------------------------------------------ */
// Macros & defines

#define NBITS(n)      ( sizeof(n) * 8 <= 32 ? sizeof(n) * 8 : 32 )
// MARK defines where to insert spaces, in a 32 bits string
#define    MARK (ulong)(       (1L<<28) | (1L<<24) | (1L<<20)| \
                    (1L<<16) | (1L<<12) | (1L<< 8) | (1L<< 4))
/* ------------------------------------------------------------ */
// Global Variables

int  nbits;
int  number;

/* ************************************************************ */
void main ( void )                                          /*()*/
{
   int i;

   for( i = 0; i < 256; i ++ ) {
       nbits  = 1;
       number = i;
       calculo( 1 );
       printf( "number = %s (%3d,0x%02X); nbits = %1d\n",
                xtobinf( (uchar)i, NBITS( (uchar)i ), MARK ),
                i, i, nbits );
   }
}
```

```c
/* ------------------------------------------------------------ */
void calculo ( int i )                                      /*()*/
{
   if( i >= number )
       return;
   nbits ++;
   i <<= 1;
   i ++;
   calculo( i );
}

/* ============================================================ */
typedef unsigned long  ulong;

#define    SPACES  7   // number of spaces in a 32 bits string
#define    N      32   // max # of bits that xtobinf may process
#define    FS     ' '  // field separator; space by default

char *xtobinf ( ulong d, int nbits, ulong mark )            /*()*/
{  /* xtobinf() */

   /* binf.c 7-Oct-84, takes a ***LONG*** (d) and makes string with
    * ..x ascii 0's and 1's, space separated. Return string address
    * xtobin, 14-may-88 (UXS.LIB)
    * xtobinf get nbits & MARK from parameter list L-August-20-2007
    */

   // ---------------------------------------------------------
   // Local Variables

   static char buf[ N + 1 + SPACES ];  // to fit 32 bits
   char *bp = buf;
   ulong  msk;

   // ---------------------------------------------------------
   // Code

   nbits = nbits <= N ? nbits : N;      // up to 32 bits
   for( msk = 1uL << (nbits - 1);  msk;  msk >>= 1 ) {
       *bp++ = (char)( ( d & msk )  !=  0 ) + '0';
       if( msk & mark ) {
           *bp++ = FS;
       }
   }
   *bp = '\0';
   return  buf;

}  /* xtobinf() */
```

El programa evalúa el número de bits necesarios para representar los primeros 256 números, comenzando en cero (`for( i = 0; i < 256; i ++ ) {}`). (NOTA: ¿Recuerdan el problema que vimos con anterioridad, que fallaba en un for como éste? Aquí no falla, porque la variable de control, **i**, es un entero)

Inicializa el número de bits en uno (`nbits = 1;`), inicializa la variable global `number = i;` (este ejercicio es la solución, en **C**, a un quiz en Assembler del HC08, de la USB, por eso las variables globales…) y llama nuestra rutina (`calculo( 1 );`), con parámetro inicial "1". Luego escribe el número y los bits que se necesitan para representarlo:

```
numero = 0000 0000 (  0,0x00); nbits = 1
numero = 0000 0001 (  1,0x01); nbits = 1
numero = 0000 0010 (  2,0x02); nbits = 2
numero = 0000 0011 (  3,0x03); nbits = 2
numero = 0000 0100 (  4,0x04); nbits = 3
numero = 0000 0101 (  5,0x05); nbits = 3
numero = 0000 0110 (  6,0x06); nbits = 3
numero = 0000 0111 (  7,0x07); nbits = 3
numero = 0000 1000 (  8,0x08); nbits = 4
numero = 0000 1001 (  9,0x09); nbits = 4
numero = 0000 1010 ( 10,0x0A); nbits = 4
numero = 0000 1011 ( 11,0x0B); nbits = 4
numero = 0000 1100 ( 12,0x0C); nbits = 4
numero = 0000 1101 ( 13,0x0D); nbits = 4
numero = 0000 1110 ( 14,0x0E); nbits = 4
numero = 0000 1111 ( 15,0x0F); nbits = 4
numero = 0001 0000 ( 16,0x10); nbits = 5
numero = 0001 0001 ( 17,0x11); nbits = 5
numero = 0001 0010 ( 18,0x12); nbits = 5
numero = 0001 0011 ( 19,0x13); nbits = 5
numero = 0001 0100 ( 20,0x14); nbits = 5
numero = 0001 0101 ( 21,0x15); nbits = 5
numero = 0001 0110 ( 22,0x16); nbits = 5
numero = 0001 0111 ( 23,0x17); nbits = 5
numero = 0001 1000 ( 24,0x18); nbits = 5
numero = 0001 1001 ( 25,0x19); nbits = 5
```

```
numero = 0001 1010 ( 26,0x1A); nbits = 5
numero = 0001 1011 ( 27,0x1B); nbits = 5
numero = 0001 1100 ( 28,0x1C); nbits = 5
numero = 0001 1101 ( 29,0x1D); nbits = 5
numero = 0001 1110 ( 30,0x1E); nbits = 5
numero = 0001 1111 ( 31,0x1F); nbits = 5
numero = 0010 0000 ( 32,0x20); nbits = 6
numero = 0010 0001 ( 33,0x21); nbits = 6

...

numero = 0011 1110 ( 62,0x3E); nbits = 6
numero = 0011 1111 ( 63,0x3F); nbits = 6
numero = 0100 0000 ( 64,0x40); nbits = 7
numero = 0100 0001 ( 65,0x41); nbits = 7

...

numero = 0111 1110 (126,0x7E); nbits = 7
numero = 0111 1111 (127,0x7F); nbits = 7
numero = 1000 0000 (128,0x80); nbits = 8
numero = 1000 0001 (129,0x81); nbits = 8
...
numero = 1111 1110 (254,0xFE); nbits = 8
numero = 1111 1111 (255,0xFF); nbits = 8
```

Lo que nos interesa por el momento no es el número de bits necesarios para representar un número, sino calcular **la potencia de dos que más se le aproxima**. Modificando un poco el programa anterior (ya no es recursivo y tiene más condiciones), vemos que el siguiente procedimiento genera el resultado que queremos. Se ha separado el programa en un header (`q2.h`) y el main (`q2.c`). `q2.h` es bastante estándar, incluye el número de bits (`#define NBITS(n) ( sizeof(n) * 8 <= 32 ? sizeof(n)*8 : 32)`) y la máscara que identifica los espacios separadores (`#define MARK (ulong)((1L<<28)|(1L<<24)| (1L<<20)|(1L<<16)|(1L<<12)|(1L<<8)|(1L<<4))`) para la función `*xtobinf();`, encargada de imprimir un entero en el equivalente binario (los espacios están en la posición 4, 8, 12, 16, 20, 24 y 28, y pueden cambiarse con solo modificar `MARK`.

```
// q2.h, Luis G. Uribe C. J30G7

// Include Files

#include <conio.h>
#include <stdio.h>

/* ------------------------------------------------------------ */
// Function Prototypes and typedefs

typedef unsigned long  ulong;
typedef unsigned char  uchar;

char *xtobinf( ulong d, int nbits, ulong mark );

/* ---------------------- for xtobinf() ---------------------- */
// MACROS & DEFINES

#define NBITS(n)      ( sizeof(n) * 8 <= 32 ? sizeof(n) * 8 : 32 )

// MARK defines where to insert spaces, in a 32 bits string
#define    MARK (ulong)(       (1L<<28) | (1L<<24) | (1L<<20) | \
                    (1L<<16) | (1L<<12) | (1L<< 8) | (1L<< 4) )
```

```
// q2.c, Luis G. Uribe C. L03S7

// Algorithm to print 'nbits', ammount of bits needed to represent
// ..a 'number'; round up 'nbits' to 'exp', least power of 2 to
// ..hold the 'number' (lp2), and (int)Log2 of the number too.
//
// NOTE: 'nbits', the # of bits of a 'number' == log2(number)+1;
// ..So, (integer part of)log2(number) == 'nbits'-1
//
//ASSUMPTION: Need 1 bit at list to represent #0; others use 0 bits

/* ------------------------------------------------------------ */
// Include Files & DEFINES

#include "q2.h"

#define BITS     8   // number of bits of numbers to be evaluated
```

```
/* ************************************************************ */
void main ( void )                                          /*()*/
{
   int i, nbits, number, exp;

   for(number = 0; number < 1<<BITS /* i.e. 2^BITS */; number++) {
       if( number > (1<<(BITS))/2 ) {  // 129...255, for BITS == 8
           nbits = exp = 0;    // Error: need > BITS to represent
       }
       else if( number == 0 || number == 1 )
           nbits = exp = 1;
       else {
       //---- Algorithm to get 'nbits' to represent 'number' ----
           nbits  = i = 1; //'0'needs one bit; else use: nbits=i=0
           while( i < number ) {
               nbits ++;
               i <<= 1;  i |= 1;   // LSL, right introducing 1
           }  // endwhile            LSL: Logical Shift Left
       }  // endif

   //----- ROUND UP 'NBITS' TO NEXT (LEAST) POWER OF 2 (lp2) ----
   // ASSUMPTION: 0 rounds up to 1; i.e.: it is needed one bit at
   // ..least to represent 0 (others make 0 round up to 0)
   // - If number is already 2's power use it; else use next number.
   //   2, for example, has 2 bits; 1<<(2-1) is 2; so use 2.
   //   3 also has 2 bits; 1<<(2-1) != 3, use 1<<(2):4 (next to 3)
   //
   // ASSUMPTIONS: log2(0) == 0; Represent ERROR_LOG as -1.

       if( nbits > 1 ) {
           if( ( exp = 1<<nbits-1 ) != number ) {
               exp = 1<<(nbits);
           }
       }
       printf( "number = %s (%3d,0x%02X); nbits = %3d, "
               "exp_lp2 = %3d, Log2 = %3d\n",
                xtobinf((uchar)number, NBITS((uchar)number ),MARK),
                number, number, nbits, exp, nbits ? nbits - 1 : 0);
   }   // endfor
}


/* ============================================================ */
char *xtobinf ( ulong d, int nbits, ulong mark )            /*()*/
{
   /* xtobinf() */

   // SEE THIS ROUTINE IN THE PRECEEDING PROGRAM, ABOVE

}  /* xtobinf() */
```

El programa imprime el 'number' (binario, decimal y hexadecimal: =0000 0000(0,0x00)), 'nbits', la próxima potencia de dos, 'exp_lp2' y el Log2, para los primeros 256 números, comenzando en 0 (`#define BITS 8`). Supone que se necesita al menos un bit para representar el número cero (otros autores asumen que el cero necesita cero bits).

El 'for': `for( number = 0; number < 1<<BITS /* i.e. 2^BITS */; number++ ) {` es estándar; una curiosidad es la forma como representa el límite superior (`number < 1<<BITS`, o 2^8).

Anticipa que los números mayores a la mitad del rango no pueden representarse con el número de bits (`BITS`) indicado: `if( number > (1<<(BITS))/2 ) { nbits = exp = 0;`. También maneja de manera separada (*fuera del for*) el número '0' y el '1' (`else if( N==0 || N==1 ) nbits = exp = 1;`).

La parte genérica comienza obteniendo primero los `'nbits' necesarios para representar el 'number'`, suponiendo un mínimo de un bit (`nbits = i = 1;`), y aplica el mismo algoritmo que vimos en el programa anterior, en la página de arriba, sólo que no emplea recursión:

```
while( i < number ) { nbits ++; i <<= 1;  i |= 1;   // logical shift left, LSL, right introducing 1.
```

Luego se obtiene el redondeo (`ROUND UP 'NBITS' TO NEXT (LEAST) POWER OF 2 (lp2)`). Las suposiciones son: 0 redondea a 1; si un número ya es potencia de 2, se lo usa, de lo contrario se usa la siguiente potencia de dos. Por ejemplo, 2 tiene 2 bits, así que `1<<(2-1) is 2;` por tanto se usa el **2**. En cambio, el 3 también tiene 3 bits, pero como `1<<(2-1) es distinto de 3`, se usa 4. La última suposición está con respecto a: `log2(0) == 0;`.

Todo lo recientemente dicho, tomando el 2 y el 3 como ejemplos, puede expresarse así:

```
if( nbits > 1 ) {
    if( ( exp = 1<<nbits-1 ) != number ) {
        exp = 1<<(nbits);
    }
}
```

```
N=0000 0000(  0,0x00);bits=1,lp2=  1,Log2=0
N=0000 0001(  1,0x01);bits=1,lp2=  1,Log2=0
N=0000 0010(  2,0x02);bits=2,lp2=  2,Log2=1
N=0000 0011(  3,0x03);bits=2,lp2=  4,Log2=1
N=0000 0100(  4,0x04);bits=3,lp2=  4,Log2=2
...
N=0000 0111(  7,0x07);bits=3,lp2=  8,Log2=2
N=0000 1000(  8,0x08);bits=4,lp2=  8,Log2=3
N=0000 1001(  9,0x09);bits=4,lp2= 16,Log2=3
...
N=0000 1111( 15,0x0F);bits=4,lp2= 16,Log2=3
N=0001 0000( 16,0x10);bits=5,lp2= 16,Log2=4
N=0001 0001( 17,0x11);bits=5,lp2= 32,Log2=4
...
N=0001 1111( 31,0x1F);bits=5,lp2= 32,Log2=4
```

```
N=0010 0000( 32,0x20);bits=6,lp2= 32,Log2=5
N=0010 0001( 33,0x21);bits=6,lp2= 64,Log2=5
...
N=0011 1111( 63,0x3F);bits=6,lp2= 64,Log2=5
N=0100 0000( 64,0x40);bits=7,lp2= 64,Log2=6
N=0100 0001( 65,0x41);bits=7,lp2=128,Log2=6
...
N=0111 1111(127,0x7F);bits=7,lp2=128,Log2=6
N=1000 0000(128,0x80);bits=8,lp2=128,Log2=7
N=1000 0001(129,0x81);bits=0,lp2=  0,Log2=0
N=1000 0010(130,0x82);bits=0,lp2=  0,Log2=0
...
N=1111 1110(254,0xFE);bits=0,lp2=  0,Log2=0
N=1111 1111(255,0xFF);bits=0,lp2=  0,Log2=0
...
```

Si lo único que pretendemos es obtener la potencia de dos que más se le aproxima a un número, podemos simplificar el programa anterior y dedicarnos sólo a eso. La rutina principal puede escribirse de otra forma. A diferencia de los ejemplos que le anteceden, éste supone que cero no necesita bits para representarse (se representa con cero bits). Aplica, además, hasta **unsigned long**, así:

```
/* ------------------------------------------------------------ */
/* Round up to a power of 2. */

ulong lp2 ( ulong x )
{
   --x;
   x |= x >> 1;
   x |= x >> 2;
   x |= x >> 4;
   x |= x >> 8;
   x |= x >>16;
   return ++x;
}
```

Para ver cómo funciona, hice un ejemplo sólo de 4 bits (>>1 y >>2 únicamente):

```
init: 0000 ( 0,0x0)
--x:  1111 (15,0xF)
>>1:  1111 (15,0xF)
>>2:  1111 (15,0xF)
++x   0000 ( 0,0x0)
====================
init: 0001 ( 1,0x1)
--x:  0000 ( 0,0x0)
>>1:  0000 ( 0,0x0)
>>2:  0000 ( 0,0x0)
++x   0001 ( 1,0x1)
====================
init: 0010 ( 2,0x2)
--x:  0001 ( 1,0x1)
>>1:  0001 ( 1,0x1)
>>2:  0001 ( 1,0x1)
++x   0010 ( 2,0x2)
====================
init: 0011 ( 3,0x3)
--x:  0010 ( 2,0x2)
>>1:  0011 ( 3,0x3)
>>2:  0011 ( 3,0x3)
++x   0100 ( 4,0x4)
====================
init: 0100 ( 4,0x4)
--x:  0011 ( 3,0x3)
>>1:  0011 ( 3,0x3)
>>2:  0011 ( 3,0x3)
++x   0100 ( 4,0x4)
====================
init: 0101 ( 5,0x5)
--x:  0100 ( 4,0x4)
>>1:  0110 ( 6,0x6)
>>2:  0111 ( 7,0x7)
++x   1000 ( 8,0x8)
====================
init: 0110 ( 6,0x6)
--x:  0101 ( 5,0x5)
>>1:  0111 ( 7,0x7)
>>2:  0111 ( 7,0x7)
++x   1000 ( 8,0x8)
====================
init: 0111 ( 7,0x7)
--x:  0110 ( 6,0x6)
>>1:  0111 ( 7,0x7)
>>2:  0111 ( 7,0x7)
++x   1000 ( 8,0x8)
====================
init: 1000 ( 8,0x8)
--x:  0111 ( 7,0x7)
>>1:  0111 ( 7,0x7)
>>2:  0111 ( 7,0x7)
++x   1000 ( 8,0x8)
====================
Error for x = 0x9: need more than4 bits
```

Todo lo anterior está muy bien, cuando se desea calcular los valores mencionados de manera programática (en run time), pero para un **static assert** no sirve. Lo que se necesita es una macro que calcule, en "compile time", **la potencia de dos que más se le aproxima** a un número.

Aquí presento el programa anterior en forma de macro (**lp2Macro.c**):

Tomen aliento...

```
#define lp2(x) (((((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                    (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)      |\
                   ((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                    (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)>>8) |\
                  (((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                    (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)      |\
                   ((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                    (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)>>8)>>16)+1)
```

Ahora, para tratar de entender la macro voy a <u>describirla</u> por partes...

```
#define lp2a_(x)      ((x)-1)
#define lp2a_1(x)     (lp2a_(x)>>1)
#define lp2a1(x)      (lp2a_(x) | lp2a_1(x))
#define lp2a1_2(x)    (lp2a1(x)>>2)
#define lp2a2(x)      (lp2a1(x) | lp2a1_2(x))
#define lp2a2_4(x)    (lp2a2(x)>>4)
#define lp2a4(x)      (lp2a2(x) | lp2a2_4(x))
#define lp2a4_8(x)    (lp2a4(x)>>8)
#define lp2a8(x)      (lp2a4(x) | lp2a4_8(x))
#define lp2a8_16(x)   (lp2a8(x)>>16)
#define lp2M(x)       (lp2a8(x) | lp2a8_16(x))

#define lp2(x)        (lp2M(x)+1)
```

Pueden usar cualquiera de las dos, pero una vez obtenida la primera, y visto que funciona, yo usaría esa. Para probarla, pueden modificar el programa anterior q2.c, incluyendo la macro y agregando en el printf: **lp2(number)**; el resto queda igual, así:

```
printf( "number=%s(%3d,0x%02X); nbits=%3d, "
        " exp_lp2=%3d, exp_lp2M=%3d, Log2=%3d\n",
         xtobinf((uchar)number, NBITS( (uchar)number ), MARK),
         number, number, nbits, exp, lp2( number ),
         nbits ? nbits - 1 : 0 );
```

Y, el **Sassert()** que buscábamos puede escribirse ahora como:

```
/* 2n-1V2.c, Luis G. Uribe C., Tuesday, August 28, 2007 */
// macro lp2 Rounds up to a power of 2, up to 32 bits

// if Sassert fail print: " expected constant expression"
#define  Sassert(e)  do{ enum { Sassert__ = 1/(e) }; } while (0)

#define lp2(x) (((((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                  (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)      |\
                 ((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                  (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)>>8) |\
                (((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                  (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)      |\
                 ((((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)          |\
                  (((x)-1|((x)-1)>>1)|((x)-1|((x)-1)>>1)>>2)>>4)>>8)>>16)+1)

#include <stdio.h>

/* ============================================================== */
void main( void )
{

#define TST 100  // ensayar con 128, 255, 256...

   Sassert( TST == lp2( TST ) );  // si TST es potencia de 2
   printf( "Size is %u\n", lp2( TST ) );

}
```

Desde luego, la macro '**lp2**' debe ir en un include file, para ocultarla.

➢ PROGRAMACIÓN POR REGLAS

Analicen el siguiente código, que es autoexplicativo, según presumo:

```
// rules02a.c, Luis G. Uribe C., M04J2007. Coding by the Rules...
//
//                    PARADIGMS IN MODERN PROGRAMMING
//
// Some real life problems may be programmed accordingly to rules
// ..used to define diverse circumstances and the actions that we
// ..must do. Some times we may include Tables, and look-up table
// ..routines to model our code like in event driven programming
// ..(similar as combinatorial digital circuits; there we would use
// ..ROMs to store the tables). But some times the tables could be
// ..very large, and not so easy to store them in our scarce Flash
// ..resources...
//
// The following example will show how to deal with this rules in
// .."C" language. We have an -input section to read all external
// ..information used by the rules; -then we apply the rules,
// ..-verify the system consistency, as depicted bellow, -and
// ..finally perform some actions, following the rules...
//
// A very big concern should be to *Warrant* the CONSISTENCY of the
// ..rules; i.e: ALL CASES MUST BE COVERED. French Ariane-5 space
// ..rocket blew up because a condition existed that was not covered
// ..by the rules. We use here 'assert(verify_rules())' to detect
// ..any inconsistencies, that will appear as "no decisions made"
// ..based on the rules. But ALL cases/ranges must be TESTED in the
// ..bench, to see if the assert fires under any circumstances.
// ..Asserts are NOT a runtime/production solutions: They abort the
// ..process (and the Ariane will blow up!)
//
// NOTE: The nature of this problem does not reside inside the code;
// ..it belongs to the external specifications... See that, no
// ..matter how good you code a program, it will fail badly to work
// ..properly if the specs it mimic are all wrong. This is the
// ..reason why you must postpone writing code, the most that it is
// ..possible.
//
// The solution is to WARRANT THAT ALL CASES ARE COVERED.
//
// See that one could take some default action if the verify routine
// ..detects "no action defined", but you MUST be very confident on
// ..your decisions in this matter.
//
// In the Ariane-5, the default action was to abort the program (as
// ..in here), under the assumption that the back-up computer would
// ..take over, and solve the problem. The second computer began to
// ..work, indeed, to enface the same problem!, because it was not a
// ..hardware malfunction; they were mere inconsistent rules that
// ..did not cover all cases and not covered such input data
// ..combination.
//
// As an example, we have tolerated inconsistencies in this program.
// ..Try and run the it with: 100 kph, 1 hour, NOT: prefer
// ..scenery over speed and, YES: I am a pilot. THE PROGRAM ABORTS!
// ..This case is NOT covered by our rules...

/* ------------------------------------------------------------ */
// Include Files

#include <assert.h>    // do NOT #define NDEBUG to disable asserts:
#undef assert          // THIS will disable asserts BUT continue to
#define assert(x)  (x) // .. execute the code passed to them!!!
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* ------------------------------------------------------------ */
// Standard Typedefs

#undef FALSE       // Some of our booleans are 3 valued variables:
#undef TRUE        // ..True, False, and UNKNOW: not already defined

typedef enum { FALSE, TRUE, UNKNOWN } bool;

/* ------------------------------------------------------------ */
// Defines

#define  SPEED_HIGH    200
#define  SPEED_MEDIUM  100
#define  SPEED_LOW      60

#define  BUF_SIZE        6    // 'gets' represents a security breach
                              // ..Space to hold: 300<CR><LF><0> max

/* ------------------------------------------------------------ */
// Function Prototypes

bool rules( void );
bool verify_rules( void );

/* ------------------------------------------------------------ */
// External Variables Definition; make up the rules

int  av_speed;          // kph

// NOTE: In small embedded systems you should avoid 'enums' because
// ..    they define integers (2 bytes), instead of chars or bits!
// ..    that would preserve more your scarcely memory resources

bool like_scenery,      // this is up to the user to decide
     is_pilot,          // ...and this

     // ---------------------------
     fly,               // depends on "calculated" average speed
     fly_commercial,    // if you must FLY, and you are NOT a pilot
     fly_cessna,        // if you must FLY, you ARE a pilot, LIKE the
                        // ..scenery & AVE. SPEED  <=  SPEED_MEDIUM
     fly_jet,           // if you must FLY, are a pilot
                        // ..and the ave. speed is <= SPEED_HIGH kph
```

```
    // ----------------------------
    drive,              // drive, not fly, for ave. speed<= SPEED_LOW
    drive_m_cycle,      // TRUE if you must DRIVE and LIKE scenery
                        // 0: you must DRIVE but DON'T like scenery
    drive_car,          // drive a car if you should drive (not fly)
                        // ..and won't use a motor cycle

    // ----------------------------
    impossible_speed; // for calculated average speeds > SPEED_HIGH


bool test_rules_assert,         // this variables used for testing
     test_verify_rules_assert; // ..purposes only


/* ************************************************************** */
void main ( void )                                          /*()*/
{
   // ---------------------------------------------------------------
   // Local Variables Definition
   // ..NOTE: Buffers should be static or dynamically allocated
   // ..(malloc), to keep them away from the STACK!

   int  distance,              // rounded to kilometers
        time;                  // rounded to hours

   static char buff[BUF_SIZE]; // to accept users' input responses

   // ---------------------------------------------------------------
   // Present program, then loop for ever (Press Ctrl-C to finish)

   printf( " I am programmed to help you with your"
           " traveling plans (press ^C to finish)" );
   printf( "\n=========================================="
            "==========================================\n\n" );

   while( TRUE ) {      // This 'while' mimics the ever lasting loop
                        // ..normally found in all embedded systems
      // ==============================================================
      // 1) Always init variables dynamically, inside the loop

      like_scenery      =
      is_pilot          =
      fly               =
      fly_commercial    =
      fly_cessna        =
      fly_jet           =
      drive             =
      drive_m_cycle     =
      drive_car         =
      impossible_speed  =   UNKNOWN;

      test_rules_assert = test_verify_rules_assert = FALSE;
```

```
      // ==============================================================
      // 2) This mimics DATA ACQ portion of Real Time systems

      // ----------------------------
      // Distance

      printf( "How far are you going (kilometers)? " );
      gets( buff );          // 'gets' represents a SECURITY BREACH
      distance = atoi( buff );

      // ----------------------------
      // Time. Evaluate average speed, av_speed

      printf( "How much time do you have for the trip (hours)? " );
      gets( buff );
      assert( time = atoi( buff ) );
      av_speed = distance / time;    // Internally calculated value

      // ----------------------------
      // Pleasure or business?

      printf( "Do you prefer scenery over speed? (y/N)? " );
      gets( buff );
      like_scenery = ( tolower( buff[ 0 ] ) == 'y' );

      // -------------- only for debugging purposes:
      test_rules_assert = ( buff[ 0 ] == 'Y' );  // 'Y' trigger var

      // ----------------------------
      // Are you a pilot by yourself?

      printf( "Are you a pilot? (y/N)? " );
      gets( buff );
      is_pilot = ( tolower( buff[ 0 ] ) == 'y' );

      // -------------- only for debugging purposes:
      test_verify_rules_assert = ( buff[ 0 ] == 'Y' );  // 'Y'

      // ==============================================================
      // 3) Apply Rules with the recently acquired external data
      // .. Abort program if rules does NOT verify

      assert( rules() );              // ASSERTS: NOT INTENDED TO BE
      assert( verify_rules() );        // ..USED ON FINAL PRODUCTS


      // ==============================================================
      // 4) Perform some Actions! (This is what they pay us for!!)

      if( impossible_speed == TRUE )
         printf("\nAv. speed (%d) is impossible: more than %d k/h",
                 av_speed, SPEED_HIGH );
```

```
        if( fly_commercial == TRUE )
            printf( "\nFly commercial." );

        if( fly_cessna == TRUE )
            printf( "\nRent a cessna and fly low." );

        if( fly_jet == TRUE )
            printf( "\nRent a Jet and fly high." );

        if( drive_m_cycle == TRUE )
           printf("\nTake your motorcycle and ride the back roads.");

        if( drive_car == TRUE )
            printf( "\nThere's nothing for it but to drive a car." );

       printf("\n-----------------------------------------------\n");

    }   // endWhile

}  /* main() */


/* ================================================================== */
bool rules ( void )                                                /*()*/
{
   // Some problems may be seen as a set of inputs used to choose a
   // ..set of outputs (pure combinatorial, as in digital circuits).
   // ..But in other cases, some outputs may depend on previously
   // ..evaluated outputs. In this case, the evaluation must be
   // ..integrated into a loop, that cycles if in the last iteration
   // ..some values were changed, that could imply new changes in
   // ..results already evaluated.
   //
   // STRATEGY: use 'evaluating'= TRUE to force loop cycling. Inside
   // ..the loop, begin assuming 'evaluating' to be FALSE, and turn
   // ..'evaluating' TRUE if some variables have been changed.
   // ..This insures that the module will repeat until it exhaust
   // ..all changes.
   //
   // We must prevent rules that have fired in a previous iteration
   // ..from firing again, which would cause the function to loop
   // ..endlessly. This is accomplished using variables that are
   // ..three-valued (TRUE, FALSE and UNKNOWN), initialized to
   // ..UNKNOWN. Now, shall two rules affect the same variable, the
   // ..final value of the variable will DEPEND ON THE ORDER of the
   // ..rules. The one that fires first prevents the other from
   // ..firing. You must be aware of this behavior... Note that you
   // ..may reformulate the rules, in such a manner to stop one
   // ..decision from being reevaluated later.
   //
   // NOTE: In some cases there could be the possibility for the
   // ..evaluating loop not to end. We include 'safe', a safety
   // ..counter to take this in account

   /* ------------------------------------------------------------ */
   // Defines (these are globals...)

   #define  SAFE  10u    // Max nmbr of iterations through eval loop

   // ------------------------------------------------------------
   // Local Variables Definition

   bool evaluating    = TRUE;       // Continue on the decision loop
   unsigned char safe = SAFE + 1;   // Limit number of iterations

   // -------------- only for debugging purposes:
   if( test_rules_assert )
       safe = 1;

   // ------------------------------------------------------------
   // Main evaluation loop

   while( evaluating  &&  --safe ) {
       evaluating = FALSE;          // Set-up loop break by default

       // ------------------------------------------------------
       // Dependencies on impossible_speed

       if( impossible_speed == TRUE ) { // then all output variables
           fly_commercial = FALSE;      // ..shall be set to FALSE
           fly_cessna     = FALSE;
           fly_jet        = FALSE;
           drive_m_cycle  = FALSE;
           drive_car      = FALSE;

           break;               // this rule ends 'while' immediately
       }

       // ------------------------------------------------------
       // Dependencies on av_speed

       if( av_speed > SPEED_HIGH && impossible_speed == UNKNOWN ) {
           impossible_speed = TRUE;     // now var may be reevaluated
           evaluating = TRUE;           // continue in the while loop
       }

       if( av_speed > SPEED_LOW && fly == UNKNOWN ) {
           fly        = TRUE;
           evaluating = TRUE;
       }
       if( av_speed <= SPEED_LOW && drive == UNKNOWN ) {
           drive      = TRUE;
           evaluating = TRUE;
       }

       // ------------------------------------------------------
       // Should I fly? If so: commercial, cessna or jet?
```

```
        if(    fly == TRUE && is_pilot == FALSE
            && fly_commercial == UNKNOWN ) {

            fly_commercial = TRUE;
            evaluating     = TRUE;
        }
        if(   fly == TRUE && is_pilot == TRUE && like_scenery == TRUE
            && av_speed <= SPEED_MEDIUM && fly_cessna == UNKNOWN ) {

            fly_cessna = TRUE;
            evaluating = TRUE;
        }

        if(    fly == TRUE && is_pilot == TRUE
            && SPEED_MEDIUM < av_speed && av_speed <= SPEED_HIGH
            && fly_jet == UNKNOWN ) {

            fly_jet    = TRUE;
            evaluating = TRUE;
        }

        // ------------------------------------------------------------
        // should I drive instead? And, if so, car or motorcycle?

        if(    drive == TRUE && drive_m_cycle == FALSE
            && drive_car == UNKNOWN ) {

            drive_car  = TRUE;
            evaluating = TRUE;
        }
        if(    drive == TRUE && like_scenery == TRUE
            && drive_m_cycle == UNKNOWN ) {

            drive_m_cycle = TRUE;
            evaluating    = TRUE;
        }
        if(    drive == TRUE && like_scenery == FALSE
            && drive_m_cycle == UNKNOWN ) {

            drive_m_cycle = FALSE;
            evaluating    = TRUE;
        }

    } /* endwhile */

    if( safe )
        return TRUE;
    return FALSE;

}  /* rules() */


/* ------------------------------------------------------------ */
```

```
bool verify_rules ( void )                                    /*()*/
{
    // CRITERIA: Some action must be taken so, at least one output
    // ..variable should take a TRUE value. Note that 'NOT TRUE'
    // ..is NOT same as FALSE, due to the UNKNOWN value.
    //
    // In other circumstances, it could be a failure if more than one
    // ..action is fired by the rules. Not this case...,

    // -------------- only for debugging purposes:
    if( test_verify_rules_assert )
        return FALSE;       // bad

    if(    impossible_speed != TRUE && fly_commercial != TRUE
        && fly_cessna       != TRUE && fly_jet        != TRUE
        && drive_m_cycle    != TRUE && drive_car      != TRUE )

        return FALSE;       // bad: no one action was activated

    return TRUE;        // good: one or more actions were activated

}  /* verify_rules() */
```

La Programación por Reglas va en el sentido de separar la Política de los Mecanismos. Este código no está tan nítidamente dividido como el de la animación ASCII que vimos al principio, pero se le aproxima bastante.

Algunos comentarios en relación al programa: La forma que se estila para deshabilitar los **asserts**, una vez que el programa ha sido depurado es, como ya hemos dicho, incluir un `#define NDEBUG` ***antes*** del **`<assert.h>`**. Sin embargo, a veces, uno incluye código que debe ejecutarse tanto si se está depurando el programa como si ya está en operación. **No se debe andar editando el programa cuando ya funciona**, pues con alta probabilidad LO HECHARÁ A PERDER, *DESPUÉS* DE "DEBUGEADO".

Recomiendo emplear la siguiente estrategia ***en este caso***: En vez del `#define NDEBUG`, cuando ya no desee los asserts, **incluya las siguientes DOS LÍNEAS de código**, ***debajo*** del `#include <assert.h>`:

```
#undef assert           // THIS will disable asserts BUT continue to
#define assert(x)   (x) // ...execute the code passed to them!!!
```

Se ha redefinido la llamada a assert, incluyéndole un parámetro: `assert(x)`, simplemente para lograr la ejecución del código identificado por dicho parámetro: **`(x)`**.

Note también la definición preferida de los valores booleanos, que se han resaltado en este código, porque se está empleando un tercer valor: **UNKNOWN**. Primero se los "borra" (`#undef FALSE`), por si estuvieran definidos con anterioridad (en algún include file, por ejemplo). Luego se los define con tres valores: `typedef enum { FALSE, TRUE, UNKNOWN } bool;`

Durante la definición de parámetros se hace un comentario en el código: Que `'gets' represents a security breach`. Esta función está desaprobado ("**deprecated**"), porque es el sitio por donde más aprovechan los piratas para irrumpir en el código de los demás. Es debido a que `'gets'` **no** valida que la cantidad de información que se le suministra, esté confinada al buffer en el que la está leyendo. Si uno define un buffer de 10 posiciones, y el usuario le proporciona 100, o 10.000 letras, o más, la información va a quedar almacenada en sitios que no le pertenecen, y el programa va a operar mal, o abortar.

Es preferible emplear `fgets()`, uno de cuyos parámetros es el tamaño máximo que el sistema va a permitir pasar a nuestro buffer. De esa manera estamos protegidos. Note que en los ejercicios anteriores donde he usado `'gets'`, la información va a llegar por el teclado, y el sistema operativo limita la entrada al tamaño que hemos definido para nuestro buffer. En todo caso, si el intruso logra "redirigir" la entrada para tomarla desde un archivo, el tamaño del buffer ya **no** seguirá protegido por el sistema...

Se menciona también que los sistemas embebidos deben evitar el uso de `'enums'`, porque C los define como enteros, y éstos suelen ser 2 bytes hasta en las máquinas más pequeñas...

En el mismo orden de ideas, definir booleanos como enteros es un desperdicio de memoria. Hay que tratar de declarar campos de bits y definirlos allí (bit fields). En nuestro ejemplo actual, se necesitan 2 bits por cada booleano, y como la idea es hacer una presentación del tema de Reglas, no hemos puesto objeción al empleo de este tipo de booleanos, pero **esté avisado**.

Pregunta: ¿Por qué se coloca **static** `char buff[BUF_SIZE];`? Porque, si no es 'static', será automático, y las variables automáticas emplean espacio en el stack, que suele ser ***escaso***, sobre todo en microcontroladores. Definirlo como 'static' hace que su espacio se tome de otra parte (el "**heap**"), que **no** compromete el stack. ¿Por qué no se puso global, si tampoco las variables globales ocupan stack? Porque hay que hacer el máximo posible por **ocultar las variables** (y las estructuras de datos, en general) de unas rutinas, de las demás funciones, para eliminar engranajes entre ellas.

Observe que, luego de leer algunas variables, las pasamos por "assert": `assert(time= atoi(buff))`. De esta forma garantizamos que se cumpla con ciertas condiciones. Recuerde que los "assert" **NO van a dejarse para el programa en producción** (ya que **abortan** la operación del programa, en caso de falla); lo que se suele hacer, en caso de querer validar la entrada que el operador nos suministra, es establecer una rutina de validación, que probablemente repita la solicitud de entrada de información, en caso de detectar alguna anomalía.

Abortar un programa en la cara del usuario, solo porque éste introdujo un valor inválido, no es muy amable por parte del programador...

Por eso se coloca una advertencia bien grande:

```
assert( rules() );              // ASSERTS: NOT INTENDED TO BE
assert( verify_rules() );        // ..USED ON FINAL PRODUCTION
```

Fíjense cómo se llamó a la rutina que procesa las reglas: **`rules()`**, y cómo se verifica su consistencia: **`verify_rules()`**

Más adelante se hace la advertencia de que, en ciertas condiciones, puede darse el caso de que una rutina de evaluación, que se repite para recalcular ciertos valores con algunos otros que también han sido calculados ahí, nunca termine. Es por esto que hemos incluido un contador:

```
#define  SAFE  10u    // Max number of iterations through eval loop.
```

ANTI REBOTE (Debouncing)

Los contactos electromecánicos de teclados y otras entradas a nuestros circuitos digitales, como los relés, ejecutan varias conexiones y separaciones (rebotes) cuando los oprimimos, y a veces también al momento de abrirlos; el tiempo entre comenzar a actuar, hasta poder decir que ya operó, puede ir desde menos de 100 nanosegundos hasta 50 milisegundos, siendo una figura comercial común: Menos de 10 milisegundos (si sus interruptores rebotan por más de 16 milisegundos, ¡**deséchelos**!).

Algunos switches tienen problema solo cuando cierran; otros, cuando abren, y algunos en ambos casos. Como de ellos hay gran cantidad de clases: DPDT (**D**ouble **P**ole, **D**ouble "**T**hrow" [doble "polo", doble conmutación; es decir, hay un contacto galvánico en la posición cerrado y otro en la posición abierto]), SPDT (**S**ingle **P**ole, **D**ouble "**T**hrow" [un "polo", doble conmutación; es decir, hay un contacto galvánico en la posición cerrado, que se abre al hacer la conmutación para la posición inactiva]), DPST, SPST, etc.), hay que cuidarse del rebote tanto al cerrar como al abrir.

Hay dos criterios si la solución es circuital; **1)** emplear un flip-flop Set-Reset sin reloj (biestable "Eccles-Jordan", o "latch" S-R [Set-Reset]), lo cual requiere que el interruptor tenga 3 salidas, una en el medio, que se conecta a tierra, y las otras dos, una que hace contacto cuando el interruptor está "abierto" y la otra que hace conexión cuando lo cerramos. **2)** usar interruptores más baratos, que sólo tienen 2 terminales (Single "Pole"): Se coloca un filtro pasabajos, RC, y un Schmitt-trigger (dispositivo con histéresis). Note que la circuitería externa (RC y Schmitt-trigger) puede llegar a ser una alternativa más costosa que utilizar un SPST; sin embargo, el "costo" escondido es que el SPST utiliza dos (2) contactos de la lógica, en tanto que la otra aproximación usa uno sólo. Cuando trabajamos con microcontroladores o FPGAs, los **pines del chip son el recurso más escaso** (ésta es la razón por la cual, en ciertos microcontroladores, **un mismo pin** puede emplearse como entrada digital, o salida, o entrada analógica, o terminal de comunicaciones seriales, o comparador, o generador de pulsos PWM, o entrada del cristal, u oscilador externo... En el chip caben todos eso recursos, **pero pines no hay**)

Cuando tenemos un microcontrolador, tratará de maximizarse el uso del software a fin de mantener al mínimo el hardware externo; por eso es común que el interruptor SPST se conecte directamente al micro, sin filtros pasabajos ni Schmitt-triggers. (**Nota**: Si las señales vienen "**del campo**", y traen ruido, por ejemplo, de una planta industrial, u otras anomalías, como voltajes de modo común, permanentes o inducidos, etc., es indispensable **adecuar** dichas señales antes de entrar a la circuitería; pero esta etapa es anterior a todo lo que estamos mencionando, y muy probablemente ¡nada de eso se pueda arreglar dentro del microcontrolador!, y hay que tenerlo en cuenta también cuando se usan "latches")

Hay varias estrategias que suelen aplicarse por software. Comencemos diciendo que (casi) nunca debe colocarse un SPST sin filtraje electrónico externo, a una línea de interrupciones (IRQ, KB). Eso suele complicar el diseño, pues los rebotes interrumpen repetidas veces, y si se incluye un temporizador por software, que a su vez emplea interrupciones, hay que estar pendientes de desactivar las interrupciones del switch en cuestión, rehabilitar las interrupciones generales hasta que *la rutina de interrupciones* del temporizador señale que ha expirado el lapso programado, y proceder a dejar todo tal como estaba.

La opción más sencilla es leer, en el momento que corresponda (sin que el switch interrumpa…), el estado del interruptor de entrada, y en caso de detectar alguna transición, activar un temporizador de 20 milisegundos. Solo al expirar el lapso se vuelve a leer el interruptor, y si ha mantenido el valor que ocasionó la entrada a la rutina de **debouncing**, se acepta la transición; de lo contrario se asume que no hubo ninguna, y se continúa en el sitio apropiado del programa, según el problema. Algo así como:

```
Forever {
    // Here inputPin was 0. If inputPin == 1; there has been a change

    if( inputPin() == 1 ) {
        WaitMS_on( 0, 20 );      // debounce on close
        if( inputPin() == 1 ) } // inputPin really is closed (ON)
            Exec What You Do If InputPin is ON();
            Wait( inputPin == 0 );
            WaitMS_on( 0, 20 ); // debounce on open too...
        }   // else, inputPin had a glitch; ignore it and continue
    }else { // inputPin is OFF
        Exec What You Do Whenever InputPin is OFF();
    }
}   // endForever
```

Nota: Al terminar la actividad que se ejecuta cada vez que hay un cambio a "1", la máquina de estados debería esperar que la entrada retorne a 0; (lo que se hace con: `Wait(inputPin == 0))` porque de lo contrario, al repetir el ciclo, volvería a ser cierto el: `if(inputPin() == 1){`, pero esa activación del usuario **¡ya fue atendida!** La secuencia es: Esperar la activación del interruptor por parte del usuario; cuando esto sucede, ejecutar lo que se le solicitó y, luego, esperar a que el usuario retire el dedo del control para volver a comenzar la secuencia.

Otra cosa: Se pensaría que mientras se ejecuta la espera final, `Wait( inputPin == 0 );` y se da la vuelta, pueden perderse activaciones del interruptor por parte del usuario. Hay que resaltar que los tiempos de reacción de una persona (la capacidad de activar, desactivar, y volver a activar uno u otro de los botones) son del orden de los **centenares** de milisegundos (nadie oprime un botón más de 10 veces por segundo; no, si no se está jugando con el teclado). Veinte milisegundos es poco en relación a nuestra velocidad de respuesta.

DEBOUNCING Y MÁQUINAS DE ESTADO

Solo como ilustración se repite el ejemplo, en forma de **Máquina de Estados**:

```
enum STATES { WaitInputPin_1, DebounceOn_1,
              TestInputPinIsReally_1,
              WaitInputPinReturnTo_0 };
state = WaitInputPin_1;
Forever {
    switch( state ) {
      case WaitInputPin_1:
        if(inputPin == 1) state= DebounceOn_1;
        else Exec_Code_For_InputPin_0();
      break;
      case DebounceOn_1:
        WaitMS_on( 0, 20 ); // debounce on 1
        state = TestInputPinIsReally_1;
      break;
      case TestInputPinIsReally_1:
        if( inputPin == 1 ) {  // really "1"
            Exec_Code_For_InputPin_1();
            state = WaitInputPinReturnTo_0;
        }else state = WaitInputPin_1; //glitch
      break;
      case WaitInputPinReturnTo_0:
        if( inputPin == 0 ) {
            WaitMS_on(0, 20); // debounce "0"
            state = WaitInputPin_1;
        }
      break;
    }   // endswitch
}   //endForever
```

Esta es una de varias maneras de implementar la **Máquina de Estados**, que son una alternativa **excelente** para codificar este y otros tipos de problemas. Se ha exagerado la cantidad de estados (en realidad, con uno o dos bastarían para el ejemplo), a fin de mostrar cómo sería la mecánica.

Ahora bien: En una aplicación en la que hay otras cosas por hacer, y donde el microcontrolador no puede darse el lujo de sentarse a perder el tiempo esperando a que se elimine el rebote (`WaitMS_on(0,20)`), el programa suele correr en un ciclo dentro del cual, una actividad es preguntar por una bandera que indica si el temporizador ya terminó. La presentación de una rutina así puede resultar muy oscurecida con sus detalles, máxime si son dos o más los interruptores sujetos a debouncing. Es decir, andar pendientes de la lógica del programa, dirigida por los interruptores, y a la vez tener que incluir la rutina de debouncing, puede crear un programa bastante retorcido.

Para que se vea cómo resultaría un código así, anexo el programa de la *siguiente* página.

Una alternativa para mejorar la presentación y el manejo del problema de mezclar el debouncing con la lógica del programa consiste en que sea la misma rutina temporizadora, que supuestamente corre por interrupciones, la que realice en el trasfondo (*background*) la eliminación del rebote. Así, habrá una o varias posiciones de memoria (variables) cuyos bits serán una copia "limpia" de los verdaderos interruptores. Cuando el programa lee una variable, ya no tiene que lidiar con el problema de debouncing. Fíjense que si se le quita el manejo del rebote al programa original, éste quede muy simple:

```
Forever {
    // Here, you know inputPin is 0. If inputPin == 1; there
    // ..has been a change
    if( inputPin() == 1 ) {
        Exec What You Do If InputPin ON();
        Wait( inputPin == 0 );
        }
    }else { // inputPin is OFF
        Exec What You Do If InputPin OFF();
    }
}   // endForever
```

La solución, con forma de Máquina de Estados:

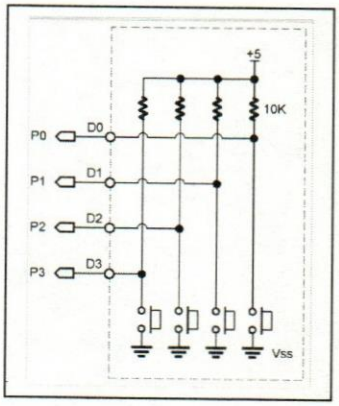
```
enum STATES { WaitInputPin_1,
              WaitInputPinReturnTo_0 };
state = WaitInputPin_1;
Forever {
    switch( state ) {
      case WaitInputPin_1:
        if( inputPin == 1 ) {
            Exec_Code_For_InputPin_1();
            state = WaitInputPinReturnTo_0;
        }else Exec_Code_For_InputPin_0();
      break;

      case WaitInputPinReturnTo_0:
        if( inputPin == 0 ) {
            state = WaitInputPin_1;
        }
      break;

    }   // endswitch

}   //endForever
```

```
// fil-tst.c, Luis G. Uribe C., D09S7
// Código para mostrar las complicaciones de un programa cuando
//..se mezcla la lógica principal, y el debouncing.

// Definiciones FICTICIAS, que sirven como ayuda en
// ..la verificación de la operación del programa.
// ..Desde el debugger se puede cambiar inputPin = 1, para simular
// ..una entrada; t = 1 simula que ya expiró el timer (Timeout(t))
// Las dos rutinas Exec_ sólo llaman a 'kbhit()', para poder
// ..detener el programa con Ctrl-C, si se entra en un loop
// ..infinito.

#define SetimerMS( t, T ) (t=0)
#define Timeout( t )    (t)
#define Exec_What_You_Do_If_InputPin_ON()  kbhit()
#define Exec_What_You_Do_If_InputPin_OFF() kbhit()

#define TRUE  1
#define FALSE 0
#define Forever    for(;;)

void main( void )
{
   int  t = 0;
   int  state, inputPin = 0;
   enum STATES { WaitInputPin_1, Debounce_1,
                 TestInputPinIsReally_1,
                 WaitInputPinReturnsTo_0,
                 WaitTimeoutForInputPin_0 };
   state = WaitInputPin_1;

   Forever {
       // Here, you know inputPin == 0; if inputPin == 1;
       // ..there has been a change

       switch( state ) {
         case WaitInputPin_1:
           if( inputPin == 1 ) {
               state = Debounce_1;
           }else {
               Exec_What_You_Do_If_InputPin_OFF();
           }
         break;

         case Debounce_1:
           SetimerMS( t, 20 );
           state = TestInputPinIsReally_1;
         break;


       case TestInputPinIsReally_1:
           if( Timeout( t ) ) {
               if( inputPin == 1 ) {   // really ON
                   Exec_What_You_Do_If_InputPin_ON();
                   state = WaitInputPinReturnsTo_0;
               }else {
                   state = WaitInputPin_1;    // ignore glitch
               }
           }
         break;

         case WaitInputPinReturnsTo_0:
           if( inputPin == 0 ) {
               SetimerMS( t, 20 );     // debounce on open too...
               state = WaitTimeoutForInputPin_0;
           }
         break;

         case WaitTimeoutForInputPin_0:
           if( Timeout( t ) ) {
               state = WaitInputPin_1;    // ignore glitch
           }
         break;

       }   // endswitch

   }   // endForever
}
```

Como ya se dijo, `#define SetimerMS( t, T ) (t=0)` cambia la función del subsistema de timers, que ya hemos usado con anterioridad, a fin de permitirnos manipular el código con el debugger y así demostrar su funcionamiento. En este caso, `(t=0)` sirve para inicializar la rutina de `Timeout( t )`. Esta, a su vez, devuelve el valor "t"; por tanto, para simular que ya expiró el timer, basta con ***cambiar desde el debugger*** el valor de "t".

Se definen las constantes `enum STATES`, dándoles a los estados nombres que sean muy sugestivos de la actividad que en ellos habrá de realizarse. El `Forever` es bastante parecido al programa anterior, excepto que en vez de los **`WaitMS_on( 0, 20 );`** que bloquean el procesador, se usa ahora **`SetimerMS(t,20);`** acompañada luego por el **`if( Timeout( t ) ) {`**.

DEBOUNCING DE VARIOS BITS



Ya que se ha visto la importancia de colocar en la rutina de interrupciones del timer, la funcionalidad de eliminar el rebote de varios interruptores a la vez, y suministrar una visión "virtual" de estos bits de entrada, "limpios" (sin rebote), pongamos como ejemplo el debouncing de los 4 bits mostrados en la figura; note que se activan bajo y los supondremos conectados a la parte inferior del puerto de entrada `inputPins4`.

```
int Get4Switches ( void ) {                          /*()*/
   int LocalVirtualSwitches=0x000F;// 00000000 00001111
   for( x = 0; x < 5; x ++ ) {
       LocalVirtualSwitches &= ~inputPins4;
       WaitMS_on( 0, 4 );
   }   // endFor
   return LocalVirtualSwitches & ~inputPins4;
}
```

En primer lugar hay que hacer notar que se inicializan los bit en "unos" (`LocalVirtualSwitches = 0x000F;`), ya que la activación externa es con ceros. Aquí se han consideran solo los 4 bits de la derecha. Dentro del ciclo se leen los 4 bits externos, `inputPins4;` se los niega para convertir a "unos" los que están activos (`~inputPins4`), y se hace and (**&**) con el valor previo de `LocalVirtualSwitches` (`LocalVirtualSwitches &= ~inputPins4;`); luego se espera 4 ms, repitiendo el lazo 5 veces, para un total de 20 ms. Para producir el resultado se hace un último and: `return LocalVirtualSwitches` **&=** `~inputPins4;`, al cual **no** hay que agregarle un `WaitMS_on( 0, 4 )` final. Observe que la variable se llama **`Local`**`VirtualSwitches` porque ella no puede ser leída en cualquier momento por otras rutinas, pues durante los 20 ms. que dura `Get4Switches`, su contenido es espurio. Debe haber una rutina que haga: **`Global`**`VirtualSwitches = Get4Switches();`

Si las entradas fueran activas en uno, se **quitaría** el negador:

`LocalVirtualSwitches &= inputPins4;`.

Una entrada tiene que mantener su valor, **inalterado** durante el ciclo completo; o la rutina devuelve un **cero** en su posición. Una rutina genérica debería permitir identificar cada bit, con un parámetro, como activo en 0 o en 1.

SECUENCIADOR DE EVENTOS, SOE

```
/* ************************************************************** *
 *                                                                *
 *                    SOE  FILTERING  ALGORITHM                   *
 *                    =========================                   *
 *                                                                *
 *                                  Luis G. Uribe C, L15E1 D07Y6 *
 *                                                                *
 *                                                                *
 *     The output will be 0 if it has been 0 for the last  NFILTER *
 *         milliseconds at least                                  *
 *                                                                *
 *     The output will be 1 if it has been 1 for the last  NFILTER *
 *         milliseconds at least                                  *
 *                                                                *
 *     Otherwise, the output will remain in its actual value      *
 *         (ActualState)                                          *
 *                                                                *
 *                                                                *
 * FILTER DATA STRUCTURES:                                        *
 * ======================                                         *
 *                                                                *
 * We will need the actual  InputData  plus one array of size     *
 * 'NFILTER' (Filter time in milliseconds, NFILTER > 0), to hold  *
 * the last NFILTER  one millisecond separated input values.      *
 * Normally, we  processs 8/16/32 input bits at a time, so the    *
 * array will be NFILTER bytes (8), words (16) or longs (32)      *
 *                                                                *
 * The Data, for any given  NFILTER  size, could be held in a     *
 * matrix of size  ** 'NFILTER + 1' **  with the following layout: *
 *                                                                *
 * uchar, uint or ulong                                           *
 *        Filter[ 0 ],                                            *
 *          Filter[ 1 ],                                          *
 *          Filter[ 2 ],                                          *
 *          ...                                                   *
 *          Filter[ NFILTER - 1 ],                                *
 *        InputData;                  // i.e.: Filter[ NFILTER ]  *
 *                                                                *
 *                                                                *
 * PROCEDURE:                                                     *
 * =========                                                      *
 *                                                                *
 * For any given input value, we will stablish the AND and the OR *
 * of it with all the last NFILTER stored values. The output value *
 * (NextState) will become:                                       *
 *                                                                *
 *     NextState = Filter_AND  |  ( ActualState  &  Filter_OR )   *
 *                                                                *
 *     being:                                                     *
 *                                                                *
 *         Filter_AND:  the AND of all PREVIOUS  NFILTER  bits    *
 *                      with the current input value              *
 *                                                                *
 *         Filter_OR:   the OR  of all PREVIOUS  NFILTER  bits    *
 *                      with the current input value              *
 *                                                                *
 *         ActualState: Last filter output                        *
 *                                                                *
 *         NextState:   Next filter output                        *
 *                                                                *
 *                                                                *
 *     This means:                                                *
 *                                                                *
 *         NextState = 1 if ALL Filter_AND are 1: 1 | ...         *
 *         NextState = 0 if ALL Filter_OR  are 0: 0 | ( PS & 0 )  *
 *                                                                *
 *     Otherwise, some bits are 1  and  some bits are 0, so:      *
 *                                                                *
 *         NextState = ActualState                                *
 *                                                                *
 *                                                                *
 * For each one millisecond separated input value, the main       *
 * processing loop will be as follows:                            *
 *                                                                *
 *     // Read next value into 'InputData'; i.e: Filter[ NFILTER ] *
 *                                                                *
 *     For( ( Filter_AND = 0xFFFF, Filter_OR  = 0x0000, j = 0 );  *
 *          j < NFILTER;  j++ )                                   *
 *                                                                *
 *     // include *ALL* ANDs, ORs, and the required data shifts  *
 *                                                                *
 *         Filter_AND  &=  Filter[ j ];                           *
 *         Filter_OR   |=  Filter[ j ];                           *
 *         Filter[ j ]  =  Filter[ j + 1 ];                       *
 *                                                                *
 *     Endfor                                                     *
 *                                                                *
 *     // Output Global Data                                      *
 *                                                                *
 *     Filter_AND = 1: All Ones                                   *
 *     Filter_OR  = 0: All Zeroes                                 *
 *                                                                *
 *     Filter_AND = 0: One or more Zeroes                         *
 *     Filter_OR  = 1: One or more Ones                           *
 *                                                                *
 *     So: True Table and Karnaugh Map are:                       *
 *                                                                *
 *     Filter_AND Filter_OR | NextState                           *
 *            1   1         | 1                                  *
 *            1   0         | Imposible                          *
```

```
 *            0   0        |  0                                          *
 *            0   1        |  ActualState (Don't change)                 *
 *                                                                        *
 *                                                                        *
 *                 \Filter_AND,Filter_OR                                  *
 *                  \                                                     *
 *       ActualState \ 00  01 |11  10 |                                  *
 *                     +---+---+---+---+                                  *
 *                  0 | 0 | 0 |(1 | -)|                                  *
 *                 ---+---+---+---+---+                                  *
 *                  1 | 0 |(1 |(1)| -)|                                  *
 *                 ---+---+---+---+---+                                  *
 *                        |       |                                       *
 *                                                                        *
 *                        NextState                                       *
 *                                                                        *
 *     NextState    = Filter_AND  |  ActualState  &  Filter_OR;           *
 *     ChangeInData = ActualState ^  NextState;                           *
 *                                                                        *
 *     // Set Auxiliary Data                                              *
 *                                                                        *
 *     ActualState  = NextState;                                          *
 *
 */

/* ************************************************************** */
#include "ezc.h"

ID = "Soe_Tst  Luis G. Uribe C, L15E1 S06Y6: \n"
     "         Soe Filtering Algorithm\n";

USAGE = "USAGE: Soe_Tst\n";

/* ============================================================== */
/* External Variables
*/

/* 33 input values on this example; must be  >  NFILTER, at least. *
 * In the example, values are valid only in the lower BYTE. Values *
 * on Upper byte are all spureous
*/

uint   Values[] = {
/* t00 */ 0x0182, // 1--- ----

/* t01 - t08 */ 0x0283, 0x0381, 0x0480, 0x0582,
               0x0683, 0x0781, 0x0880, 0x0982,

/* t09 */ 0x1080, // change 1--- ---- in t0 seen in t10, even if it
/* t10 */ 0x1102, // ..changes back to 0: 0--- --1- here!
/* t11 */ 0x2203, // ---- ---1

/* t12 - t19 */ 0x3303, 0x4403, 0x5503, 0x6603,
               0x7703, 0x8803, 0x9903, 0xAA03,

/* t20 */ 0xBB03, // seen here change 0--- --1- on t10
/* t21 */ 0x0102, // seen change ---- ---1 in t11; chg. to ---- ---0

/* t22 - t30 */ 0x0206, 0x0300, 0x0402, 0x0506,
               0x0602, 0x0700, 0x0806, 0x0902, 0x1000,

/* t31 */ 0x0102, // seen ---- ---0 value from t21
/* t32 */ 0x0000

};

/* ============================================================== */
/* Filter Global Parameter                                        */

#define NFILTER    10      // Filter Milliseconds; CHANGE as needed


/* -------------------------------------------------------------- */
/* Include Files                                                  */

#include <stdio.h>

/* ============================================================== */
/* Soe_Filter Prototype & Private Global Accessible Data;         */
/* ... *** move this into  "soe_filter.h" ***                     */

void Soe_Filter( uint Input );

uint   Filter[ NFILTER + 1 ],  // Global data init on 0's by linker
       NextState,
       ChangeInData;


/* -------------------------------------------------------------- */
/* Number of Elements on this Example                             */
/*   Following expresion automatically computes de size of the    */
/*   data array defined above by the user, for this example       */

#define    EX_SIZE     ( sizeof( Values ) / sizeof( Values[ 0 ] ) )
```

```
/* ============================================================ */
void   main( void )
Begin      /* soe_tst.c */

   /* ------------------------------------------------------------ */
   /* Local Variables                                              */

   uint    i;

   /* ------------------------------------------------------------ */
   /* Process                                                      */

   puts( _id );

   /* ------------------------------------------------------------ */
   /* For each incomming value, call Soe_Filter()                  */

   For( i = 0; i < EX_SIZE; i++ )
       printf( "i = %02u, Input = %04X,", i, Values[ i ] );

       Soe_Filter( Values[ i ] );

       printf( " Output = %04X", NextState );
       printf( ChangeInData ? ", Change = %04X\n" : "\n",
               ChangeInData );

   Endfor

End        /* soe_tst.c */


/* ============================================================ */
void   Soe_Filter( uint Input )                          /*()*/
Begin      /* Soe_Filter() */

/* Returns:                                                     *
 *                                                              *
 * NextState    = Filter_AND  |  ActualState  &  Filter_OR;     *
 * ChangeInData = ActualState ^  NextState;                     *
 *                                                              *
 */

   /* ------------------------------------------------------------ */
   /* Local Variables                                              */

   static  uint    ActualState = 0;

   uint    Filter_AND,
           Filter_OR,
           i;

   /* ------------------------------------------------------------ */
   /* Load read data                                               */

   Filter[ NFILTER ] = Input;     // sizeof(Filter) is 'NFILTER + 1'


   /* ------------------------------------------------------------ */
   /* Process each input value through the Filter                  */

   For( ( Filter_AND = 0xFFFF, Filter_OR  = 0x0000, i = 0 );
        i < NFILTER;  i++ )

       Filter_AND  &=  Filter[ i ];
       Filter_OR   |=  Filter[ i ];
       Filter[ i ]  =  Filter[ i + 1 ];

   Endfor

   /* Output Global Data   */

   NextState    = Filter_AND  |  ActualState  &  Filter_OR;
   ChangeInData = ActualState ^  NextState;

   /* Set Auxiliary Data   */

   ActualState  = NextState;

End        /* Soe_Filter() */
```

En este ejercicio hay varias cosas interesantes qué resaltar. En primer lugar, es un subsistema que operó en equipos comerciales, supervisando muchos centenares de puntos (1024); la rutina corría cada milisegundo (la resolución requerida era de 1 ms); si se encontraba alguna diferencia entre el estado presente y el anterior (algún bit había cambiado), se le hacía un "time stamp", registrándolo con la marca de tiempo, a fin de establecer la hora exacta de la ocurrencia de dicho cambio. El proceso posterior, para almacenar un registro histórico, hacer los reportes correspondientes y demás manejo de alarmas, etc., es una función de la Unidad Maestra. Para alguno de los protocolos de comunicaciones que emplearon este subsistema, como el DNP, la clase de datos SOE es nativa; para otros, como el Modbus, hubo que hacer una expansión de las capacidades en las Maestras, porque la función de secuenciador de eventos, SOE, no es nativa en Modbus. Sin embargo, la transferencia de todas maneras resultó muy sencilla, y se escogieron funciones dentro del grupo que los usuarios pueden definir. Esto no evitó que hubiera que reprogramar la Maestra Modbus.

El objetivo de la rutina SOE es producir una salida que será 0 si la variable digital ha sido 0 continuo por lo menos durante los últimos NFILTER milisegundos; y producirá un 1 a la salida si la variable digital ha sido 1 continuo al menos durante los últimos NFILTER milisegundos. De lo contrario la salida permanecerá en su valor presente.

Las estructuras de información necesarias para apoyar el desarrollo del algoritmo de este filtro son la siguientes: La entrada actual, `InputData`, más un arreglo de tamaño `'NFILTER' (Filter time in milliseconds, NFILTER > 0)`, que almacenará los últimos valores de entrada, separados un milisegundo (La cifra de un milisegundo no es esencial para la operación del filtro; éste puede actuar más lento, o a mayor velocidad si hay tiempo disponible en la rutina del temporizador en un equipo en particular)

Normalmente se realiza el proceso de `8/16/32` bits a la vez, así que el arreglo será de `bytes (8), words (16) o longs (32)`. Para cualquier tamaño de `NFILTER`, la información puede almacenarse en una matriz de tamaño `'NFILTER + 1'`, con la siguiente definición:

```
uchar, uint or ulong
       Filter[ 0 ],     // este, al menos, es mandatorio;
         Filter[ 1 ],  // los demás dependen de la profunidad
         Filter[ 2 ],  // ..del filtro
         ... hasta:
         Filter[ NFILTER - 1 ], // A continuación se almacena:
       InputData;               // i.e.: Filter[ NFILTER ]
```

El procedimiento se expone a continuación: Para cada vector de entrada, se establece el `AND` y el `OR` entre él y los últimos `NFILTER` valores almacenados; el valor resultante obedece a la siguiente ecuación:

`NextState = Filter_AND | ( ActualState & Filter_OR )` siendo:

`Filter_AND:` el AND de todos los PREVIOS NFILTER bits con el valor de entrada actual,
`Filter_OR:` el OR de todos los PREVIOS NFILTER bits con el valor de entrada actual;
`ActualState:` la última salida del filtro, y
`NextState:` la siguiente salida del filtro.

De acuerdo a lo anterior, `NextState` será 1 si TODOS los `Filter_AND son 1; NextState` será 0 si TODOS los `Filter_OR` son 0, y si no, algunos bits serán 1 y algunos serán 0, por lo que: `NextState = ActualState.`

El procesamiento que se hace cada milisegundo será, entonces:

```
// Read next value into 'InputData'; i.e: Filter[ NFILTER ]
   For( ( Filter_AND = 0xFFFF, Filter_OR  = 0x0000, j = 0 );
        j < NFILTER;  j++ )
   // include *ALL* ANDs, ORs, and the required data shifts
       Filter_AND  &=  Filter[ j ];
       Filter_OR   |=  Filter[ j ];
       Filter[ j ]  =  Filter[ j + 1 ];
   Endfor
```

Al final del ciclo, las posibilidades de salida son:

```
Filter_AND = 1, para todos unos
Filter_OR  = 0, para todos ceros
Filter_AND = 0, uno o más ceros
Filter_OR  = 1, uno o más unos
```

Pueden representarse mediante una tabla de verdad:

Filter AND,	Filter OR	NEXTSTATE
1	1	1
1	0	Imposible (don't care)
0	0	0
0	1	ActualState (Don't change)

También pueden representarse las posibilidades mediante un Mapa de Karnaugh, de donde pueden sacarse las ecuaciones de salida:

```
                \Filter_AND,Filter_OR
                 \
     ActualState \ 00  01 |11  10 |
                   +---+---+---+---+
                 0 | 0 | 0 |(1 | -)|
                ---+---+---+---+---+
                 1 | 0 |(1 |(1)| -)|
                ---+---+---+---+---+
                       |       |

                     Z = NextState

NextState    = Filter_AND  |  ActualState  &  Filter_OR;

ChangeInData = ActualState ^  NextState;

// Variable interna, auxiliar:

ActualState  = NextState;
```

Además de las rutinas hay un programa de prueba. Las secuencias de prueba que han de llevar las variables han sido estudiadas con mucho cuidado; hay 33 valores de entrada que se comentan a continuación (note que la cantidad de valores de entrada debe ser mayor que **NFILTER** para apreciar una salida):

```
uint   Values[] = {
/* t00 */ 0x0182, // 1--- ---- (estos son 8 bits: 0x0082. Los 8 bits más altos
//                               representan valores de entrada espúreos)
/* t01 - t08 */ 0x0283, 0x0381, 0x0480, 0x0582,
                0x0683, 0x0781, 0x0880, 0x0982,

/* t09 */ 0x1080, // el cambio 1--- ---- en t0 es visto in t10, aún si después
/* t10 */ 0x1102, // ..se devuelve el cambio a 0: 0--- --1- aquí!
/* t11 */ 0x2203, // ---- ---1

/* t12 - t19 */ 0x3303, 0x4403, 0x5503, 0x6603,
                0x7703, 0x8803, 0x9903, 0xAA03,
```

```
/* t20 */ 0xBB03, // aquí se ve el cambio 0--- --1- de t10
/* t21 */ 0x0102, // aquí se ve el cambio ---- ---1 de t11; chg. a ---- ---0

/* t22 - t30 */ 0x0206, 0x0300, 0x0402, 0x0506,
               0x0602, 0x0700, 0x0806, 0x0902, 0x1000,

/* t31 */ 0x0102, // seen ---- ---0 value from t21
/* t32 */ 0x0000
```

El parámetro global del filtro es `#define NFILTER 10` (milisegundos), que puede cambiarse a voluntad. En el programa se anota la información que debería pasarse a un include file `"soe_filter.h"`.

Fíjense cómo el programa calcula el número de elementos con los que va a trabajar: `#define EX_SIZE (sizeof(Values)/sizeof(Values[0]))`; es el tamaño total de `Values`, en bytes, dividido en la cantidad de bytes que mide un elemento, el `Values[0]`.

El resultado de aplicar el algoritmo a ese conjunto de datos es el siguiente:

```
/* The above input values on this example will produce the following
 * results:
 *
 * Soe_Tst2 Luis G. Uribe C, L15E1: Soe Filtering Algorithm
 *
 * i = 00, Input = 0182, Output = 0000
 * i = 01, Input = 0283, Output = 0000
 * i = 02, Input = 0381, Output = 0000
 * i = 03, Input = 0480, Output = 0000
 * i = 04, Input = 0582, Output = 0000
 * i = 05, Input = 0683, Output = 0000
 * i = 06, Input = 0781, Output = 0000
 * i = 07, Input = 0880, Output = 0000
 * i = 08, Input = 0982, Output = 0000
 * i = 09, Input = 1080, Output = 0000
 * i = 10, Input = 1102, Output = 0080, Change = 0080
 * i = 11, Input = 2203, Output = 0080
 * i = 12, Input = 3303, Output = 0080
 * i = 13, Input = 4403, Output = 0080
 * i = 14, Input = 5503, Output = 0080
 * i = 15, Input = 6603, Output = 0080
 * i = 16, Input = 7703, Output = 0080
 * i = 17, Input = 8803, Output = 0080
 * i = 18, Input = 9903, Output = 0080
 * i = 19, Input = AA03, Output = 0080
 * i = 20, Input = BB03, Output = 0002, Change = 0082
 * i = 21, Input = 0102, Output = 0003, Change = 0001
 * i = 22, Input = 0206, Output = 0003
 * i = 23, Input = 0300, Output = 0003
 * i = 24, Input = 0402, Output = 0003
 * i = 25, Input = 0506, Output = 0003
 * i = 26, Input = 0602, Output = 0003
 * i = 27, Input = 0700, Output = 0003
 * i = 28, Input = 0806, Output = 0003
 * i = 29, Input = 0902, Output = 0003
 * i = 30, Input = 1000, Output = 0003
 * i = 31, Input = 0102, Output = 0002, Change = 0001
 * i = 32, Input = 0000, Output = 0002
 *
 */
```

Solo falta, en el código, extraer aquellos valores para los que hubo cambio, y agregarles el "time stamp".

BIBLIOGRAFÍA

Hamming, R. W., Coding and Information Theory, 2e, Prentice-Hall, 1986, p. 27

Verhoeff, J., Error Detecting Decimal Codes, Mathematical Centre Tract 29, The Mathematical Centre, Amsterdam, 1969.

http://www.augustana.ab.ca/~mohrj/algorithms/checkdigit.html, (Jonathan Mohr, 1999)

http://en.dahnielson.com/2006/09/verhoeff.html, Anders Dahnielson

http://mathworld.wolfram.com/DihedralGroupD5.html

http://mathworld.wolfram.com/DihedralGroup.html

Capítulo 5

MICROCONTROLADORES

"Neglected topics…"

INTRODUCCIÓN

DOS puntos de vista opuestos son tradicionales en nuestro ambiente, al momento de focalizar los objetivos que deben perseguirse en el estudio de la Arquitectura del Computador y sus *a lateres*; uno, pretende ***comprender*** los virtuosos alcances logrados, cada vez más esotéricos, y centran su atención en multiprocesadores y multicores, cachés de múltiples niveles y su coherencia; virtualización, y las profundidades de la segmentación, el pipelining; la ejecución de instrucciones fuera de secuencia, identificar de registros internos mediante sinónimos, temas todos esenciales a quienes deseen integrar el plantel de compañías como Intel.

Yo prefiero un enfoque más pragmático, centrado en la ***utilización*** que de esa tecnología podemos hacer, que potencie el estudio de las interfaces de entrada y salida (***I/O Interfacing***): Hardware y software; rutinas de interrupción, colas; reentrancia, recursividad y, en nuestro ámbito –que bastante difiere del Ingeniero de Computación, la programación en tiempo real, "debouncing", manejo de secuencias de eventos (SOE), la serialización de sucesos asíncronos, el modelaje de problemas como Máquinas de Estado; el handshaking entre periféricos, los filtros de entrada y salida (para señales analógicas, y digitales), el problema del "aliasing" y su solución moderna; fuentes de error de conversión analógico-digital (en ambos sentidos), estudio de periféricos de comunicación serial, I2C…

No lo tomen a mal; al autor le fascina el tema del diseño de computadoras (fue su tesis de grado) y se arroba ante la astucia de nuestros tiempos, pero en el aspecto didáctico prefiere el **pragmatismo**, que le asigna prelación al **uso apropiado de la tecnología**, previendo que a muy pocos de nosotros nos estará dado incursionar en la concepción y desarrollo de tan avanzados sistemas. Si bien es cierto que trabajando con FPGAs y similares siempre podremos incluir en ellos procesadores especiales, no es menos real reconocer que la cantidad de problemas que pueden resolverse con un microcontrolador es altísima. En su vida profesional, nuestros ingenieros enfrentarán 99 proyectos basados en un micro, contra 1 desarrollo sustentado en FPGA. Yo prefiero enseñar para esos 99.

Debo decir, a vía de ejemplo, que de todos los desarrollos en los que participé durante mi carrera solo **un (1)** diseño no se hizo con un microprocesador y requirió hardware especializado (una tarjeta de conversión ADC, para 32/64 canales, a suficiente velocidad como para hacer tambalear el bus de entrada y salida del Pentium). Todos los demás proyectos se centraron en un micro que, en mi caso, siempre fue un **80x86**-Pentium, configurado la mayor parte del tiempo como un "Credit Card PC", por alrededor de los U.S.$ 100 (un regalo).

Los libros de texto tienden también a soslayar conceptos básicos, importantísimos si las circunstancias dictan que debamos trabajar con microcontroladores en bajo nivel, lo que suele ocurrir cuando parte de nuestro desarrollo incluye dispositivos pequeños y económicos, como los de la familia PIC 16Fxxx (el 16F88, por ejemplo). A veces encontramos que el diseñador en realidad desconoce, por ejemplo, la diferencia entre hacer comparaciones numéricas con cantidades sin signo o entre números en complemento a dos. Él no distingue muy bien entre el Carry y el Overflow, ni sabe cómo y cuándo (y por qué) emplear el uno o el otro. Es claro que si trabajamos en lenguajes de alto nivel como el "C", lo cual es recomendable siempre que sea posible, el problema pierde relevancia pues el lenguaje y el compilador se encargan del asunto; si usamos el Assembler del Pentium o del HC08, tampoco es tan importante porque esos conjuntos de instrucciones son ricos e incluyen códigos especializados para cada tipo de dato, con y sin signo (signed, unsigned), pero en una buena parte de los diseños es fundamental diferenciar bien entre estos dos tipos de representación numérica.

La confusión reinante es tan grande a la hora de comprender cabalmente conceptos fundamentales, que una de las instrucciones más importantes del HC08 (**TAP**, **T**ransfer **A**ccumulator to **P**rocessor Status Word) tiene esta única anotación en el "*CPU08 Central Processor Unit, Reference Manual, CPU08RM/AD, Rev. 3, 2/2001*", pág. 219:

"Note: The TAP instruction was added to improve testability of the CPU08 and so, few practical applications of the instruction exist."

También en el mismo manual, pág. 50:

"In rare cases it may be useful to allow an interrupt routine to be interrupted. However, nesting is discouraged because it greatly complicates a system and rarely improves system performance."

El anónimo personaje perpetrador de tan infames párrafos, incluidos en el Manual de Referencia de tan prestigiosa y avanzada empresa, ***no tiene ni la menor idea de lo que dice***. Uno de los ejercicios que incluyo en este capítulo está concebido para poder medir tiempos (por interrupciones) ¡***dentro*** de una rutina de interrupción! (al hacer el debouncing de la entrada "INT" [en su rutina de interrupciones] empleando el timer [interrupción anidada]). Esto, como se verá, difícilmente puede realizarse sin hacer caso omiso a los párrafos arriba resaltados.

CONDITION CODE REGISTER: C, Z, N y V

COMPARAR ES RESTAR

Los micros más pequeños ni siquiera incluyen instrucciones para comparar 2 cantidades. Supongamos que deseamos tomar decisiones del tipo:

`A>B, A==B y A<B`, y las compuestas: `A>=B, A!=B y A<=B` ①

en un indefenso microcontrolador como el 16F88, cuyo conjunto de instrucciones es uno en el que precisamente **no** figuran operaciones de comparación. Lo primero es notar que este grupo de relaciones puede reescribirse de la siguiente manera, sin alterar su significado:

`A-B>0, A-B==0, A-B<0, A-B>=0, A-B!=0 y A-B<=0` ②

El cambio es conveniente porque en un procesador siempre podemos restar, bien sea mediante el correspondiente código de substracción (casi todos en la tierra, hasta el PIC), o sumando al minuendo el negativo del sustraendo. Además, todos los computadores tienen un indicativo, **Z** (cero), almacenado como un bit del registro **CCR** (Condition Code Register), **PSW** (Processor Status Word), **Flags** o similar. Cuando **Z** vale 1 significa que el resultado de la última operación fue **cero**; de lo contrario **Z** vale 0. El bit **Z** resuelve directamente las comparaciones por igualdad: `A-B==0` y `A-B!=0`.

Para establecer las demás relaciones hace falta más información. Ahora, piense que la resta indicada en ② tiene diferentes significados si las dos cantidades involucradas tienen signo o no, y que la mecánica involucrada en ambos casos es diferente. Por eso el resto del desarrollo lo estableceremos en dos partes, una para cantidades que no tienen signo (ordinales, índices, direcciones de memoria), y otra para las que sí tienen signo (cardinales, variables físicas, cuentas contables), y que están codificadas en complemento a dos (2´s complement).

La nomenclatura convencional emplea las siguientes relaciones (en inglés), que se expresan primero para cantidades SIN signo, y se hace un parangón con las informaciones CON signo:

- Unsigned: Higher than. Signed, Greater than.
- Unsigned: Higher than or Same.
  Signed, Greater than or Equal to.
- Unsigned: Lower than. Signed: Less than.
- Unsigned: Lower than or Same.
  Signed: Less than or Equal to.
- Unsigned: Same (it could use "Equal" too).
  Signed: Equal
- Unsigned: Different. Signed: Not Equal.

Se emplea "superior" (higher) para "unsigned", y "mayor" (greater) para "signed". Algo similar ocurre con "inferior" (lower) para "unsigned", y "menor" (less) para "signed". Es fundamental no confundir la terminología, o los resultados ***pueden*** resultar erróneos. Si emplea C, las relaciones se escriben igual para ambos tipos de datos (A<B, por ejemplo); el compilador sabe si usar una clase de instrucción de la máquina, o la otra, de acuerdo al tipo asociado (signed, unsigned) a las variables. En un programa en C, uno normalmente lee siempre "A<B" como; "A menor que B", sin importar el tipo…

UNSIGNED

Necesitamos al menos tres ejemplos para mostrar cómo funciona la comparación, o resta, al determinar

```
A-B>0, A-B==0, A-B<0
```

Los otros casos son combinaciones lógicas que incluyen mezclas de las anteriores (mayor o igual, etc). Por simplicidad emplearemos cantidades que puedan representarse en dos (2) bits.

1) Comparar 2 con 1; ver cómo se comporta la <u>resta</u> y qué produce que nos indique que 2 > 1:

```
                    Resta
Borrows  0   -1
                 1   |  0  ||  +2
             -   0   |  1  ||  -1
             ⓪   0   |  1  ||  +1      Z == 0
```

Lo primero es decir que no hubo borrow final (**Borrow==0** en la columna izquierda, la del tercer bit). Si **A >= B** <u>no</u> hay que pedir prestado nada a la izquierda y, por tanto, B==0 indica que A >= B. Como el resultado no fue cero (fue +1), entonces el bit **Z** vale 0. Esta bandera descarta que A sea igual a B; por tanto entre las dos condiciones, **Borrow ==0 y Z==0**, indican a las claras que **A>B**.

Hay que indicar también que todos los procesadores modernos codifican los enteros y las cantidades de punto fijo, en complemento a 2, y los que tienen código de operación para la resta, internamente la ejecutan sumando as minuendo el negativo (en complemento a 2) del sustraendo. ¿Cómo podemos estar hablando de representar números negativos, si acabamos de decir que estamos en la sección en que las cantidades no poseen signo (unsigned)? Bueno; la contradicción consiste en que, si no hubiera restas, no habría necesidad de signos. El signo se necesitó en la aritmética cuando comenzaron a hacerse restas, en donde a una cierta cantidad se le trataba de quitar una más grande. Obviamente eso no es posible si no se incluye el signo. Así que, si hay instrucción de resta, hay signo involucrado, queramos o no.

En este caso, lo que el hardware ejecuta en realidad, en vez de la resta señalada, es la siguiente suma:

```
              2's complement Add
Carries  1
                 1   |  0  ||  +2
             +   1   |  1  ||  +(-1)
             ①   0   |  1  ||  +1      Z == 0
```

Observe que si se usan sin modificar los mismos "full-adders" para hacer la suma del negativo del sustraendo: +(-1), el resultado no es el Borrow, sino el Carry final, QUE ES EL NEGADO DEL BORROW que resultó al hacer la resta a mano. Por eso se dice que los circuitos sumadores completos entregan: Carry/Borrow negado (C/~B).

Ahora el indicativo de que 2 >1, es entonces que **C == 1 y Z==0**.

2) Comparar 2 con 2; ver cómo se comporta la <u>resta</u> y qué produce que nos indica que 2 == 2:

			Resta		
Borrows	*0*				
		1	0	+2	
	-	1	0	-2	
	⓪	0	0	+0	Z == 1

Hay que indicar que aquí tampoco hubo borrow final (**Borrow==0** en la columna izquierda, la del tercer bit). Es obvio que si **A == B**, no es necesario pedir prestado nada a la izquierda y, por tanto, Borrow==0. Sin embargo, lo que en realidad indica que A == B es el hecho de que **Z==1**. Eso es suficiente.

Otra vez vemos que lo que el hardware ejecuta es la siguiente suma, en vez de la resta señalada:

			2's complement Add		
Carries	*1*				
		1	0	+2	
	+	1	0	+(-2)	
	①	0	0	+0	Z == 1

C==1 (negado del Borrow de la tabla anterior), pero es **Z==1** lo que señala que las dos cantidades son iguales.

3) Por último, comparar 2 con 3; determinar cómo la <u>resta</u> nos indica que 2 < 3:

			Resta		
Borrows	*-1*	*-1*			
		1	0	+2	
	-	1	1	-3	
	①	1	1	-1	Z == 0

En primer lugar vemos que ahora sí hubo borrow final (**Borrow==1** en la columna izquierda, la del tercer bit). Si **A < B** <u>sí</u> hay que pedir prestado a la columna de la izquierda y, por tanto, B==1 indica que A < B. Como el resultado no fue cero (fue -1), entonces el bit **Z** vale 0, pero esta bandera no se necesita, ya que siempre que A sea menor que B, el que **Borrow==1**, indican a las claras que **A<B** (A menor que B).

Como indicamos antes, en este caso, unsigned o no unsigned, el resultado es negativo. Así que una manera que algunos usan para ver la última fila de la tabla anterior, es como si los números en realidad tuvieran tres (3) bits en la representación (habíamos dicho que eran solo dos), y que la tercera columna (la del **Borrow**) corresponde al signo (que era positivo, o cero, en todas partes, menos ahora que dio una cantidad negativa). Y podemos constatar que el negativo de ①11 es ⓪01, es decir, que el resultado fue **<u>menos 1</u>**.

Resumamos diciendo que si el resultado es negativo (**Borrow==1**), **A<B**.

Y si el resultado es positivo (**Borrow==0**), entonces **A>B** o **A==B**, y el discriminador ahora es:
Si **Z==1**, **A==B**; si **Z==0**, **A>B**.

Para ver otra vez lo que el hardware realiza (una suma, en vez de la resta señalada), tenemos:

2's complement Add

Carries				
	1	0	+2	
+	0	1	+(-3)	
©	1	1	-1	Z == 0

C==0 (negado del Borrow de la tabla anterior); eso indica, *per se*, que **A<B**.

Estas condiciones pueden incorporarse en un Mapa de Karnaugh (MK); como el Borrow no existe, sino el carry, según acabamos de indicar, las variables que definen las entradas al MK son C y Z:

C \ Z	0	1
0	A>B	A==B
C 1	A<B	-X-

Así, las ecuaciones que resultan para comparar números CON signo, son las siguientes:

- A>B = C' * Z'
- A==B = Z
- A!=B = Z'
- A<B = C
- A>=B = C'
- A<=B = C + Z

Una nota adicional, en relación a la familia PIC; Microchip suministra el Carry **negado** (el Borrow original de nuestros ejercicios); por tanto, en el Mapa de Karnaugh para los microcontroladores 16F88, por ejemplo, hay que invertir las dos filas del MK de arriba. El manual de Microchip indica que, para la operación de resta, el Carry asume el valor de lo que ellos denominan: Signo del resultado (exactamente **C'** de nuestro desarrollo). Intel, Motorola, IBM (mainframes), DEC, y todos los demás procesadores que conozco, ***excepto el PIC***, utilizan la tabla que acabamos de desarrollar. Desde luego, en las ecuaciones hay que intercambiar **C** por **C'** para el PIC.

SIGNED

Establezcamos las relaciones necesarias para `A-B>0, A-B==0, A-B<0`, cuando la información tiene signo. Los ejemplos los representaremos con 3 bits, siendo el más de la izquierda el del signo. La información, como ya se dijo, se representa en complemento a dos. En este caso haremos directamente lo que el hardware realiza, es decir, en vez de restar, se suma al minuendo el negativo del sustraendo:

1) Comparar 2 con 1; ver cómo se comporta la resta y qué produce que nos indique que 2 > 1:

```
                2's complement Add
                S:
               Sign
Carries        1
                0   1   0   +2
          +     1   1   1   +(-1)
          0     0   0   1   +1      Z == 0
```

Ahora sí, la tercera columna es el signo con propiedad. Si el resultado de la resta es **S==0** (positivo), significa que **A >= B**. Como **Z==0**, necesariamente la respuesta es que **A >=B**.

Ahora el indicativo de que 2 >1, es entonces que **S == 0 y Z==0**.

Comparemos -2 con -3; tendría que dar igual -2 > -3:

```
                2's complement Add
                S:
               Sign
Carries        1
                1   1   0   -2
          +     0   1   1   -(-3): +3
          1     0   0   1   +1      Z == 0
```

S == 0 y Z==0 indican de que -2 >-1.

Uno más: Comparemos 1 con -1; tendría que dar igual, que 1 > -1:

```
                2's complement Add
                S:
               Sign
Carries        1
                0   0   1   +1
          +     0   0   1   -(-1): +1
          1     0   1   0   +2      Z == 0
```

S == 0 y Z==0 indican de que 1 >-1.

Ahora, cuando trabajamos con números con signo, puede ocurrir un fenómeno nuevo. Si sumamos dos números positivos, y el resultado es negativo, o si sumamos dos números negativos, y el resultado es positivo, esto es un indicativo de que se produjo una cifra tal, que no puede representarse con el número de bits asignados (normalmente, 8, 16, 32, 64, en las máquinas actuales). Este inconveniente se denomina OVerflow, y en casi todos los micros (***excepto en el PIC***), existe una variable **V**, que vale 1 si hubo overflow, lo que indica además, que el signo actual del resultado está mal. Esto es conveniente, porque si tenemos el signo **N**, y sabemos que está mal (**V==1**), entonces podemos averiguar cuál era el signo correcto del resultado, que es lo que nos indica si A>B o no. La corrección se hace así: Si V=0, S=N; pero si V=1, S=N' (lo contrario de lo que tenemos).

La ecuación para S, en términos de N (signo obtenido del resultado, *sin corregir*), y el V, es la siguiente:

```
S = N ^ V      (^ es XOR)
```

Y **A>B** = **S' * Z'**, que se corrige como: **A>B** = **(N ^ V) ' * Z'**

Un ejemplo con Overflow: Comparemos 3 con -3; tendría que dar igual, que 3 > -3:

```
          2's complement Add
          S:
         Sign
Carries  1   1
          0   1   |  1  |  +3
     +    0   1   |  1  |  -(-3): +3
     0    1   1   |  0  |  -2      Z == 0
```

V==1 (suma de dos números positivos, con resultado negativo; 3+3 dio -2 cuando ha debido ser +6, que no puede representarse en dos bits de mantisa y uno de signo)

Así que **(N ^ V) ' * Z' = (1 ^ 1) ' * Z' = 1**; por tanto, 3>-3.

Note que el resultado real (-6) no nos interesa para la comparación; sólo el signo del resultado, que ya sabemos cómo corregirlo.

2) Comparar 2 con 2; ver cómo se comporta la resta y qué produce que nos indica que 2 == 2:

```
          2's complement Add
          S:
         Sign
Carries  1
          0   1   |  0  |  +2
     +    1   1   |  0  |  +(-2)
     1    0   0   |  0  |  +0      Z == 1
```

Como en el caso unsigned, lo importante aquí es que **Z==1**; por tanto las dos cantidades son iguales. Cuando Z==1, no miramos el signo (que siempre es positivo); por tanto no hay que corregir nada (no hay overflow cuando el resultado es cero). Al comparar por igualdad, restando, da lo mismo que los números tengan signo o no.

3) Por último, comparar 2 con 3; determinar cómo la resta nos indica que 2 < 3:

```
          2's complement Add
          S:
         Sign
Carries
          0   1   |  0  |  +2
     +    1   0   |  1  |  -(+3): +(-3)
     0    1   1   |  1  |  -1      Z == 0
```

A<B = S, o sea que corregido queda: **A<B = N ^ V**

En nuestro ejemplo: **N==1**; **V == 0**; por tanto S==1 ^ 0 = 1; por tanto, 2<3.

Estas condiciones pueden incorporarse en un Mapa de Karnaugh (MK); las variables que definen las entradas al MK son S y Z:

S \ Z	0	1
0	A>B	A==B
1	A<B	-X-

Y S = N ^ V (^ es XOR)

Así, las ecuaciones que resultan para comparar números CON signo, son las siguientes:

- A>B = (N ^ V)' * Z'
- A==B = Z
- A!=B = Z'
- A<B = (N ^ V)
- A>=B = (N ^ V)'
- A<=B = (N ^ V) + Z

Los PIC solo tienen el Carry (negado) y el Z; por tanto, para realizar comparaciones con cantidades **signed** hay que averiguar **N**, el signo del resultado (que probablemente queda en el registro W, y por tanto es sencillo de encontrar: es el bit7 de W) y hay que calcular **V**, lo cual es un poco más complejo: $\mathbf{V=(S_A \wedge S_B)' * (S_A \wedge S_R)}$, que es la manera de expresar, en álgebra booleana, que el signo de A es igual al de B y distinto del signo del Resultado.

COMPARACIONES PARA EL PIC

La siguiente librería implementa con **MACROS** todos los saltos, con signo y sin signo, para la familia PIC **16Fxxx**, pues el método nativo es muy primitivo, incluye solo comparaciones sin signo, dado que no tiene el bit de overflow, **V**

```
;-----------------------------------------------------------------
; All Branches`.inc, Ing. Luis G. Uribe C., S16D6 D01L7 (V1.4)
; .. S30J7 (V1.4)    Added NEG (Negative A), NEGW (Negative W);
; ..                 .. CBGELZ, CBGTLZ, CBLELZ, CBLTLZ (w/Zero)
;
; .. D24D6 (V1.3b)   ***_NOTE_***: Signed branches are NOT REENTRANT**
;                    *IRS should save '_BR' if Sig. Br. are in use ! *
;                    ************************************************
;
; .. S16D6 (V1.3)   Included 'local' labels. Included _BR tmp variable
; .. S09D6 (V1.2c)   Some Cosmetics; CBGE/CBGEL, BHIS, simplified
; .. D04J6 (V1.2b)   Added MOVFF, MOVLF, TST, LOOP, LOOPI,
;                    .. BTFJS (btfBs), BTFJC (btfBc), BTFCS, BTFCC,
;                    .. WAITBS, WAITBC, SAVREGS, RSTREGS
; .. L02E6 (V1.1)
; .. D11D2005 (V1.0)
;
;     CMP    AF, BF
;     CMPL   AF, L
;     CMPLI  L,  AF   (if needed...)
;
;     BCC    Address    CBGE   AF, BF, Address
;     BCS    Address    CBGEL  AF, L,  Address    CBGELZ  AF, Address
;     BEQ    Address    CBGT   AF, BF, Address
;     BHI    Address    CBGTL  AF, L,  Address    CBGTLZ  AF, Address
;     BHIS   Address    CBLE   AF, BF, Address
;     BLO    Address    CBLEL  AF, L,  Address    CBLELZ  AF, Address
;     BLOS   Address    CBLT   AF, BF, Address
;     BNE    Address    CBLTL  AF, L,  Address    CBLTLZ  AF, Address
;
;-----------------------------------------------------------------
; CMP   AF, BF (compare two registers)
; CMPL  AF, L  (compare register and  literal)
; CMPLI AF, L  (compare literal  and  register [if needed...])
;
; Establish: AF > BF,  AF = BF  or  AF < BF, (for CMP),  or
;            AF > L,   AF = L   or  AF < L   (for CMPL), or
;            L  > AF,  L  = AF  or  L  < AF  (for CMPLi)
;
;                (IN THIS DUE ORDER!)
;
;
; The STATUS FLAGS C(b0) and Z(b2), FOR *SUBTRACTION* (i.e: CMP) are:
;
;                 |   Z
;     C\Z   0     |   1
;       +---------+---------+
;       | A<B(W)  |         |
;      0|   LO    |   ---   |
;    ---+---------+---------+
;      1| A>B(W)  | A=B(W)  |
;    C  |   HI    |   EQ    |
;    ---+---------+---------+
;
; (NOT so clearly shown as above, in Microchip's literature...)
;
;-----------------------------------------------------------------
; UNSIGNED Branches:
; -----------------
;
; 1) Use one of the CMP Macros to set the Condition Codes as follow:
;    CMP   AF,BF compare  2  REGISTERS
;    CMPL  AF,L  compare one REGISTER and a Literal
;    CMPLI L,AF  compare one Literal  and a REGISTER (in this order)
;
;    TST   AF    compare one REGISTER against '0'
;
; 2) Then, use Macro to branch (if cmp was: CMP AF,BF, this applies):
;    BEQ  (Branch if AF  EQual  BF),
;    BNE  (Branch if AF  Not Equal  to BF);
;    BHI  (Branch if AF  HIgher  than BF),
;    BLOS (Branch if AF  LOwer or Same  than BF);
;    BLO  (Branch if AF  LOwer  than BF),
;    BHIS (Branch if AF  HIgher or Same  than BF).
;
; EXAMPLE:
;
;    CMP REGA, LIMIT
;    BLO Loop    ; REGA < LIMIT
; ...
;
;
; *SIGNED* Branches:
; -----------------
;
;   Use the Integrated "Compare And Branch instruction", as follows:
;
;
```

```
; 1) Compare two (2) Registers:
;
;CBGT AF,BF,ADD ;Cmp AF with BF; Branch to ADDR if AF Greater Than BF
;CBLT AF,BF,ADD ;Cmp AF with BF; Branch to ADDR if AF Less    Than BF
;CBLE AF,BF,ADD ;Cmp AF with BF; Branch to ADDR if AF Less    than ..
;      ... or Equal to BF
;CBGE AF,BF,ADD ;Cmp AF with BF; Branch to ADDR if AF Greater than ..
;      ... or Equal to BF
;
;      BEQ and BNE are the same Unsigned macros. Use them (and CMP).
;
;
; 2) Compare Register and Literal:
;
;CBGTL AF,L,ADD ;Cmp AF with L;  Branch to ADDR if AF Greater Than L
;CBLTL AF,L,ADD ;Cmp AF with L;  Branch to ADDR if AF Less    Than L
;CBLEL AF,L,ADD ;Cmp AF with L;  Branch to ADDR if AF Less     than
...
;      ... or Equal to L
;CBGEL AF,L,ADD ;Cmp AF with BF; Branch to ADDR if AF Greater than
...
;      ... or Equal to L
;
;
; EXAMPLES:
; ========
;
;   CBLT REGA, LIMIT, Loop ; Branch if REGA < LIMIT  (2's complement)
; ...
;
;   CBLTL REGA,LITLIM,Loop ; Branch if REGA < LITLIM (2's complement)
; ...
;
;=================================================================
; Macro Definitions: COMPARE (for Unsigned numbers):
;
;    CMP   (Register to Register),
;    CMPL  (Register to Literal),
;    CMPLI (Register to Literal, Inverted)
;
;=================================================================
; Reserve local variables

      CBLOCK
          _BR ; ** IRS should SAVE '_BR' if using Signed Branches!**
      ENDC

;-----------------------------------------------------------------
TST    MACRO   A                ;  Compare REGISTER against '0'
       movf    A, F

       ENDM

;-----------------------------------------------------------------
CMP    MACRO   A, B             ; Compare AF to BF

       movfw   B
       subwf   A, w             ; Compare A, B(w)

       ENDM

;-----------------------------------------------------------------
CMPL   MACRO   A, L             ; Compare AF to Literal

       movlw   L
       subwf   A, w             ; Compare A, L(w)

       ENDM

;-----------------------------------------------------------------
CMPLI  MACRO   L, A             ; Compare Literal to AF

       movfw   A
       sublw   L

       ENDM

;=================================================================
; Unsigned Branches, SIMPLE:
;
;   BEQ (Z == 1), BHI (C == 1 && Z == 0), BLO (C == 0)
;
;-----------------------------------------------------------------
BEQ    MACRO   Address          ; Branch if EQuals

       btfsc   STATUS, Z        ; goto Address if Z == 1
       goto    Address

       ENDM

;-----------------------------------------------------------------
BHI    MACRO   Address          ; Branch if Higher

       local   EQ

       btfsc   STATUS, Z        ; goto EQ
       goto    EQ               ; .. if Z = 1

       btfsc   STATUS, C        ; goto Address
```

```
        goto    Address         ; .. if C === 1 (HI, && Z == 0)
EQ:
        ENDM

;-----------------------------------------------------------------
BLO     MACRO   Address         ; Branch if Lower

        btfss   STATUS, C       ; goto Address if C == 0
        goto    Address

        ENDM

;-----------------------------------------------------------------
; Unsigned Branches, COMPOSED:
;
;   BNE (Z == 0), BLOS (C == 0 || Z == 1), BHIS (C == 1)
;
;-----------------------------------------------------------------
BNE     MACRO   Address         ; Branch if Not Equals

        btfss   STATUS, Z       ; goto Address if Z = 0
        goto    Address

        ENDM

;-----------------------------------------------------------------
BLOS    MACRO   Address         ; Branch if LOwer or Same

        btfsc   STATUS, Z       ; goto Address if Z == 1 (same)
        goto    Address

        btfss   STATUS, C       ; .. or C == 0 (lower)
        goto    Address

        ENDM

;-----------------------------------------------------------------
BHIS    MACRO   Address         ; Branch if HIgher or Same

        btfsc   STATUS, C       ; goto Address if C == 1 (higher/same)
        goto    Address

        ENDM


;-----------------------------------------------------------------
; Ancillary Branches:
;
;   BCS (C == 1), BCC (C == 0)
;
;   (Usefull to signal Return Status from subroutines, via C bit)
;
;-----------------------------------------------------------------
BCC     MACRO   Address         ; Branch if Carry Clear

        btfss   STATUS, C       ; goto Address if C == 0
        goto    Address

        ENDM

;-----------------------------------------------------------------
BCS     MACRO   Address         ; Branch if Carry Set

        btfsc   STATUS, C       ; goto Address if C == 1
        goto    Address

        ENDM

;*****************************************************************
;                HOW THE SIGNED BRANCHES WORK?
;
; They follow a simple, not very well known, trick.
;
; As an example: for 3 bits signed numbers, in 2's complement, the
; .. following HIGH TO LOW ordering applies (Table 1). Obviously,
; .. for the negative numbers, even if they are LESS THAN the possi-
; .. tive ones, THEY ALL ARE HIGHER THAN THEM (if seen as unsigned!)
;
; This has been a very well known fact over the years, and this
; .. caused the adoption of two more bits to code the Condition
; .. Codes: oVerflow bit  and  Negative bit
;
; But what may be done in microcontrolers like the PIC, than only
; .. have two flags: the Z and the C bits, for the Condition Codes?
;
;
; URIBE'S OBSERVATION:
; ====================
;
; Recode the signed, cumbersome numbers into Excess 2^n (add +4 to
; .. all figures in the example; See Table 2). Now, you can see that
; .. the new table exhibit the high to lower order used in the
; .. UNSIGNED representation; so, you can APPLY TO THEM THE UNSIGNED
; .. BRANCHES of the PIC (the code shown above), and the answers
; .. will be correct, following the Signed 2's complement order!
;
; NOTE: To Recode a number you only need to CHANGE the sign bit. If
; .. it is 0, turn it to 1, and if it is 1, turn it to 0.
;
; You don't even need to check the sign bit; just only make the EXOR
```

```
; ..between the 8 bit number and the literal b'10000000'. THAT'S ALL!
;
;                    Table 1   Table 2
;                   --------  --------
;                    3: 011    3: 111
;                    2: 010    2: 110
;                    1: 001    1: 101
;                    0: 000    0: 100
;                   -1: 111   -1: 011
;                   -2: 110   -2: 010
;                   -3: 101   -3: 001
;                   -4: 100   -4: 000
;
; Ing. Luis G. Uribe C., May 2005, December 2005
;
;=====================================================================
; Signed Branches, SIMPLE:    (BEQ & BNE are same as for unsigned)
;
;   CBGT (C == 1 && Z == 0, after recoding)
;        Negated condition: Do NOT jump on:  C == 0 || Z == 1,
;   CBLT (C == 0, after recoding)
;        Negated condition: Do NOT jump on:  C == 1
;
;---------------------------------------------------------------------
CBGT    MACRO   A, B, Address    ; Compare and Branch if Greater Than

        local   EQ

        movfw   A                ; Recode A (to tmp: _BR)
        xorlw   b'10000000'
        movwf   _BR
        movfw   B
        xorlw   b'10000000'      ; Recode B (inside W)
        subwf   _BR, w           ; Compare A(_BR), B(w)
        btfsc   STATUS, Z        ; goto EQ
        goto    EQ               ; .. if Z == 1 (eq)

        btfsc   STATUS, C        ; goto Address
        goto    Address          ; .. if C == 1 (GT)
EQ:
        ENDM

;---------------------------------------------------------------------
CBLT    MACRO   A, B, Address    ; Compare and Branch if Less Than

        movfw   A                ; Recode A (to tmp: _BR)
        xorlw   b'10000000'
        movwf   _BR
        movfw   B
        xorlw   b'10000000'      ; Recode B (inside W)
        subwf   _BR, w           ; Compare A(_BR), B(w)
        btfss   STATUS, C        ; goto Address
        goto    Address          ; .. if C == 0 (lt)

        ENDM

;---------------------------------------------------------------------
; SIGNED BRANCHES: COMPOSED:    (EQ is same as for unsigned)
;
;   CBLE (C == 0 || Z == 1, after recoding),
;   CBGE (C == 1, after recoding)
;
;---------------------------------------------------------------------
CBLE    MACRO   A, B, Address    ; Compare & Branch if Less Than or
Equal

        movfw   A                ; Recode A (to tmp: _BR)
        xorlw   b'10000000'
        movwf   _BR
        movfw   B
        xorlw   b'10000000'      ; Recode B (inside W)
        subwf   _BR, w           ; Compare A(_BR), B(w)
        btfsc   STATUS, Z        ; goto Address
        goto    Address          ; .. if Z == 1 (eq)

        btfss   STATUS, C        ; goto Address
        goto    Address          ; .. if C == 0 (lt)

        ENDM

;---------------------------------------------------------------------
CBGE    MACRO   A, B, Address    ; Cmp & Branch if Greather Than or
Equal

        movfw   A                ; Recode A (to tmp: _BR)
        xorlw   b'10000000'
        movwf   _BR
        movfw   B
        xorlw   b'10000000'      ; Recode B (inside W)
        subwf   _BR, w           ; Compare A(_BR), B(w)
        btfsc   STATUS, C        ; goto Address
        goto    Address          ; .. if C == 1 (ge)

        ENDM

;=====================================================================
; SIGNED BRANCHES with Literal; SIMPLE:
;
;   CBGTL (C == 1 && Z == 0, after recoding),
```

```
;         Negated condition: Do NOT jump on:  C == 0 || Z == 1,
;   CBLTL (C == 0, after recoding)
;         Negated condition: Do NOT jump on:  C == 1
;
; Following Literal macros assume comparison between a Register and a
; .. literal.
;
; (SIGNED BRANCHES to compare between a Literal  and  a Register
; ... are left as an exercise to the student, if needed).
;
; Ing. Luis G. Uribe C.
;
;-----------------------------------------------------------------

CBGTL  MACRO   A, L, Address    ; CBGT Literal

       local   EQ

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   L
       xorlw   b'10000000'      ; Recode L (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfsc   STATUS, Z        ; goto EQ
       goto    EQ               ; .. if Z == 1 (eq)

       btfsc   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 1 (GT)
EQ:
       ENDM

;-----------------------------------------------------------------
CBGTLZ MACRO   A, Address       ; CBGT w/Zero

       local   EQ

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   b'10000000'      ; Zero; Recode Zero (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfsc   STATUS, Z        ; goto EQ
       goto    EQ               ; .. if Z == 1 (eq)

       btfsc   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 1 (GT)
EQ:
       ENDM
```

```
;-----------------------------------------------------------------
CBLTL  MACRO   A, L, Address    ; CBLT Literal

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   L
       xorlw   b'10000000'      ; Recode B (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfss   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 0 (lt)

       ENDM

;-----------------------------------------------------------------
CBLTLZ MACRO   A, Address       ; CBLT w/Zero

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   b'10000000'      ; Zero; Recode Zero (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfss   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 0 (lt)

       ENDM

;-----------------------------------------------------------------
; SIGNED BRANCHES; Compare Register to Literal: COMPOSED:
;
;   CBLEL (C == 0 || Z == 1, after recoding),
;   CBGEL (C == 1, after recoding)
;
;-----------------------------------------------------------------
CBLEL  MACRO   A, L, Address    ; CBLE Literal

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   L
       xorlw   b'10000000'      ; Recode B (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfsc   STATUS, Z        ; goto Address
       goto    Address          ; .. if Z == 1 (eq)

       btfss   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 0 (lt)

       ENDM
```

```
;-----------------------------------------------------------------
CBLELZ MACRO   A, Address       ; CBLE w/Zero

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   b'10000000'      ; Zero; Recode Zero (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)
       btfsc   STATUS, Z        ; goto Address
       goto    Address          ; .. if Z == 1 (eq)

       btfss   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 0 (lt)

       ENDM


;-----------------------------------------------------------------
CBGEL  MACRO   A, L, Address    ; CBGE Literal

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   L
       xorlw   b'10000000'      ; Recode B (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)

       btfsc   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 1 (ge)

       ENDM

;-----------------------------------------------------------------
CBGELZ MACRO   A, Address       ; CBGE w/Zero

       movfw   A                ; Recode A (to tmp: _BR)
       xorlw   b'10000000'
       movwf   _BR
       movlw   b'10000000'      ; Zero; Recode Zero (inside W)
       subwf   _BR, w           ; Compare A(_BR), L(w)

       btfsc   STATUS, C        ; goto Address
       goto    Address          ; .. if C == 1 (ge)

       ENDM

;=================================================================
; SIGNED BRANCHES to compare between a Literal  and  a Register (in
; ... this order) are LEFT AS AN EXERCISE TO THE STUDENT (if needed).
;
;-----------------------------------------------------------------


;=================================================================
LOOP   MACRO   A, Address       ; Dec as Loop Control

       DecFsZ  A, F
       goto    Address

       ENDM

;-----------------------------------------------------------------
LOOPI  MACRO   A, Address       ; Inc as Loop Control

       IncFsZ  A, F
       goto    Address

       ENDM

;-----------------------------------------------------------------
MOVFF  MACRO   A, B             ; Move File to File

       movw    A, W
       movwf   B

       ENDM

;-----------------------------------------------------------------
MOVLF  MACRO   L, A             ; Move Literal to File

       movlw   L
       movwf   A

       ENDM

;-----------------------------------------------------------------
BTFJS  MACRO   F, B, Address    ; Bit File Test and Jump if Set

       btfsc   F, B
       goto    Address

       ENDM

#define BTFBS  BTFJS            ; Bit File Test and Branch if Set

;-----------------------------------------------------------------
BTFJC  MACRO   F, B, Address    ; Bit File Test and Jump if Clear

       btfss   F, B
```

```
        goto    Address

        ENDM

#define BTFBC  BTFJC            ; Bit File Test and Branch if Clear

;-----------------------------------------------------------------------
BTFCS   MACRO   F, B, Address   ; Bit File Test and CALL if Set

        btfsc   F, B
        call    Address

        ENDM

;-----------------------------------------------------------------------
BTFCC   MACRO   F, B, Address   ; Bit File Test and CALL if Clear

        btfss   F, B
        call    Address

        ENDM

;-----------------------------------------------------------------------
WAITBS  MACRO   F, B            ; Wait for Bit Set

        local   loop

loop:   btfss   F, B
        goto    loop

        ENDM

;-----------------------------------------------------------------------
WAITBC  MACRO   F, B            ; Wait for Bit Clear

        local   loop

loop:   btfsc   F, B
        goto    loop

        ENDM

;=======================================================================
SAVREGS MACRO  WSAVE, PSWSAVE   ; Save W & PSW for ISR

    movwf   WSAVE           ; Save off current W register contents
                                ; ..MOVWF *NEVER* change STATUS
    movfw   STATUS          ; Move status register into W register
                                ; ..MOVFW change STATUS, but AFTER
                                ; ..storing it into W
    movwf   PSWSAVE         ; Save off contents of STATUS register
    bcf     STATUS, RP0     ; Select Bank 0... (by default)

        ENDM

;-----------------------------------------------------------------------
RSTREGS MACRO  WSAVE, PSWSAVE   ; Restore saved W & PSW for RFIE

        movfw   PSWSAVE         ; Retrieve copy of STATUS register
        movwf   STATUS          ; ..restore pre-isr STATUS contents
                                ; ..MOVWF *NEVER* change STATUS
        swapf   WSAVE, F        ; restore pre-isr W register contents
        swapf   WSAVE, W        ; .. Swap does not change Status flags
        retfie                  ; return from int. & int. enable

        ENDM

;=======================================================================
NEG     MACRO   A               ; Negative A

        comf    A,F
        incf    A,F

        ENDM

;-----------------------------------------------------------------------
NEGW    MACRO

        XorLW   0xFF
        AddLW   d'1'

        ENDM
```

LIBRERÍA DE TIMERS PARA HC08

La siguiente librería implementa una batería de 8 timers, por interrupciones, para el Freescale HC908, bajo WinIDE.

```
; Timers8E.inc: Luis G. Uribe C., Basic 8 Timers Support
;   L24N3 L08N4 J01D5 M27F7 V09M7 D08L7
; D08L7: Included TimeOut macro for semantics enhacement.
; M27F7: Init8MhzClock run 7,987,200 Hz instead 8Mhz, to fit 9600 bps
; ..in SCI
; J01D5: ISR is INTERRUPTIBLE from any other devices, despite the
; ..wrong advise from Motorola handbooks, seen in class.
; ..As ISR is non reentrant, Auto-disable TBM interrupts before CLI

;-----------------------------------------------------------------
; DATA DEFINITION.
; Define 8 timers Global Variables
; Note: Macro directives (DW, $IF...) cannnot go into macros... GOK!
;
; NOTE: REQUIRES  'org RAMstart' **BEFORE**  $include 'timers8C.inc'
; ..in main program, to make this data definitions work!
;
; All following variables  *Must*  reside on ZERO page!

Timer0          DS  2
Timer0.Rdy      EQU 0                   ; ..Bit #0; for brclr & brset
Timer1          DS  2
Timer1.Rdy      EQU 1                   ; ..Bit #1; for brclr & brset
Timer2          DS  2
Timer2.Rdy      EQU 2                   ; ..Bit #2; for brclr & brset
Timer3          DS  2
Timer3.Rdy      EQU 3                   ; ..Bit #3; for brclr & brset
Timer4          DS  2
Timer4.Rdy      EQU 4                   ; ..Bit #4; for brclr & brset
Timer5          DS  2
Timer5.Rdy      EQU 5                   ; ..Bit #5; for brclr & brset
Timer6          DS  2
Timer6.Rdy      EQU 6                   ; ..Bit #6; for brclr & brset
Timer7          DS  2
Timer7.Rdy      EQU 7                   ; ..Bit #7; for brclr & brset

TimerFlags      DS  1                   ; 8 bit flags: Timer0,Timer1..

;-----------------------------------------------------------------
; DEFINES

;RAMstart       EQU $40         ; <<< Put  THIS  into Main program <<<
LastRam         EQU RamStart+!512       ; First one beyond RAM
ROMstart        EQU $8000               ; ROM (32Kbytes)
VectorStart     EQU $FFDC               ; Interrupt Vectors

;---------------------------------
; Defines Bits for VCO Programming
; ..PCTL bits ($36), for bset & bclr instructions
PCTL_VCO        EQU   4
PCTL_PLLON      EQU   5

; ..PBWC bits ($37)
PBWC_LOCK       EQU   6
PBWC_AUTO       EQU   7

; ..TBCR bits ($1C)
TBON            EQU   { 1 < 1 }         ; ..Bit #1
TBIE            EQU   { 1 < 2 }         ; ..Bit #2
TBIE.bit        EQU   2                 ; ..Bit #2
TACK            EQU   { 1 < 3 }         ; ..Bit #3 (Write ONLY. Read
as 0)
TACKbit         EQU   3                 ; ..Bit #3 (Write ONLY. Read
as 0)
MS_1024         EQU   %01010000         ; bits 6,5,4: '.101...'

; CONFIG1 Register
LVIandCOP_OFF   EQU   31                ; See CONFIG1 Register...


;=================================================================
; MACRO DEFINITIONS

$MACRO TimeOut timer,gotoAddress        ; timer is a CONSTANT (K)

        brset   %1,TimerFlags,%2

$MACROEND

;-----------------------------------------------------------------
$MACRO SetimerK timer,time              ; time is a CONSTANT (K)

        pshh                            ; save H:X
        pshx                            ; ..
        psha                            ; save ACC
        tpa                             ; push CCS saves IMask status
        psha                            ; ...
        sei                             ; DISABLE INTERRUPTS (IMask=1)
        bclr    %1,TimerFlags           ; Make timer X NOT Ready
        ldhx    #%2                     ; time in milliseconds (!50)
        sthx    timer%1                 ; ..
        bne     *+4                     ; Skip 'bset'
```

```
        bset    %1,TimerFlags           ; Make timer X Ready...

        pula                            ; Pull CCS recover IMask saved
        tap                             ; ...status
        pula                            ; restore ACC
        pulx                            ; restore H:X
        pulh                            ; ..

$MACROEND

;-----------------------------------------------------------------
$MACRO SetimerV timer,time              ; time is a VARIABLE (V)

        pshh                            ; save H:X
        pshx                            ; ..
        psha                            ; save ACC
        tpa                             ; push CCS saves IMask status
        psha                            ; ...
        sei                             ; DISABLE INTERRUPTS (IMask=1)
        bclr    %1,TimerFlags           ; Make timer X NOT Ready
        ldhx    %2                      ; time in milliseconds (var)
        sthx    timer%1                 ; ..
        bne     *+4                     ; Skip 'bset'
        bset    %1,TimerFlags           ; Make timerX Ready...

        pula                            ; Pull CCS recover IMask saved
        tap                             ; ...status
        pula                            ; restore ACC
        pulx                            ; restore H:X
        pulh                            ; ..

$MACROEND

;-----------------------------------------------------------------
$MACRO Wait_onK timer,time              ; time is a Constant (K)

; SetimerK    ; NEED TO REPEAT CODE, cause Macros cannot be nested...
        pshh                            ; save H:X
        pshx                            ; ..
        psha                            ; save ACC
        tpa                             ; push CCS saves IMask status
        psha                            ; ...
        sei                             ; DISABLE INTERRUPTS (IMask=1)
        bclr    %1,TimerFlags           ; Make timer X NOT Ready
        ldhx    #%2                     ; time in milliseconds (!50)
        sthx    timer%1
        bne     *+4                     ; Skip 'bset'
        bset    %1,TimerFlags           ; TimerX Ready...

        pula                            ; Pull CCS recover IMask saved
        tap                             ; ...status
        pula                            ; restore ACC
        pulx                            ; restore H:X
        pulh                            ; ..

        brclr   %1,TimerFlags,*         ; Wait %time milliseconds on %timer

$MACROEND

;-----------------------------------------------------------------
$MACRO Wait_onV timer,time              ; time is a Variable (V)

; SetimerV    ; NEED TO REPEAT CODE. Macros cannot be nested...
        pshh                            ; save H:X
        pshx                            ; ..
        psha                            ; save ACC
        tpa                             ; push CCS
        psha                            ; ...
        sei                             ; DISABLE INTERRUPTS
        bclr    %1,TimerFlags           ; TimerX NOT Ready
        ldhx    %2                      ; time in milliseconds (var)
        sthx    timer%1
        bne     *+4                     ; Skip 'bset'
        bset    %1,TimerFlags           ; TimerX Ready...

        pula                            ; pull CCS. Will CLI if
        tap                             ; ...interrupts were Enabled
        pula                            ; restore ACC
        pulx                            ; restore H:X
        pulh                            ; ..

        brclr   %1,TimerFlags,*         ; Wait %time milliseconds on %timer

$MACROEND

;-----------------------------------------------------------------
; Program VCO to 32 MHz from 32768 Hz cristal (8 MHz VCO bus clock)
; ..V09M7: run 7,987,200 Hz instead 8Mhz, to fit 9600 bps in SCI
; ..Now: N=975; before: N=977. See below
; *** REMEMBER ***: Give 'Xtal' command in Debugger to 32768 Hz

$MACRO Init8MhzClock
        bclr    PCTL_PLLON,PCTL         ; Turn OFF PLL (ON by default)

        mov     #$02,PCTL               ; Set P=0 for PRE[1:0],
                                        ; ..E=2 for VPR[1:0]

        mov     #$CF,PMSL               ; N=975($03CF) for MUL[11:0]
        mov     #$03,PMSH               ; ..pms L:MUL[7:0],H:MUL[11:8]
```

```
        mov     #$D0,PMRS               ; Set L=208($D0) for VRS[7:0]
        mov     #$01,PMDS               ; Set R=1 for RDS[3:0]

        bset    PCTL_PLLON,PCTL         ; Turn ON  PLL
        bset      PBWC_AUTO,PBWC                ; Enable Auto Bandwidth
Control
        brclr   PBWC_LOCK,PBWC,*        ; Wait here until LOCK bit set
        bset    PCTL_VCO,PCTL           ; Set VCO clock as Sys. Clock

$MACROEND

;-----------------------------------------------------------------
; TBM INIT. This is the base for Timer Support

$MACRO Init8Timers
        clrx
        clrh

        sthx    Timer0                  ; Init Timer0 to 0
        sthx    Timer1                  ; Init Timer1 to 0
        sthx    Timer2                  ; Init Timer2 to 0
        sthx    Timer3                  ; Init Timer3 to 0
        sthx    Timer4                  ; Init Timer4 to 0
        sthx    Timer5                  ; Init Timer5 to 0
        sthx    Timer6                  ; Init Timer6 to 0
        sthx    Timer7                  ; Init Timer7 to 0

        clr     TimerFlags              ; clear ALL 8 bit timers flags

        mov     #{ TACK | MS_1024 | TBON | TBIE },TBCR
                                        ; set TACK, select 1024Hz,
                                  ; ..Set TBON, Enable TBM interrupts
$MACROEND

;=================================================================
; TBM Interrupt Service Routine (ISR). 90 cycles...
; Now ISR is interruptible from any other devices; non reentrant

$MACRO TIM8INT

        bclr TBIE.bit,TBCR  ; **SELF-DISABLE TBM Interrupts** in TBCR
        bset TACKbit,TBCR   ; Set TBCR TACK bit Acknowledges Interrupt
        cli                 ; Enable ALL (other than this) Interrupts
        pshh                ; Not saved by interrupt proc

        ;-----------------------------------------------------------
        ; Timer0

        ldhx    Timer0      ; load H:X and compare Timer0 (16 bits)
        beq     _TM1        ; ..to see if it is 0

        aix     #-1         ; not 0? decrement
        sthx    Timer0
        bne     _TM1
        bset    Timer0.Rdy,TimerFlags  ; 0? Mark Timer0 Ready

        ;-----------------------------------------------------------
        ; Timer1

_TM1:   ldhx    Timer1      ; load H:X and compare Timer1 (16 bits)
        beq     _TM2        ; ..to see if it is 0

        aix     #-1         ; not 0? decrement
        sthx    Timer1
        bne     _TM2
        bset    Timer1.Rdy,TimerFlags  ; 0? Mark Timer1 Ready

        ;-----------------------------------------------------------
        ; Timer2

_TM2:   ldhx    Timer2      ; load H:X and compare Timer2 (16 bits)
        beq     _TM3        ; ..to see if it is 0

        aix     #-1         ; not 0? decrement
        sthx    Timer2
        bne     _TM3
        bset    Timer2.Rdy,TimerFlags  ; 0? Mark Timer2 Ready

        ;-----------------------------------------------------------
        ; Timer3

_TM3:   ldhx    Timer3      ; load H:X and compare Timer3 (16 bits)
        beq     _TM4        ; ..to see if it is 0

        aix     #-1         ; not 0? decrement
        sthx    Timer3
        bne     _TM4
        bset    Timer3.Rdy,TimerFlags  ; 0? Mark Timer3 Ready

        ;-----------------------------------------------------------
        ; Timer4

_TM4:   ldhx    Timer4      ; load H:X and compare Timer4 (16 bits)
        beq     _TM5        ; ..to see if it is 0

        aix     #-1         ; not 0? decrement
        sthx    Timer4
        bne     _TM5
        bset    Timer4.Rdy,TimerFlags  ; 0? Mark Timer4 Ready
```

```
        ;-----------------------------------------------------------
        ; Timer5

_TM5:   ldhx    Timer5          ; load H:X and compare Timer5 (16 bits)
        beq     _TM6            ; ..to see if it is 0

        aix     #-1             ; not 0? decrement
        sthx    Timer5
        bne     _TM6
        bset    Timer5.Rdy,TimerFlags   ; 0? Mark Timer5 Ready

        ;-----------------------------------------------------------
        ; Timer6

_TM6:   ldhx    Timer6          ; load H:X and compare Timer6 (16 bits)
        beq     _TM7            ; ..to see if it is 0

        aix     #-1             ; not 0? decrement
        sthx    Timer6
        bne     _TM7
        bset    Timer6.Rdy,TimerFlags   ; 0? Mark Timer6 Ready

        ;-----------------------------------------------------------
        ; Timer7

_TM7:   ldhx    Timer7          ; load H:X and compare Timer7 (16 bits)
        beq     _TBISRexit      ; ..to see if it is 0

        aix     #-1             ; not 0? decrement
        sthx    Timer7
        bne     _TBISRexit
        bset    Timer7.Rdy,TimerFlags   ; 0? Mark Timer7 Ready


;-----------------------------------------------------------------
; _TBISRexit:

_TBISRexit:
        sei                     ; Disable CPU Interrupts
        bset TBIE.bit,TBCR      ; Re-enable TBM Interrupts in TBCR
        pulh                    ; Not restored by interrupt process
        rti                     ; ..Return from Interrupt

$MACROEND
```

PRUEBA LIBRERÍA DE TIMERS

Programa que demuestra cómo usar, **dentro** de una rutina de interrupciones, los temporizadores que también van por interrupciones.

```
; Tim8_IRQ.asm: Luis G. Uribe C., Timer calls inside IRQ ISR, J01M7
; Program to show interactions between TBM timers8 interrupt routines
; ..and IRQ' interrupts, when it is needed to request timer delays
; ..from IRQISR. This is a typical case when you need to rearm global
; ..interrupts, despite de advise in contrary from Motorola technical
; ..documents, as shown in class...
;=================================================================
; Include files

$NOLIST
$Include 'gpregs.inc'
$LIST
;-----------------------------------------------------------------
; EQU Definitions
; INTSCR  EQU $001d     ; IRQ status/control, in 'gpregs.inc'

IMASK.bit       EQU   1                   ; ..Bit #1
IMASK           EQU   { 1 < IMASK.bit }   ; ..Bit #1
IRQACK.bit      EQU   2                   ; ..Bit #2
IRQACK          EQU   { 1 < IRQACK.bit}   ; ..Bit #2
IRQF.bit        EQU   3                   ; ..Bit #3
IRQF            EQU   { 1 < IRQF.bit }    ; ..Bit #3
;-----------------------------------------------------------------
; DEFINES

RAMstart        EQU $40         ; 512 RAM bytes
;=================================================================
; Global Variables

        org RAMstart            ; Put this *BEFORE* 'timers8D.inc'

$Include 'timers8D.inc'         ; Timer Support: Variables & Macros

;*****************************************************************
; MAIN PROGRAM HEADER:

        org ROMstart

Main:   mov     #LVIandCOP_OFF,CONFIG1  ; a) Turn off LVI & COP
        ldhx    #LastRam                ; b) Set up SP
        txs                             ; ...
```

```
        clra                            ; Init ACC to fool Debuger...

        Init8MHzClock                   ; c)
        Init8Timers                     ; ...

        ;-----------------------------------------------------------
        ; Enable IRQ interrupts

        bclr IMASK.bit,INTSCR

;-------------------------------------------------------------------
; Main Loop

        cli                             ; Enable Global interrupts NOW

again: inca
        bra     again
;===================================================================
; PROGRAM TRAILER:
;
; 1) TIM8INT Macro
; 2) Define Interrupt Vectors
;
;-------------------------------------------------------------------
; TMB Interrupt Service Routine
TBMISR:
        TIM8INT
;-------------------------------------------------------------------
; IRQ INTERRUPT SERVICE ROUTINE
;
; IRQ has one of the highest priorities in the HC08, and the higher
; ..hardware priority, below Reset and SWI. That is ok for
; ..responsiveness but care must be taken to assure that no priority
; ..inversion ocurrs through long ISR times, using debouncing delays,
; ..for example. If this is the case, you MUST auto-desable the IRQ
; ..interrupts and renable global interruptos, to permit other lower
; ..peripherals access.

IRQISR:
        ; NOTE: NO  'bset IRQF.bit,INTSCR'  required here to Ack IRQ':
        ; ..IRQ' Interrupt is Acknowledged through 'Vector Fetch'.
        ; ..cfr. Technical Data, page 176. Verify using Simulation...

        bset IMASK.bit,INTSCR       ; **SELF-DISABLE IRQ' Interrupts**
        cli                     ; Enable ALL (other than this) Interrupts
        pshh                    ; Not saved by interrupt proc

        ;-----------------------------------------------------------
        ; delaying process 20 MSeconds the IRQISR helps to avoid
        ; ..multiple interrupts due to contact bounce on IRQ' pin

        Wait_onK 7,#!20         ; 20 MSeconds for debouncing. This only
                                ; ..works if TBM is enabled to interrupt..
                                ; ..Should you include NOP loops to delay,
                                ; ..you CAN'NOT block all other peripheral
                                ; ..interrupts. You must follow this same
                                ; ..strategy.
        ;-----------------------------------------------------------
        ; *** NOTE ***:
        ;
        ; Timer 7 MUST NOT be used anywhere else!
        ; ..It belongs exclusively to this 'IRQISR'
        ;
        ;-----------------------------------------------------------
        ; PUT HERE ANY ORIGINAL CODE FOR THIS ISR
        ; ..
        ;-----------------------------------------------------------
IRQISRexit:
        sei                     ; Disable CPU Interrupts; never mess with
                                ; ..hardware enable bits with interrupts
                                ; ..enabled
        bclr IMASK.bit,INTSCR   ; Re-enable IRQ interrupts
        pulh                    ; Not restored by interrupt process
        rti                     ; Return from Interrupt
;-------------------------------------------------------------------
; Interrupt Vectors
dummy_isr:                              ; Must be placed in Rom Space
        rti
        org  VectorStart        ; Increasing priority from bottom up
        dw  TBMISR                      ; 8Timer ISR. Minimum priority
        dw  dummy_isr                   ; ADC Conversion Complete
        dw  dummy_isr                   ; Keyboard
        dw  dummy_isr                   ; SCI Transmit
        dw  dummy_isr                   ; ..SCI Receive
        dw  dummy_isr                   ; ..SCI Error
        dw  dummy_isr                   ; SPI Transmit
        dw  dummy_isr                   ; ..SPI Receive
        dw  dummy_isr                   ; TIM2 Overflow
        dw  dummy_isr                   ; ..TIM2 Channel 1
        dw  dummy_isr                   ; ..TIM2 Channel 0
        dw  dummy_isr                   ; TIM1 Overflow
        dw  dummy_isr                   ; ..TIM1 Channel 1
        dw  dummy_isr                   ; ..TIM1 Channel 0
        dw  dummy_isr                   ; PLL
        dw  IRQISR                      ; IRQ'
        dw  dummy_isr                   ; SWI
        dw  Main                        ; Reset. Maximum priority

        END
```

EJERCICIOS ADICIONALES

PLANTEAMENTO INICIAL

Estos problemas están pensados para realizarse empleando el microcontrolador PIC 16F84A, pero pueden programarse para otros modelos, así como el Freescale HC908 (o cualquier otro microcontrolador que desee). La idea es que el estudiante debe poder hacer ***cualquiera*** de las funciones TTL, empleando para ello un microcontrolador.

La primera cosa que debe hacer es resolver todos los ejercicios anteriores, de lógica combinatoria y secuencial, empleando microcontroladores (PIC, Freescale, etc.) Lo segundo es comparar la dificultad de una solución y otra. Analice el hardware de ambas propuestas, los diagramas de estado (máquinas secuenciales) y los de flujo (programas en el micro), e intente visualizar las dificultades que se derivarían si tuviera que cambiar la especificación original, para cada una de las soluciones alternativas.

1) EJERCICIOS DE REEMPLAZO

Hacer para el PIC 16F84A, programas completos en Assembler (deben correr bajo MPLAB [WinIDE para HC08]) que ejecuten c/u de las siguientes funciones; incluir los Diagramas de Flujo (o el código en C, bien estructurado), y las simulaciones:

1) Decoder/Demultiplexer de 1 a 8 (una [1] entrada "**I**", una [1] entrada "**Enable**", tres [3] selectores, "**S2S1S0**"; ocho [8] salidas, "**O7O6…O1O0**")
2) Multiplexer de 8 a 1 (ocho [8] entradas, "**I7I6…I1I0**"; tres [3] selectores, "**S2S1S0**", una [1] entrada "**Enable**", una [1] salida "**Z**")
3) Full Adder (A+B); A y B tienen c/u 4 bits. Resultado: 5 bits. Ver si puede colocar una entrada de Carry-IN y una señal 'ADD/~SUBSTRACT' (***Nota***: Los dispositivos PIC16F628 y PIC16F88 puede operar con un oscilador interno, de 4MHz y 8MHz respectivamente, lo que deja libres los 2 pines del cristal, que pueden usarse como 2 terminales de entrada o salida, adicionales a los del PIC16F84. El PIC16F877 tiene, desde luego, más puertos de entrada y salida)
4) Comparador de A y B; A y B tienen c/u 4 bits. Resultado: A> B, A == B, A<B.
5) Emule un subset del 7486, para 4 Exors, c/u con 2 entradas y una salida
6) Contador Up/~Down de 0 a 15 (salidas DCBA). Otras entradas: ~MasterClear, ~LOAD, entradas DCBA para el Load; Clock.
7) Idem al (6), pero de 0 a 9
8) Registro de entrada serie y salida paralelo (1 entrada 'I', 1 Clock, 1 entrada Left/~Right, 4 salidas)
9) Registro de entrada paralelo y salida serie (4 entradas 'I', 1 Clock, 1 entrada Left/~Right, 4 salidas)
10) Un Procesador de "Código Hamming", para 4 bits de información (Entrada: I7I6I5I3, salidas I7I6I5P4I3P2P1), que permita ambas funciones: Generador de Código y Corrector de Error.

11) Emule una ROM de 256 posiciones, de 5 bits de longitud para cada posición. Cuando se le da la dirección (8 bits de entrada, por uno de los puertos de entrada), su programa debe localizar la posición correspondiente y sacar los 5 bits de información por un puerto de salida, de su elección.

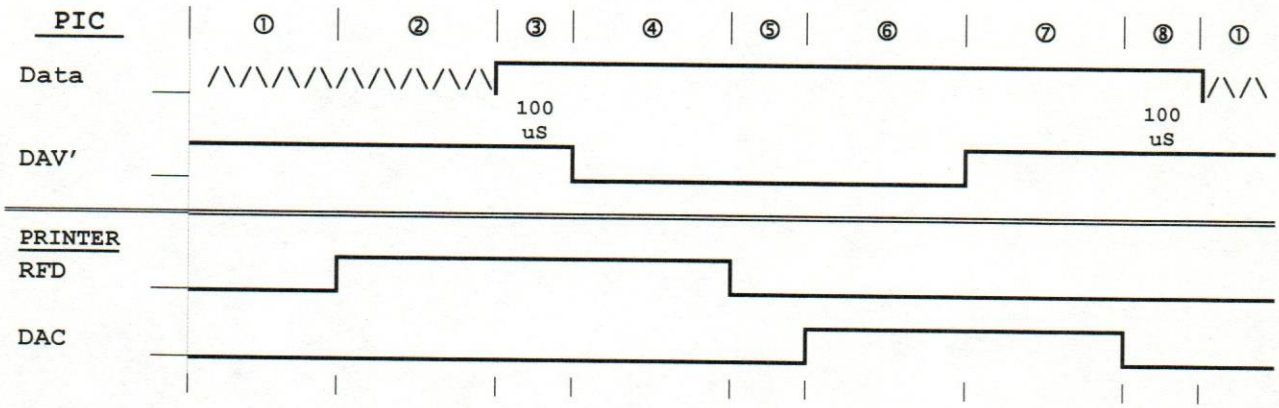
Algunas soluciones analizan las entradas una por una, en una cadena interminable de "if_s". Así no se hace, normalmente; los circuitos combinatorios (o la parte combinacional de los circuitos secuenciales) pueden resolverse mediante Tablas (DT: Define Table). Pueden emplear subrutinas (CALL), a lo mejor pasándoles como parámetros los bits (ceros y unos) leídos de las entradas, y que devuelvan el resultado en W. También pueden usarse Macros.

2) MANEJO DE UNA IMPRESORA GPIB

En 1972 **HP** produjo un estándar de interconexión, el HPIB (HP Interface Bus), adoptado como el IEEE488-75 General Purpose I.B. Esta es una sobre simplificación en cuanto a señales, mecanismos y número de periféricos que allí se definen.

Diseñe un sistema que le permita a su PIC 16F84A actuar como dispositivo emisor, para enviar datos a una impresora (LP), que tiene interfaz GPIB y se comporta sólo como dispositivo receptor de datos. El protocolo para la Transferencia de Información tiene la siguiente secuencia (ver el siguiente texto y su diagrama de tiempo asociado):

1. Esperar a que la impresora coloque **RFD** (**R**eady **F**or **D**ata) en **1** (impresora ociosa)
2. Dar a LP el código de la letra (ASCII, 7 bits)
3. Hacer que **DAV'** (**!** **D**ata **AV**ailable) valga **0**, indicando información lista para el LP
4. Esperar a que la impresora coloque **RFD** en **0** (LP **no** está disponible pues va a procesar los datos)
5. Esperar a que LP coloque **DAC** en **1**;
6. Hacer que **DAV'** valga **1** (datos **no** válidos)
7. Esperar a que la impresora coloque **DAC** en **0** (**D**ata **AC**cepted), confirmando que sabe que ya la información **no** es válida
8. El PIC *puede* retirar ya los datos (si hiciera falta); comienza la secuencia para otra letra...



NO todos los estados duran el mismo tiempo; asuma un *mínimo* de 100 microsegundos (uS) entre cada transición emanada de **su** PIC. El **t.** máximo no está definido. Para **inicializar** ***la interfaz*** del LP, su PIC debe llevar a **0** la línea **IFC** (**I**nter**F**ace **C**lear) al menos 100 uS, dejarla luego en **1** para actividad normal. (Por sencillez **no** se considera aquí código para inicializar la **Impresora** en sí)

En su Diagrama en Bloques indique **todas** las conexiones entre el PIC y la impresora, que tiene 8 bits de entrada (usted genera sólo 7; ¿qué hace con el octavo?)

Su sistema debe imprimir la frase: "Feliz 2007**n**Que pase su examen**n**"

Su código ha de seguir las técnicas de "Structured Programming y Top-Down Design" (estudiadas en los cursos de programación), con al menos 4 niveles (tiene que definir y codificar todo). Si no sabe hacer subrutinas (CALL, RETURN), vea algún texto de microprocesadores, consulte sus asignaciones y referencias del Laboratorio, o escriba macros...; en todo caso, produzca código de tal manera que se distinga claramente cada uno de los siguientes **4** niveles de subrutinas:

LEVEL 0	LEVEL 1	LEVEL 2	LEVEL 3
Main()			
initSystem()	SOME GENERAL PIC INIT ROUTINES.. INITPRINTER() MORE GENERIC PIC INIT ROUTINES..	... initGpib() ...	outpIFC() ...
printDOC(TABLE) ... TABLE: "Feliz año\n" "Que pase su curso\n" ...	PRINTLINE("FELIZ AÑO\N") printLINE("Que pase su curso\n") ...	printCHAR('F') printCHAR('e') printCHAR('l') printCHAR('i') ...	inpRFD() outpData('F') outpDAV() inpDAC() ...

(Esto es sólo un bosquejo mínimo de las funcionalidades del software; incluya **todo** lo que sea necesario: *antes, entre y después* de las subrutinas indicadas)

3) EL JUEGO DE LA VIDA [SOLUCIÓN]

PLANTEAMIENTO

Empleando 12 LEDs (**3 filas, 4 columnas**) diseñe con un PIC 16F84A un sistema que ejecute el llamado Juego de la Vida; en él, los LEDs encendidos representan **organismos** y cada celda tiene adyacentes otras 8. El arreglo "plano" formado por los 12 LEDs, representa una **esfera** (3 dimensiones), por lo que las celdas del techo se "tocan" con las del piso, y las del borde derecho son vecinas a las del izquierdo. Cada organismo puede tener hasta 8 vecinos; la sobrevivencia y reproducción de los organismos en cada generación, dependen de la cantidad de vecinos, según estas **4** reglas:

a) Si un organismo **no** tiene vecinos (**0**), **o** solo está acompañado por **uno** (**1**), **MORIRÁ** de soledad.

b) Si un organismo tiene **dos** (**2**) **o tres** (**3**) vecinos, **SOBREVIVIRÁ** hasta la próxima generación.

c) Si un organismo tiene **cuatro** vecinos **o más** (**>= 4**), **MORIRÁ** a causa de la sobrepoblación.

d) En una celda ***vacía*** que tiene ***exactamente*** **tres** (**3**) organismos vecinos, habrá un **NACIMIENTO**.

Todos los cambios ocurrirán al mismo tiempo; el futuro de cada entidad se basa solo de la actual generación, con independencia de lo que les vaya a ocurrir a sus vecinos en el estado siguiente. Incluya retardos adecuados para permitir la observación de las transformaciones en los LEDs.

Sugerencia #1: Emplee 2 arreglos, CURG (CURRENT GENERATION O GENERACIÓN ACTUAL) y NEXTG (NEXT GENERATION O PRÓXIMA GENERACIÓN), c/u de 3X4 (*BITS* ***o*** *BYTES*), que contengan un **0** en las celdas que **no** alberguen un organismo vivo, y un valor distinto de **0** en caso contrario.

Para cada celda en CURG, el programa mira sus 8 vecinos y establece los cambios para la siguiente etapa; almacena el resultado en la posición correspondiente de NEXTG y, después de procesar todas las celdas de CURG, copia NEXTG sobre CURG y procede a actualizar los LEDs de salida.

Sugerencia #2: La parte del programa que analiza las adyacencias se simplifica, al considerar los **bordes**, si usa arreglos de **5X6** [(3+*2*)X(4+*2*)] **Note**: Se **duplican**: la fila del ***piso***, *en la nueva fila*, ***encima*** *del techo*, y ***ésta***, *en la nueva fila*, ***debajo*** *del piso*; la columna derecha se **replica** *en la nueva* de la izquierda y viceversa. Así, los bordes originales ven adyacencias con mayor facilidad.

Explique si sus arreglos son de **bits** o de **bytes** y por qué; señale la correspondencia que establecerá entre sus arreglos y los terminales de salida.

Como parte de la simulación de su programa emplee esta secuencia infinita. A la izquierda, la población original; luego 4 sucesivas generaciones:

X			X
X			

X			X
X			X
X			X

	X	X	
	X	X	
	X	X	

X			X
X			X
X			X

	X	X	
	X	X	
	X	X	

Este pasatiempo fue desarrollado por el matemático J. Conway de la Universidad de Cambridge (***cfr.*** bibliografía); resulta interesante con muchos más de 16 organismos, mínimo 64. En el original los bordes **no** son adyacentes.

SOLUCIÓN (Programa life2.c)

Las técnicas del capítulo sobre programación recomiendan, entre otras cosas, separar Políticas y Mecanismos (***cfr.*** Cap. 4, Técnicas de Programación). La primera solución a este ejemplo se presenta en lenguaje C y consta de 3 archivos:

- **life2.c**
  Contiene exclusivamente las **Políticas** (**QUÉ** hay que hacer), sin la más mínima referencia a los Mecanismos (**cómo** se va a hacer).
- **life2.h**
  Los **Mecanismos** (**CÓMO** se va a hacer) se han dividido en dos: **1**) Macros y código en línea se encuentran en este archivo
- **life2tr.c**
  2) Las Funciones están en este otro documento.

Para compilar: **cc life2.c**. Debe correrse el programa sólo, y observarlo bajo el debugger a fin de verificar su comportamiento.

También anexo la solución completa que escribí **en Assembler** para el PIC 16F84A (**life.asm**; 996 líneas; 462 sin comentarios ni líneas en blanco). Ciertas particularidades del PIC (por ejemplo, código que debe estar situado en ciertas posiciones), hacen que la versión actual en Assembler y la del original en C no aparezcan tan apegadas, pero siempre pueden observarse las diferentes funciones del C representadas en Assembler. Se ha incluido como comentario, parte del código en C, para ayudar a asociar un texto con otro. ***Este podría ser el programa en Assembler***

más extenso que un estudiante novel de PICs llegue a ver en mucho tiempo. Gran esfuerzo del profesor; ojalá los alumnos tengan similar dedicación...

El programa debe simularse bajo MPLab; verifique que las figuras presentadas en los puertos A y B corresponden con las del ejemplo, arriba en esta misma página. Para correrlo en un PIC real hay que añadirle rutinas de retardo, o algún otro mecanismo que permita separar la presentación de generaciones sucesivas, y así poder observar el juego en los Leds.

```
// life2.c, Ing. Luis G. Uribe C., V03G7
// ...Politics & Mechanisms Separation
// ==================================================================
//     This file serves well to illustrate the algorithm, instead
//     of using a flow diagram; also you may get the assembly
//     program working directly from here
//
// This file has ONLY ***POLITICS*** (WHAT will be done);
// ..There are NO implied Mechanisms (or How things will be done).

/* ----------------------------------------------------------------*/
/* Include Files                                                   */

#include "life2.h"

DefineGlobals();

/* ***************************************************************** */
void   main ( int argc, char ** argv )                        /*()*/
{

   DefineLocals();

   InitProcess();

   LoopForever() {
       copyNext2ActualAndDisplay();
       getNextGen();
   }

}

#include "life2Tr.c"
```

```
// life2.h, Ing. Luis G. Uribe C., V03G7
// This file has ONLY ***MECHANISMS*** (HOW Politics will be done)
// ..(Politics: What will be done)
// Mechanisms are divided in two: macros and in-line code (this file)
// ..and functions (life2tr.c)

/* ---------------------------------------------------------------- */
/* Include Proyect Specific Files */

#include <stdio.h>
#include <stdlib.h>
```

```
/* ================================================================*/
// Typedefs, Defines, macros (DoForEver), Function Prototypes, and
// (perhaps) external variables, should be hidden into a header file

/* ----------------------------------------------------------------*/
/* Typdefs                                                         */

typedef unsigned int uint;

typedef struct {                // Structure to define the 8 bits
    uint Bit0 : 1;              // ..from a byte Port
    uint Bit1 : 1;
    uint Bit2 : 1;
    uint Bit3 : 1;
    uint Bit4 : 1;
    uint Bit5 : 1;
    uint Bit6 : 1;
    uint Bit7 : 1;
} Port;


/* ----------------------------------------------------------------*/
/* Defines                                                         */

// Define L0 .. L11 as shortcuts to the PortX.BitY names:

#define L0     PortB.Bit0      // Leds have following order:
#define L1     PortB.Bit1      // L0  L1  L2  L3
#define L2     PortB.Bit2      // L4  L5  L6  L7
#define L3     PortB.Bit3      // L8  L9  L10 L11
#define L4     PortB.Bit4
#define L5     PortB.Bit5      // First 8 Leds in PortB
#define L6     PortB.Bit6      // ..Reminding 4 Leds in PortA
#define L7     PortB.Bit7

#define L8     PortA.Bit0      // NOTE: We use PORTS A & B to drive
#define L9     PortA.Bit1      // ..the LEDs, and to store the
#define L10    PortA.Bit2      // ..Current Generation too, so we
#define L11    PortA.Bit3      // ..preserve RAM memory


// Shortcuts to the nextGenX.BitY names:

#define N0     nextGenB.Bit0   // This bits represent the NextGen
#define N1     nextGenB.Bit1   // .. array, and use 2 bytes of RAM
#define N2     nextGenB.Bit2
#define N3     nextGenB.Bit3
#define N4     nextGenB.Bit4
#define N5     nextGenB.Bit5
#define N6     nextGenB.Bit6
```

```
#define N7      nextGenB.Bit7

#define N8      nextGenA.Bit0
#define N9      nextGenA.Bit1
#define N10     nextGenA.Bit2
#define N11     nextGenA.Bit3

#define X       1               // To show initial data in a more
#define _       0               // .. descriptive way

#define FIRSTCELL   0
#define LASTCELL   12
#define NEIGHBORS   8

/* ----------------------------------------------------------------*/
/* Macro Definitions (General)                                     */

#define LoopForever()  while( 1 )

/* ----------------------------------------------------------------*/
/* Function Prototypes                                             */

int  applyRule( int cellPosition, int neighborOrganisms );
int  fVecino( int neighbor_i, int cellPosition );
void getNextGenf( int cellPosition, int applyRule );


// ================================================================
// Macro for Global Variables definitions

// This variables need to be seen for several subroutines; they could
// ..be passed as parameters, but in the Assembly language for the
// .. small PIC, we prefer they to be global

#define DefineGlobals()                                          \
                                                                 \
Port PortA, PortB, nextGenB, nextGenA;                           \


// ---------------------------------------------------------------
// Macro for Local Variables definitions

#define DefineLocals()                                           \
                                                                 \
int cellPosition;           /* Twelve: 0..11                 */  \
int neighborOrganisms;      /* Living beings surrounding a cell */ \
int neighbor_i;             /* The i'th neighbor             */  \


// ---------------------------------------------------------------
```

```
#define InitProcess()                                            \
                                                                 \
   N0  = X;  N1  = _;  N2  = _;  N3  = X;  /* This implies that */ \
   N4  = _;  N5  = _;  N6  = _;  N7  = _;  /* ..PIC drives the  */ \
   N8  = X;  N9  = _;  N10 = _;  N11 = _;  /* ..LEDs to ground. */ \


// ---------------------------------------------------------------
#define forEachCell()                                            \
                                                                 \
   for( cellPosition = FIRSTCELL; cellPosition < LASTCELL;       \
       cellPosition ++ )                                         \

// ---------------------------------------------------------------
#define forEachNeighbor()                                        \
                                                                 \
   for( neighborOrganisms = neighbor_i  =  0;                    \
        neighbor_i < NEIGHBORS; neighbor_i ++ )                  \

// ---------------------------------------------------------------
#define getNextGen()                                             \
                                                                 \
       forEachCell() {                                           \
           forEachNeighbor() {                                   \
               neighborOrganisms +=                              \
                   fVecino( neighbor_i, cellPosition );          \
               getNextGenf( cellPosition,                        \
                   applyRule( cellPosition, neighborOrganisms ) ); \
           }                                                     \
       }                                                         \

// ---------------------------------------------------------------
#define copyNext2ActualAndDisplay()                              \
                                                                 \
/* Copy nextGenX array into PortX bits; this will light the LEDs*/ \
                                                                 \
   L0  = N0;  L1  = N1;  L2  = N2;  L3  = N3;                    \
   L4  = N4;  L5  = N5;  L6  = N6;  L7  = N7;                    \
   L8  = N8;  L9  = N9;  L10 = N10; L11 = N11;                   \
                                                                 \
/* The following lines are used to verify program under PC. */   \
/* Comment them out for running in the PIC microcontroller */    \
                                                                 \
   printf( "%c ",   L0  == X ? 'X' : '_' );    /* 1st line */    \
   printf( "%c ",   L1  == X ? 'X' : '_' );                      \
   printf( "%c ",   L2  == X ? 'X' : '_' );                      \
   printf( "%c \n", L3  == X ? 'X' : '_' );                      \
   printf( "%c ",   L4  == X ? 'X' : '_' );    /* 2nd line */    \
   printf( "%c ",   L5  == X ? 'X' : '_' );                      \
   printf( "%c ",   L6  == X ? 'X' : '_' );                      \
```

```
   printf( "%c \n", L7  == X ? 'X' : '_' );    /* 3rd line */      \
   printf( "%c ",   L8  == X ? 'X' : '_' );                       \
   printf( "%c ",   L9  == X ? 'X' : '_' );                       \
   printf( "%c ",   L10 == X ? 'X' : '_' );                       \
   printf( "%c \n", L11 == X ? 'X' : '_' );    /* finish frame */  \
                                                                  \
   getchar();  /* This will simulate DELAY; ENTER continues. */    \
               /* Press Control-C or Control-Break to finish.*/    \
```

```
// lifetr.c, Ing. Luis G. Uribe C., trailing file V03G7
// This file has ONLY ***MECHANISMS*** (HOW Politics will be done)
// ..(Politics: What will be done)
// Mechanisms are divided in two: macros and in-line code (life2.h)
// ..and functions (this trailing file)

/* ================================================================ */
int applyRule( int cellPosition, int neighborOrganisms )        /*()*/
{

   // Returns 1 if the rules imply that there will be an organism in
   // ..the next generation for this amount of neighborOrganisms.
   // Note that returning 1 implies to light the LED, and it will
   // ..happen if the cell was empty and a new organism has born, or
   // ..if a preexistent organism has survived. Zero means to turn
   // ..off the LED.

   int thereIsAnOrganism;

   switch( cellPosition ) {
       case 0:  thereIsAnOrganism = L0;  break;
       case 1:  thereIsAnOrganism = L1;  break;
       case 2:  thereIsAnOrganism = L2;  break;
       case 3:  thereIsAnOrganism = L3;  break;
       case 4:  thereIsAnOrganism = L4;  break;
       case 5:  thereIsAnOrganism = L5;  break;
       case 6:  thereIsAnOrganism = L6;  break;
       case 7:  thereIsAnOrganism = L7;  break;
       case 8:  thereIsAnOrganism = L8;  break;
       case 9:  thereIsAnOrganism = L9;  break;
       case 10: thereIsAnOrganism = L10; break;
       case 11: thereIsAnOrganism = L11; break;
   }

   if( thereIsAnOrganism ) {   // in this cellPosition
       switch( neighborOrganisms ) {   // Who many?

           case 0:             // Will be DEAD, as if by loneliness
           case 1:
               return 0;
           break;

           case 2:             // Will SURVIVE
           case 3:
               return 1;
           break;

           case 4:
           default:
               return 0;       // Will be DEAD, as if by overcrowding
           break;
       }
   }

   // NOTE: There are implied else statements here, above and below.
   // ..Above, because all cases execute a 'return' statement and
   // ..give up program flow control to the calling routine.
   // Same occurs bellow.
   //
   // CELL IS EMPTY if program comes to run this instructions, so,
   // the rules now are:

   if( neighborOrganisms == 3 )
       return 1;   // WILL BORN !!!

   return 0;                // cell was, and will continue to be empty

}  /* applyRule() */


/* ---------------------------------------------------------------- */
int fVecino ( int neighbor_i, int cellPosition )                /*()*/
{

   // Function Vecino returns 1 if there is a i'th neighbor organism
   // .. (0..7) for some actual 'cellPosition' (0..12);
   // Returns 0 if the neighbor_i is an empty cell.
   //
   // NOTE: The Current Generation information is stored in the same
   //       ..bits of Ports A & B used to light the LEDs, so
   //       ..returning the LED bit from PORT is returning it's value
   //
```

```
// THIS ARE THE 8 NEIGHBORS FOR EACH CELL:
// ---------------------------------------
// 0:  11,  8,  9,  3,  1,  7,  4,  5
// 1:   8,  9, 10,  0,  2,  4,  5,  6
// 2:   9, 10, 11,  1,  3,  5,  6,  7
// 3:  10, 11,  8,  2,  0,  6,  7,  4
// 4:   3,  0,  1,  7,  5, 11,  8,  9
// 5:   0,  1,  2,  4,  6,  8,  9, 10
// 6:   1,  2,  3,  5,  7,  9, 10, 11
// 7:   2,  3,  0,  6,  4, 10, 11,  8
// 8:   7,  4,  5, 11,  9,  3,  0,  1
// 9:   4,  5,  6,  8, 10,  0,  1,  2
// 10:  5,  6,  7,  9, 11,  1,  2,  3
// 11:  6,  7,  4, 10,  8,  2,  3,  0
//
// HOW WE IMPLEMENT fVecino:
//
// We use a 12*8 = 96 positions table to implement this function;
// ..this 'brute force' approach takes RAM but is straightforward:
// ..you only need to read the neighbors to each cell from the
// ..check board for the game, and make-up following switch/cases
// ..control structures.
//
// MINIMIZATIONS COULD BE DONE.
// For example, store TWO values in each byte, using it's 2
// ..nibbles (half the RAM).
// ..Or develop some algorithmic relationships between files and
// ..columns IDs for each cell and it's neighbors; you may help
// ..yourself replying border rows and columns to make processing
// ..easy.
// Note that working with rows and columns need the main program
// ..change current strategy, from looping through 12 cells to
// ..looping through 3 rows, then looping through 4 columns.
//
// *** NONE OF THIS WAS DONE HERE ***

switch( cellPosition ) {            // Twelve values: form 0..11
    case 0:
        switch( neighbor_i ) {      // Eight neighbors: from 0..7
            case 0: return L11; case 1: return L8;
            case 2: return L9;  case 3: return L3;
            case 4: return L1;  case 5: return L7;
            case 6: return L4;  case 7: return L5;
        }
    break;

case 1:
    switch( neighbor_i ) {
        case 0: return L8;  case 1: return L9;
        case 2: return L10; case 3: return L0;
        case 4: return L2;  case 5: return L4;
        case 6: return L5;  case 7: return L6;
    }
break;

case 2:
    switch( neighbor_i ) {
        case 0: return L9;  case 1: return L10;
        case 2: return L11; case 3: return L1;
        case 4: return L3;  case 5: return L5;
        case 6: return L6;  case 7: return L7;
    }
break;

case 3:
    switch( neighbor_i ) {
        case 0: return L10; case 1: return L11;
        case 2: return L8;  case 3: return L2;
        case 4: return L0;  case 5: return L6;
        case 6: return L7;  case 7: return L4;
    }
break;

case 4:
    switch( neighbor_i ) {
        case 0: return L3;  case 1: return L0;
        case 2: return L1;  case 3: return L7;
        case 4: return L5;  case 5: return L11;
        case 6: return L8;  case 7: return L9;
    }
break;

case 5:
    switch( neighbor_i ) {
        case 0: return L0;  case 1: return L1;
        case 2: return L2;  case 3: return L4;
        case 4: return L6;  case 5: return L8;
        case 6: return L9;  case 7: return L10;
    }
break;

case 6:
    switch( neighbor_i ) {
        case 0: return L1;  case 1: return L2;
        case 2: return L3;  case 3: return L5;
        case 4: return L7;  case 5: return L9;
```

```
            case 6: return L10; case 7: return L11;
        }
    break;

    case 7:
        switch( neighbor_i ) {
            case 0: return L2;  case 1: return L3;
            case 2: return L0;  case 3: return L6;
            case 4: return L4;  case 5: return L10;
            case 6: return L11; case 7: return L8;
        }
    break;

    case 8:
        switch( neighbor_i ) {
            case 0: return L7;  case 1: return L4;
            case 2: return L5;  case 3: return L11;
            case 4: return L9;  case 5: return L3;
            case 6: return L0;  case 7: return L1;
        }
    break;

    case 9:
        switch( neighbor_i ) {
            case 0: return L4;  case 1: return L5;
            case 2: return L6;  case 3: return L8;
            case 4: return L10; case 5: return L0;
            case 6: return L1;  case 7: return L2;
        }
    break;

    case 10:
        switch( neighbor_i ) {
            case 0: return L5;  case 1: return L6;
            case 2: return L7;  case 3: return L9;
            case 4: return L11; case 5: return L1;
            case 6: return L2;  case 7: return L3;
        }
    break;

    case 11:
        switch( neighbor_i ) {
            case 0: return L6;  case 1: return L7;
            case 2: return L4;  case 3: return L10;
            case 4: return L8;  case 5: return L2;
            case 6: return L3;  case 7: return L0;
        }
    break;
    }

} /* fVecino() */


/* ------------------------------------------------------------------ */
void getNextGenf ( int cellPosition,                          /*()*/
                   int resultsOfApplyingRules )               /*()*/
{

    // Produce the Next Generation Element for this cellPosition,
    // ..into internal nextGen B & A variables (they are GLOBALS).
    // Uses the result of applying game's Rule to this cell, with it's
    // ..calculated amount of neighborOrganisms (See calling routine)
    // Return values accessing global variables.

    switch( cellPosition ) {
        case 0:  N0  = resultsOfApplyingRules; break;
        case 1:  N1  = resultsOfApplyingRules; break;
        case 2:  N2  = resultsOfApplyingRules; break;
        case 3:  N3  = resultsOfApplyingRules; break;
        case 4:  N4  = resultsOfApplyingRules; break;
        case 5:  N5  = resultsOfApplyingRules; break;
        case 6:  N6  = resultsOfApplyingRules; break;
        case 7:  N7  = resultsOfApplyingRules; break;

        case 8:  N8  = resultsOfApplyingRules; break;
        case 9:  N9  = resultsOfApplyingRules; break;
        case 10: N10 = resultsOfApplyingRules; break;
        case 11: N11 = resultsOfApplyingRules; break;
    }

} /* getNextGen() */
```

life.asm

```
; Ing. Luis G. Uribe C., M06F7

 __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON &
_XT_OSC
    LIST        P=16F84A
    INCLUDE  <P16F84A.INC>

; Defines
#define     L0  PORTB,0
#define     L1  PORTB,1
#define     L2  PORTB,2
#define     L3  PORTB,3
#define     L4  PORTB,4
#define     L5  PORTB,5
#define     L6  PORTB,6
#define     L7  PORTB,7
#define     L8  PORTA,0
#define     L9  PORTA,1
#define     L10 PORTA,2
#define     L11 PORTA,3

#define     N0  nextGenB,0
#define     N1  nextGenB,1
#define     N2  nextGenB,2
#define     N3  nextGenB,3
#define     N4  nextGenB,4
#define     N5  nextGenB,5
#define     N6  nextGenB,6
#define     N7  nextGenB,7

#define     N8  nextGenA,0
#define     N9  nextGenA,1
#define     N10 nextGenA,2
#define     N11 nextGenA,3

; Variables Definition
   CBLOCK  0x10
       nextGenB          ; internal copy of
            ;..PORTB, for next Generation
       nextGenA  ; internal copy of PORTA
       cellPosition        ; Twelve: 0..11
       neighborOrganisms        ; Living
            ;..beings surrounding a cell
       neighbor_i        ; The i'th neighbor
       resultsOfApplyingRules  ; auxiliary
            ;.. variable for getNextGen
   ENDC

   ORG     0
   goto main;

   ORG     0x004      ; Interrupt vector
;  goto delay;
   nop

; int applyRule( int cellPosition,
            ;..int neighborOrganisms ) /*()*/
;
; Returns 1 if the rules imply that there
;..will be an organism in the next
;..generation for this amount of
;..neighborOrganisms.
; Note that returning 1 implies to light on
;..the LED, and it will happen if the cell
;..was empty and a new organism has born,
;..or if preexistent organism has survived.
;Zero means to turn off the LED.
;
; NOTE: This routine header must be placed
;..near the beginning, to avoid problems
;..with  addwf PCL,F

applyRule:
   CBLOCK
       thereIsAnOrganism
   ENDC

   clrf    thereIsAnOrganism
   movfw   cellPosition          ;; ;TODO:
            ;..restrain cellPosition 0..11
   addwf   PCL,F
applyReadTable:
   goto    readL0
   goto    readL1
   goto    readL2
   goto    readL3
   goto    readL4
   goto    readL5
   goto    readL6
   goto    readL7
   goto    readL8
   goto    readL9
   goto    readL10
   goto    readL11

; this code, from 'applyRule', need to be
;..in lower memory

lowerThan4:
   movfw   neighborOrganisms   ; for this
                          ;..cellPosition?
   addwf   PCL,F
applyRuleTable:
   goto    rule0
   goto    rule1
   goto    rule2
   goto    rule3

; this code, from 'fVecino', need to be in
;..lower memory

fVecino:
   clrf    vecino
   movfw   cellPosition
   addwf   cellPosition,W
   addwf   cellPosition,W
   addwf   cellPosition,W
   addwf   cellPosition,W
   addwf   cellPosition,W
   addwf   cellPosition,W
   addwf   cellPosition,W

   addwf   neighbor_i,W
   addwf   PCL,F

fVecinoTable:
   goto RL11
   goto RL8
   goto RL9
   goto RL3
   goto RL1
   goto RL7
   goto RL4
   goto RL5

   goto RL8
   goto RL9
   goto RL10
   goto RL0
   goto RL2
```

```
goto RL4
goto RL5
goto RL6

goto RL9
goto RL10
goto RL11
goto RL1
goto RL3
goto RL5
goto RL6
goto RL7

goto RL10
goto RL11
goto RL8
goto RL2
goto RL0
goto RL6
goto RL7
goto RL4

goto RL3
goto RL0
goto RL1
goto RL7
goto RL5
goto RL11
goto RL8
goto RL9

goto RL0
goto RL1
goto RL2
goto RL4
goto RL6
goto RL8
goto RL9
goto RL10

goto RL1
goto RL2
goto RL3
goto RL5
goto RL7
goto RL9
goto RL10
goto RL11

goto RL2
goto RL3
goto RL0
goto RL6
goto RL4
goto RL10
goto RL11
goto RL8

goto RL7
goto RL4
goto RL5
goto RL11
goto RL9
goto RL3
goto RL0
goto RL1

goto RL4
goto RL5
goto RL6
goto RL8
goto RL10
goto RL0
goto RL1
goto RL2

goto RL5
goto RL6
goto RL7
goto RL9
goto RL11
goto RL1
goto RL2
goto RL3

goto RL6
goto RL7
goto RL4
goto RL10
goto RL8
goto RL2
goto RL3
goto RL0
```

```
; this code, from 'getNextGen', need to be
;..in lower memory
getNextGen:

   movfw   cellPosition
   addwf   PCL,F

getNextGenTable:
   goto NextN0
   goto NextN1
   goto NextN2
   goto NextN3
   goto NextN4
   goto NextN5
   goto NextN6
   goto NextN7
   goto NextN8
   goto NextN9
   goto NextN10
   goto NextN11
```

```
main:
init:
   clrf    PORTB        ; First, set Port B
             ;..to SAFE/Innocuous value Now
             ;.. Program Port B for output:
   clrf    PORTA        ; Set Port A too
   bsf     STATUS,RP0   ; .. STATUS bit (5)
                     ;.. = 1: Select Bank 1
   clrf    TRISB        ; .. All bits of
            ;..Port B configured for Output
   movlw   b'1110000'   ; .. PortA, Bits
                ;..0, 1, 2 &  3, for Output
   movwf   TRISA
   bcf     STATUS,RP0   ; .. STATUS bit
                  ;..5 = 0: ReSelect Bank 0

   ; Use following example (beginning and
   ;..following states shown)
;RB b0 b1 b2 b3 X _ _ X   X _ _ X   _ X X _
;RB b4 b5 b6 b7 _ _ _ _   X _ _ X   _ X X _
;RA b0 b1 b2 b3 X _ _ _   X _ _ X   _ X X _
```

```
   ; init next Generation with above
   ;..example pattern:

   movlw   b'00001001' ; NOTE: Bits are in
          ;..REVERSE order from the above!!
   movwf   nextGenB    ; .. table

   movlw   b'00000001' ; Note: only bits
                       ;..0..3 are in use
   movwf   nextGenA    ; .. in PORTA
```

```
; MAIN PROGRAM FLOW:
; - First, copy internally stored
; ..'nextGeneration'   data into
; ..actualGeneration  (this will light
; ..the LEDs!!!). Note that above
; ..initializes 'nextGeneration' instead
; ..of actualGeneration LEDs ...
;
; - Loop through all twelve
; ..cellPositions; for each, inspect
; ..all it's eight neighbors and count
; ..how many are alive, into
; ..'neighborOrganisms' variable
;
; - Then, get the nextGeneration cell
; ..position into nextGen array,
; ..applying game's rules to actual
; ..cellPosition
;
; - Loop for ever
```

```
LoopForever:

copyNext2ActualAndDisplay:      ;; ;TODO:
                        ;..DELAY w/Timer0
   movfw   nextGenB
   movwf   PORTB

   movfw   nextGenA
   movwf   PORTA
   nop
```

```
;  for( cellPosition = 0;
;..cellPosition < 12; cellPosition ++ ) {
for1:

   clrf    cellPosition

for1test:
   movlw   .12
   subwf   cellPosition,W
   btfsc   STATUS,Z     ; Goto nextGen if
                       ;..cellPosition == 12
   goto    for1end      ; Fall into for2, if
                        ;..cellPosition < 12

for1body:
;     for(neighborOrganisms=neighbor_i=0;
            ;..neighbor_i<8;neighbor_i++) {
for2:
   clrf    neighborOrganisms
   clrf    neighbor_i
for2test:
   movlw   .8
   subwf   neighbor_i,W
   btfsc   STATUS,Z     ; Goto for2end if
                          ;..neighbor_i == 8
   goto    for2end

for2body:
;     neighborOrganisms +=
;..        fVecino(neighbor_i,cellPosition);

   call    fVecino
         ; fVecino(neighbor_i,cellPosition)
         ;..RetVal: W
   movfw   vecino
   addwf   neighborOrganisms,F

for2loop:
   incf    neighbor_i,F
   goto    for2test

for2end:
; getNextGen( cellPosition,
;..           applyRule(cellPosition,
;..                     neighborOrganisms));

   call    applyRule    ; for cellPosition
                  :..and neighborOrganisms.
                        ; RetVal in
                 ;..'resultsOfApplyingRules'
   movwf   resultsOfApplyingRules

   call    getNextGen   ; for cellPosition
            ;..and resultsOfApplyingRules

for1loop:
   incf    cellPosition,F
   goto    for1test

for1end:
   goto LoopForever
```

```
; int applyRule( int cellPosition,
;..          int neighborOrganisms ) /*()*/
;
; Returns 1 if the rules imply that there
;..will be an organism in the next
;..generation for this amount of
;..neighborOrganisms.
; Note that returning 1 implies to light
;..the LED, and it will happen if the cell
;..was empty and a new organism has born,
;..or if a preexistent organism has
;..survived.
;..Zero means to turn off the LED.
; NOTE: This routine header must be placed
;..near the beginning, to avoid problems
;..with  addwf PCL,F
;; FOLLOWING CODE HAS BEEN MOVED UP IN MEM.
;; ;applyRule:
;; ;
;; ;   CBLOCK
;; ;       thereIsAnOrganism
;; ;   ENDC
;; ;
;; ;   clrf    thereIsAnOrganism
;; ;   movfw   cellPosition        ;; ;TODO:
            ;..restrain cellPosition 0..11
;; ;   addwf   PCL,F
;; ;
;; ;applyReadTable:
;; ;   goto    readL0
;; ;   goto    readL1
;; ;   goto    readL2
;; ;   goto    readL3
;; ;   goto    readL4
;; ;   goto    readL5
;; ;   goto    readL6
;; ;   goto    readL7
;; ;   goto    readL8
;; ;   goto    readL9
;; ;   goto    readL10
;; ;   goto    readL11

readL0:
   btfsc   L0
   incf    thereIsAnOrganism,F
   goto    theRules
readL1:
   btfsc   L1
   incf    thereIsAnOrganism,F
   goto    theRules
readL2:
   btfsc   L2
   incf    thereIsAnOrganism,F
   goto    theRules
```

```
readL3:
   btfsc   L3
   incf    thereIsAnOrganism,F
   goto    theRules
readL4:
   btfsc   L4
   incf    thereIsAnOrganism,F
   goto    theRules
readL5:
   btfsc   L5
   incf    thereIsAnOrganism,F
   goto    theRules
readL6:
   btfsc   L6
   incf    thereIsAnOrganism,F
   goto    theRules
readL7:
   btfsc   L7
   incf    thereIsAnOrganism,F
   goto    theRules
readL8:
   btfsc   L8
   incf    thereIsAnOrganism,F
   goto    theRules
readL9:
   btfsc   L9
   incf    thereIsAnOrganism,F
   goto    theRules
readL10:
   btfsc   L10
   incf    thereIsAnOrganism,F
   goto    theRules
readL11:
   btfsc   L11
   incf    thereIsAnOrganism,F
;  goto    theRules
```

```
theRules:
;    if( thereIsAnOrganism ) {    // in this
                                 ;..cellPosition
;        switch( neighborOrganisms ){// How
                                     ;..many?
;            case 0:              // DEAD by
                                 ;.. loneliness
;            case 1:
;                return 0;
;            break;
;
;            case 2:              // SURVIVE
;            case 3:
;                return 1;
;            break;
;
;            case 4:
;            default:
;                return 0;        // DEAD by
                                 ;..overcrowding
;            break;

   movf    thereIsAnOrganism,F ; ... in
                          ;..this cellPosition?
   btfsc   STATUS,Z              ; goto born
                          ;..if cell is empty
   goto    born
organismInCell:
   movlw   .4                    ; First,
                   ;..test to see if >= 4
   subwf   neighborOrganisms,W ; Compare
              ;..neighborOrganisms, L(w)
   btfsc   STATUS,C              ; goto
      ;..Address if C == 1 (higher/same)
   goto    rule4       ; goto rule4 is better
                          ;..than: retlw .0
   goto   lowerThan4             ; to place
          ;..'lowerThan4' in lower memory
```

```
; this code, from 'applyRule', need to be
;..in lower memory
;; FOLLOWING CODE HAS BEEN MOVED UP IN MEM.
;; ;lowerThan4:
;; ;   movfw   thereIsAnOrganism    ; in
                          ;..this cellPosition?
;; ;   addwf   PCL,F
;; ;applyRuleTable:
;; ;   goto    rule0
;; ;   goto    rule1
;; ;   goto    rule2
;; ;   goto    rule3
rule0:
rule1:
rule4:
   retlw   .0

rule2:
rule3:
   retlw   .1
```

```
born:
; CELL IS EMPTY if program comes to run
;..this instructions, so now the rules are:
;    if( neighborOrganisms == 3 )
;        return 1;   // WILL BORN !!!
;   return 0;              // cell was, and
                ;..will continue to be empty
   movfw   neighborOrganisms
   sublw   .3
   btfsc   STATUS,Z     ; Goto for2end if
                        ;..neighbor_i == 8
   retlw   .1
   retlw   .0
```

```
; fVecino:
; int fVecino ( int neighbor_i,
                ;..int cellPosition ) /*()*/
; Function Vecino returns 1 if there is a
;..i'th neighbor organism (0..7) for some
;..actual 'cellPosition' (0..12);
; Returns 0 if the neighbor_i is an empty
;..cell.
;
; NOTE: The Current Generation information
;..is stored in the same bits of Ports A &
;..B used to light the LEDs, so returning
;..the LED bit from PORT is returning it's
;..value
;
; THIS ARE THE 8 NEIGHBORS FOR EACH CELL:
; ---------------------------------------
;  0: 11,  8,  9,  3,  1,  7,  4,  5
;  1:  8,  9, 10,  0,  2,  4,  5,  6
;  2:  9, 10, 11,  1,  3,  5,  6,  7
;  3: 10, 11,  8,  2,  0,  6,  7,  4
;  4:  3,  0,  1,  7,  5, 11,  8,  9
;  5:  0,  1,  2,  4,  6,  8,  9, 10
;  6:  1,  2,  3,  5,  7,  9, 10, 11
;  7:  2,  3,  0,  6,  4, 10, 11,  8
;  8:  7,  4,  5, 11,  9,  3,  0,  1
;  9:  4,  5,  6,  8, 10,  0,  1,  2
; 10:  5,  6,  7,  9, 11,  1,  2,  3
; 11:  6,  7,  4, 10,  8,  2,  3,  0
;
; HOW WE IMPLEMENT fVecino:
;
; We use a 12*8 = 96 positions table to
;..implement this function; this 'brute
;..force' approach takes RAM but is
;..straightforward: you only need to read
```

```
;..the neighbors to each cell from the
; ..check board for the game, and make-up
;..following switch/cases control
;..structures.

; MINIMIZATIONS COULD BE DONE.
; For example, store TWO values in each
;..byte, using it's 2 nibbles (half the
;..RAM).
; ..Or develop some algorithmic
;..relationships between files and columns
;..IDs for each cell and it's neighbors;
;..you may help yourself replying border
;..rows and columns to make processing easy
; Note that working with rows and columns
;..need the main program change current
;..strategy, from looping through 12 cells
;..to looping through 3 rows, then looping
;..through 4 columns.
; *** NONE OF THIS WAS DONE HERE ***      .

;   switch( cellPosition ) {// 12 values:
                                ;..form 0..11
;       case 0:
;           switch( neighbor_i ) {     // 8
                        ;..neighbors: from 0..7
;   case 0: return L11; case 1: return L8;
;   case 2: return L9;  case 3: return L3;
;   case 4: return L1;  case 5: return L7;
;   case 6: return L4;  case 7: return L5;
;       }; break;
;       case 1:
;           switch( neighbor_i ) {
;   case 0: return L8;  case 1: return L9;
;   case 2: return L10; case 3: return L0;
;   case 4: return L2;  case 5: return L4;
;   case 6: return L5;  case 7: return L6;
;       }; break;
;       case 2:
;           switch( neighbor_i ) {
;   case 0: return L9;  case 1: return L10;
;   case 2: return L11; case 3: return L1;
;   case 4: return L3;  case 5: return L5;
;   case 6: return L6;  case 7: return L7;
;       }; break;
;       case 3:
;           switch( neighbor_i ) {
;   case 0: return L10; case 1: return L11;
;   case 2: return L8;  case 3: return L2;
;   case 4: return L0;  case 5: return L6;
;   case 6: return L7;  case 7: return L4;
;       }; break;
;       case 4:
;           switch( neighbor_i ) {
;   case 0: return L3;  case 1: return L0;
;   case 2: return L1;  case 3: return L7;
;   case 4: return L5;  case 5: return L11;
;   case 6: return L8;  case 7: return L9;
;       }; break;
;       case 5:
;           switch( neighbor_i ) {
;   case 0: return L0;  case 1: return L1;
;   case 2: return L2;  case 3: return L4;
;   case 4: return L6;  case 5: return L8;
;   case 6: return L9;  case 7: return L10;
;       }; break;
;       case 6:
;           switch( neighbor_i ) {
;   case 0: return L1;  case 1: return L2;
;   case 2: return L3;  case 3: return L5;
;   case 4: return L7;  case 5: return L9;
;   case 6: return L10; case 7: return L11;
;       }; break;
;       case 7:
;           switch( neighbor_i ) {
;   case 0: return L2;  case 1: return L3;
;   case 2: return L0;  case 3: return L6;
;   case 4: return L4;  case 5: return L10;
;   case 6: return L11; case 7: return L8;
;       }; break;
;       case 8:
;           switch( neighbor_i ) {
;   case 0: return L7;  case 1: return L4;
;   case 2: return L5;  case 3: return L11;
;   case 4: return L9;  case 5: return L3;
;   case 6: return L0;  case 7: return L1;
;       }; break;
;       case 9:
;           switch( neighbor_i ) {
;   case 0: return L4;  case 1: return L5;
;   case 2: return L6;  case 3: return L8;
;   case 4: return L10; case 5: return L0;
;   case 6: return L1;  case 7: return L2;
;       }; break;
;       case 10:
;           switch( neighbor_i ) {
;   case 0: return L5;  case 1: return L6;
;   case 2: return L7;  case 3: return L9;
;   case 4: return L11; case 5: return L1;
;   case 6: return L2;  case 7: return L3;
;       }; break;
;       case 11:
;           switch( neighbor_i ) {
;   case 0: return L6;  case 1: return L7;
;   case 2: return L4;  case 3: return L10;
;   case 4: return L8;  case 5: return L2;
;   case 6: return L3;  case 7: return L0;
;       }; break;
; }
```

```
;  make a table, using
;.. cellPosition * 8 + neighbor_I
;.. as an index:

   CBLOCK
       vecino
   ENDC
```

```
; this code, from 'fVecino', need to be in
;..lower memory
;; FOLLOWING CODE HAS BEEN MOVED UP IN MEM.
;; ;fVecino:
;; ;   clrf    vecino
;; ;   movfw   cellPosition
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;   addwf   cellPosition,W
;; ;
;; ;   addwf   neighbor_i,W
;; ;   addwf   PCL,F
;; ;
;; ;fVecinoTable:
;; ;   goto RL11
;; ;   goto RL8
;; ;   goto RL9
;; ;   goto RL3
;; ;   goto RL1
;; ;   goto RL7
;; ;   goto RL4
;; ;   goto RL5
;; ;
;; ;   goto RL8
;; ;   goto RL9
;; ;   goto RL10
;; ;   goto RL0
```

```
;; ;    goto RL2
;; ;    goto RL4
;; ;    goto RL5
;; ;    goto RL6
;; ;
;; ;    goto RL9
;; ;    goto RL10
;; ;    goto RL11
;; ;    goto RL1
;; ;    goto RL3
;; ;    goto RL5
;; ;    goto RL6
;; ;    goto RL7
;; ;
;; ;    goto RL10
;; ;    goto RL11
;; ;    goto RL8
;; ;    goto RL2
;; ;    goto RL0
;; ;    goto RL6
;; ;    goto RL7
;; ;    goto RL4
;; ;
;; ;    goto RL3
;; ;    goto RL0
;; ;    goto RL1
;; ;    goto RL7
;; ;    goto RL5
;; ;    goto RL11
;; ;    goto RL8
;; ;    goto RL9
;; ;
;; ;    goto RL0
;; ;    goto RL1
;; ;    goto RL2
;; ;    goto RL4
;; ;    goto RL6
;; ;    goto RL8
;; ;    goto RL9
;; ;    goto RL10
;; ;
;; ;    goto RL1
;; ;    goto RL2
;; ;    goto RL3
;; ;    goto RL5
;; ;    goto RL7
;; ;    goto RL9
;; ;    goto RL10
;; ;    goto RL11
;; ;
```

```
;; ;    goto RL2
;; ;    goto RL3
;; ;    goto RL0
;; ;    goto RL6
;; ;    goto RL4
;; ;    goto RL10
;; ;    goto RL11
;; ;    goto RL8
;; ;
;; ;    goto RL7
;; ;    goto RL4
;; ;    goto RL5
;; ;    goto RL11
;; ;    goto RL9
;; ;    goto RL3
;; ;    goto RL0
;; ;    goto RL1
;; ;
;; ;    goto RL4
;; ;    goto RL5
;; ;    goto RL6
;; ;    goto RL8
;; ;    goto RL10
;; ;    goto RL0
;; ;    goto RL1
;; ;    goto RL2
;; ;
;; ;    goto RL5
;; ;    goto RL6
;; ;    goto RL7
;; ;    goto RL9
;; ;    goto RL11
;; ;    goto RL1
;; ;    goto RL2
;; ;    goto RL3
;; ;
;; ;    goto RL6
;; ;    goto RL7
;; ;    goto RL4
;; ;    goto RL10
;; ;    goto RL8
;; ;    goto RL2
;; ;    goto RL3
;; ;    goto RL0

RL0:
   btfsc   L0
   incf    vecino,F
   return
RL1:
```

```
   btfsc   L1
   incf    vecino,F
   return
RL2:
   btfsc   L2
   incf    vecino,F
   return
RL3:
   btfsc   L3
   incf    vecino,F
   return
RL4:
   btfsc   L4
   incf    vecino,F
   return
RL5:
   btfsc   L5
   incf    vecino,F
   return
RL6:
   btfsc   L6
   incf    vecino,F
   return
RL7:
   btfsc   L7
   incf    vecino,F
   return
RL8:
   btfsc   L8
   incf    vecino,F
   return
RL9:
   btfsc   L9
   incf    vecino,F
   return
RL10:
   btfsc   L10
   incf    vecino,F
   return
RL11:
   btfsc   L11
   incf    vecino,F
   return
```

```
;void getNextGen (int cellPosition,
;..      int resultsOfApplyingRules) /*()*/
;..
; Produce the Next Generation Element for
;..this cellPosition, into internal nextGen
```

```
;..B & A variables (they are GLOBALS).
;..
; Uses the result of applying game's Rule
;..to this cell, with it's calculated
;..amount of neighborOrganisms (See
;..calling routine)
; Return values accessing global variables.
;       switch( cellPosition ) {
;      case 0:  N0 = resultsOfApplyingRules;
;      break;
;      case 1:  N1 = resultsOfApplyingRules;
;      break;
;      case 2:  N2 = resultsOfApplyingRules;
;      break;
;      case 3:  N3 = resultsOfApplyingRules;
;      break;
;      case 4:  N4 = resultsOfApplyingRules;
;      break;
;      case 5:  N5 = resultsOfApplyingRules;
;      break;
;      case 6:  N6 = resultsOfApplyingRules;
;      break;
;      case 7:  N7 = resultsOfApplyingRules;
;      break;
;      case 8:  N8 = resultsOfApplyingRules;
;      break;
;      case 9:  N9 = resultsOfApplyingRules;
;      break;
;      case 10: N10= resultsOfApplyingRules;
;      break;
;      case 11: N11= resultsOfApplyingRules;
;      break;
;this code getNextGen needed in low memory
;; FOLLOWING CODE HAS BEEN MOVED UP IN MEM.

;; ;getNextGen:
;; ;    movfw   cellPosition
;; ;    addwf   PCL,F
;; ;getNextGenTable:
;; ;    goto NextN0
;; ;    goto NextN1
;; ;    goto NextN2
;; ;    goto NextN3
;; ;    goto NextN4
;; ;    goto NextN5
;; ;    goto NextN6
;; ;    goto NextN7
;; ;    goto NextN8
;; ;    goto NextN9
;; ;    goto NextN10
;; ;    goto NextN11

NextN0:
   bcf     N0
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N0   ; set bit N0 conditionally
   return

NextN1:
   bcf     N1
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N1   ; set bit N1 conditionally
   return

NextN2:
   bcf     N2
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N2   ; set bit N2 conditionally
   return

NextN3:
   bcf     N3
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N3   ; set bit N3 conditionally
   return

NextN4:
   bcf     N4
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N4   ; set bit N4 conditionally
   return

NextN5:
   bcf     N5
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N5   ; set bit N5 conditionally
   return

NextN6:
   bcf     N6
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N6   ; set bit N6 conditionally
   return

NextN7:
   bcf     N7
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N7   ; set bit N7 conditionally
   return

NextN8:
   bcf     N8
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N8   ; set bit N8 conditionally
   return

NextN9:
   bcf     N9
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N9   ; set bit N9 conditionally
   return

NextN10:
   bcf     N10
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N10   ; set bit N10 conditionally
   return

NextN11:
   bcf     N11
   movf    resultsOfApplyingRules,F
             ; test resultsOfApplyingRules
   btfss   STATUS,Z
   bsf     N11   ; set bit N11 conditionally
   return

   END
```

BIBLIOGRAFÍA

Introduction to Embedded Microcomputer Systems. ***Jonathan W. Valvano***, Thomson Brooks/Cole, 2003. ISBN 0-534-39177-X

Computers as Components, ***Wayne Wolf***, Elsevier, 2005. ISBN 0-12-369459-0

Microcontrolador PIC16F84 Desarrollo de Proyectos, 2ª ed. 2006, ***Palacios, Remiro, López***, Alfa-Omega

Microcontroladores PIC16F84, Diseño práctico de aplicaciones, ***Angulo Usategui, et. al***, 2ª parte, 2ª ed., 2006, McGraw Hill, ISBN 84-481-4627-1

Embedded Software, The Works, ***Colin Walls***, Elsevier, 2006, ISBN 0-7506-7954-9

High-Performance Embedded Computing, ***Wayne Wolf***, Elsevier, 2007, ISBN 0-12-369485-X