Tema 3: Herramientas de diseño

- Dificultades en el diseño de un SE
- Herramientas de diseño
  - Requerimientos del entorno de desarrollo
  - Simuladores
  - Analizador Lógico
  - Sistemas de depuración. Monitores
  - Emuladores ROM
  - Emuladores In-Circuit (ICE)
- Test

Bibliografía: Capítulos 4, 5, 6, 7 y 8

"Embedded Systems Design: An introduction to Processes, Tools, & Techniques". Arnold S. Berger. CMP Books, 2002. Cap.

Dificultades en el diseño de un SE

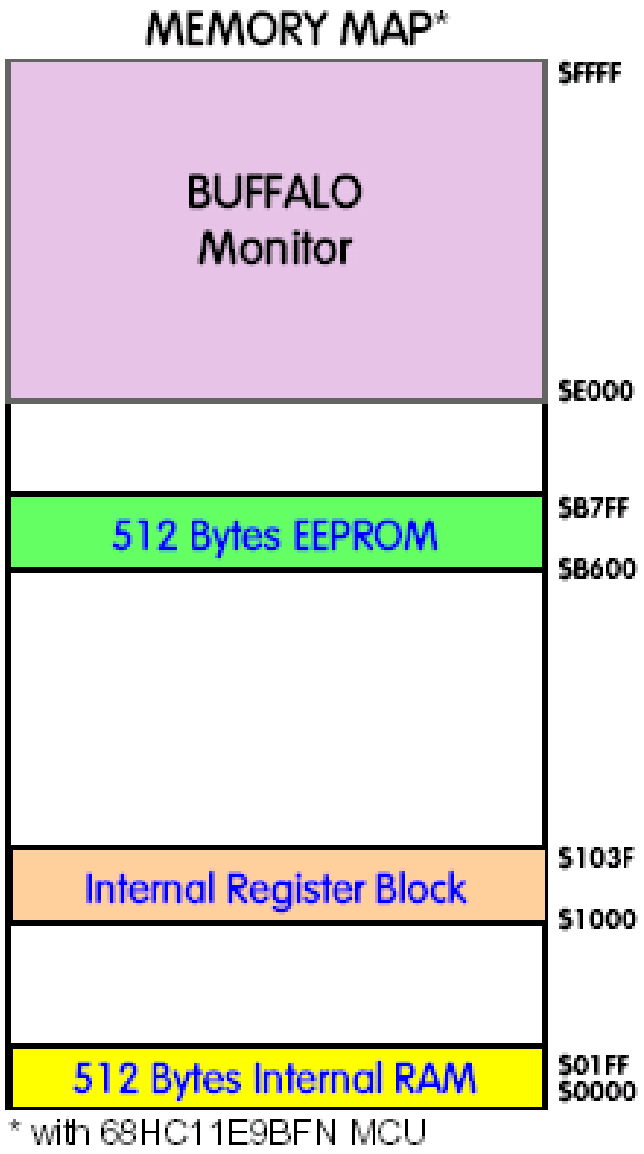
- El desarrollo de aplicaciones para un PC se realiza con herramientas que funcionan sobre la misma plataforma donde esta se ejecutará

- El software de un SE se ejecuta en un hardware único
  - Distinto del *host* donde se desarrolla
  - Es posible que éste no existiera cuando se creó el entorno de desarrollo

- Al escribir código para un SE debemos conocer de éste
  - Uso y organización de la memoria, incluyendo manejo de la pila
  - Qué ocurre al arrancar el sistema (direcc. Inicio, qué ejecuta, etc.)
  - Cómo se manejan las interrupciones y las excepciones

Organización de la memoria

- Mapa de Memoria
  - Espacio de código
    - ROM/EPROM/EEPROM/Flash/
  - Espacio de datos
    - Pila
    - Memoria Libre (variables)
  - Espacio de E/S
    - Registros E/S, Timer, Conv. A/D, etc.
    - Reset, excepciones e interrupciones
  - Espacio de expansión
    - RAM
    - ROM
    - E/S



Sistema de arranque (Reset)

- Supondremos que el SE ejecuta un código almacenado en el espacio de memoria propio → Memoria ROM/EPROM

- Fases del arranque
  - Fase Hardware (después de activar la línea de RESET)
    - Fuerza al procesador a ejecutar el programa de arranque o transferir el control a este programa → La dirección inicial siempre es fija
  - Fase Software (primeras instrucciones del programa de arranque)
    - Unas pocas instrucciones que inicializan los elementos principales del hardware (conv. A/D, timer, etc.) y estructuras de datos en memoria (pila, excepciones, etc.)

Ciclo de respuesta de las interrupciones

- Diferencias en el manejo de las interrupciones según el tipo de procesador
- ¿Cómo sabe la CPU dónde encontrar el código de manejo de la interrupción?
  - Vectorización? Direcciones fijas?
- ¿Qué hace para salvar y restaurar el "contexto" del hilo principal?
  - Qué registros intervienen?
- ¿Cuándo deberían habilitarse las interrupciones?

Llamada a funciones y marco de pila

- Instrucciones de llamada y retorno
- Instrucciones de manejo de la pila
  - Load/store sobre el puntero de pila → manual (MIPS R2000)
  - Instrucciones *push* (apilar) y *pop* (desapilar) (MC68HC11)
- Código generado por los compiladores
  - Apilado de argumentos en la pila
  - Llamada a la función (almacenamiento direcc. retorno en pila)
  - Reserva almacenamiento variable locales (pila)
  - Trabajo de la función
  - Liberación almacenamiento variables locales
  - Retorno de la función
  - Desapilado del espacio usado para los argumentos

Emplazamiento/ubicación de código

- El código diseñado debe ser independiente de las posiciones de memoria absolutas (Position Independent Code (PIC))
  - Uso de etiquetas/símbolos
  - Uso de referencias relativas
- Código reubicable
  - Simplifica el emplazamiento de código generado desde compiladores de alto nivel
  - Permite a módulos individuales ser actualizados independientemente y recompilados
  - En sistemas de propósito general simplifica la gestión de la memoria y facilita el uso de librerías compartidas precompiladas.
- El enlazador o linker
  - Se encarga de unir los objetos reubicables y generar el código ejecutable final

Herramientas de diseño

- Un sistema de diseño mínimo debe:
  - Proporcionar un control de ejecución para el SE (total, paso a paso)
  - Permitir un medio para reemplazar el código del SE
  - Proporcionar una monitorización no intrusiva (que no afecte al comportamiento del sistema monitorizado) de la ejecución en el SE
- El sistema de diseño más barato que cumple estas condiciones está formado por:
  - Núcleo de depuración (conectado a un programa depurador remoto en un *host*)
  - Un analizador lógico
  - A veces, se requiere además un emulador de ROM (aconsejable)

Simuladores

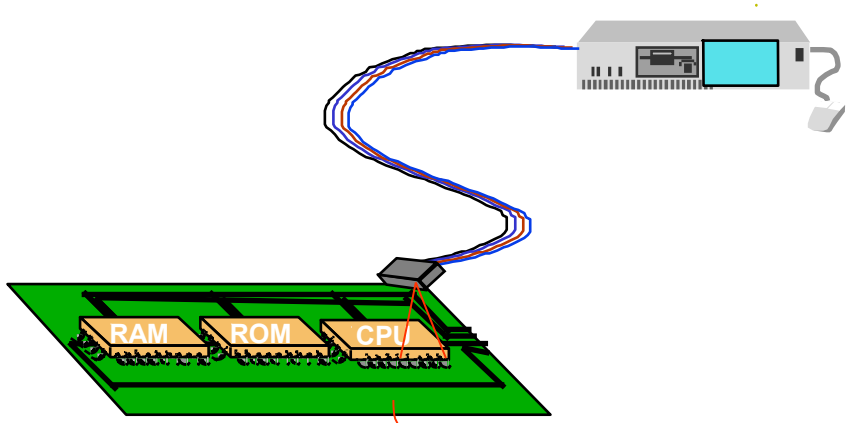
- ISS (Instruction Set Simulator)
  - Programa que crea una versión virtual del microprocesador / microcontrolador
  - Permiten compilar y/o desarrollar código ejecutándolo en la propia estación de trabajo (*host*)

- Algunos están muy elaborados y permiten
  - Compilar/Ensamblar código (incluso desde lenguajes de alto nivel)
  - Ejecución paso a paso
  - Visualizar memoria, registros, etc
  - Manejar puertos de E/S, interrupciones, timers

  - Ejemplos HC11:
    - Wookie
    - Shadow11

Analizador Lógico

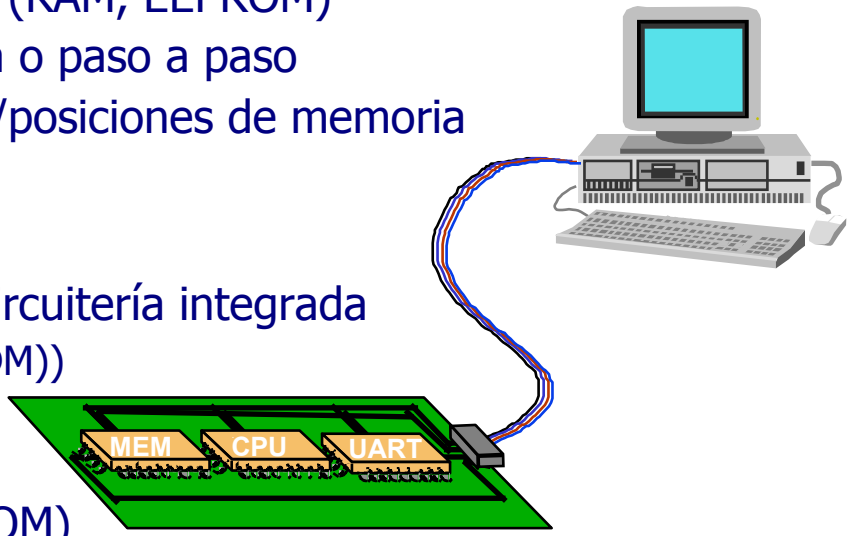
- Complementa otras → Depuradores (debuggers)
- Toma simultánea de señales digitales
- Análisis en Tiempo Real del procesador
- Trazas de ejecución CPU/ acceso a memoria
- Sin perturbación significativa del sistema
- Modos
  - Tiempos →Sinc. reloj interno
    - Transiciones, glitchs (pulsos indeseados)
  - Estados → Sincronizado con el reloj del SE (Clock, E)
    - Posibilidad de desensamblado
- Triggers → disparo toma datos
  - Nivel, flancos
  - Estados
  - Transición estados



Depuradores

- Servicios de ejecución
  - Puntos de ruptura (Breakpoints)
  - Carga de programas desde el *host*
  - Visualizar/modificar registros, memoria (RAM, EEPROM)
  - Ejecutar desde una dirección: completa o paso a paso
  - Ensamblar/desensamblar instrucciones/posiciones de memoria

- Núcleo de depuración → en el SE
  - Hardware: Capacidad del procesador/circuitería integrada
    - Motorola: Background Debug Mode (BDM))
    - Joint Test Action Group (JTAG)
  - Software: Monitor (HC11: buffalo en ROM)

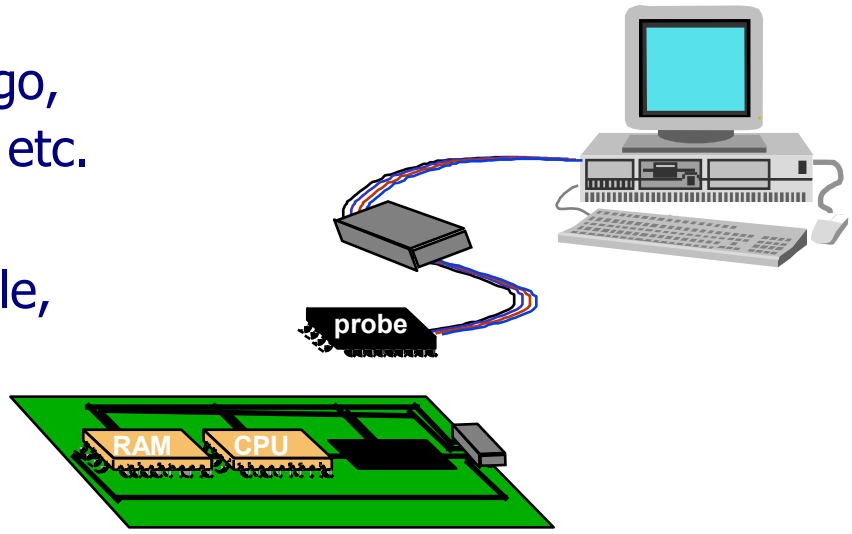


- Depurador Remoto → en el host
  - Simple (terminal) o sofisticado (debugger, máquina virtual, etc.)

- Se conectan mediante una interfaz serie

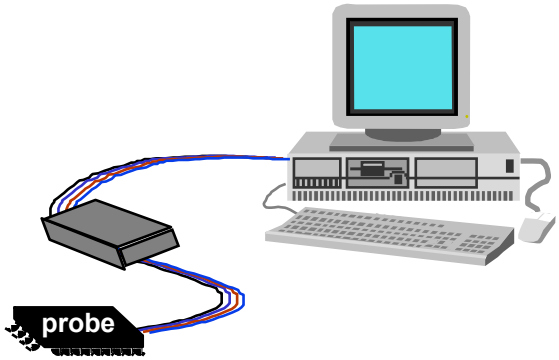
Emulador de ROM

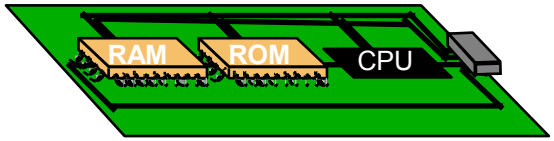
- Permite cargar rápidamente la imagen del código en una RAM
- RAM de doble puerto → acceso simultaneo SE/*HOST*
- Elementos
  - Cable con conector para sustituir la ROM
  - RAM rápida para emular la ROM
  - Procesador de control local
  - Comunicación con el *host*
  - Otras: memoria de trazas (buffer), alg. de programación
- Ventajas
  - Genérico, descarga rápida de código, trazado de ROM, UART, breakpoints, etc.
- Inconvenientes
  - Memoria de SE en condición estable, código en ROM, algunos SE ejecutan el código en RAM



ICE

- ICE: In Circuit Emulator
- Solución integrada (debugger, emulador ROM, AL y más)
  - Control de ejecución
  - Sustitución de memoria en el *host*
  - Trazado en tiempo real
- Bloques que lo componen:
  - Lógica de control de NMI (No Mask Interrupt)
  - Lógica de selección de memoria
  - ROM y RAM de respaldo (*shadow*)
- Desventajas:
  - Díficiles de usar
  - Caros
  - Específicos de un procesador/familia de procesadores
- Tipos
  - Sistemas de desarrollo completos
  - Conectados a un *host*
- Etapas del diseño → configuraciones





Test

- Por qué?
  - Buscar errores (bugs) en el software/hardware
  - Reducir costes de desarrollo y mantenimiento
  - Aumentar las prestaciones
- Cuando?
  - Tan pronto como sea posible → Coste = exp (tiempo proyecto)
- Qué comprobar?
  - Funcionalidades diseñadas/modificadas/ampliadas
  - Hacer test por pares: uno diseña y el otro hace el test.
- Tipos de test
  - Caja blanca: comprobación de elementos internos del SE
  - Caja negra: Sin conocer el funcionamiento interno del SE
- Cuando parar?
  - Lo dice el jefe / errores < k / errores = 0
- Tests más críticos
  - Tiempo real

Conclusiones

- Problemas al abordar el diseño de un SE
- Características Herramientas de Diseño
- Alternativas en la instrumentación necesaria
- Facilidades de los fabricantes para depuración
- Importancia del test en el producto final