

7919 | **Sistemas Embebidos**

2º Cuatrimestre de 2017

CLASE 2: HERRAMIENTAS DE DESARROLLO EMBEBIDO

Prof: José H. Moyano

Autor original: Sebastián Escarza

Dpto. de Cs. e Ing. de la Computación
Universidad Nacional del Sur
Bahía Blanca, Buenos Aires, Argentina



GrIDSE

The background of the slide is a detailed, high-resolution image of a printed circuit board (PCB). It shows a complex network of copper traces, various electronic components like capacitors, resistors, and integrated circuits, and connectors. The image is slightly blurred and has a warm, golden-brown color palette. A semi-transparent dark blue horizontal band is overlaid across the middle of the image, containing the title text.

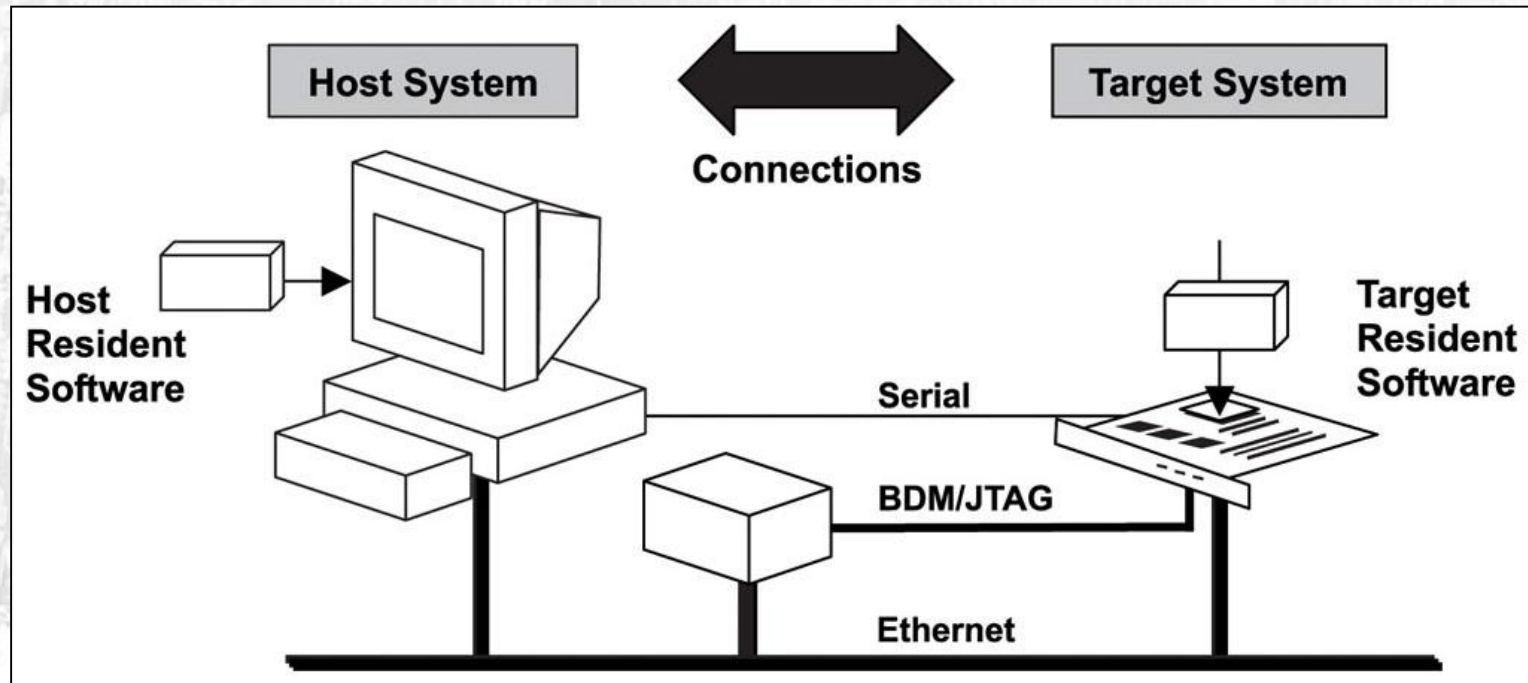
Desarrollo cruzado

Desarrollo cruzado

- **El desarrollo cruzado es una característica distintiva de los sistemas embebidos.**
 - **El host tiene más recursos para alojar las herramientas de desarrollo. Por ej.**
 - un disco y sistema de archivos
 - periféricos que facilitan el desarrollo (teclado, pantalla)
 - recursos para ejecutar las herramientas de desarrollo
 - **El target difiere en arquitectura, recursos y prestaciones respecto del host. El software embebido no puede ejecutarse directamente en el host.**

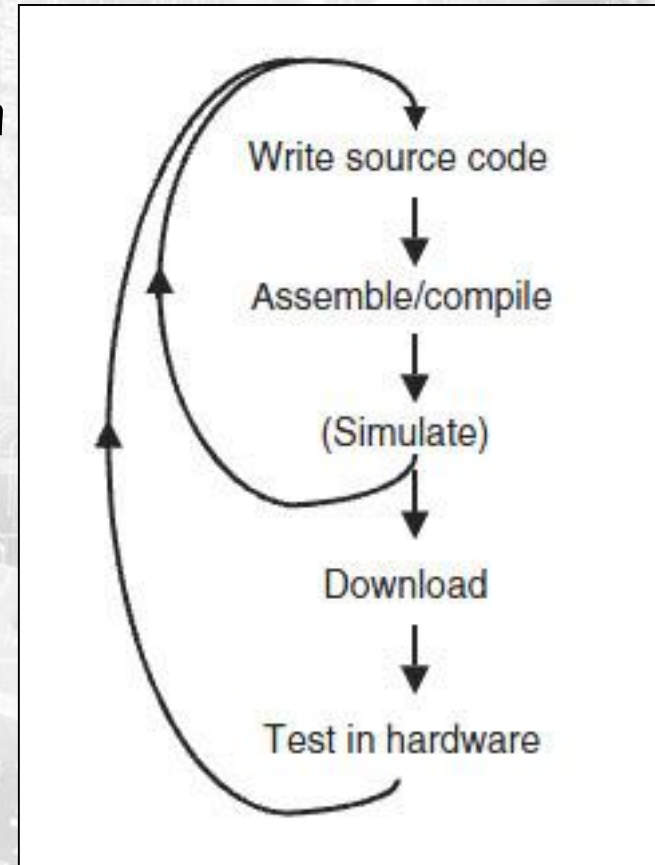
Desarrollo cruzado

- El desarrollo cruzado es una característica distintiva de los sistemas embebidos.



Desarrollo cruzado

- Para la construcción del ejecutable, en gral. se genera código de otra arquitectura.
- Una vez que se obtuvo la imagen ejecutable, a partir de ella se puede:
 - simular en el host
 - descargar en el target
- El proceso, a partir de este paso, se diferencia del desarrollo no cruzado.

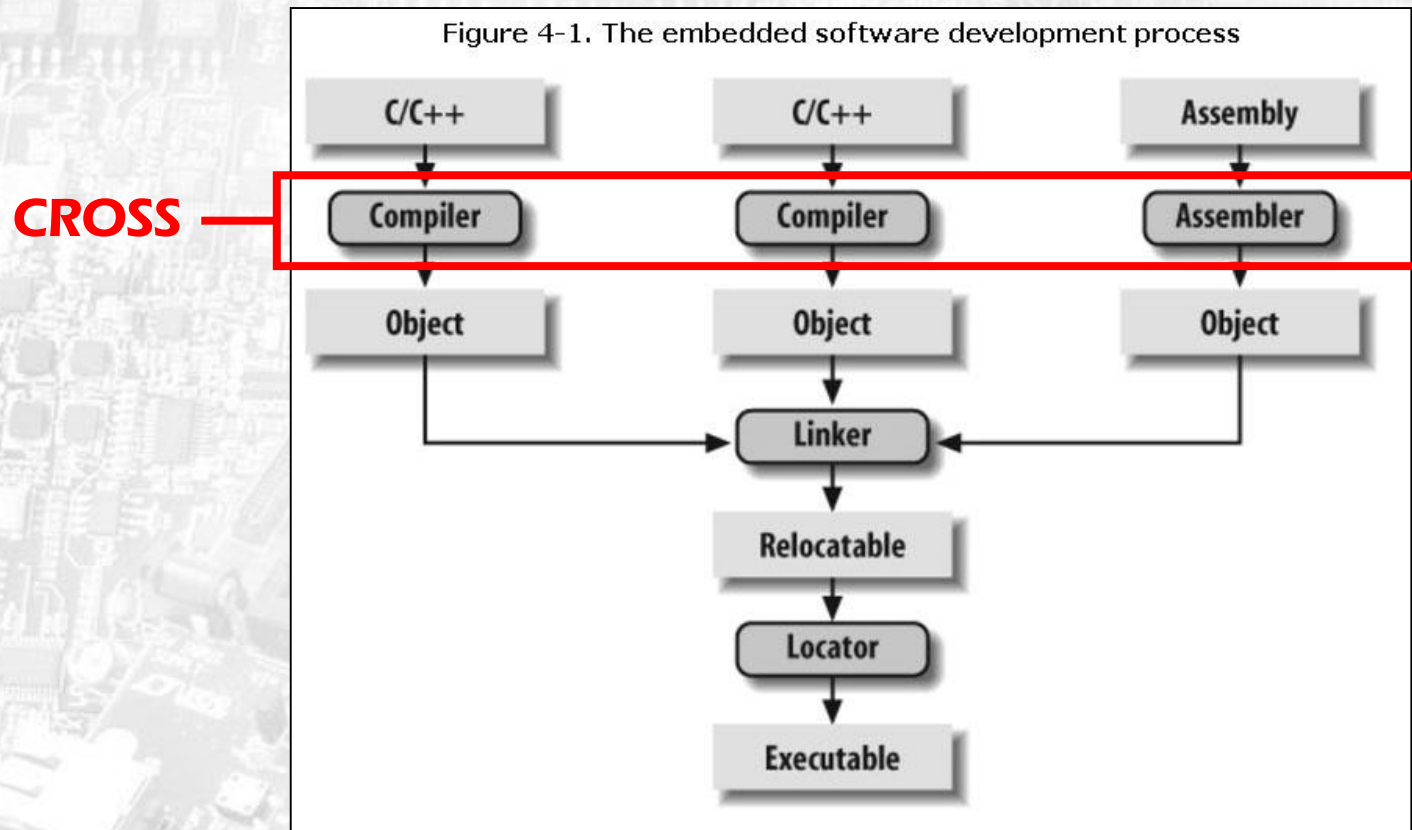


The background of the slide is a detailed, high-resolution image of a printed circuit board (PCB). It shows various electronic components such as integrated circuits, capacitors, and resistors, along with intricate copper traces. A semi-transparent blue horizontal band is superimposed across the middle of the image, serving as a backdrop for the title text.

El “toolchain” embebido

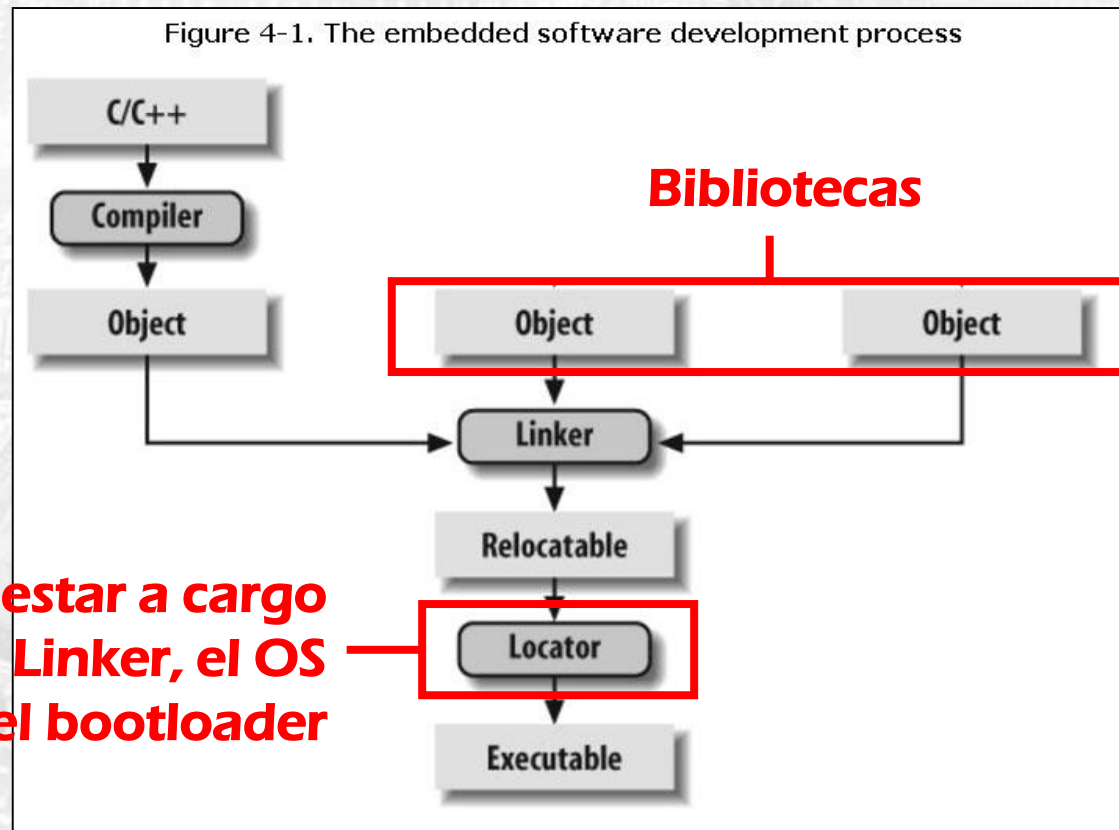
Construcción Cruzada del Ejecutable

- Proceso de construcción de una aplicación



Construcción Cruzada del Ejecutable

- Proceso de construcción de una aplicación



Ensamblado/Compilación Cruzada

- **La implementación del sistema comienza por el código fuente:**
 - **Ensambladores: lenguaje ensamblador. Mapeo 1 a 1 entre mnemónicos e instrucciones del procesador (target).**
 - **Compiladores: lenguajes de alto nivel (C, C++, Java, etc). La traducción es más compleja e incorpora un mayor número de chequeos.**
- **Ambas herramientas traducen el código fuente a código objeto para la arquitectura target seleccionada.**

Ensamblado/Compilación Cruzada

- Ej. Desensamblando en Arduino

```
0000011a <setup>:  
11a: 8d e0 ldi r24, 0x0D ; 13  
11c: 61 e0 ldi r22, 0x01 ; 1  
11e: dc c0 rjmp .+440 ; 0x2d8 <pinMode>
```

Dirección PC relativa al inicio de la
función pinMode(...)

```
void setup() {  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  digitalWrite(13, HIGH);  
  delay(1000);  
  digitalWrite(13, LOW);  
  delay(1000);  
}
```

Ensamblado/Compilación Cruzada

- Ej. Desensamblando en Arduino

```
000000fa <loop>:
fa: 8d e0 ldi r24, 0x0D ; 13
fc: 61 e0 ldi r22, 0x01 ; 1
fe: 12 d1 rcall .+548 ; 0x324 <digitalWrite>

100: 68 ee ldi r22, 0xE8 ; 232
102: 73 e0 ldi r23, 0x03 ; 3
104: 80 e0 ldi r24, 0x00 ; 0
106: 90 e0 ldi r25, 0x00 ; 0
108: 53 d0 rcall .+166 ; 0x1b0 <delay>

10a: 8d e0 ldi r24, 0x0D ; 13
10c: 60 e0 ldi r22, 0x00 ; 0
10e: 0a d1 rcall .+532 ; 0x324 <digitalWrite>

110: 68 ee ldi r22, 0xE8 ; 232
112: 73 e0 ldi r23, 0x03 ; 3
114: 80 e0 ldi r24, 0x00 ; 0
116: 90 e0 ldi r25, 0x00 ; 0

118: 4b c0 rjmp .+150 ; 0x1b0 <delay>
```

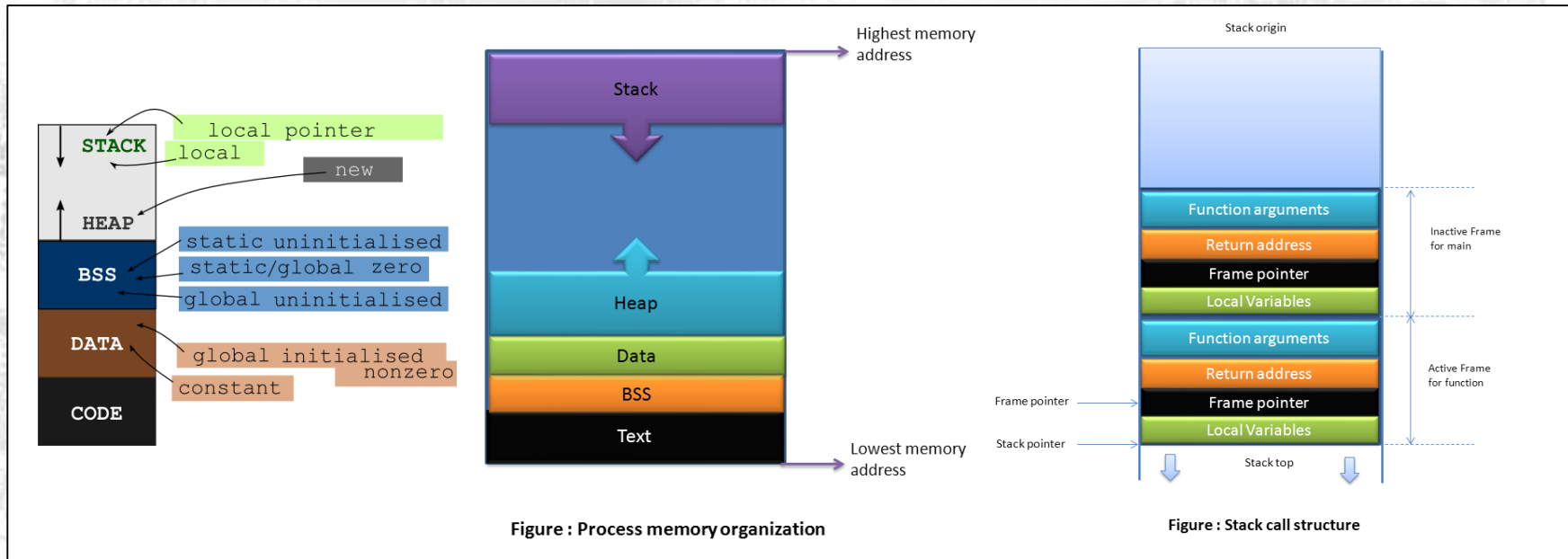
```
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

1000 (little endian)
AVR es una arq. de 8 bits

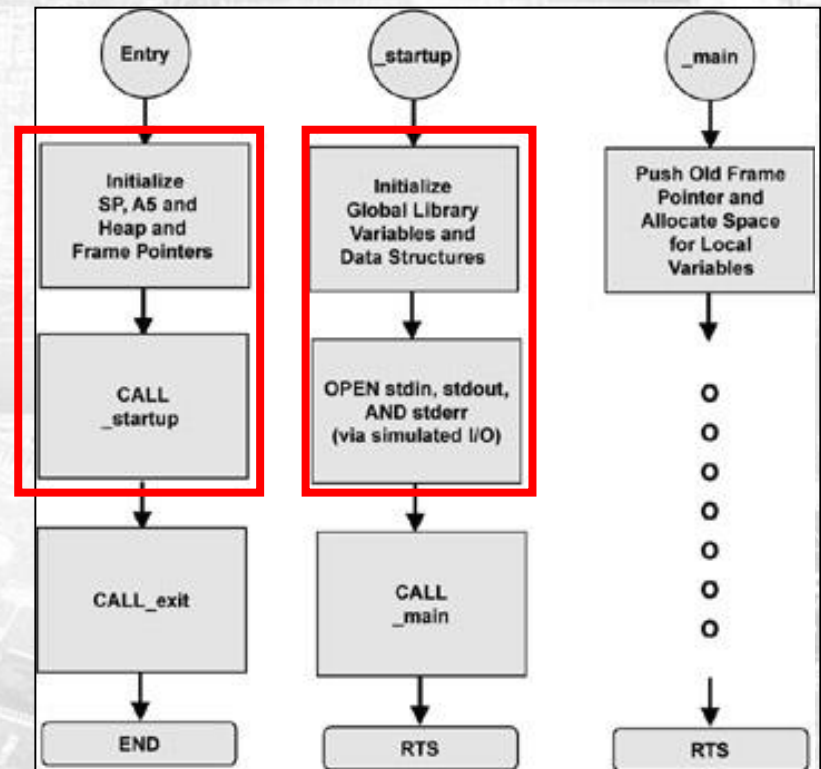
Código de inicialización

- Cuando se trabaja con lenguajes de alto nivel, una de las tareas del compilador es incluir código de inicialización para preparar el ambiente de tiempo de ejecución.



Código de inicialización

- Cuando se trabaja con lenguajes de alto nivel, una de las tareas del compilador es incluir código de inicialización para preparar el ambiente de tiempo de ejecución.

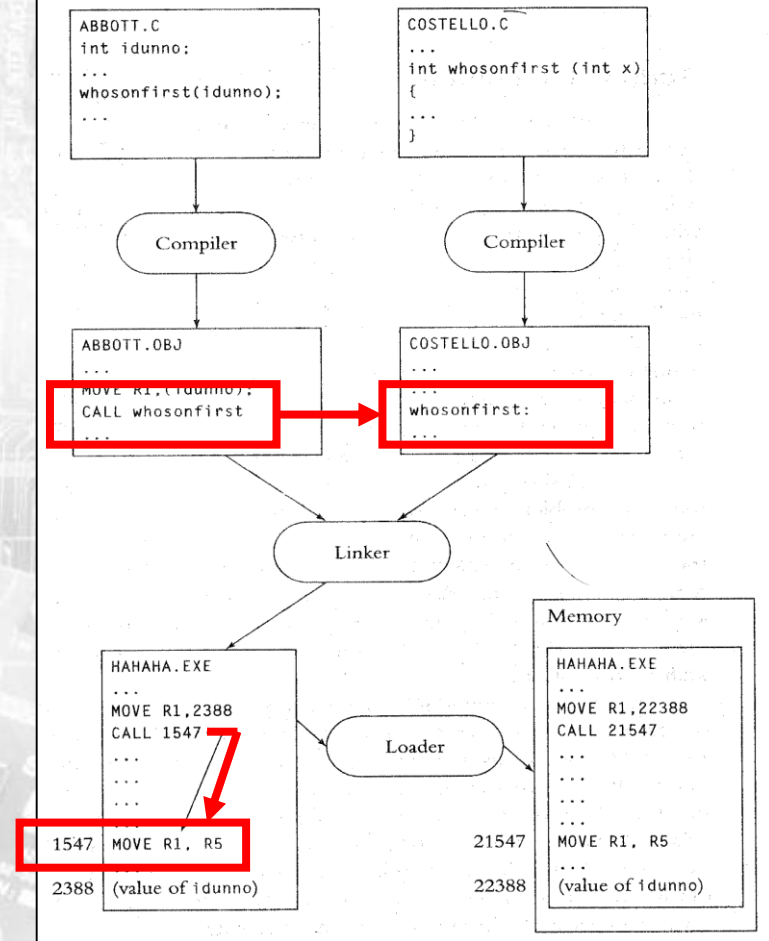


Linkeando y localizando...

- **El linker:**

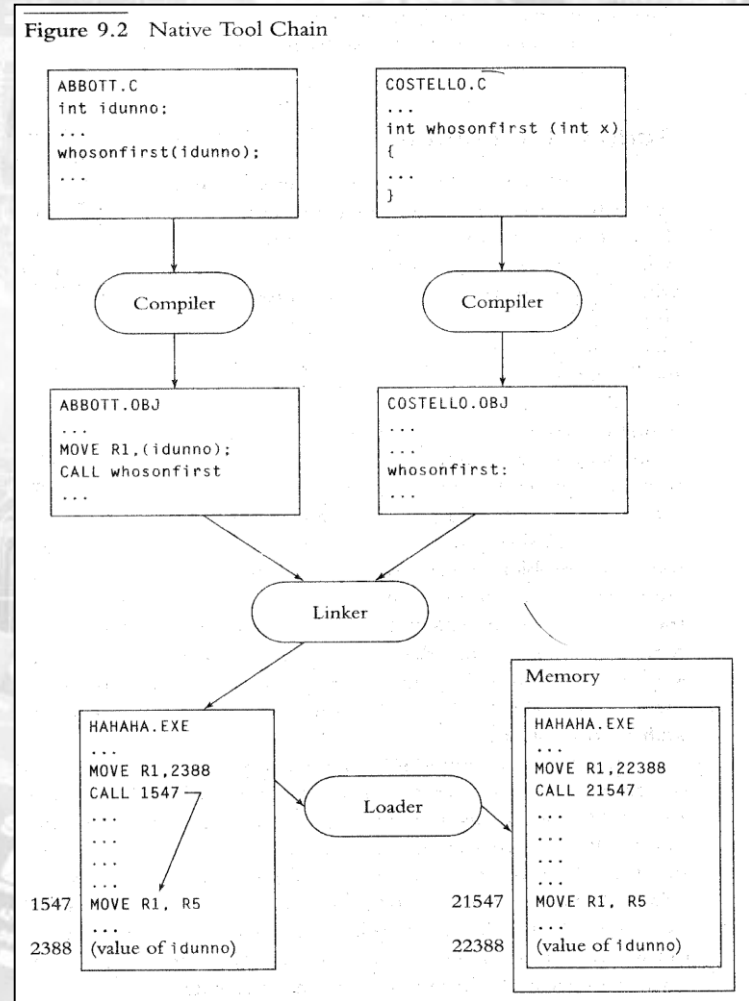
- Combina los códigos objeto en uno s/referencias externas.
- Resuelve las ref. simbólicas definiendo locaciones para los símbolos (código absoluto, reubicable, independiente de la posición, ...)
- El linkeo generalmente es estático (ocurre en el host, en gral. No se utilizan DLLs).
- La responsabilidad de la locación suele ser compartida entre el linker y el loader.

Figure 9.2 Native Tool Chain



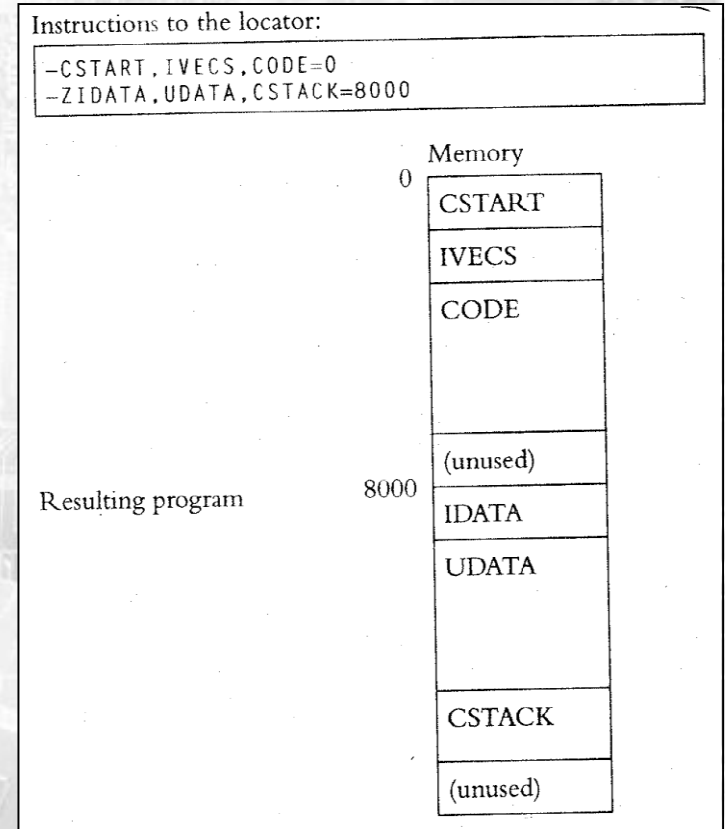
Linkeando y localizando...

- La responsabilidad de la locación suele ser compartida entre el linker y el loader.
- Sin embargo, puede existir un locator standalone.



Linkeando y localizando...

- La responsabilidad de la locación suele ser compartida entre el linker y el loader.
- Sin embargo, puede existir un locator standalone (con su script correspondiente).



Linkeando y localizando...

- Ej: Un script de linkeo (p/PIC 18F242).

```
// $Id: 18f242.lkr,v 1.1 2003/12/16 14:53:08 GrosbaJ Exp $
// File: 18f242.lkr
// Sample linker script for the PIC18F242 processor

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f242.lib

CODEPAGE    NAME=vectors      START=0x0          END=0x29          PROTECTED
CODEPAGE    NAME=page        START=0x2A         END=0x3FFF
CODEPAGE    NAME=idlocs      START=0x200000     END=0x200007     PROTECTED
CODEPAGE    NAME=config      START=0x300000     END=0x30000D     PROTECTED
CODEPAGE    NAME=devid       START=0x3FFFFE     END=0x3FFFFFFF   PROTECTED
CODEPAGE    NAME=eedata      START=0xF00000     END=0xF000FF     PROTECTED

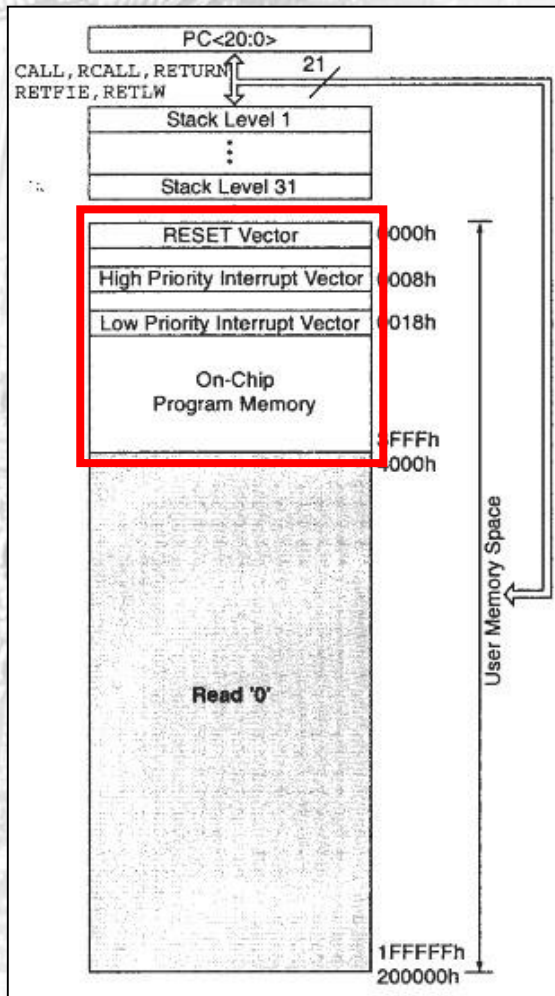
ACCESSBANK  NAME=accessram    START=0x0          END=0x7F
DATABANK    NAME=gpr0         START=0x80         END=0xFF
DATABANK    NAME=gpr1        START=0x100        END=0x1FF
DATABANK    NAME=gpr2        START=0x200        END=0x2FF
ACCESSBANK  NAME=accesssfr    START=0xF80        END=0xFFFF       PROTECTED

SECTION     NAME=CONFIG      ROM=config

STACK SIZE=0x100 RAM=gpr2

Program Example 17.6 Linker Script for 18F242
```


Linkeando – Mapa de memoria



```
// $Id: 18f242.lkr,v 1.1 2003/12/16 14:53:08 GrosbaJ Exp $
// File: 18f242.lkr
// Sample linker script for the PIC18F242 processor
```

```
LIBPATH .
```

```
FILES c018i.o
FILES clib.lib
FILES p18f242.lib
```

CODEPAGE	NAME=vectors	START=0x0	END=0x29	PROTECTED
CODEPAGE	NAME=page	START=0x2A	END=0x3FFF	
CODEPAGE	NAME=idlocs	START=0x200000	END=0x200007	PROTECTED
CODEPAGE	NAME=config	START=0x300000	END=0x30000D	PROTECTED
CODEPAGE	NAME=deviid	START=0x3FFFFE	END=0x3FFFFF	PROTECTED
CODEPAGE	NAME=eedata	START=0xF00000	END=0xF000FF	PROTECTED

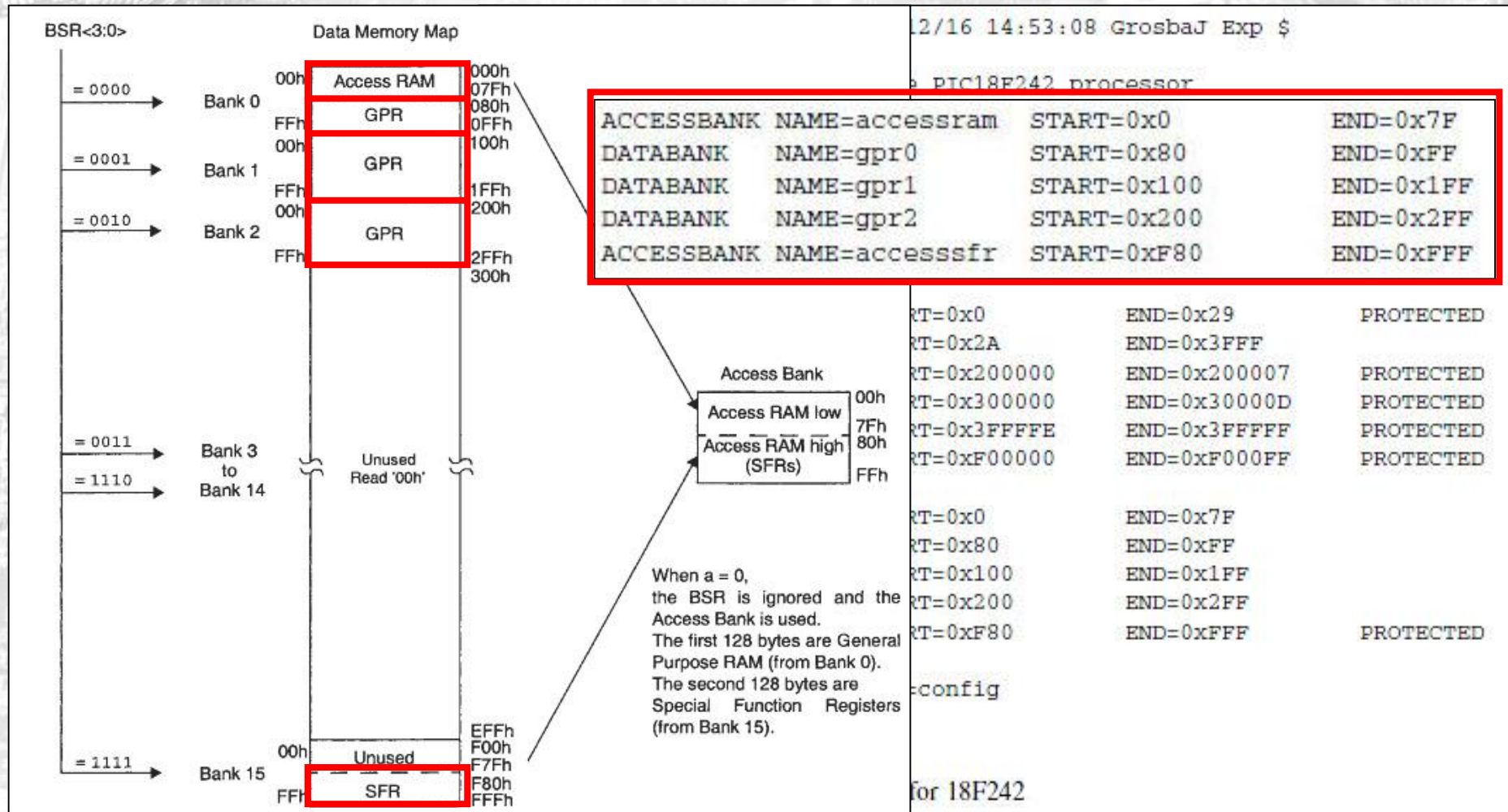
ACCESSBANK	NAME=accessram	START=0x0	END=0x7F	
DATABANK	NAME=gpr0	START=0x80	END=0xFF	
DATABANK	NAME=gpr1	START=0x100	END=0x1FF	
DATABANK	NAME=gpr2	START=0x200	END=0x2FF	
ACCESSBANK	NAME=accesssfr	START=0xF80	END=0xFFFF	PROTECTED

```
SECTION NAME=CONFIG ROM=config
```

```
STACK SIZE=0x100 RAM=gpr2
```

Program Example 17.6 Linker Script for 18F242

Linkeando – Mapa de memoria

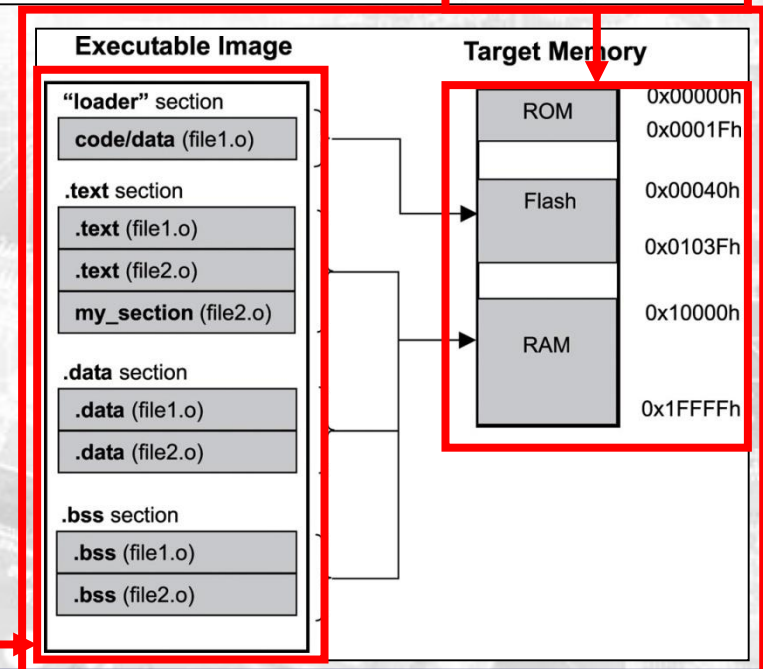
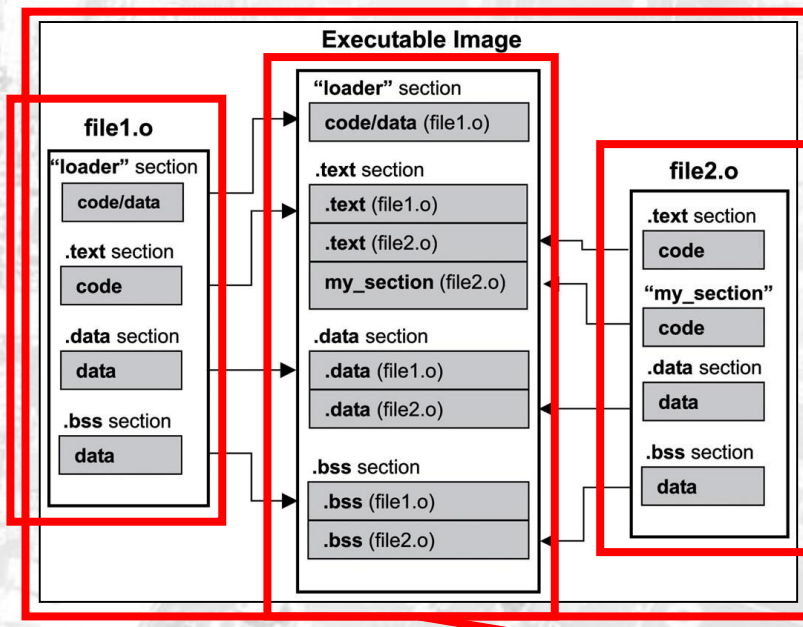
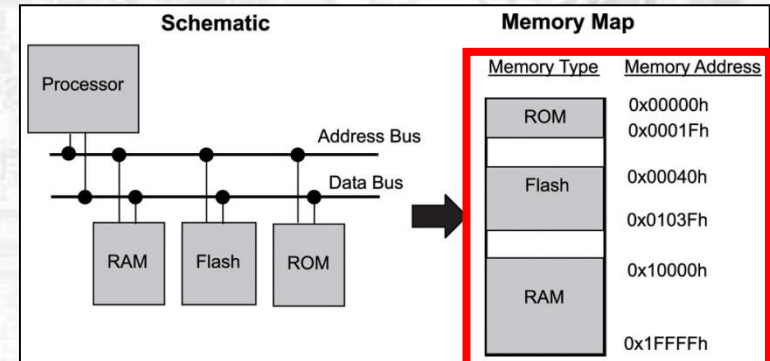


Linkeando y localizando...

- El proceso de locación refleja el mapa de memoria del sistema, y no siempre es directo.
 - Muchos sistemas embebidos copian su programa almacenado en una ROM a memoria RAM para ejecutar más rápido, como parte de su inicialización.
- El locador tiene que ser capaz de generar código que se aloje en cierto espacio de direcciones (ej. ROM), pero que ejecute correctamente en otro (ej. RAM).
 - Trivial si el código es independiente de la posición.
 - Requiere un manejo especial de las direcciones de código en caso contrario.

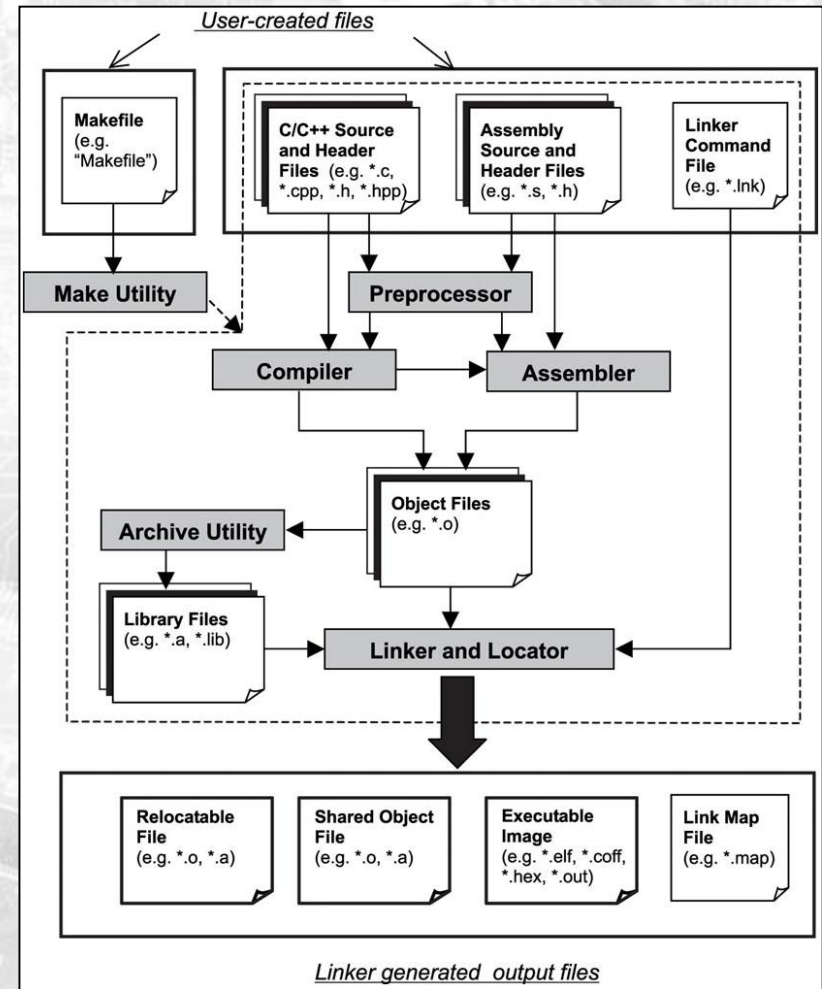
Obteniendo la imagen ejecutable...

- Se parte del mapa de memoria
- Ensamblado/Compilación
- Linkeado
- Localización



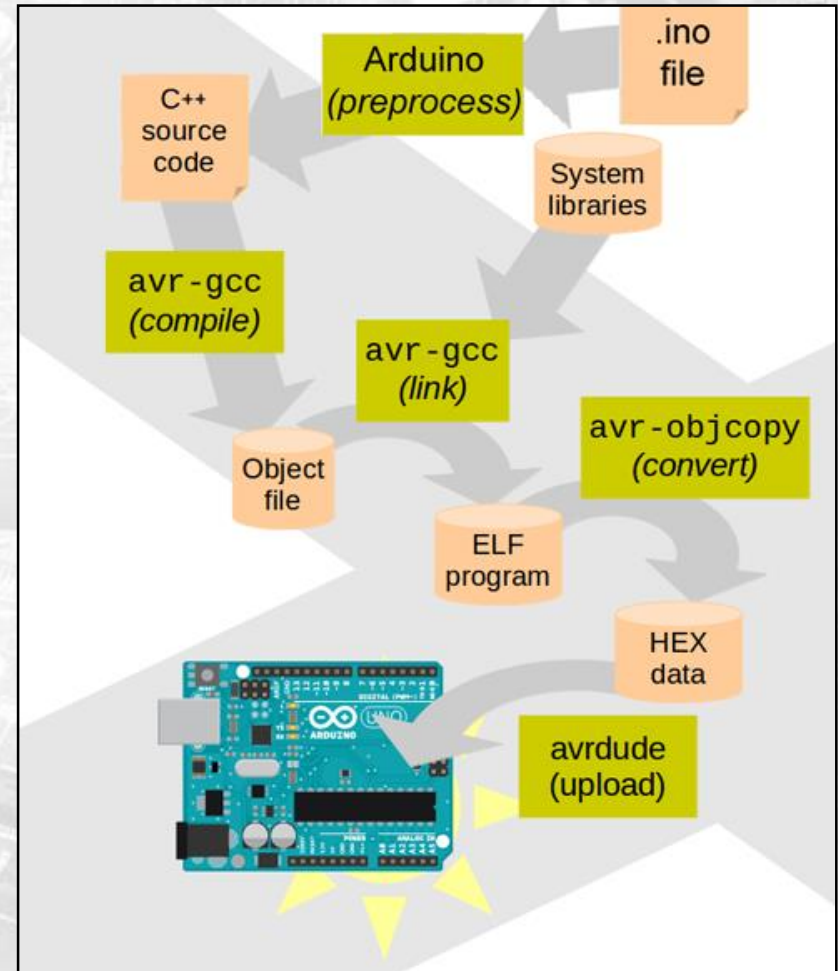
Herramientas adicionales

- Puede haber herramientas adicionales:
 - La utilidad Make
 - El preprocesador de C
 - La utilidad para generar librerías
 - etc.
- Con este encadenamiento de herramientas, obtenemos una imagen ejecutable del sistema en el host.



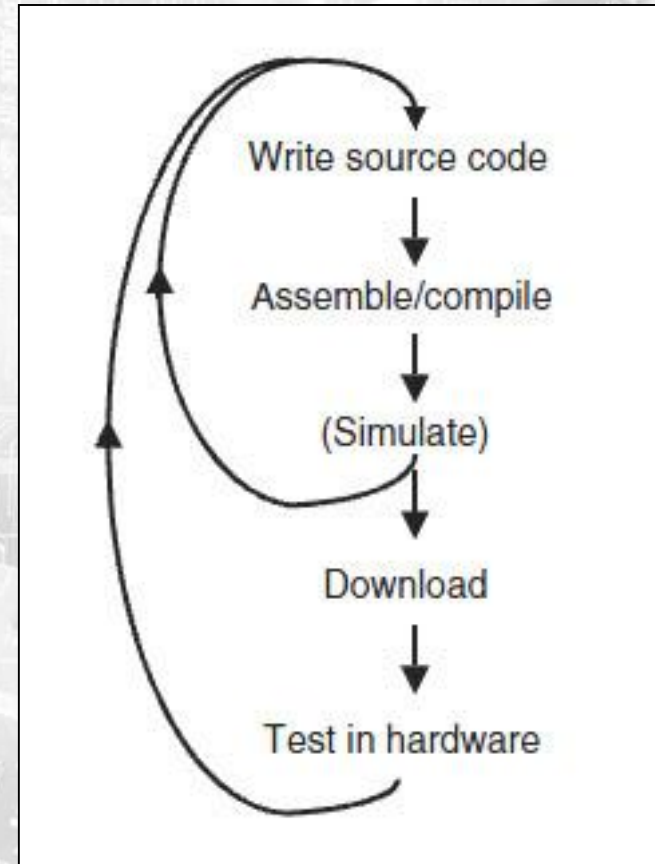
Procesos adicionales – Ej. Arduino


- Arduino realiza un pre-procesamiento que genera código C/C++ a partir de los sketches:
 - Añade el main()
 - Añade #includes para librerías
 - Etc.
- Luego provee el código generado al toolchain de AVR (Atmel).



Desarrollo cruzado

- Una vez que se obtuvo la imagen ejecutable, a partir de ella se puede:
 - simular en el host
 - descargar en el target

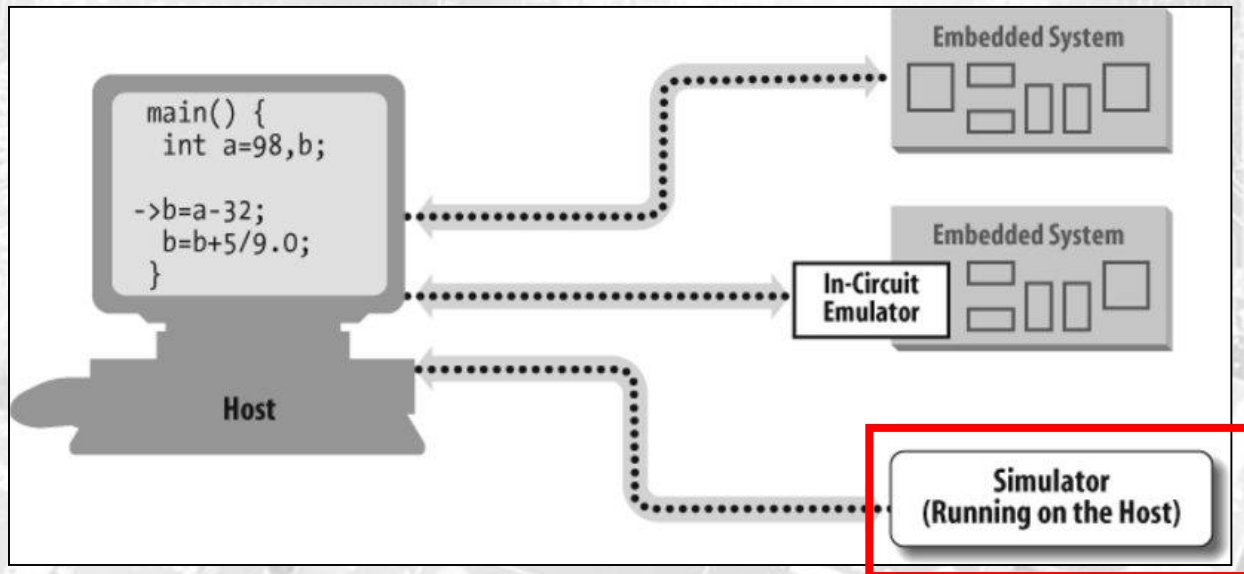


The background of the slide is a detailed, high-resolution image of a printed circuit board (PCB). It shows various electronic components such as integrated circuits, capacitors, and resistors, along with intricate copper traces. The image is slightly faded and has a blueish tint, serving as a technical backdrop for the text.

Simuladores (depuración en el host)

Simuladores – Depuración en el host

- Tres alternativas para depurar un Sist, Embebido:
 - Host based debugging / Simuladores
 - Debugging remoto y agentes de debug en el target
 - HW de debug: ICEs e interfaces de debug “in-circuit”

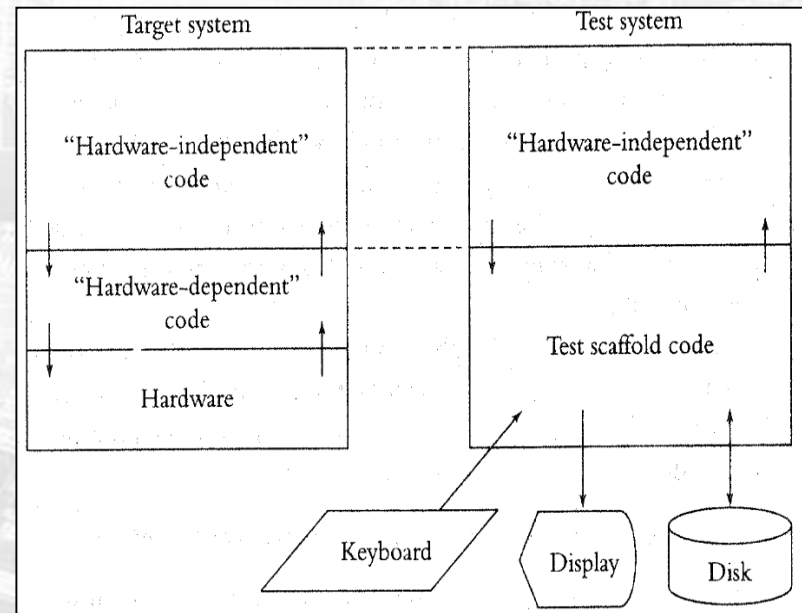


Simuladores – Depuración en el host

- Los simuladores (“Instruction Set Simulators” o “Máquinas Virtuales”),
 - permiten ejecutar las imágenes del software imitando el comportamiento del target en el host.
 - permiten depurar el sistema sin necesidad de descargarlo en la memoria del target.
 - son útiles para las etapas iniciales del desarrollo.
- Existen limitaciones en cuanto a la fidelidad de la simulación lograda (periféricos, temporizado, etc).

Simuladores – Depuración en el host

- Existen limitaciones en cuanto a la fidelidad de la simulación lograda (periféricos, temporizado, etc).
- Sólo se simula el CPU:
 - Scripts complejos para simular Entrada / Salida (archivos de estímulos y logs)
 - Módulos “dummy” para simular el hardware que depende del software.

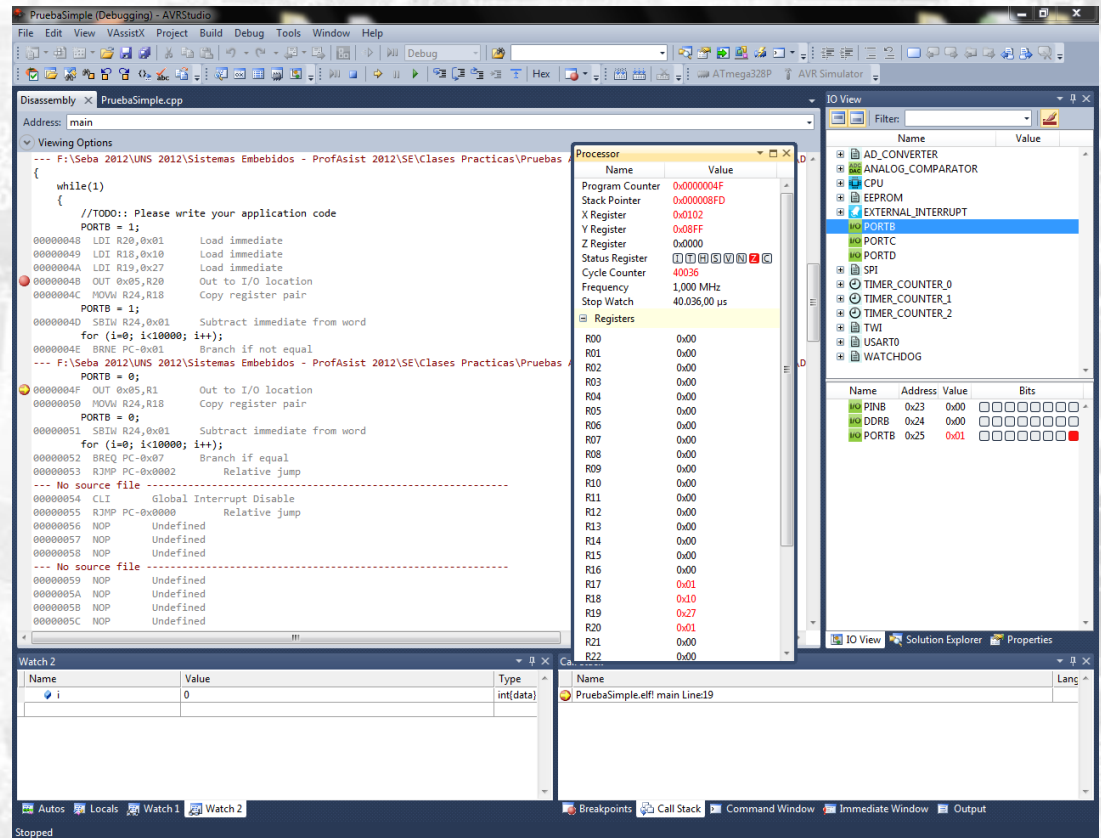


Simuladores – Depuración en el host

- **Host-based debugging:**
 - Pueden surgir problemas de portabilidad:
 - anchos de palabra diferentes (CPU target vs CPU host),
 - little/big endianess,
 - se mitigan usando Instr. Set Simulators / Máquinas Virtuales (usando mismo toolchain y logrando el mismo comportamiento en los CPU simulados y reales).
 - No se puede testear:
 - Interacción del hardware con el software (manejo concreto de dispositivos).
 - Tiempos de respuesta y throughput (se pueden estimar con Instr. Set. Simulators o Máquinas Virtuales).
 - Condiciones de carrera surgidas a partir de ISRs.

Simuladores

- Ejemplo: AVR Studio – AVR Simulator
 - Breakpoints
 - Step by step
 - Watches
 - Traces
 - Simulación de estímulos (E/S temporizadas)
 - Etc.





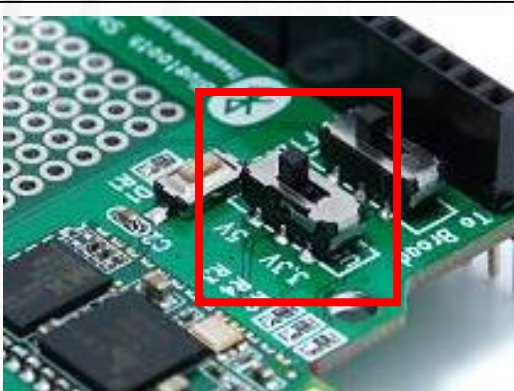
Configuración del target

Configuración del target

- Luego de simular el sistema (verificación independiente del hardware), llega el momento de dejar el host y pasar al target.
- El target (microcontrolador) debe configurarse para su uso, mediante:
 - Niveles lógicos en pines:
 - Existencia de llaves y/o jumpers en el circuito target, de accionamiento manual
 - Bits de configuración (fuses):
 - Determinan parámetros de funcionamiento del sistema
 - No son accesibles al firmware
 - Se configuran durante la descarga del firmware

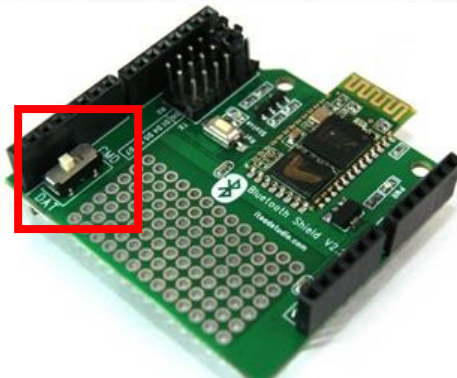
Configuración del target

- Niveles lógicos en pines - ejemplos:



Switch p/voltaje

- Define el voltaje de operación del circuito
- Ej. 3.3 v / 5 v



Modo Cmd/Data

- En Shield Bluetooth
- Modo comandos (para configurar el shield)
- Modo datos (para establecer comunicaciones)



Arduino DFU

- Un puente/jumper define si el bootloader del ATmega16U2 inicia en modo DFU (firmware upgrade) o normal.

Configuración del target

- Los bits de configuración (fuses) definen aspectos esenciales de funcionamiento tales como:
 - Origen y frecuencia del oscilador utilizado con el microcontrolador
 - Modos de programación (bajo voltaje, vía SPI, etc)
 - Multiplexado de ciertos pines, como por ej:
 - Si se habilita el pin de reset
 - Si se utiliza una salida de reloj (CLKOUT)
 - Habilitación de ciertos subsistemas (ej. Watchdog Timer)
 - Selección de temporizado de inicialización del sistema
 - Protección de memoria, tamaño del bootloader, etc
- Ej: <http://www.engbedded.com/fusecalc>

Configuración del target

- Ej. Fuses
PIC18F4550
(Microchip)

Kitsrus.com FUSE Edit

USBPLL	Divide by 2	CPUDIV	Div/6 wPLL, Div/4
PPLDIV	Osc/2 (8Mhz)	OSCSSEN	Enabled
Oscillator	HS OSC w/PLL, HS u	VREGEN	Disabled
Brownout Voltage	2.7V	Brownout Detect	Disabled
Powerup Timer	Disabled	Watchdog Timer	Disabled
Watchdog Postscale	1:32768	MCLRE	Disabled
LPT1OSC	LOW Power Mode	PBADEN	AD 4:0 Reset=Analog
CCP2MX	MUC RC1	ECCMUX	ENH CCP1 RE6 3
XINST	Disabled	Low Voltage Program	Disabled
Stack Overflow Reset	Enabled	BOOT ROM Protect	0000-01FF Disabled
ROM Protect	0200-1FFF Disabled	ROM Protect	2000-3FFF Disabled
ROM Protect	4000-5FFF Disabled	ROM Protect	6000-7FFF Disabled
EEPROM Protect	Disabled	BOOT Table Write Protect	0000-01FF Disabled
ROM Table Write Protect	0200-1FFF Disabled	ROM Table Write Protect	2000-3FFF Disabled
ROM Table Write Protect	4000-5FFF Disabled	ROM Table Write Protect	6000-7FFF Disabled
EEPROM Table Write Protect	Disabled	CONFIG Table Write Protect	Disabled
BOOT Table Read Protect	0000-01FF Disabled	ROM Table Read Protect	2000-3FFF Disabled

ID: FFFFFFFF

OK Cancel Default Help

Configuración del target

- Ej. Fuses ATmega328P (Atmel)
- Con electrónica prediseñada (por ej. Arduino) no hace falta ajustarlos. Cada uC ya está se encuentra configurado de manera acorde.

Table 27-8. Fuse High Byte for ATmega328P

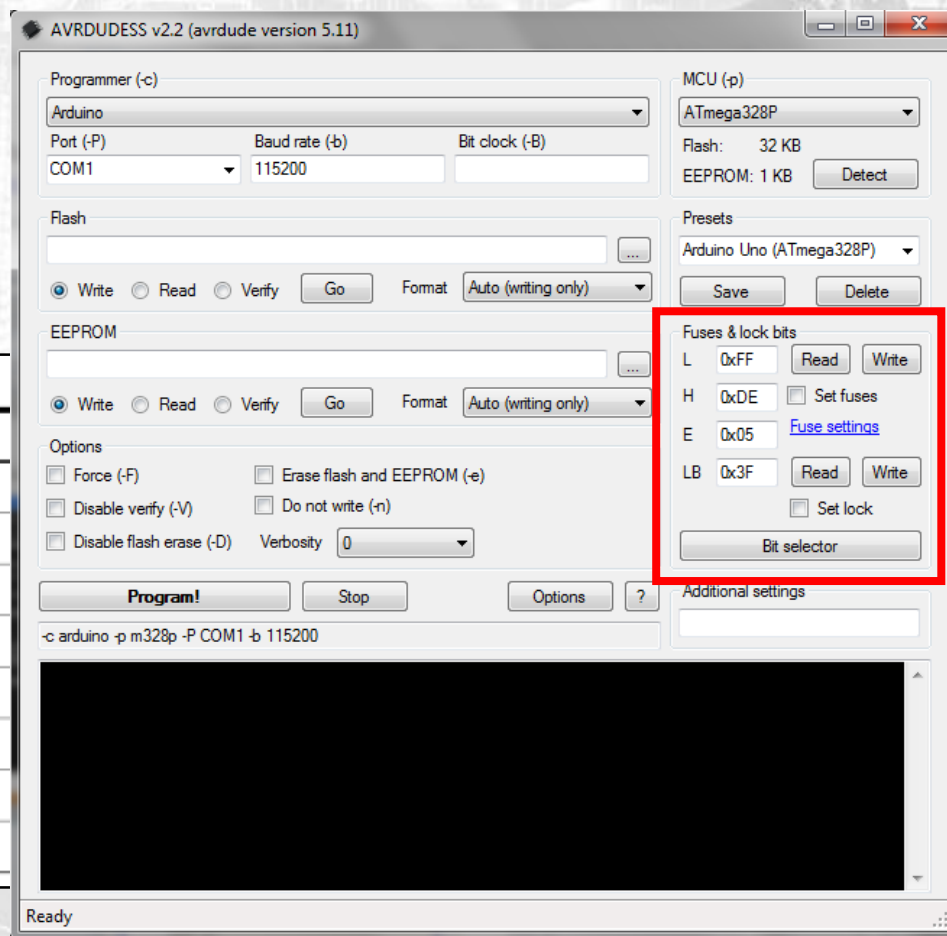
High Fuse Byte	Bit No	Description	Default Value
RSTDISBL ⁽¹⁾	7	External Reset Disable	1 (unprogrammed)
DWEN	6	debugWIRE Enable	1 (unprogrammed)
SPIEN ⁽²⁾	5	Enable Serial Program and Data Downloading	0 (programmed, SPI programming enabled)
WDTON ⁽³⁾	4	Watchdog Timer Always On	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed), EEPROM not reserved
BOOTSZ1	2	Select Boot Size (see Table 26-7 on page 289 , Table 26-10 on page 290 and Table 26-13 on page 291 for details)	0 (programmed) ⁽⁴⁾
BOOTSZ0	1	Select Boot Size (see Table 26-7 on page 289 , Table 26-10 on page 290 and Table 26-13 on page 291 for details)	0 (programmed) ⁽⁴⁾
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Configuración del target

- Ej. Fuses ATmega328P (Atmel)
- Fuses en AVRDudess

Table 27-9. Fuse Low Byte

Low Fuse Byte	Bit No	Description
CKDIV8 ⁽⁴⁾	7	Divide clock by 8
CKOUT ⁽³⁾	6	Clock output
SUT1	5	Select start-up time
SUT0	4	Select start-up time
CKSEL3	3	Select Clock source
CKSEL2	2	Select Clock source
CKSEL1	1	Select Clock source
CKSEL0	0	Select Clock source

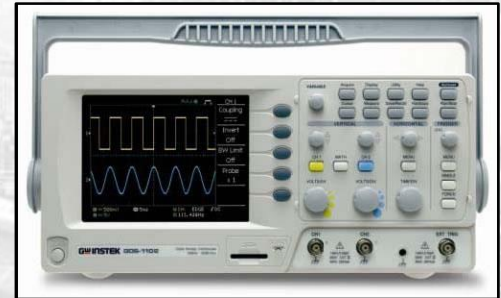


The background of the slide is a detailed, high-resolution image of a printed circuit board (PCB). It shows various electronic components such as integrated circuits, capacitors, and resistors, along with intricate copper traces. The image is slightly faded and serves as a technical backdrop for the text.

Herramientas de Testeo del target

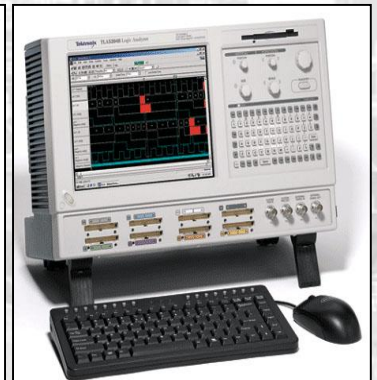
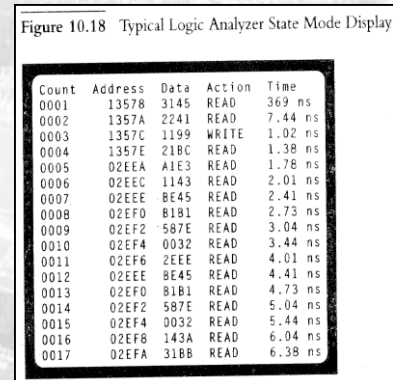
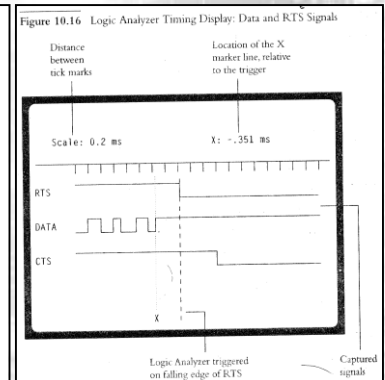
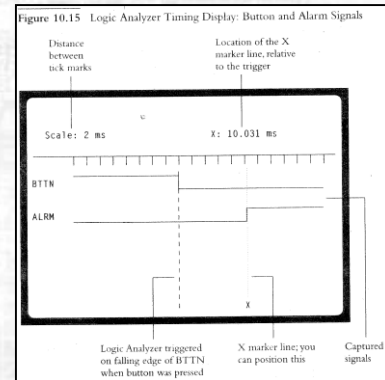
Herramientas de Testeo

- **Herramientas de laboratorio:**
 - Testers
 - Osciloscopios
 - Detectar problemas temporales en señales analógico/digitales (por ej. señal de clock defectuosa)
 - Analizadores lógicos
 - Trackean múltiples señales (Vcc, Gnd solamente) al mismo tiempo
 - Puede almacenar las capturas (a partir de un disparador)
 - Timing mode/State mode



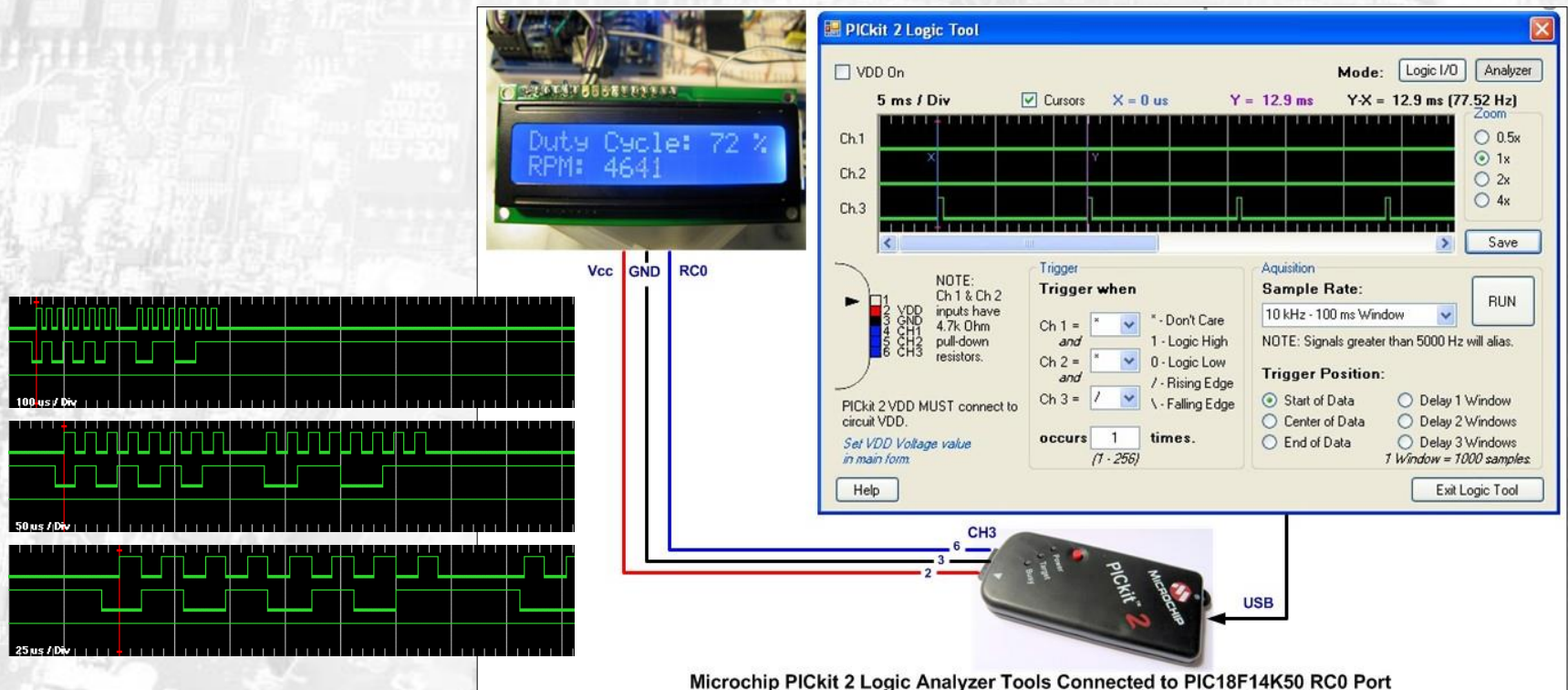
Herramientas de Testeo


- **Analizadores lógicos:**
 - **Timing mode:** para capturar patrones temporales de las señales (sincrónico)
 - **State mode:** captura datos ante la ocurrencia de ciertos eventos (asincrónico)



Herramientas de Testeo

- Analizadores lógicos por Software:



The background of the slide is a detailed, high-resolution image of a printed circuit board (PCB). It shows various electronic components such as integrated circuits, capacitors, and resistors, along with intricate copper traces. The image is slightly blurred and has a warm, yellowish tint. A semi-transparent dark blue horizontal band is overlaid across the middle of the image, containing the main title text.

Descarga y depuración del ejecutable en el target

Descargando el ejecutable al target

- Existen varias alternativas para descargar la imagen:
 - Manufacturar el ejecutable en memoria ROM.
 - Usar un programador de (E)PROMs /Flash
 - Usar una interface de programación “in-circuit”, ya sea propietaria o estándar. (*)
 - Usar un programa Monitor (boot)loader en el target (*).
 - In-Circuit Emulators: mediante overlay de memoria (*).

(*): admiten realizar debug del sistema con los elementos adecuados.

Descargando el ejecutable al target

- **Manufacturar memoria ROM**
 - Sólo apto para el momento de salir al mercado.
 - El fabricante de la ROM estampa la misma con la imagen que ejecutará el target.
 - Imposible realizar cambios.
 - No apto para realizar debugging.

Descargando el ejecutable al target

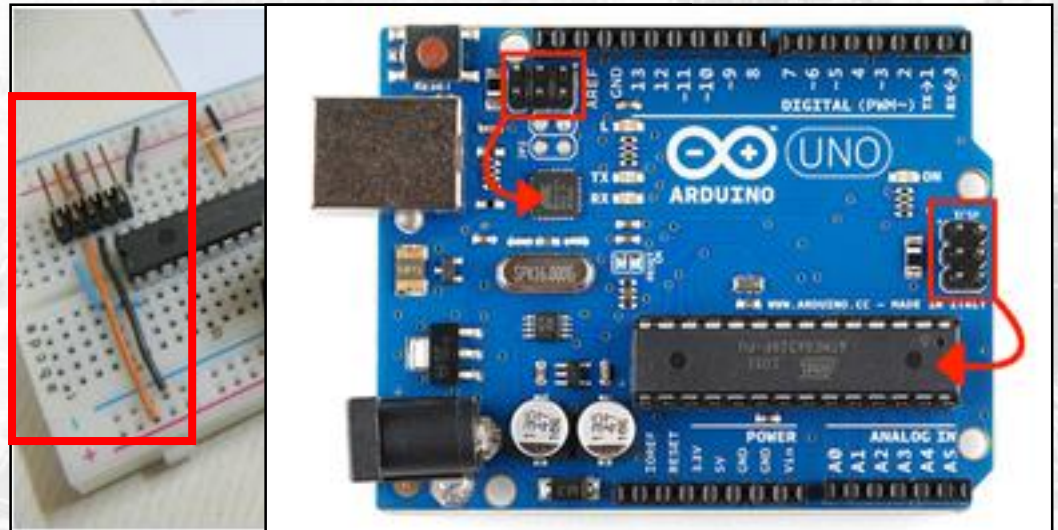
- Programador de memorias (E)PROMs/Flash
 - se conecta al host (generalmente por RS232 o USB).
 - se coloca la memoria (E)PROM o el uC flash, y se les aplica un voltaje diferenciado de programación
 - el host transmite (en gral.) secuencialmente el programa, el cual es cargado en la memoria por el programador.
 - se pueden programar SoCs.

Programador PICSTART Plus



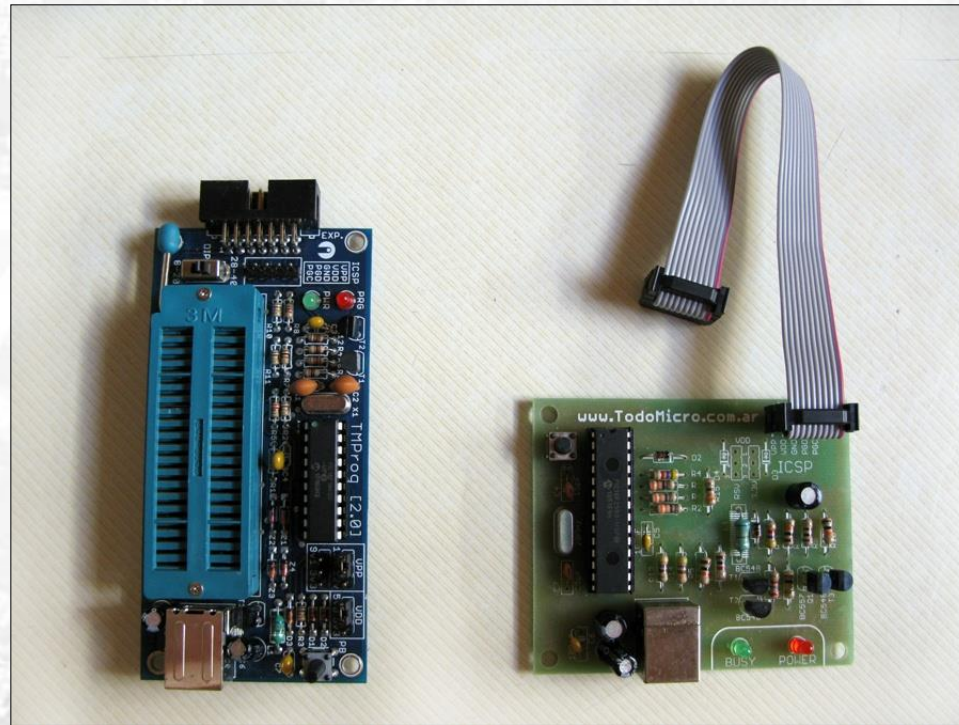
Descargando/depurando en el target

- Interfaces de programación “in-circuit”
 - Se utiliza la misma lógica y dispositivo de programación que en el caso de programadores (E)EPROM / Flash.
 - En el target se implementa un conector (estándar o propietario) que facilita el acceso del programador o debugger a los pines de programación.
 - No requiere extraer chips y es apto para SoCs.
 - Posibilidad de actualizaciones de firmware.



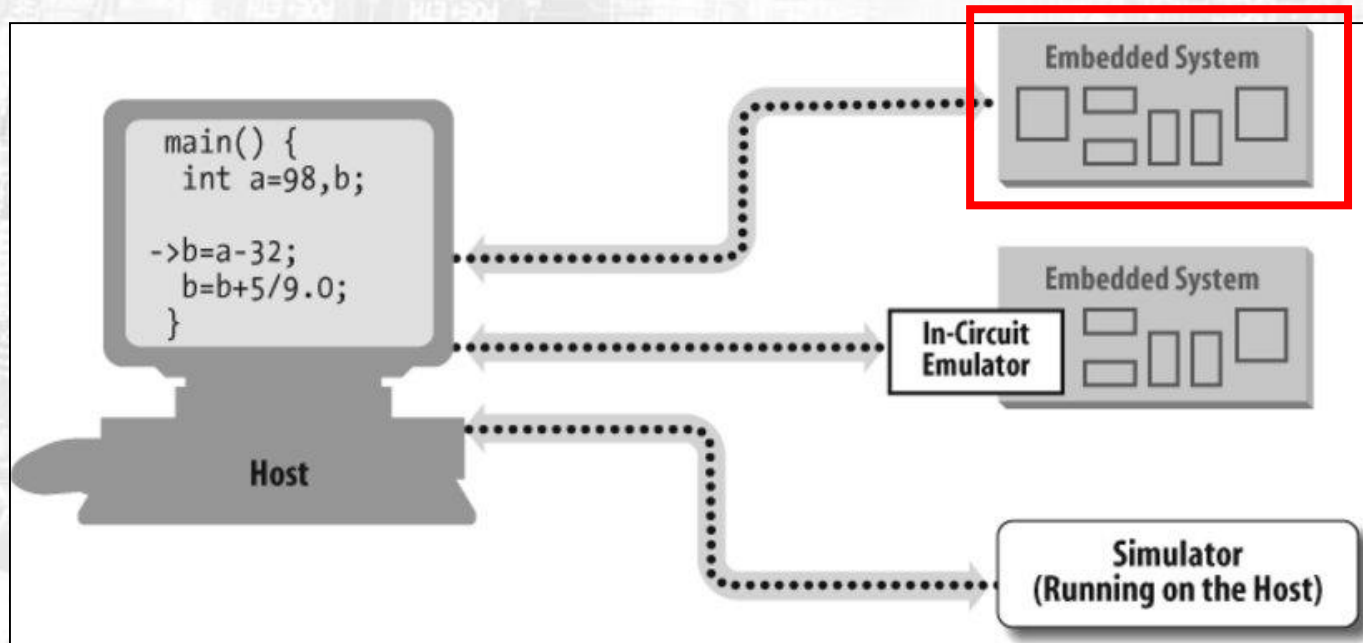
Descargando/depurando en el target

- Programadores de memoria flash:
 - Extraer el microcontrolador vs programar “in-circuit”.



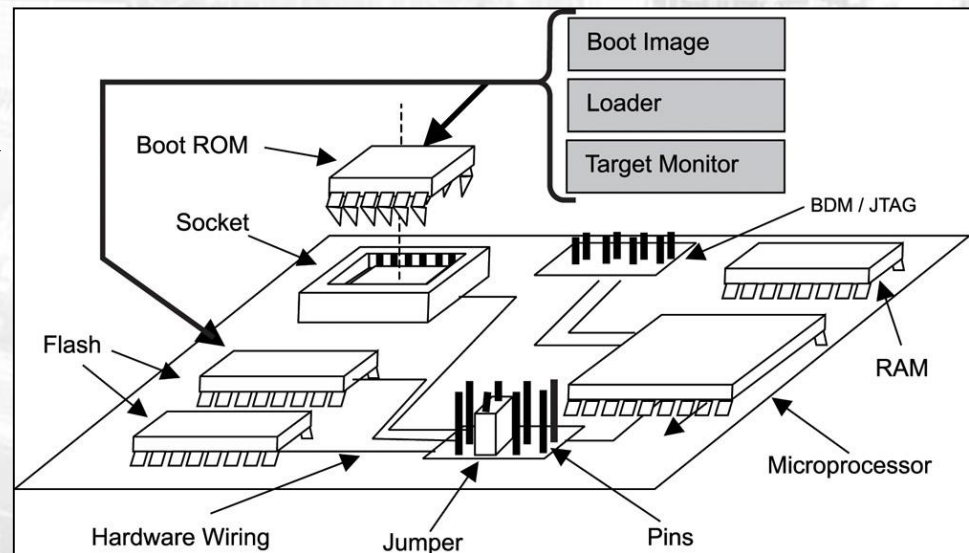
Descargando/depurando en el target

- Programa Monitor / (boot)loader en target:
 - Se incorpora software en el Target.
 - Ofrecerá posibilidad de descarga (y quizás debug)



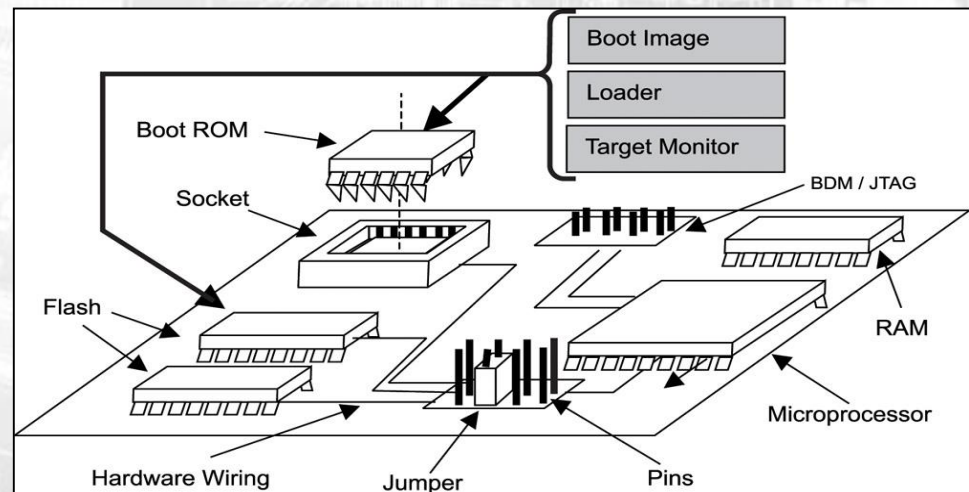
Descargando/depurando en el target

- Programa Monitor / (boot)loader en target:
 - Se necesita un puerto de comunicación en el target para comunicarse con el host.
 - El Monitor reside en el target y es capaz de cargar una imagen (recibida del host) en la RAM/Flash del target.
 - El esquema de boot image, embedded loader y monitor se relaciona con el concepto de bootstrapping.



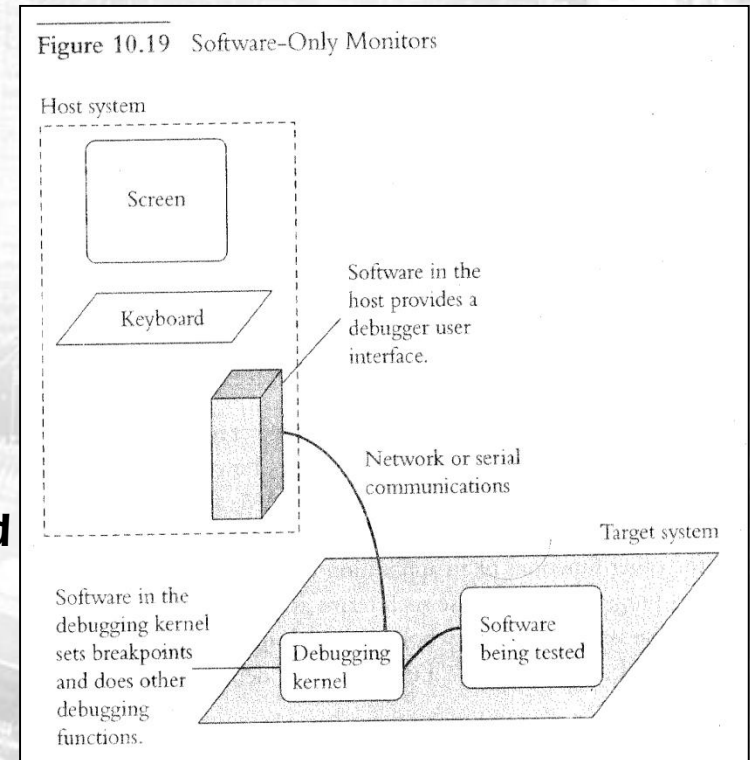
Descargando/depurando en el target

- **Uso de programa Monitor (boot)loader en target:**
 - El embedded loader para memoria flash debe copiarse a la RAM para ejecutar, o alojarse en un espacio especial de flash.
 - Se deben evitar actualizaciones corruptas o incompletas. Evitar perder el loader en una actualización.
 - Es la alternativa usada por Arduino y por Intel Galileo (cuando se la usa con el IDE de Arduino).



Descargando/depurando en el target

- **Uso de programa Monitor (boot)loader en target:**
 - **Si soporta debugging:**
 - El agente recibe comandos de depuración del host mediante la interface de comunicación.
 - Permite establecer breakpoints, correr el programa, paso a paso, etc.
 - Contar con esta funcionalidad no requiere modificaciones al HW (sólo SW).

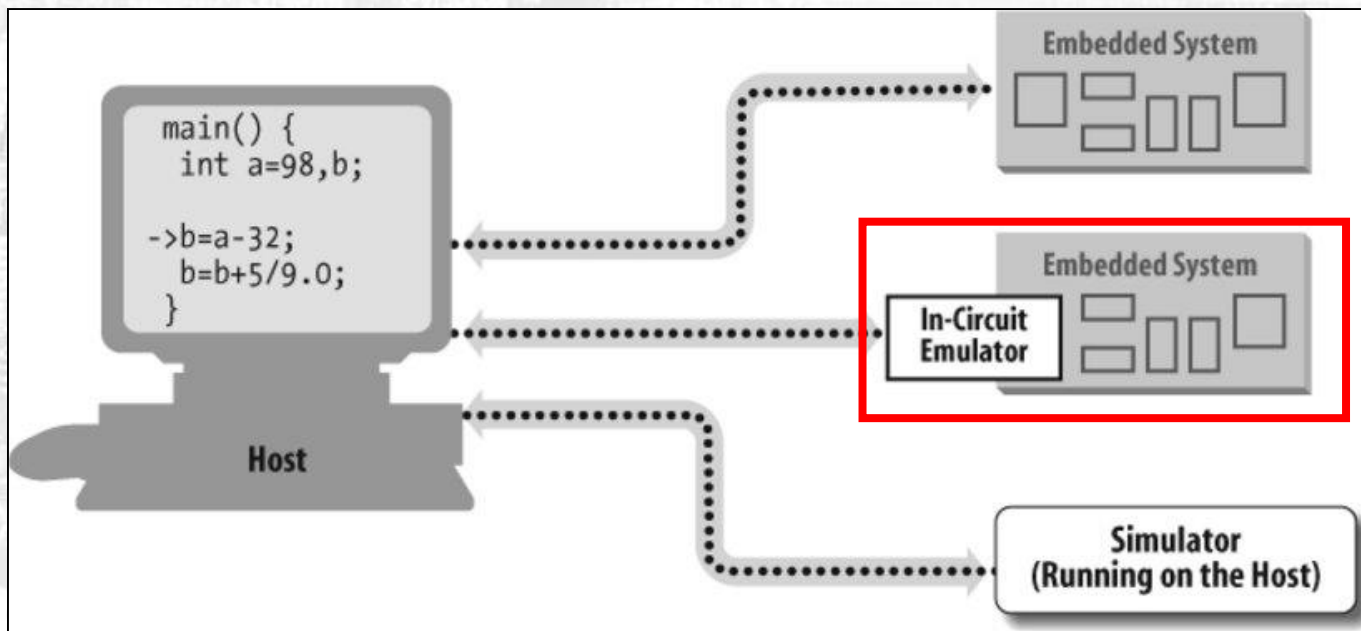


Descargando/depurando en el target

- **Uso de programa Monitor (boot)loader en target:**
 - **Algunas desventajas:**
 - Requiere puerto de comunicación.
 - Tracing limitado (no se pueden capturar algunas de las señales reportadas por emuladores y analizadores lógicos).
 - El entorno de producción podría diferir del entorno de testeo (el kernel de debugging podría no formar parte del producto una vez lanzado).
 - Cuidado con breakpoints y restricciones de tiempo real.
 - Debería poderse asegurar que el ambiente de ejecución del kernel de debugging funciona correctamente por otros medios (no puedo usar el kernel de debugging para debuggearse a sí mismo).

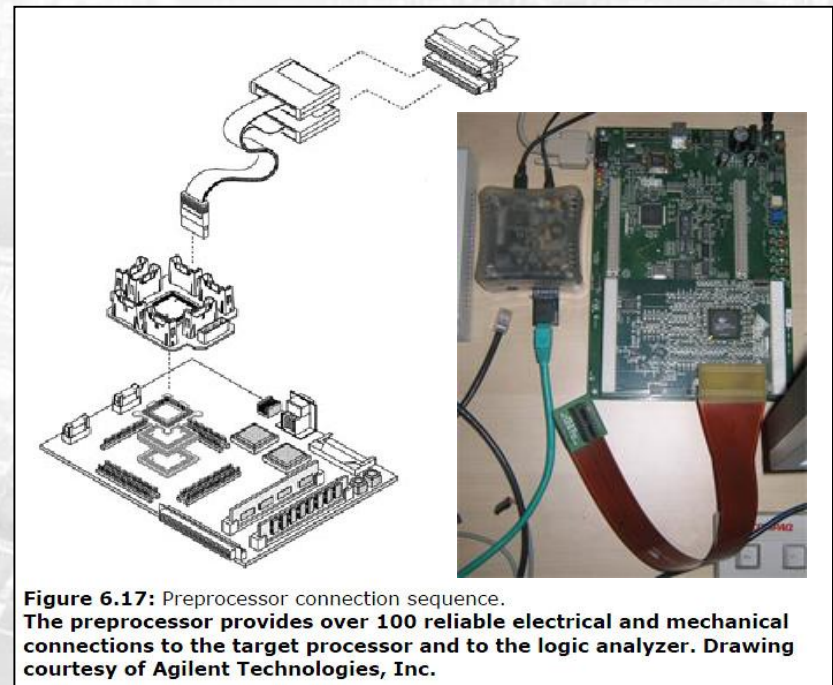
Descargando/depurando en el target

- **Hardware de debugging:**
 - Se utiliza hardware (temporal o permanente) en el target.



Descargando/depurando en el target

- **Hardware de debugging:**
 - Es hardware diseñado con el propósito de facilitar el acceso al target para obtener señales y datos.
 - Menos intrusivo (se depura el target sin agregados).
 - Más adecuado para profiling y evaluar requerimientos temporales.
 - Caen en esta categoría:
 - In-Circuit Emulators (ICE)
 - Interfaces de depuración “in-circuit” / “on-chip”



Descargando/depurando en el target

- **Hardware de debugging: In-Circuit Emulators**
 - Reemplaza al procesador en el target (se conecta a los buses y gestiona todas las señales como si fuera este).
 - Provee facilidades de debug (breakpoints, watches, etc).
 - Además puede realizar capturas temporales similares, aunque menos flexibles, que los analizadores lógicos.
 - Soportan overlay de memoria:
 - reemplazar bloques de memoria del target con bloques de memoria del ICE (útil para “descargar” la imagen al target).




Descargando/depurando en el target

- **Hardware de debugging: In-Circuit Emulators**
 - Se aproximan a la flexibilidad de un analizador lógico, pero:
 - Los analizadores lógicos tienen más alternativas de disparo y mejores filtros de tracing
 - Los analizadores lógicos ejecutan en timing mode (muestreo independiente del clock del target).
 - Los analizadores lógicos funcionan con cualquier microprocesador, no sólo con aquellos que “emulan”.
 - Los emuladores son más invasivos que los analizadores lógicos.

Descargando/depurando en el target

- **Hardware de Debugging: Interfaces de Debug:**
 - Implementan On-Chip debugging.
 - Los procesadores modernos implementan:
 - circuitería específica de debugging en su núcleo
 - una interface/puerto especial para manejar la información de debugging, establecer breakpoints, etc.
 - No se necesita hardware adicional, ya se encuentra implementado en el mismo target.
 - Menos intrusivo que el uso de kernels de debugging (sin software adicional en el target).
- **Ej: BDM, JTAG, Nexus (las veremos más adelante)**



Inicialización y carga

Inicialización y carga

- La ejecución de todo microcontrolador comienza en el vector de reset (o dirección equivalente).
- Antes de que el sistema esté listo para funcionar hay que realizar ciertas tareas:
 - inicializar el hardware
 - inicializar el ambiente de runtime (lenguaje de alto nivel)
 - podría suceder que haya que cargar el software embebido (booteo)
- Luego de esos pasos recién se puede transferir el control a la lógica del sistema.
- Es un proceso similar al de sistemas de escritorio.

Carga en sistemas de escritorio

- El ejecutable se genera en un formato soportado por el loader del sistema operativo (ELF – Linux y Unix, PE – Windows, etc).
- El cargador en un ambiente de escritorio:
 - Puede concluir la localización de la imagen
 - Crea el proceso en el sistema
 - Inicializa el ambiente de runtime
 - Transfiere el control al programa cargado
- Ante librerías de enlace dinámico, el loader localiza y carga dichas librerías.

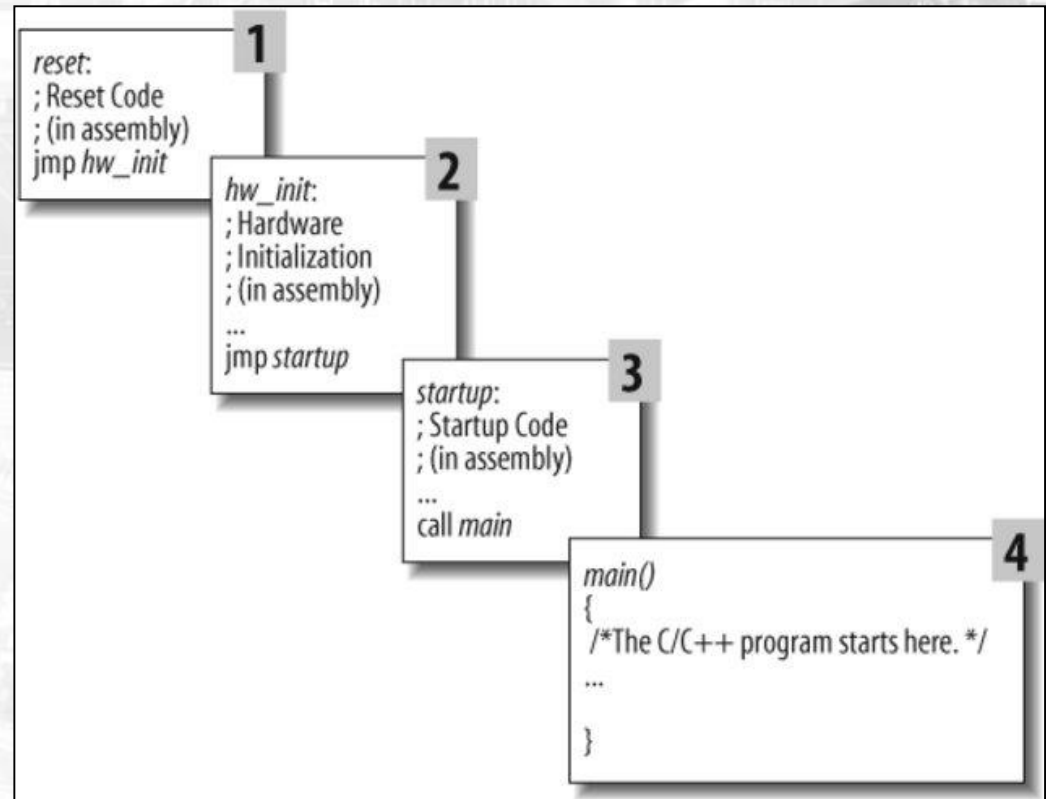
Linkable File	Executable File
ELF Header	ELF Header
Program Header Table (optional)	Program Header Table
Section 1 data	Segment 1 data
Section 2 data	
⋮	⋮
Section <i>n</i> Data	Segment <i>n</i> Data
Section Header Table	Section Header Table (optional)

Carga en sist. Emb. - Bootstrapping

- En sistemas embebidos la carga puede darse:
 - Durante la programación del microcontrolador (no existe software de carga)
 - Como consecuencia de las tareas de un (boot)loader, en sistemas embebidos simples
 - A cargo de un sistema operativo embebido y su (application)loader (proceso de carga similar al contexto de escritorio).
- Bootstrapping: para iniciar un sistema
 - a partir del inicio sucesivo de subsistemas menores
 - cada uno aporta mayor funcionalidad hasta completar la inicialización y carga

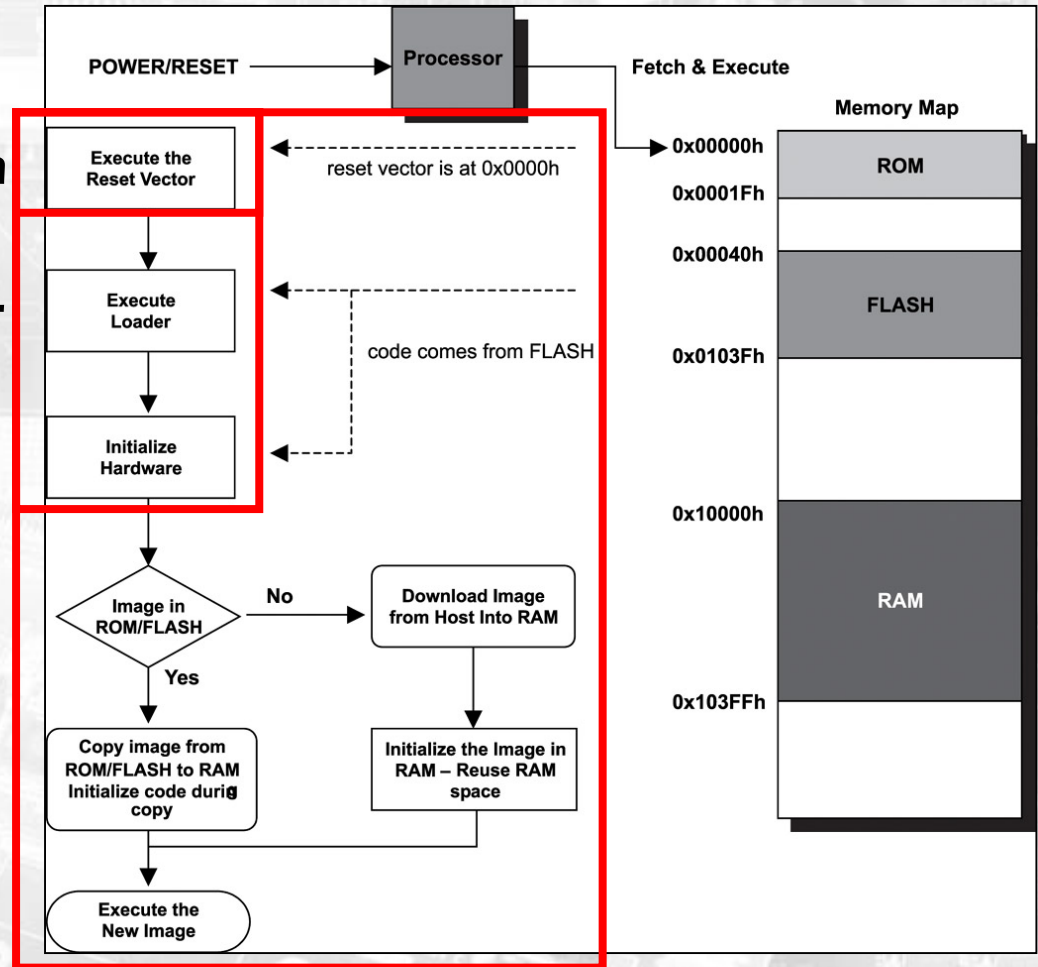
Inicialización - Bootstrapping

- **Bootstrapping de SE Simple:**
 - Salta según el vector de reset.
 - Inicializa el hardware.
 - **Startup:** inicializa el runtime de C/C++ (añadido por el compilador).
 - Invoca a main.



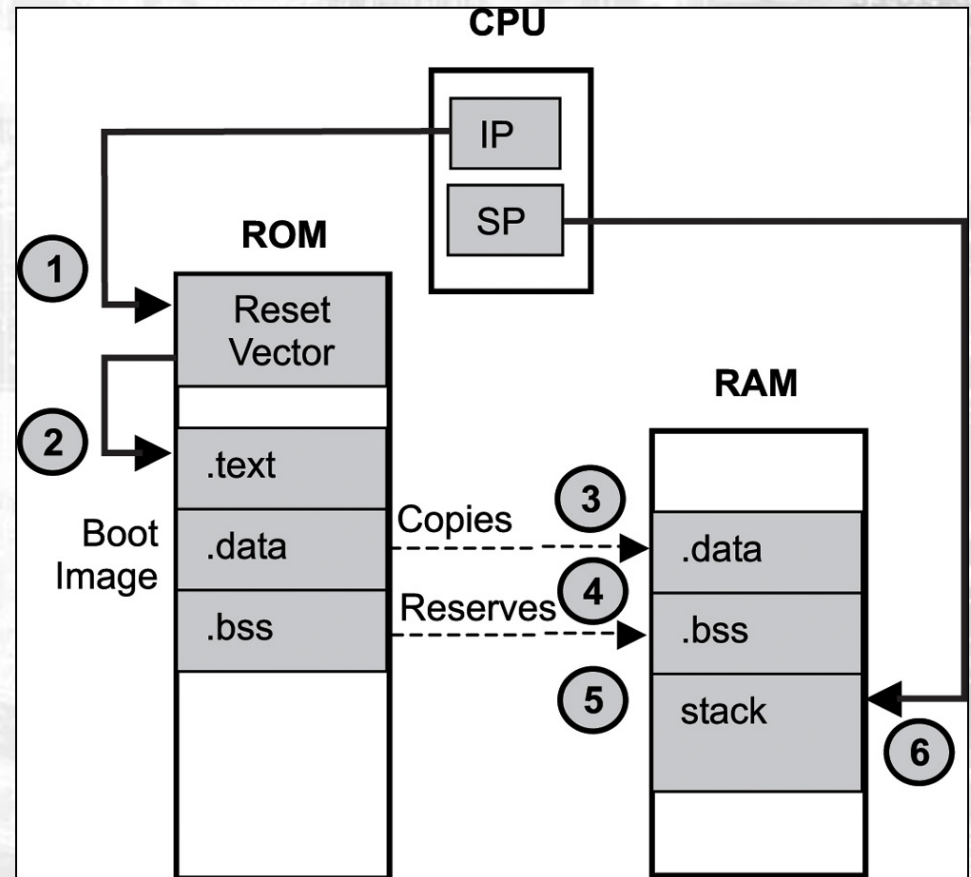
Inicialización - Bootstrapping

- **Ejemplo:**
 - inicialización de un sistema que posee un monitor/loader.
- **3 pasos similares:**
 - se inicia el sistema
 - se inicia el loader y el hardware
 - se inicia la imagen cargada



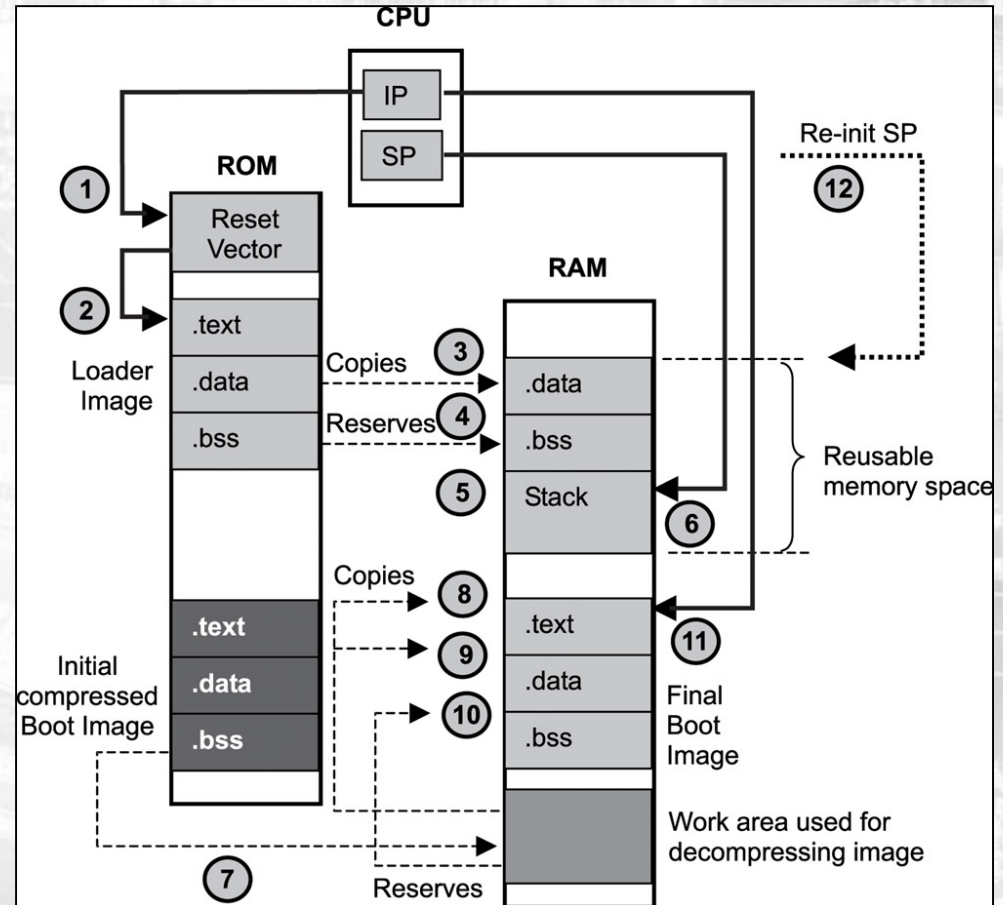
Inicialización - Bootstrapping

- Ejecución en RAM
(se copia la imagen desde la ROM)



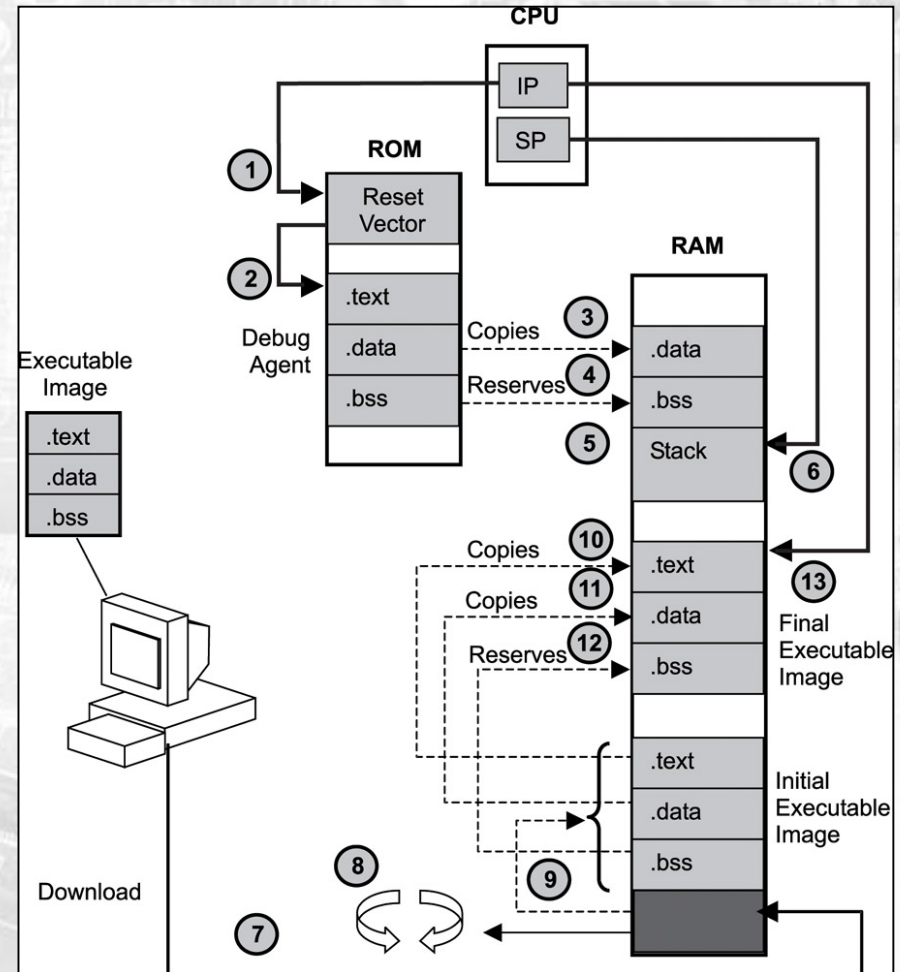
Inicialización - Bootstrapping

- Ejecutando en RAM (se copia y descomprime la imagen desde la ROM)
- Datos en RAM (se copian desde la ROM)



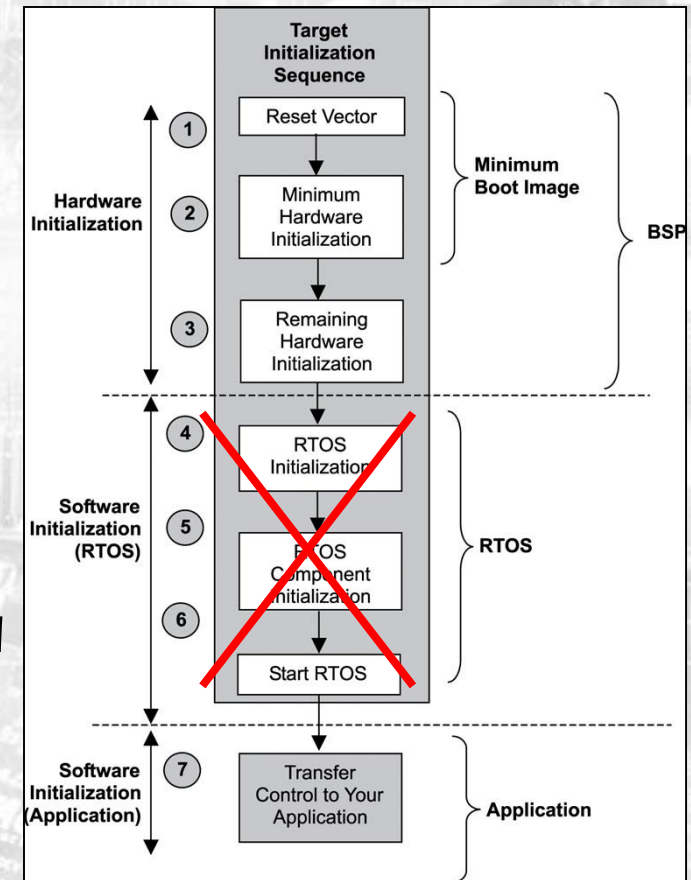
Inicialización - Bootstrapping

- Ejecutando en RAM (la imagen se descarga desde el host)
- El bootloader se copia a RAM para su ejecución, y el ejecutable se transfiere desde el host



Inicialización - Bootstrapping

- **Proceso de inicialización del software.**
 - carga el boot loader...
 - e inicializa el hardware mínimo
 - se completa la inicialización del hardware (board support pack.)
 - se carga el sistema operativo (si hubiese uno)
 - finalmente se transfiere el control a la aplicación



Referencias

- Barr, M., Massa, A. Programming Embedded Systems: With C and GNU Development Tools, 2nd Edition. O'Reilly Media. 2006. ISBN: 978-0596009830. Capítulos 3,4 y 5.
- Berger, A. Embedded Systems Design: An Introduction to Processes, Tools & Techniques. CMP Books. 2001. ISBN: 978-1578200733. Capítulos 4 y 6.
- Li, Q., Yao, C. Real-time concepts for Embedded Systems. CMP. 2003. ISBN: 978-1578201242. Capítulos 2 y 3.
- Simon, D. An Embedded Software Primer. Addison-Wesley Professional. 1999. ISBN: 978-0201615692. Capítulos 9 y 10.
- Wilmshurst, T. Designing Embedded Systems with PIC Microcontrollers: Principles and Applications. Newnes. 2006. ISBN: 978-0750667555. Capítulos 4 y 5.

