Entorno de Desarrollo de Software

PEDRO IGNACIO MARTOS

PMARTOS@FI.UBA.AR

Contenido

- **Introducción**
- Entorno SDK
- Creación de Proyectos
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- Linker script

Desarrollo de software para sistemas embebidos

Software Tradicional

- Se desarrolla, depura y ejecuta en la misma estación de trabajo.
- Un sistema operativo de alto nivel carga el programa en memoria cuando va a ejecutarlo.
- Se utiliza memoria virtual, la traducción de memoria virtual a física la hace el procesador.
- El programa al finalizar devuelve el control al sistema operativo.

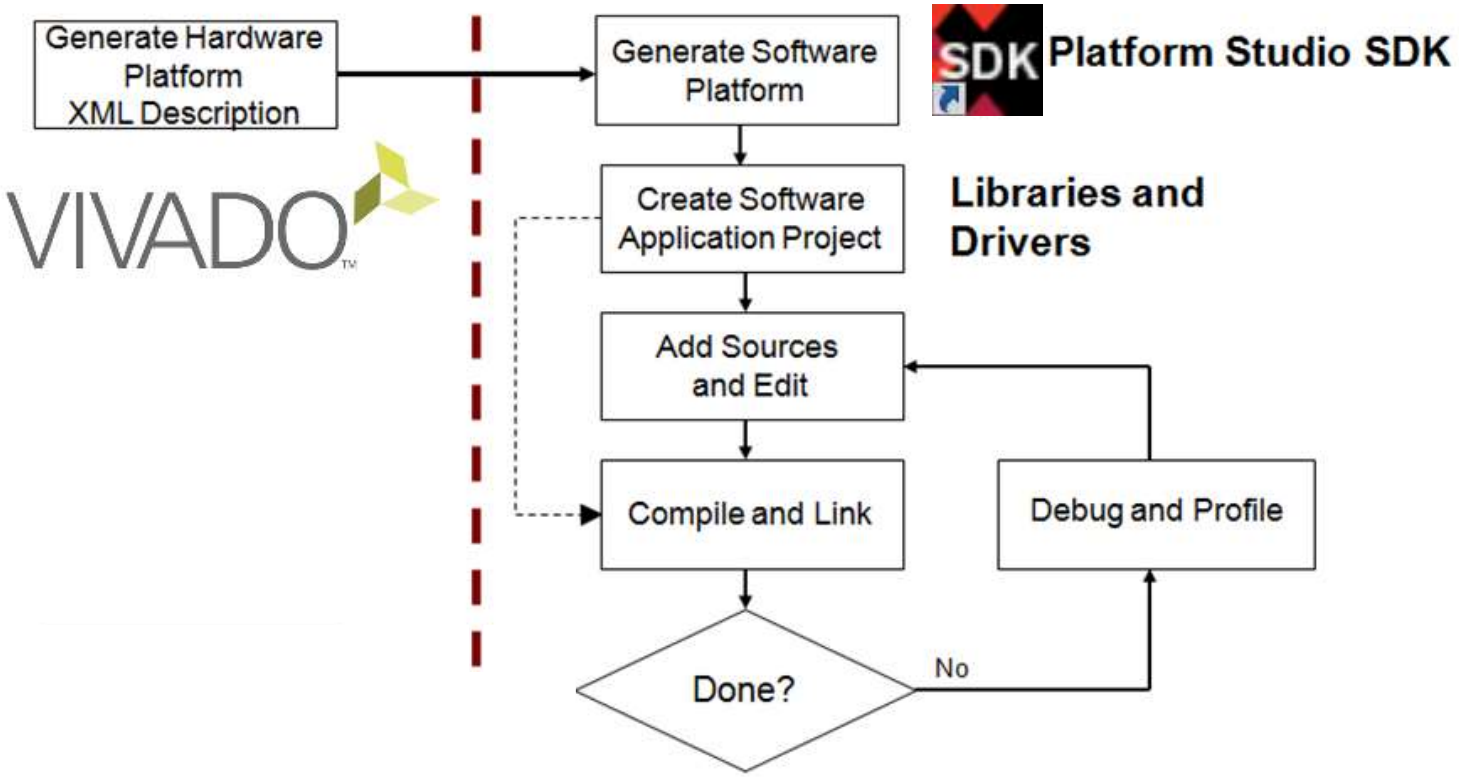
Software para sistemas embebidos

- Se desarrolla en una estación de trabajo y se ejecuta en un dispositivo de computo que puede tener otra arquitectura. La depuración se hace en parte en la estación de trabajo y en parte en el dispositivo que ejecutará el software.
- No hay división sistema operativo/aplicación, se utiliza una imagen ejecutable que contiene todo el software (Boot, Rtos, Aplicación, ISR, controladores, etc).
- Se utiliza solo memoria física.
- El software de aplicación no tiene un punto de finalización.

Contenido

- Introducción
- **Entorno SDK**
- Creación de Proyectos
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- Linker script

Ciclo de desarrollo utilizando SDK



Entorno: Workspace y Perspectivas

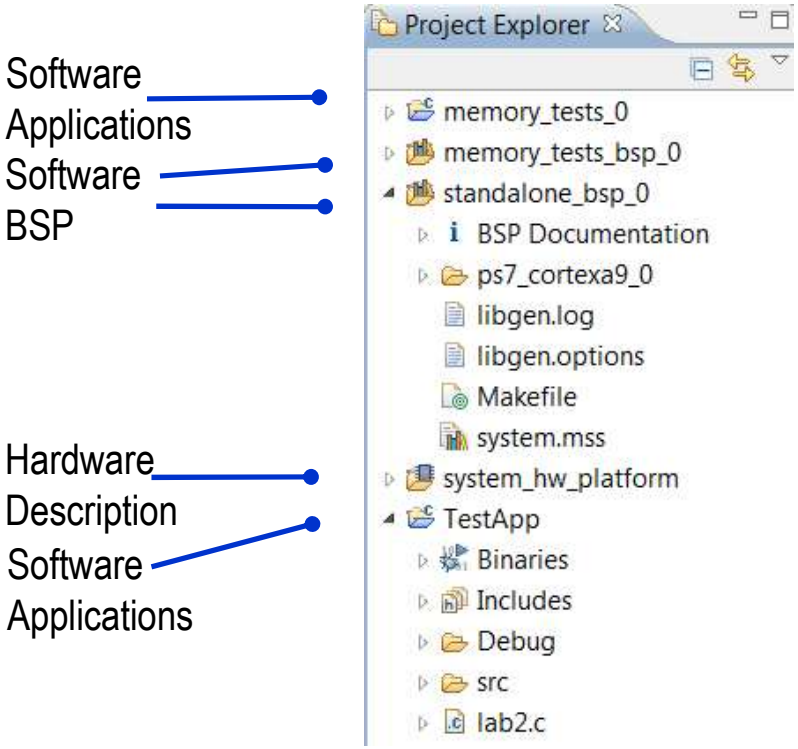
- Workspace:
  - Es el repositorio físico de los proyectos y donde se guarda la configuración general.
  - Normalmente solo se utiliza un workspace con varios proyectos
  - Si se tiene distinto hardware, conviene definir un workspace propio para cada uno

- Vistas y Editores:
  - Son los elementos donde se desarrolla el proyecto: en los editores se trabaja código (normalmente en lenguaje C) y en las vistas se ve información del software a través de interfaces gráficas.

- Perspectivas:
  - Son un conjunto de vistas y editores cuya funcionalidad esta relacionada (p.ej. Depuración)

Vista C/C++ Project

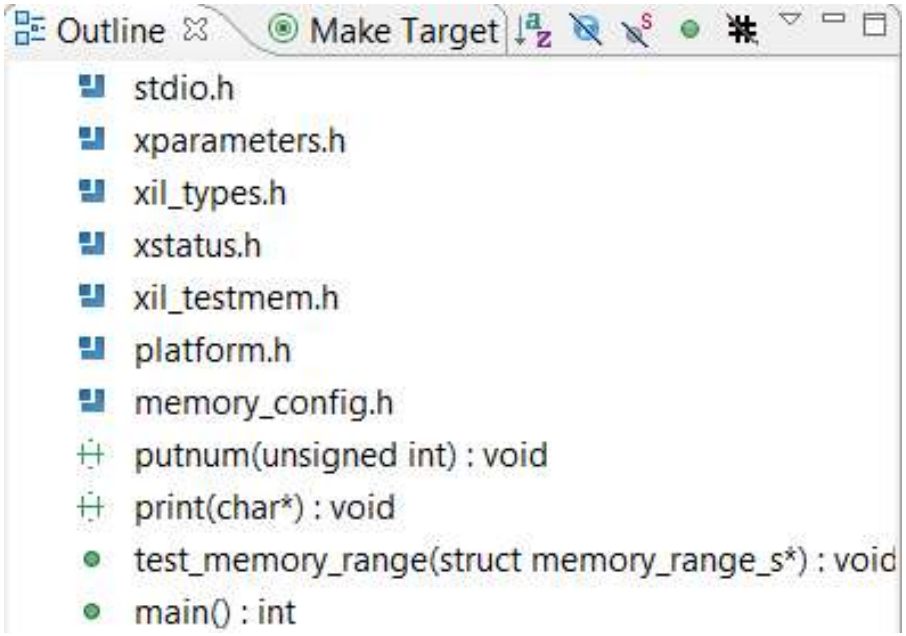
- Muestra los proyectos dentro del workspace en forma jerárquica
- Se accede a cada archivo en forma directa (doble click sobre el archivo)
- Se puede acceder a la configuración de cada proyecto en forma directa (click derecho sobre el proyecto)



Vista Outline

- Muestra la estructura del archivo abierto en el editor.
- El contenido de la vista depende del editor que se esta utilizando.
- Se identifica el tipo de contenido mediante iconos.

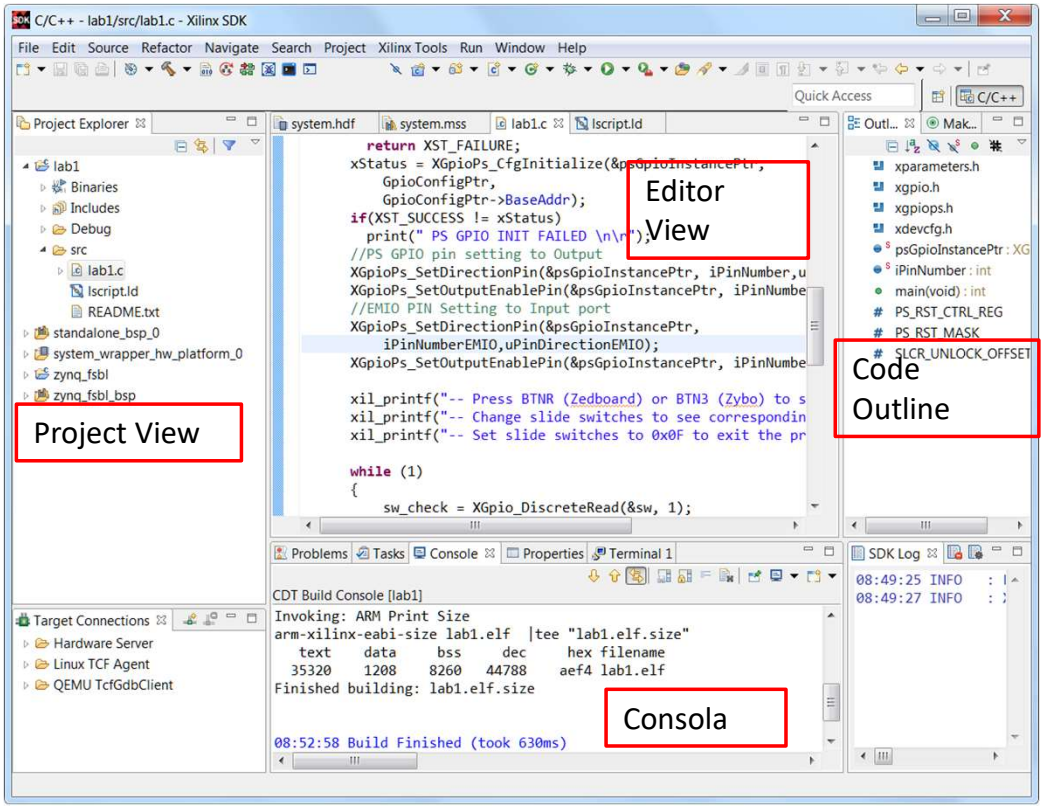
  Ejemplo: en un archivo de C los iconos representan
  - Sentencias #define
  - Archivos cabecera (.h)
  - Prototipos de funciones
  - Definición de funciones (cabecera + cuerpo)
- Seleccionar un elemento en la vista permite navegar al mismo elemento en el editor.



Perspectiva C/C++

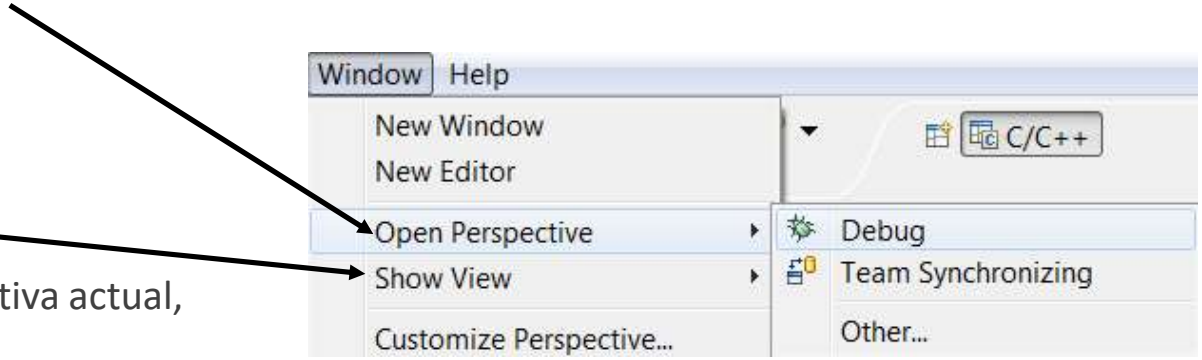
Elementos:

- Vista Project: muestra los archivos del proyecto identificando con iconos cada tipo de archivo
- Vista Outline: muestra elementos del archivo actualmente en el editor de código
- Editor de código: Orientado a C/C++ con resaltado de texto
- Vistas Console, Problems: muestran información de compilación



Acceso a Vistas y Perspectivas

- Perspectivas
  - Menu Window -> OpenPerspective
- Vistas
  - Menu Window -> Show View
  - Si la vista esta presente en la perspectiva actual, quedará resaltada



Contenido

- Introducción
- Entorno SDK
- **Creación de Proyectos**
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- Linker script

Iniciar la herramienta SDK

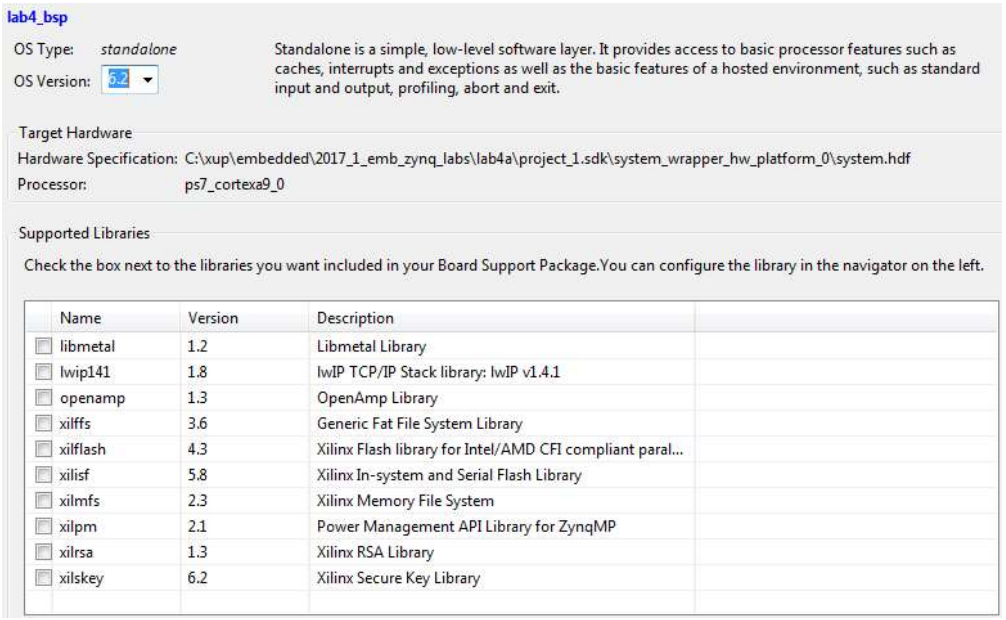
- En forma independiente
  - Permite especificar el workspace
  - Permite especificar la plataforma de hardware a utilizar

- Desde la herramienta Vivado
  - Exportar el hardware (Menu File -> Export Hardware) para especificar la plataforma de hardware
  - Iniciar la herramienta (Menu File -> Launch SDK)

- En ambos casos:
  - La plataforma de hardware se describe mediante un archivo HDF (Hardware Description File)
  - Se genera un proyecto Hardware Platform Specification en forma automática
  - Adicionalmente se puede crear un proyecto BSP (board support package) y un proyecto de aplicación asociados a la plataforma de hardware

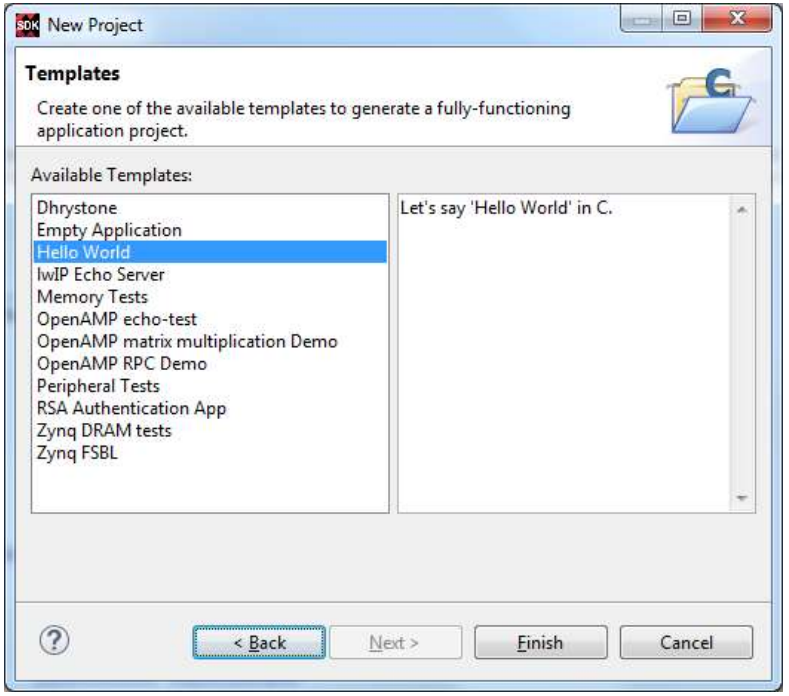
Proyecto BSP

- Este proyecto contiene todo el middleware (controladores de periféricos y RTOS) necesarios para trabajar con el sistema descripto en la plataforma de hardware.
- No incluye los controladores asociados a IP Propia en el hardware. Estos controladores están en el proyecto Hardware Platform Specification, junto con el software de inicialización necesario para configurar el hardware.
- Siempre tiene asociado un proyecto Hardware Platform Specification
- Se puede crear en forma independiente o cuando se crea un proyecto de aplicación



Proyecto de Aplicación

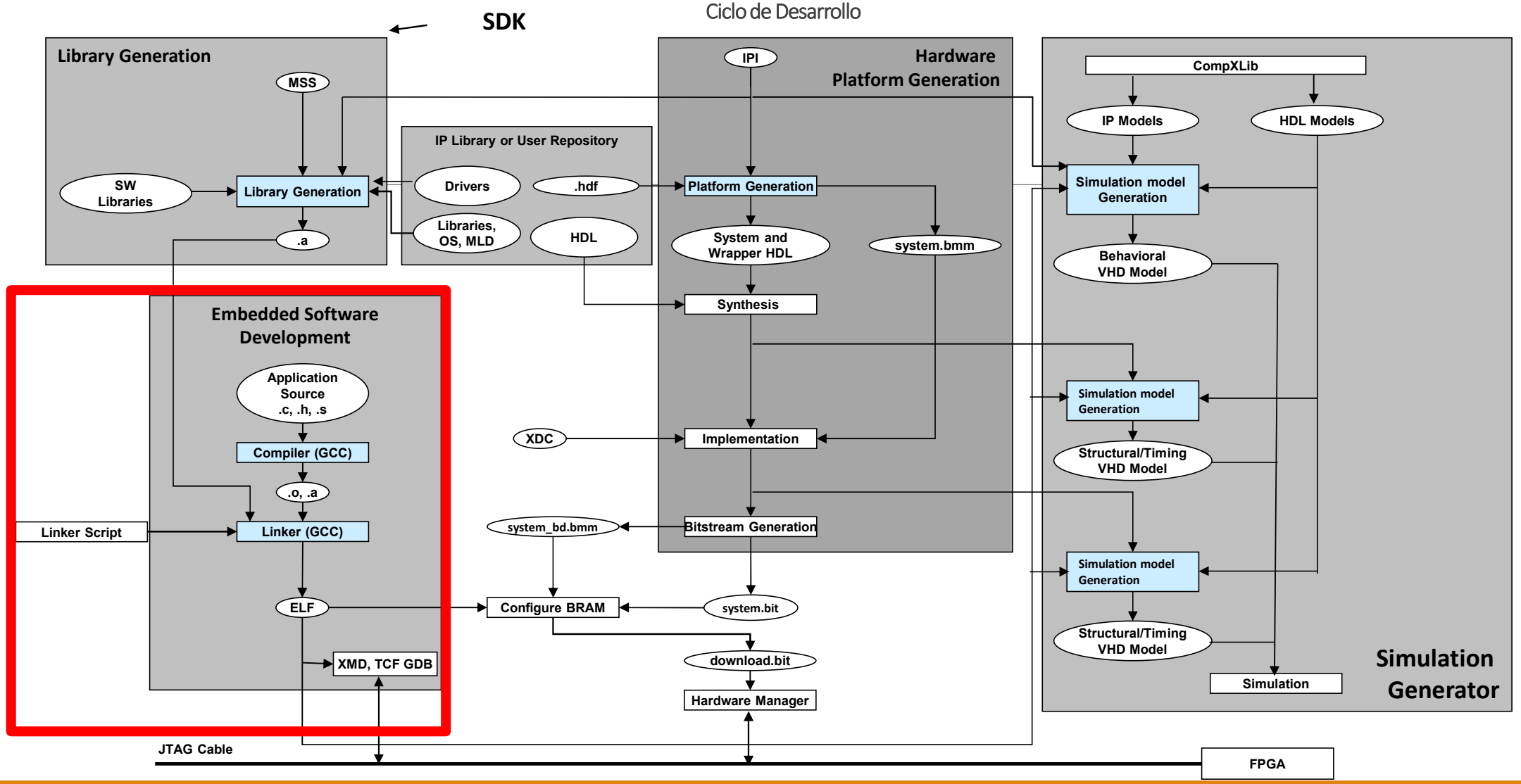
- Un proyecto de Aplicación siempre tiene asociado un proyecto BSP
- Puede haber varios proyectos de aplicación asociados al mismo proyecto BSP
- Hay disponibles plantillas con proyectos de aplicación básicos para verificar la funcionalidad de los periféricos y como base para desarrollar aplicaciones propias.
- Opcionalmente se puede crear un proyecto vacio (Empty Application) para desarrollar una aplicación propia desde el principio.
- Estos proyectos pueden tener distintas configuraciones
  - Debug: Para depuración (configuración por defecto)
  - Release: Codigo de Producción
  - Profile: Para medición de tiempos de ejecución



Contenido

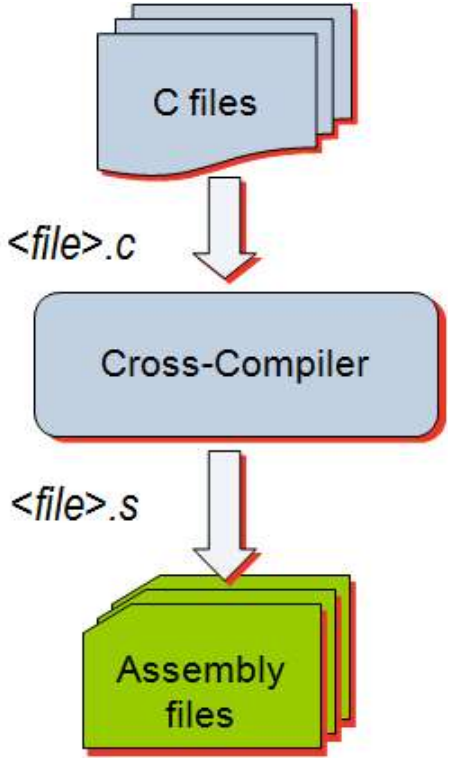
- Introducción
- Entorno SDK
- Creación de Proyectos
- **Herramientas de desarrollo GNU**
- Configuración del software de sistema
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- Linker script

Ciclo de Desarrollo



Herramientas: GCC

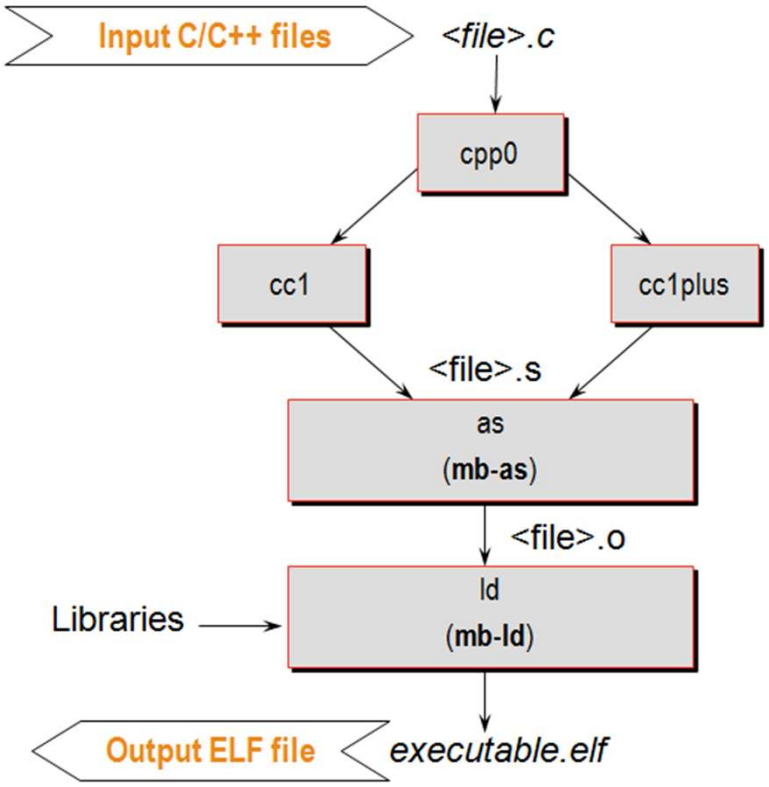
- GCC es el compilador de código fuente en lenguaje C a assembler del procesador (ARM)
- Tambien es la interfaz a otras herramientas (GNU Assembler – GAS y GNU Linker – Ld) invocándolas con los parámetros apropiados.
- Es un compilador cruzado (cross compiler), se ejecuta en una arquitectura x86_64 y compila a una arquitectura ARM32
- Es una herramienta de línea de comandos que se invoca con parámetros configurados dentro del entorno.



Herramientas: GCC

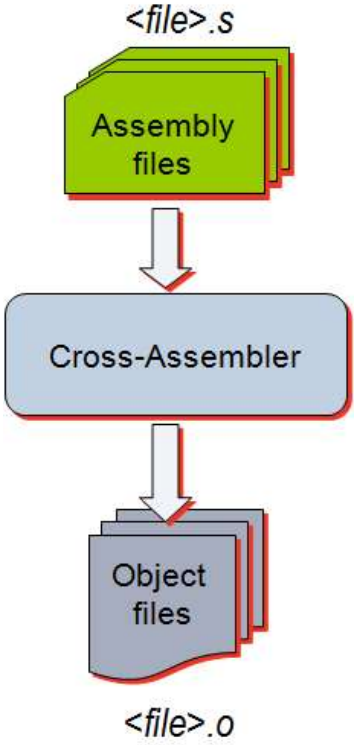
Es la interfaz para estas herramientas:

- Preprocesador (cpp0): Reemplaza las etiquetas por sus definiciones en código fuente y cabeceras
- Compilador
  - Cc1: compilador de C
  - Cc1plus: compilador de C++
- Assembler (arm-xilinx-eabi-as): compila el assembler generado a código objeto
- Enlazador (arm-xilinx-eabi-ld): resuelve las direcciones de memoria del código objeto e integra todos los archivos de código objeto en una imagen ejecutable.



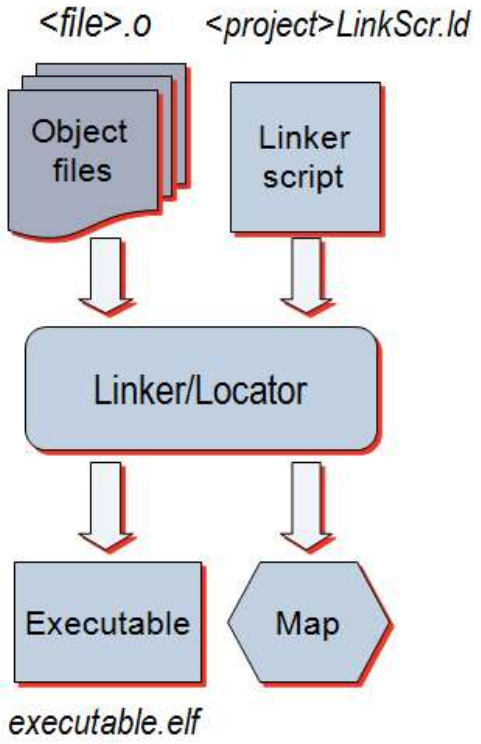
Herramientas: AS

- Es el compilador de assembler a código objeto
- El código objeto contiene:
  - Código de máquina de las instrucciones
  - Constantes
  - Referencias externas
  - Información de depuración
  - Direcciones de memoria relativas



Herramientas: LD

- Integra los distintos archivos de código objeto en una imagen ejecutable
- Utiliza un archivo *linker script* para resolver la ubicación en memoria de los distintos elementos
- Genera una imagen ejecutable en formato .ELF
- Genera un mapa de memoria con la ubicación de cada elemento del software (constantes, variables, estructuras, funciones, etc)



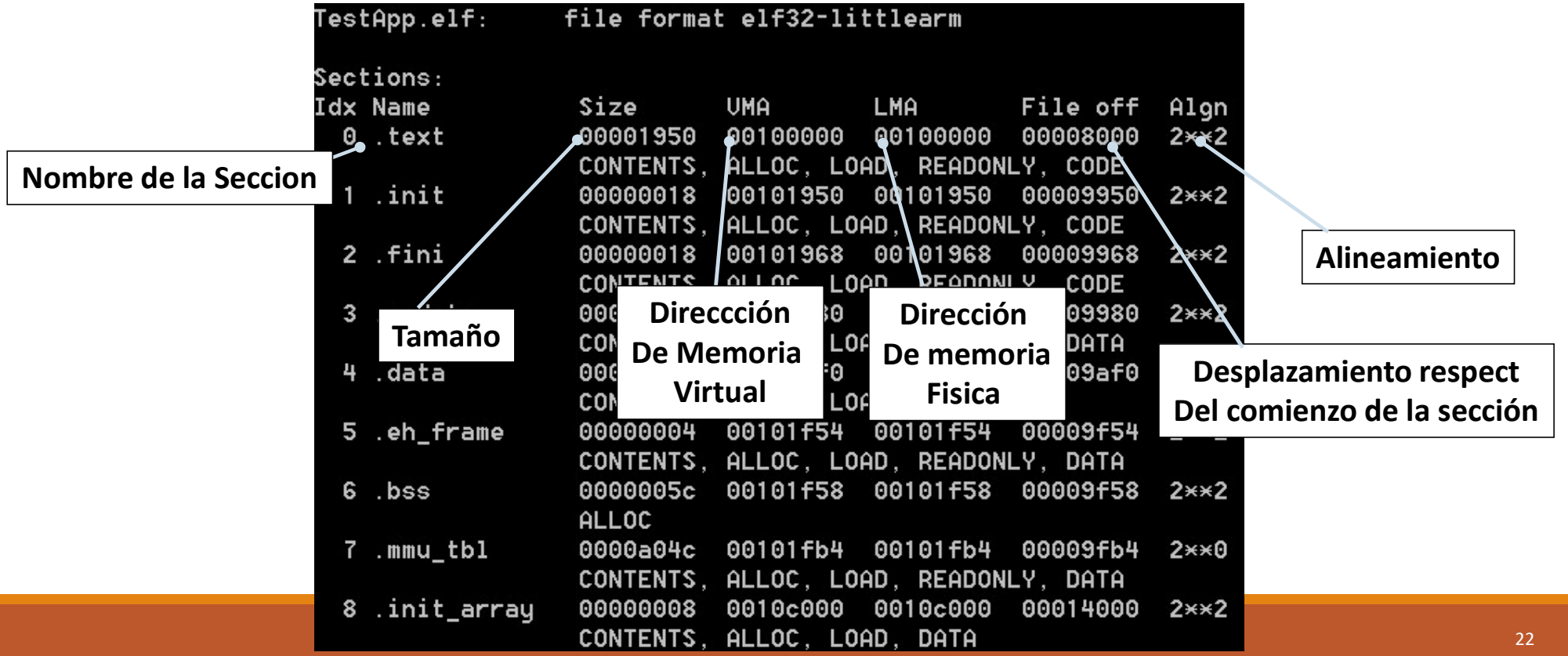
Herramientas Auxiliares

- Archiver (ar)
  - Crea, modifica y extrae código objeto de librerías
  - Integra el código objeto del proyecto BSP asociado en una librería del proyecto de aplicación

- Object Dump (objdump): Muestra información de archivos objeto e imágenes ejecutables
  - Arquitectura para la que se compilo el código
  - Ubicación en memoria de los distintos elementos
  - Código desensamblado

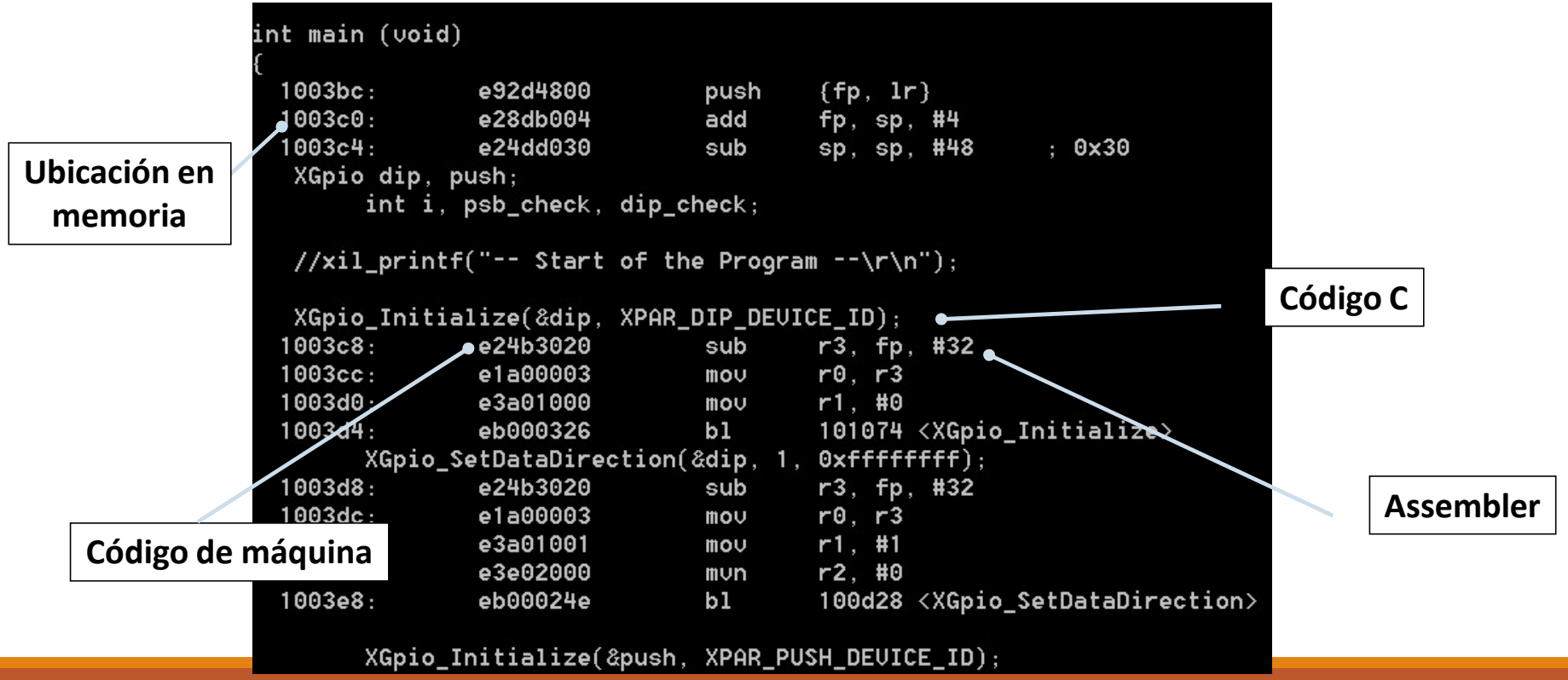
Herramienta Object Dump

Información de las distintas secciones: arm-xilinx-eabi-objdump –h <imagen ejecutable>.elf



Herramienta Object Dump

Código fuente y código desensamblado: arm-xilinx-eabi-objdump –S <imagen ejecutable>.elf



Contenido

- Introducción
- Entorno SDK
- Creación de Proyectos
- Herramientas de desarrollo GNU
- **Configuración del software de sistema**
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- Linker script

Servicios mínimos

El software de sistema (middleware) debe proveer distintos servicios para que el software de aplicación pueda ejecutarse

- Servicios standard del lenguaje C
  - Stdin y Stdout
  - Librerias (math, stdlib, conio, etc)
  - Gestion de memoria dinámica (malloc, free)

- Servicios de soporte al procesador
  - Gestión de interrupciones
  - Inicialización
  - Acceso al hardware (p.ej. Temporizador Interno)

Sistemas Operativos

- El sistema operativo es un software de sistema que permite ofrecer al software de aplicación un conjunto standard de servicios (acceso a hardware, almacenamiento permanente, interfaces de entrada y salida, etc)
- Si no se utiliza un sistema operativo (configuración standalone/bare metal), el software de sistema provee un conjunto mínimo de estos servicios a través de funciones.
- Se pueden configurar distintos sistemas operativos:
  - Linux (sistema operativo de alto nivel)
  - FreeRTOS (sistema operativo de bajo nivel con requerimientos de tiempo real)
  - XilKernel (sistema operativo básico)
- El sistema operativo forma parte del software de sistema y forma parte del Proyecto BSP

Sistemas Operativos

Los sistemas operativos facilitan la provisión de servicios al software de aplicación

- Interfaz gráfica
- Conectividad (ethernet, wifi, TCP/IP)
- Gestión de Tareas/Aplicaciones
- Gestión de recursos de hardware
- Comunicación entre aplicaciones
- Lanzamiento, ejecución y finalización de aplicaciones
- Gestión de sistemas de archivos (almacenamiento permanente)

Configuración Standalone / Bare Metal

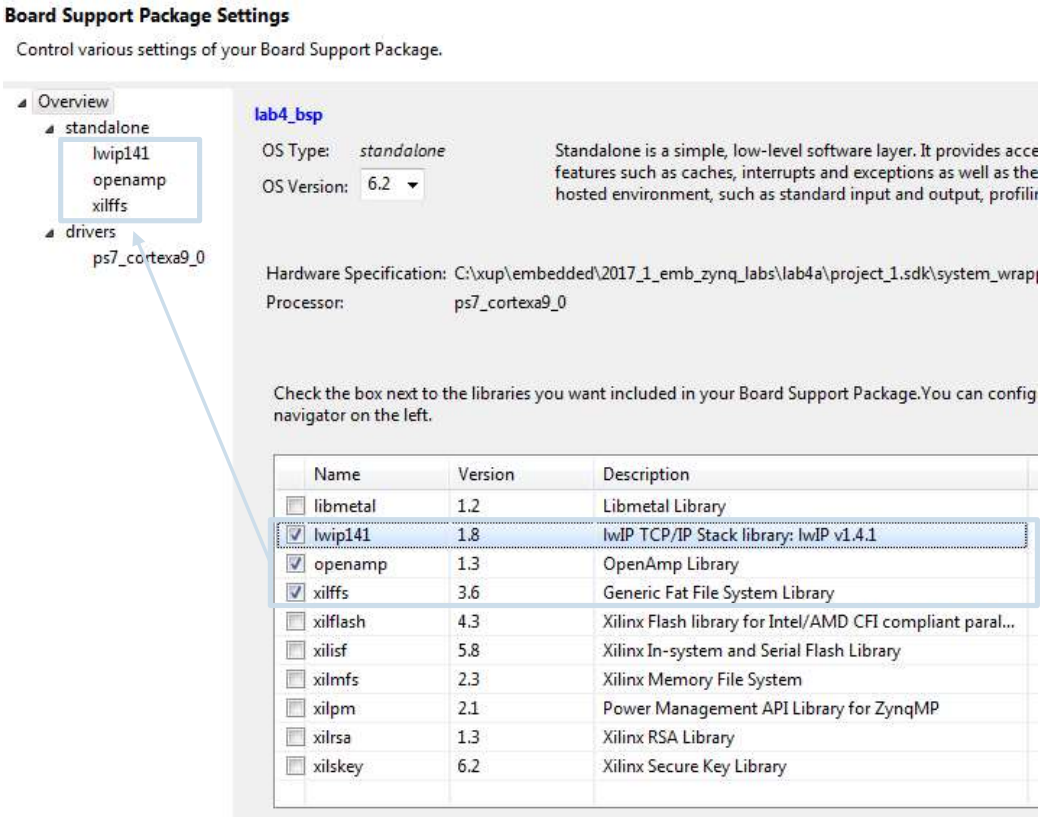
Esta configuración es recomendable cuando:

- Todos los servicios necesarios pueden incluirse en el Proyecto BSP
- La aplicación es estática (no es necesario reiniciarla)
- La imagen ejecutable puede ubicarse en BlockRAM, OnChip RAM o DDR RAM
- La aplicación consiste en una sola tarea

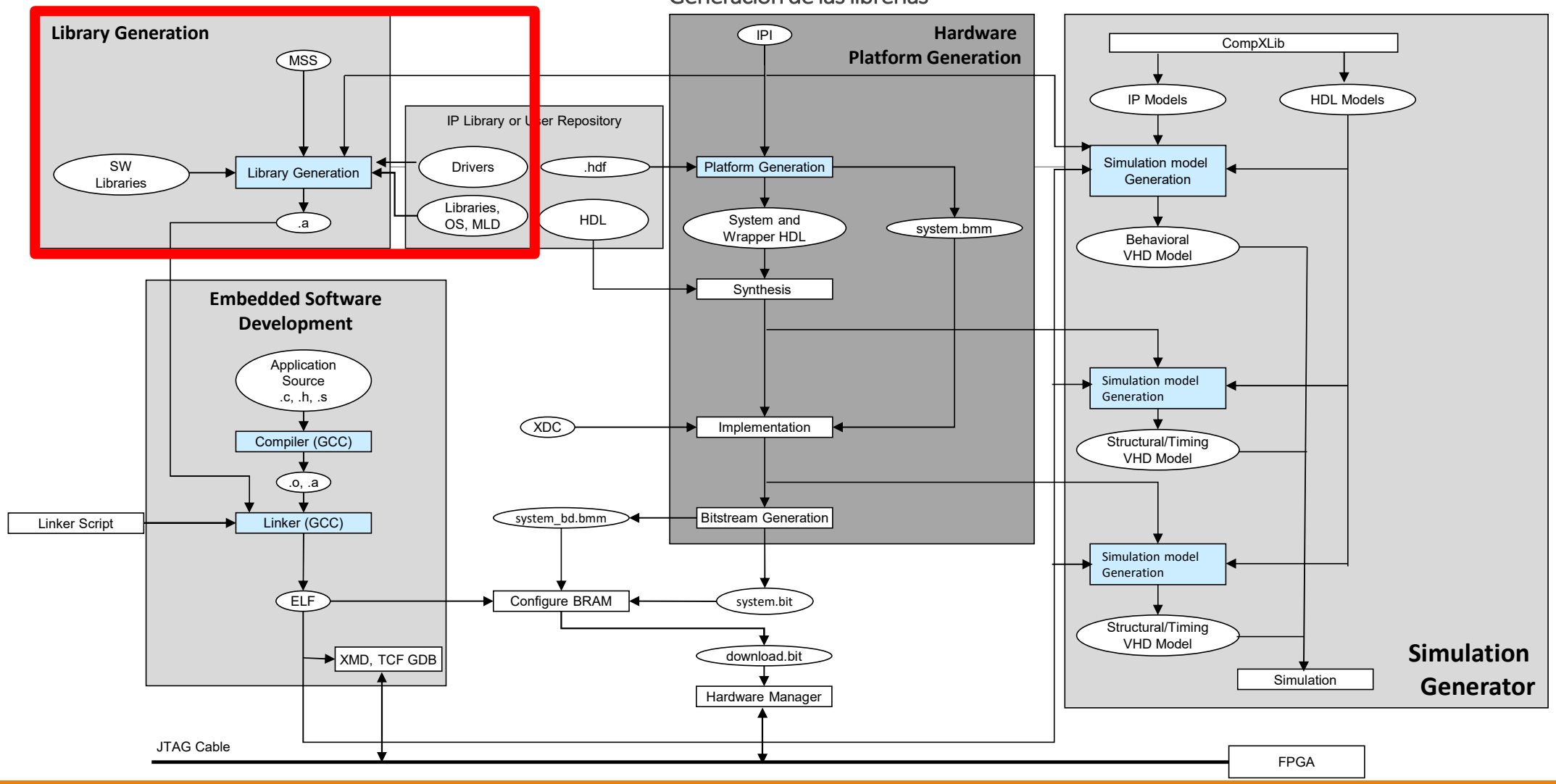
Esta configuración permite la gestión de interrupciones a través de llamadas a función

Configuración Standalone / Bare Metal

- La configuración se accede desde el menú Xilinx -> Board Support Package Settings
- En la opción “Standalone” se puede seleccionar los servicios que van a estar disponibles para la aplicación desde el software de sistema.
- Por cada servicio disponible se genera una subentrada dentro del menú “Standalone” que permite configurar el software de ese servicio.



Generación de las librerias

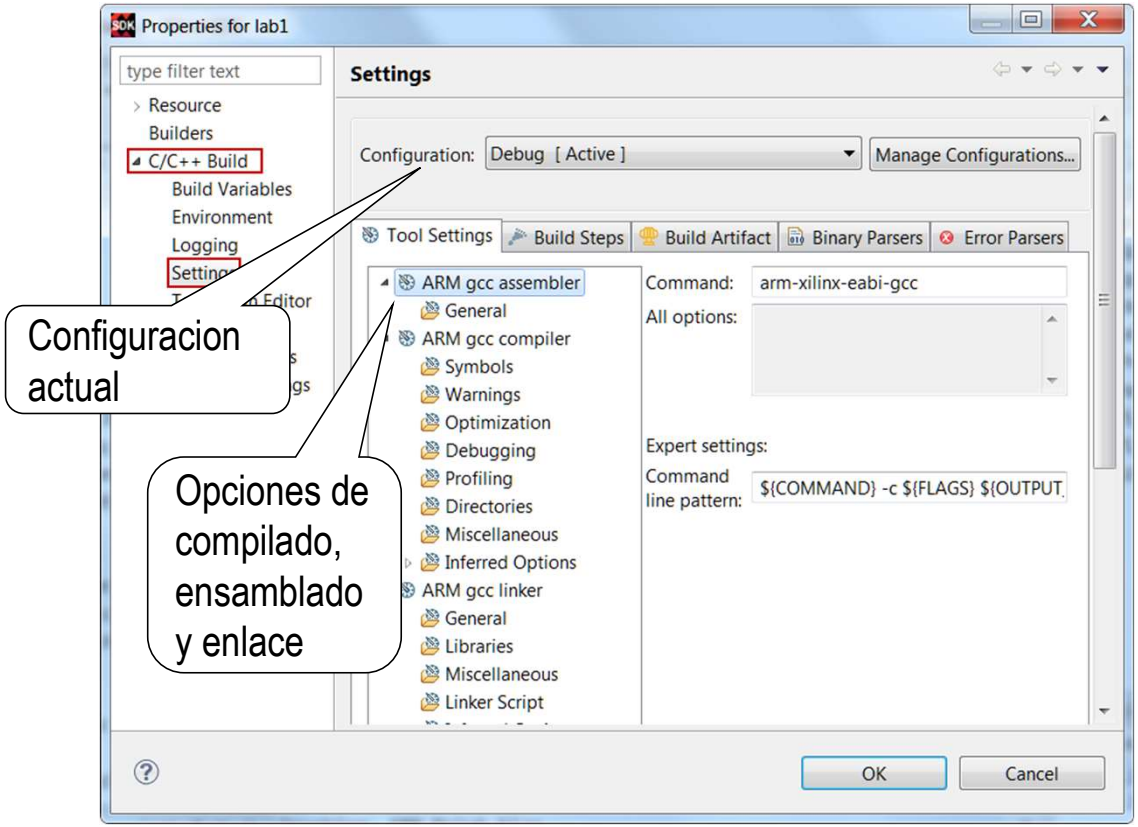


Generación de las librerias

- La herramienta libgen genera el software de sistema disponible para las aplicaciones
- Toma información del archivo .MSS, que define los controladores asociados a periféricos, los dispositivos stdin y stdout y demás características del software de sistema
- Genera las librerías que utilizará la aplicación
  - libc.a: Librería standard de C con las funciones del lenguaje
  - libXil.a: librería con los controladores de los periféricos
  - libm.a: librería para soporte de funciones matematicas

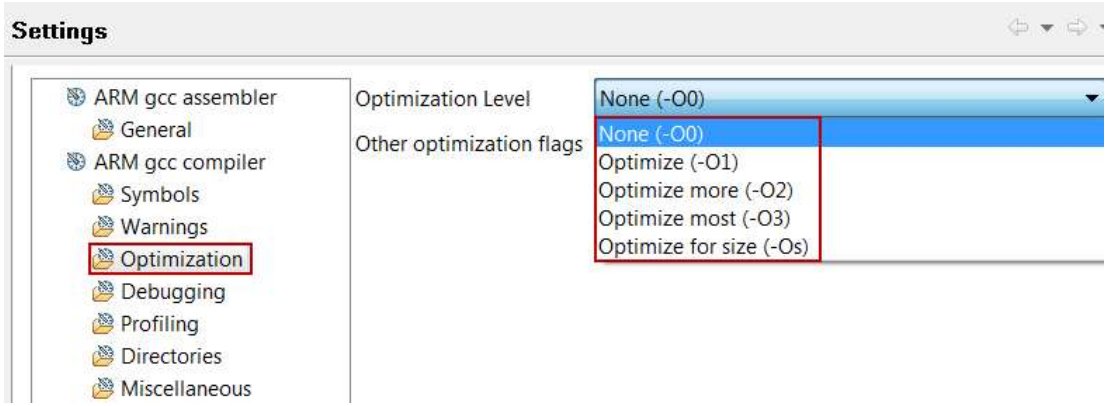
Configuración de las opciones de compilado

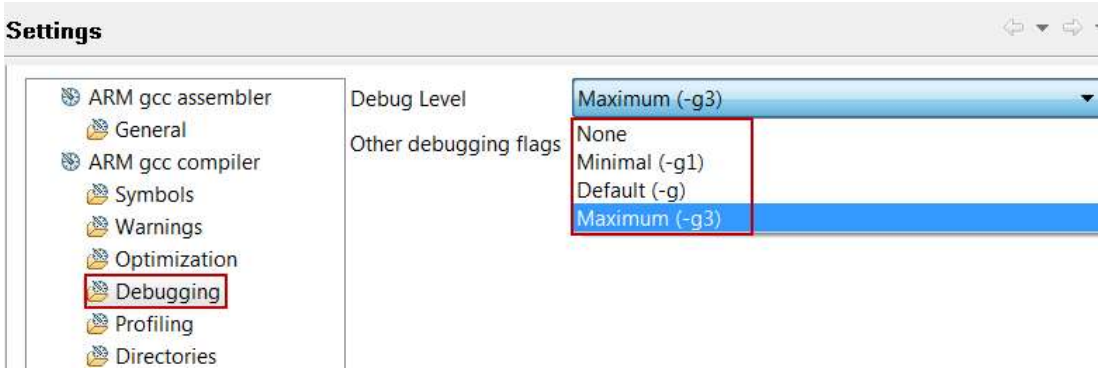
- El compilador se invoca mediante una línea de comando cuyos parámetros se pueden configurar desde las opciones del proyecto
- Hay 3 configuraciones por defecto (debug, reléase, profile). Cada una de ellas tiene un juego de parámetros específicos para cada herramienta del proceso de compilado
- Es posible generar configuraciones personalizadas



Configuración de las opciones de depuración y optimizado

- Nivel de optimización:
  - None
  - Low
  - Medium
  - High
  - Optimizado para tamaño (código mas compacto)
- Nivel de información de depuración:
  - Necesario para poder depurar, pero genera una imagen mas grande y lenta
  - Es conveniente fijar el nivel de optimización en “None” para depurar





Configuración de símbolos para compilado condicional

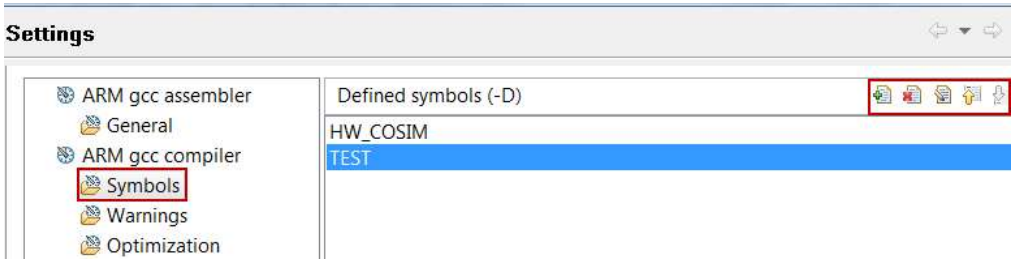
- Se puede agregar, editar o borrar símbolos
- Se utilizan de la siguiente forma:

  #ifdef <símbolo>

  (código compilado condicionalmente)

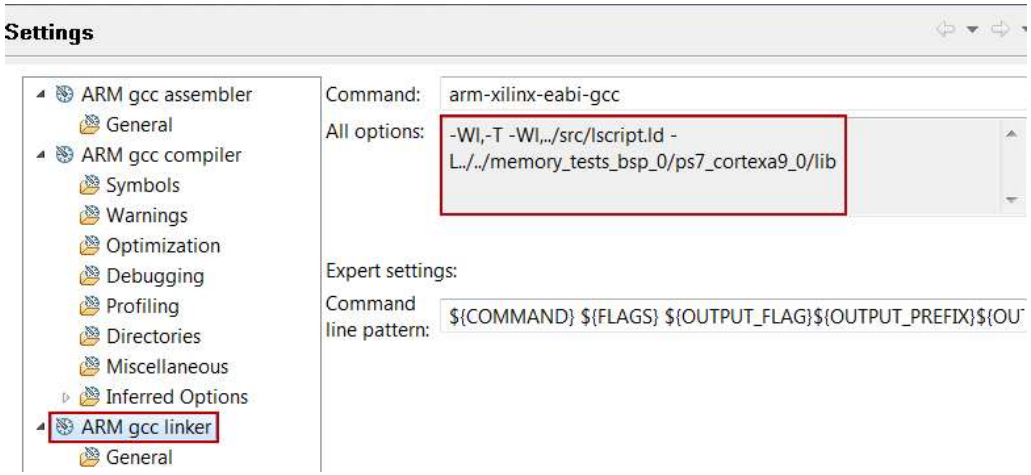
  #endif

- En la línea de comando corresponden a la opción “-D”



Configuración del Enlazado

- Se puede configurar el nombre del archivo de enlazado y las librerías con las que se enlazara el código de la aplicación.



- En general no es necesario variar esta configuración

Contenido

- Introducción
- Entorno SDK
- Creación de Proyectos
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- **Gestión de direcciones de memoria**
- Partes de un archivo objeto
- Linker script

Gestión de Direcciónes de Memoria

En el PS el procesador tiene asignados determinados espacios de memoria para funciones específicas, por lo que es necesario definir:

- Ubicación y rango de direcciones de los periféricos
- Ubicación y rango de los bloques de memoria de código (BlockRAM, Flash, DDR, etc)
- Asignación de memoria del procesador:
  - 1 GB: DDR RAM
  - 2 GB: Periféricos en PL (IP Propia)
  - 1 GB: Periféricos en PS

Start Address	Size	Description
0x0000_0000	1GB	External DDR RAM
0x4000_0000	2GB	Custom Peripherals (Programmable Logic including PCIe)
0xE000_0000	256MB	PS I/O Peripherals
0xF800_0000	32MB	Fixed Internal Peripherals (Timers, Watchdog, DMA, Interconnect)
0xFC00_0000	64MB	Flash Memory
0xFFFC_0000	256KB	On-Chip Memory

Espacio de memoria del bloque PL

- Espacio total de 2 GB
- 1 GB para cada Maestro AXI (GP0 y GP1)
  - Según el GPx al que este conectada la IP Propia es el rango de direcciones disponible.
- Espacio accesible por los periféricos AXI Maestro
  - Procesadores ARM Cortex A9
  - Controlador DMA
  - Periférico ethernet
  - Periférico USB
  - Periférico SD/SDIO

Start Address	Description
0x4000_0000	Accelerator #1 (Video Scaler)
0x6000_0000	Accelerator #2 (Video Object Identification)
0x8000_0000	Peripheral #1 (Display Controller)

```
int main() {

int *data = 0x1000_0000;
int *accel1 = 0x4000_0000;

// Pure SW processing
Process_data_sw(data);

// HW Accelerator-based processing
Send_data_to_accel(data, accel1);
process_data_hw(accel1);
Recv_data_from_accel(data, accel1);
}
```

Espacio de memoria de los periféricos PS

Register Base Address	Description
E000_0000, E000_1000	UART Controllers 0, 1
E000_2000, E000_3000	USB Controllers 0, 1
E000_4000, E000_5000	I2C Controllers 0, 1
E000_6000, E000_7000	SPI Controllers 0, 1
E000_8000, E000_9000	CAN Controllers 0, 1
E000_A000	GPIO Controller
E000_B000, E000_C000	Ethernet Controllers 0, 1
E000_D000	Quad-SPI Controller
E000_E000	Static Memory Controller (SMC)
E010_0000, E010_1000	SDIO Controllers 0, 1
E020_0000	IOP Bus Configuration

Espacio de memoria de los registros de control de sistema (SLCR)

Register Base Address	Description
F800_0000	SLCR write protection lock and security
F800_0100	Clock control and status
F800_0200	Reset control and status
F800_0300	APU control
F800_0400	TrustZone control
F800_0500	CoreSight SoC debug control
F800_0600	DDR DRAM controller
F800_0700	MIO pin configuration
F800_0800	MIO parallel access
F800_0900	Miscellaneous control
F800_0A00	On-chip memory (OCM) control
F800_0B00	I/O buffers for MIO pins (GPIOB) and DDR pins (DDRIOB)

Espacio de memoria de los registros del bloque PS

Register Base Address	Description
F800_1000, F800_2000	Triple timer counter 0, 1
F800_3000	DMAC when secure
F800_4000	DMAC when non-secure
F800_5000	System watchdog timer (SWDT)
F800_6000	DDR DRAM controller
F800_7000	Device configuration interface (DevC)
F800_8000	AXI_HP 0 high performance AXI interface w/ FIFO
F800_9000	AXI_HP 1 high performance AXI interface w/ FIFO
F800_A000	AXI_HP 2 high performance AXI interface w/ FIFO
F800_B000	AXI_HP 3 high performance AXI interface w/ FIFO
F800_C000	On-chip memory (OCM)
F800_D000	Reserved
F880_0000	CoreSight debug control

Espacio de memoria de los registros internos de los CPU

Register Base Address	Description
F890_0000 to F89F_FFFF	Top-level interconnect configuration and Global Programmers View (GPV)
F8F0_0000 to F8F0_00FC	SCU control and status
F8F0_0100 to F8F0_01FF	Interrupt controller CPU
F8F0_0200 to F8F0_02FF	Global timer
F8F0_0600 to F8F0_06FF	Private timers and private watchdog timers
F8F0_1000 to F8F0_1FFF	Interrupt controller distributor
F8F0_2000 to F8F0_2FFF	L2-cache controller

Contenido

- Introducción
- Entorno SDK
- Creación de Proyectos
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- Gestión de direcciones de memoria
- **Partes de un archivo objeto**
- Linker script

Archivo objeto

Un archivo objeto es código ensamblado, es decir código de maquina

li r31,0x00 → 0x3BE0 0000

Además puede contener:

- Constantes
- Referencias a elementos que están definidos en otros archivos objeto
- Información de depuración

Secciones del archivo objeto

.text	Text section (codigo)
.rodata	Read-only data section (constantes)
.data	Read-write data section (variables inicializadas)
.bss	Uninitialized data section (variables no inicializadas)

```
int ram_data[10] = {0,1,2,3,4,5,6,7,8,9};        /* DATA */
const int rom_data[10] = {9,8,7,6,5,4,3,2,1};    /* RODATA */
int I;   /* BSS */
main(){
...
I = I + 10;  /* TEXT */
...
}
```

Contenido

- Introducción
- Entorno SDK
- Creación de Proyectos
- Herramientas de desarrollo GNU
- Configuración del software de sistema
- Gestión de direcciones de memoria
- Partes de un archivo objeto
- **Linker script**

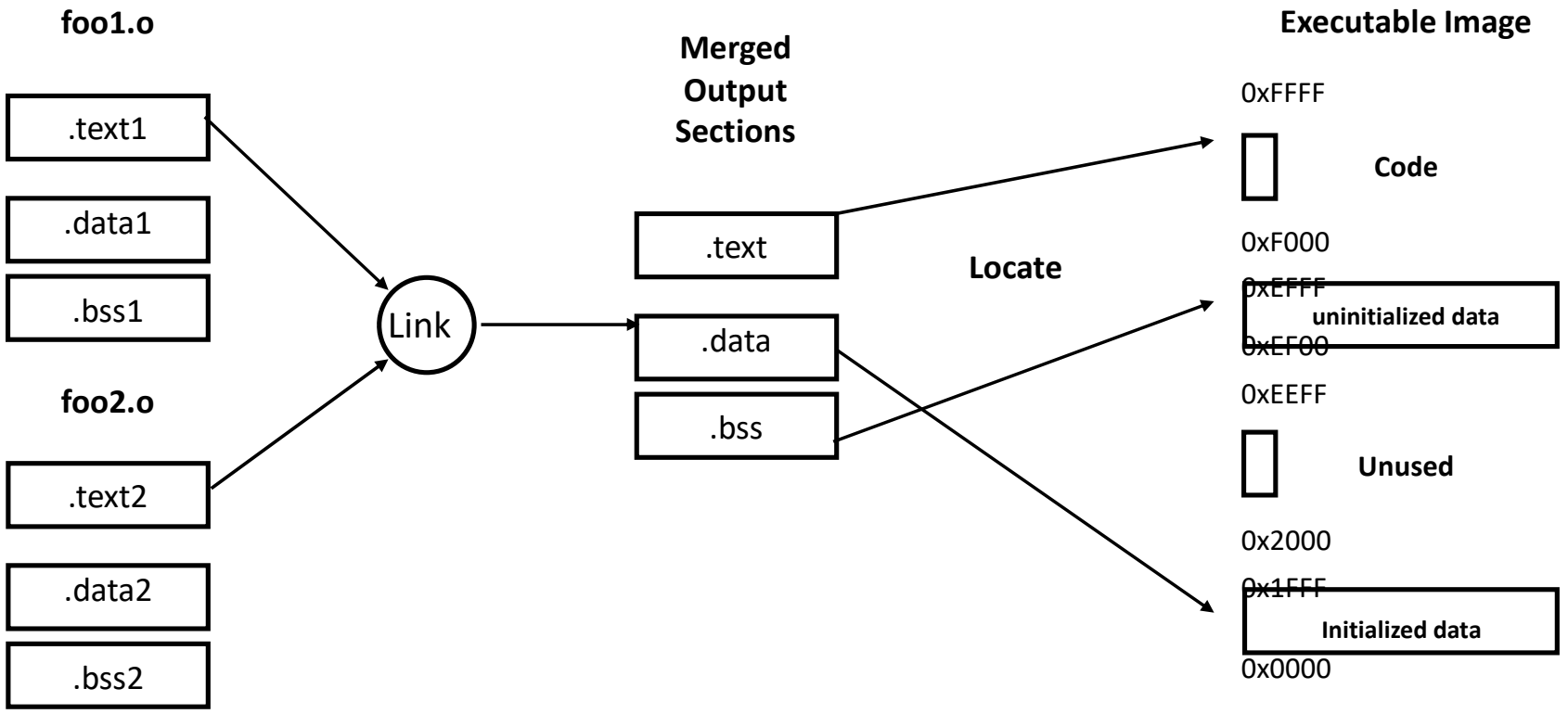
Linker script

El archivo linker script (lscript.ld) controla el proceso de enlazado de todos los archivos objeto

- Establece la ubicación dentro del mapa de memoria del código y los datos
- Define el punto de entrada de la imagen ejecutable
- Reserva espacio para el stack.

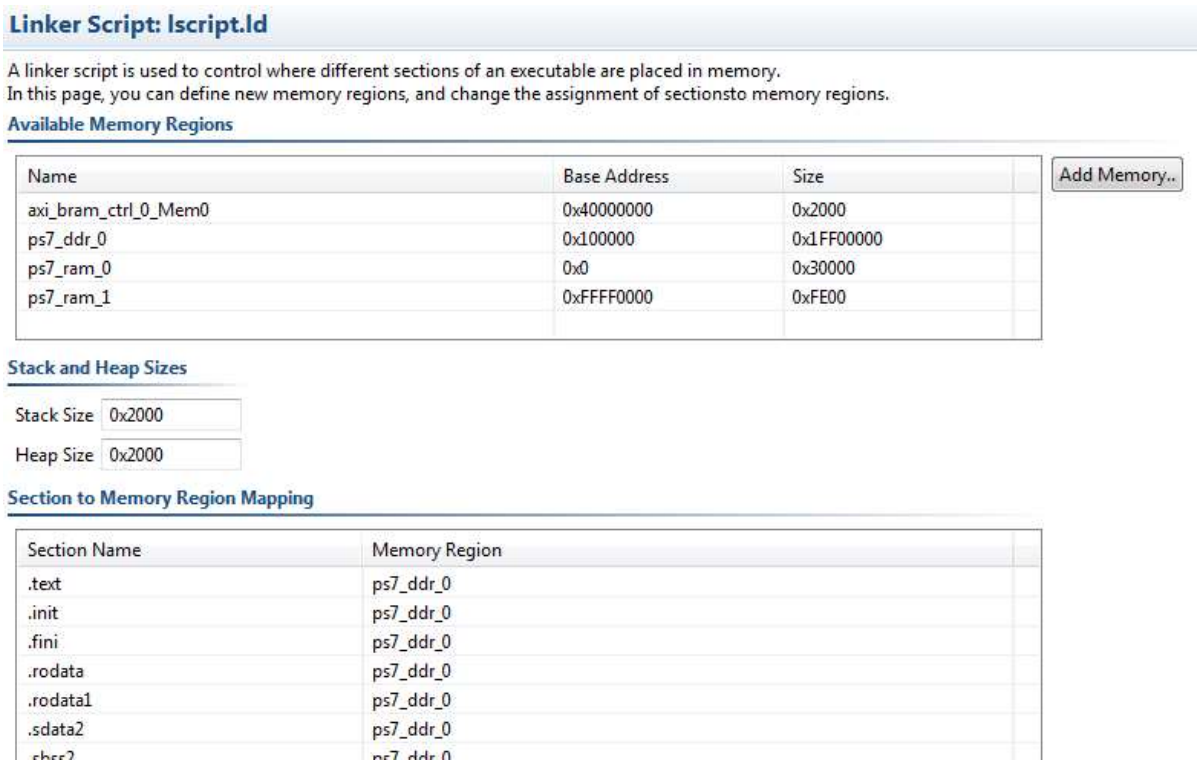
Es necesario si el mapa de memoria no es continuo

Enlazado


foo1.o
.text1
.data1
.bss1
foo2.o
.text2
.data2
.bss2
Link
Merged
Output
Sections
.text
.data
.bss
Locate
Executable Image
0xFFFF
Code
0xF000
0xEFFF
uninitialized data
0xEF00
0xEEFF
Unused
0x2000
0x1FFF
Initialized data
0x0000

Generación del linker script

- Se configura mediante una interfaz gráfica en formato de tabla
- Se inicia desde el menú Xilinx -> Generate Linker Script



Practica 4

- En esta Práctica se utiliza el hardware creado en la práctica anterior.
- Se creará una aplicación de software que utilizará un controlador (driver) para comunicarse con la IP Propia.
- Se modificará el linker script para asignar distintas zonas de memoria a las secciones de la aplicación.