Software Development

Objectives

After completing this module, you will be able to:

- Identify the functionality included in the GNU tools: GCC, AS, LD, GDB
- Understand the basic concepts of the Eclipse IDE
- List Xilinx Software Development Kit (SDK) features
- Examine the IP driver's functionality and design
- Examine the Xilinx Libraries
- Determine what a BSP is and what is included

Outline

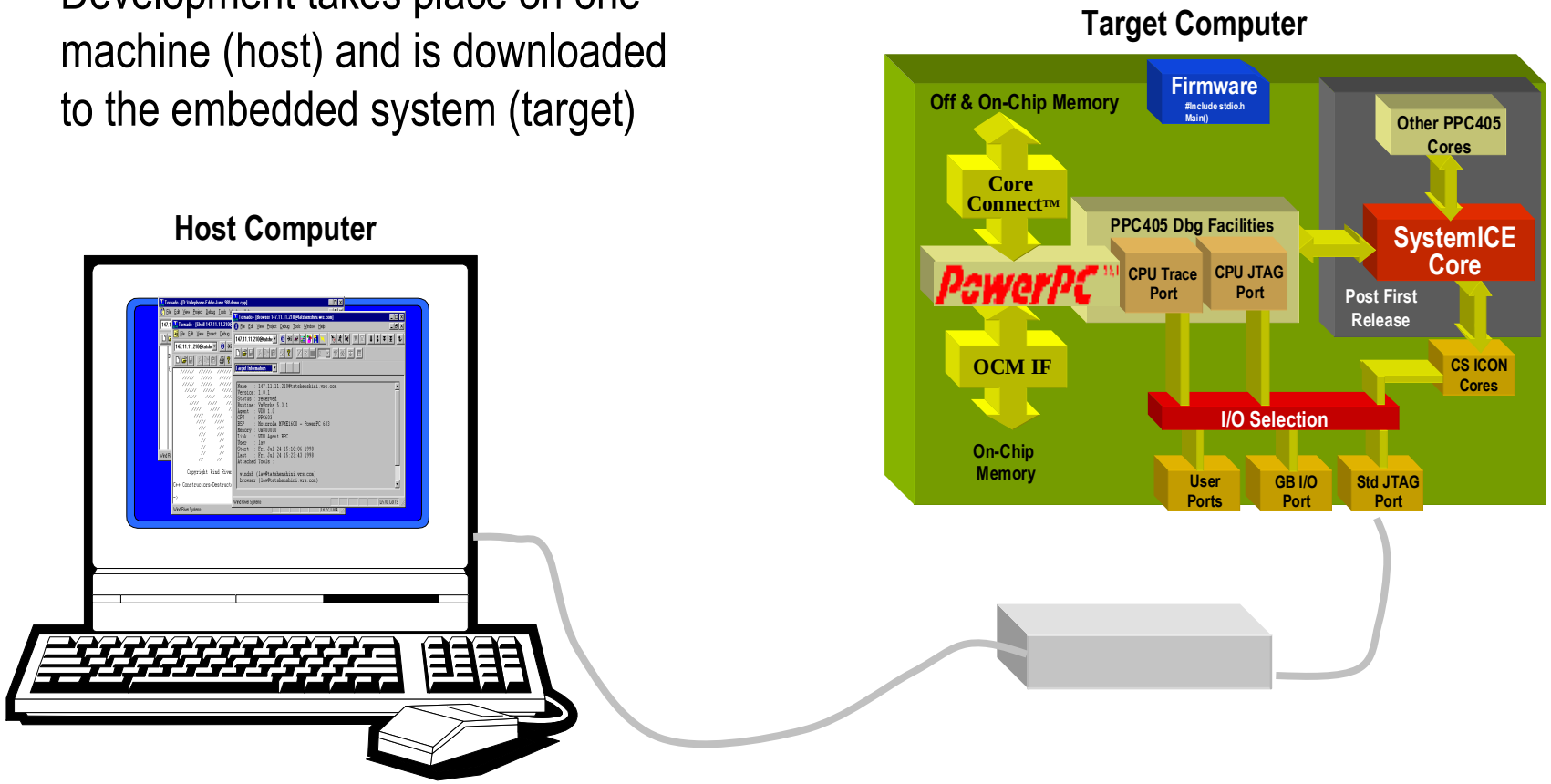
- **Introduction**
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- Board Support Packages
  - Boot Files and Sequence

Desktop versus Embedded

- Desktop development: written, debugged, and run on the same machine
- OS loads the program into the memory when the program has been requested to run
- Address resolution takes place at the time of loading by a program called the loader
  - The loader is included in the OS
- The programmer glues into one executable file called ELF
  - Boot code, application code, RTOS, and ISRs
  - Address resolution takes place during the *gluing* stage
- The executable file is downloaded into the target system through different methods
  - Ethernet, serial, JTAG, BDM, ROM programmer

Embedded versus Desktop

- Development takes place on one machine (host) and is downloaded to the embedded system (target)



A cross-compiler is run on the host

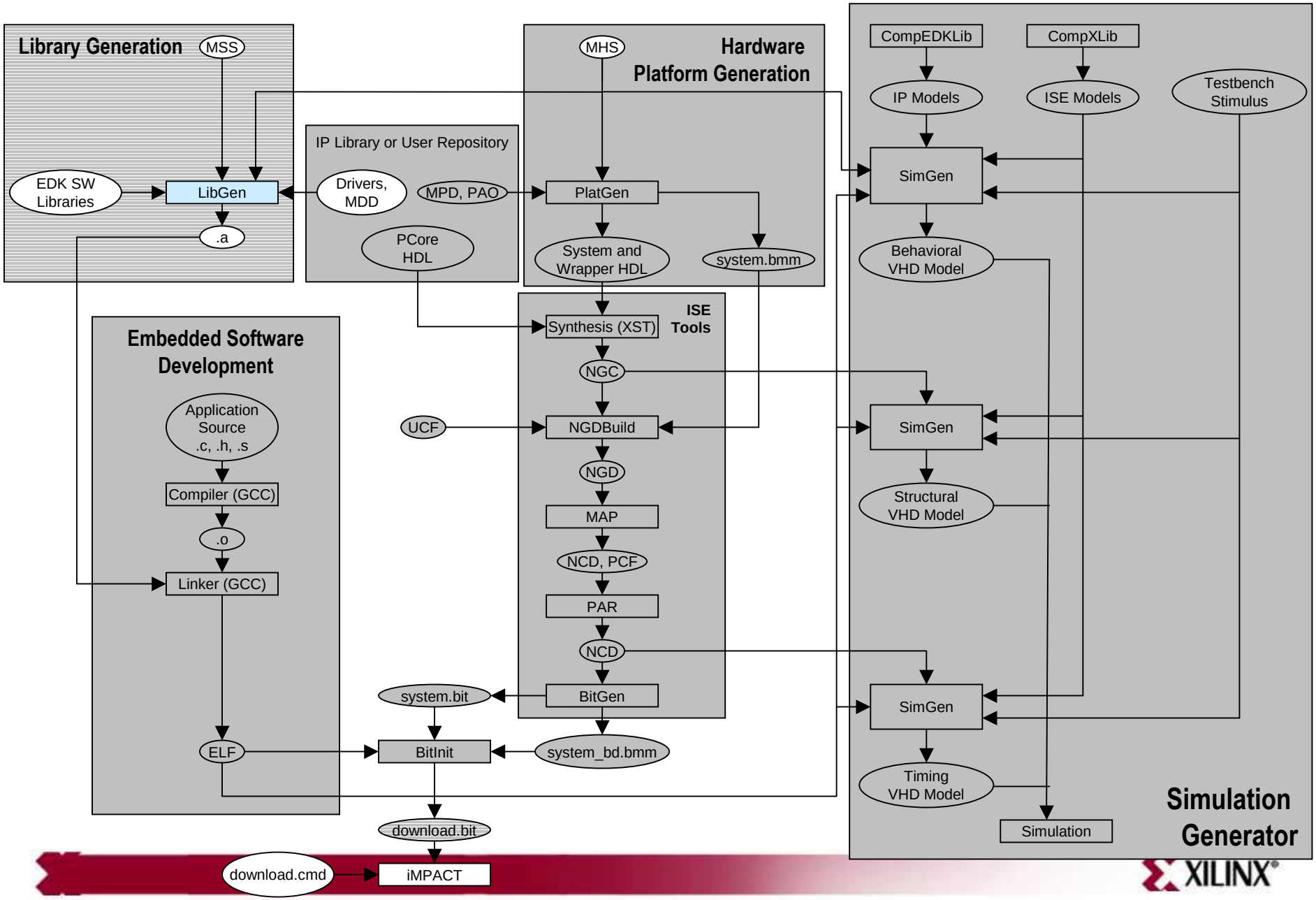
Embedded Development

- Different set of problems
  - Unique hardware for every design
  - Reliability
  - Real-time response requirement (sometimes)
    - RTOS versus OS
  - Code compactness
  - High-level languages and assembly

Outline

- Introduction
- **Software Settings**
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Library Generation (LibGen)



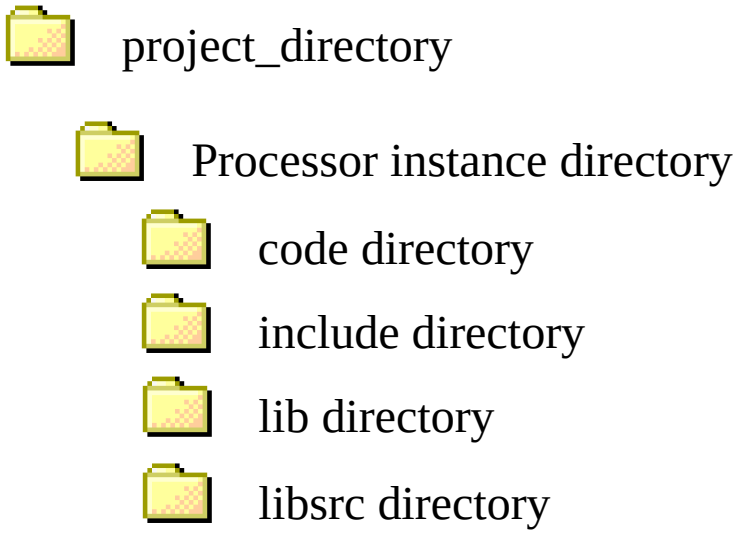
Software Design
Environment

- The Library Generator (LibGen) utility generates the necessary libraries and drivers for the embedded system
- LibGen takes an MSS (Microprocessor Software Specification) file created by the user as input. The MSS file defines the drivers associated with peripherals, standard input/output devices, interrupt handler routines, and other related software features
- The MSS file is generated by XPS by using the software settings specified

LibGen

Configures libraries and device drivers

LibGen Generated Directories

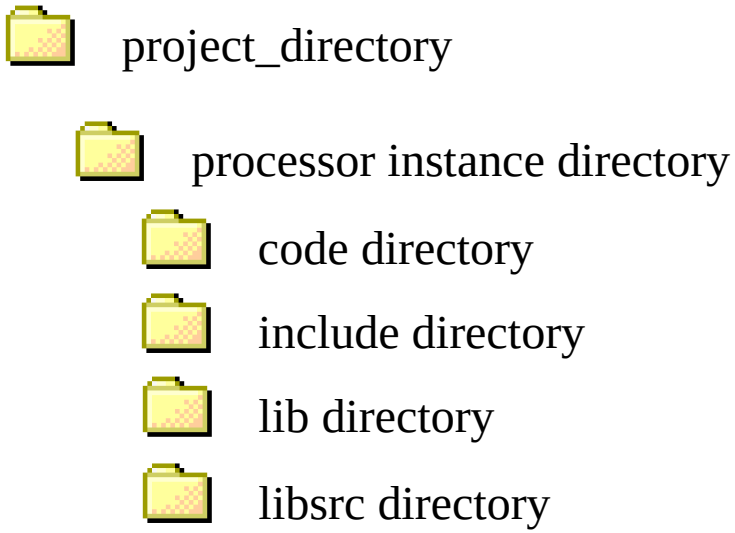


Note: The number of processor instance directories generated is related to the number of processor instances present in the system

- code directory
  - A repository for EDK executables
  - Creates xmdstub.elf for MB here
- include directory
  - C header files that are required by drivers
  - xparameters.h
    - Defines base and high addresses of the peripherals in the system
    - Defines the peripheral IDs required by the drivers and user programs
    - Defines the function prototypes

LibGen

LibGen Generated Directories



Note: The processor instance directories content is overwritten every time LibGen is run

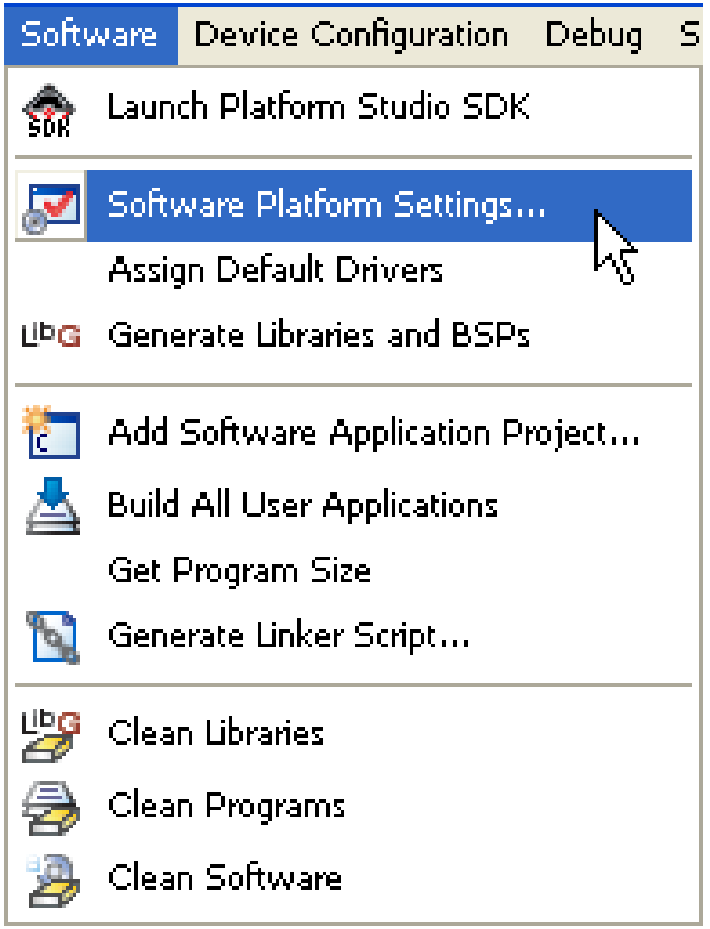
- lib directory
  - **libc.a**, **libm.a** and **libxil.a** libraries
    - The libxil library contains driver functions that the particular processor can access
- libsrc directory
  - Intermediate files and makefiles that compile the libraries and drivers
  - Peripheral-specific driver files that are copied from the EDK and user driver directories

Outline

- Introduction
- Software Settings
  - **Software Platform Settings**
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

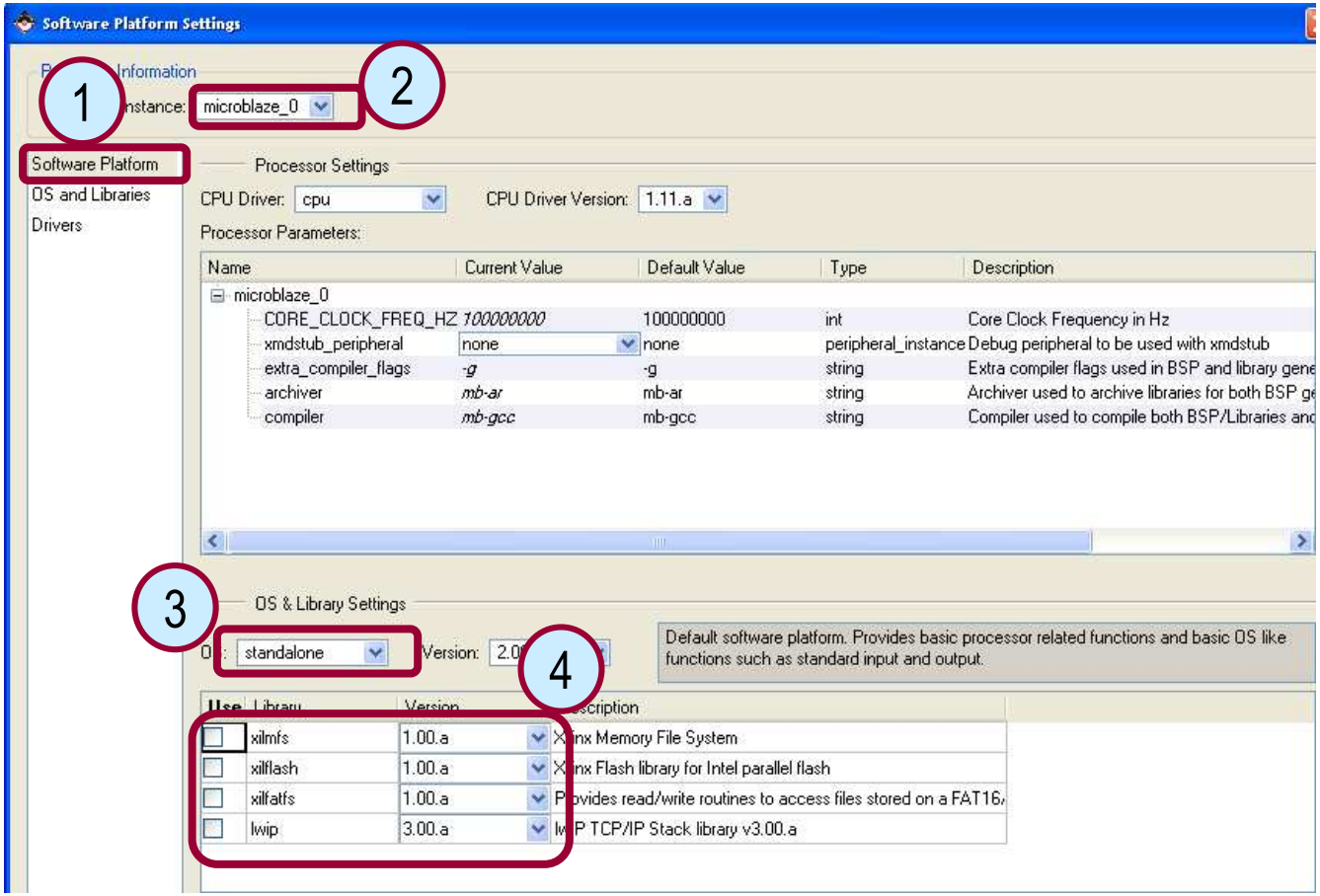
Software Platform Settings

- Software settings can be assigned to individual processor instance by selecting **Software** → **Software Platform Settings** or clicking button on the toolbar
- In case of multiple processors in the design software platform settings allow you to select each processor instance and set parameters



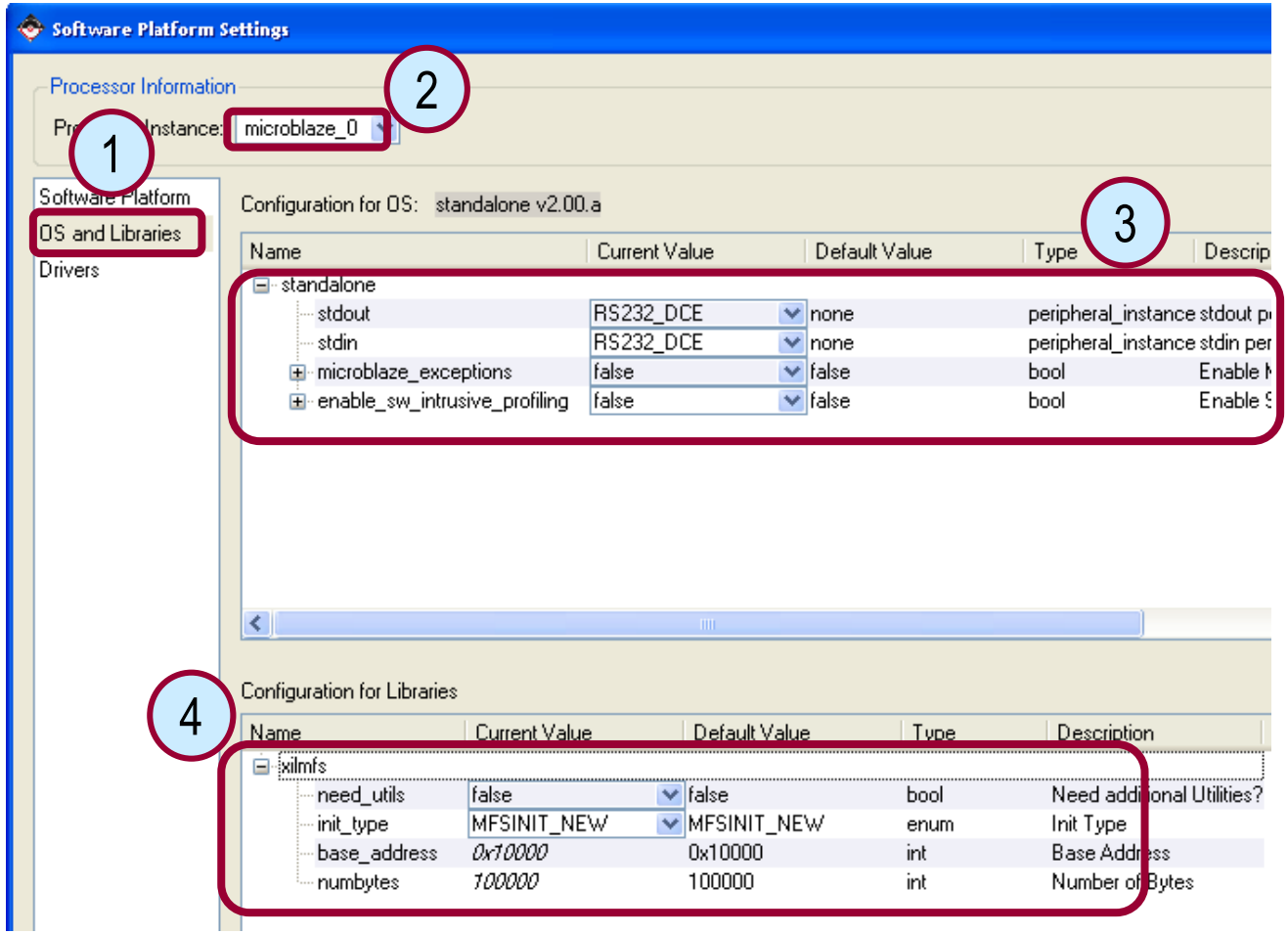
Software Platform Settings (1)

1. Select Software Platform panel
2. Select processor instance
3. Select OS
4. Check desired libraries and their version



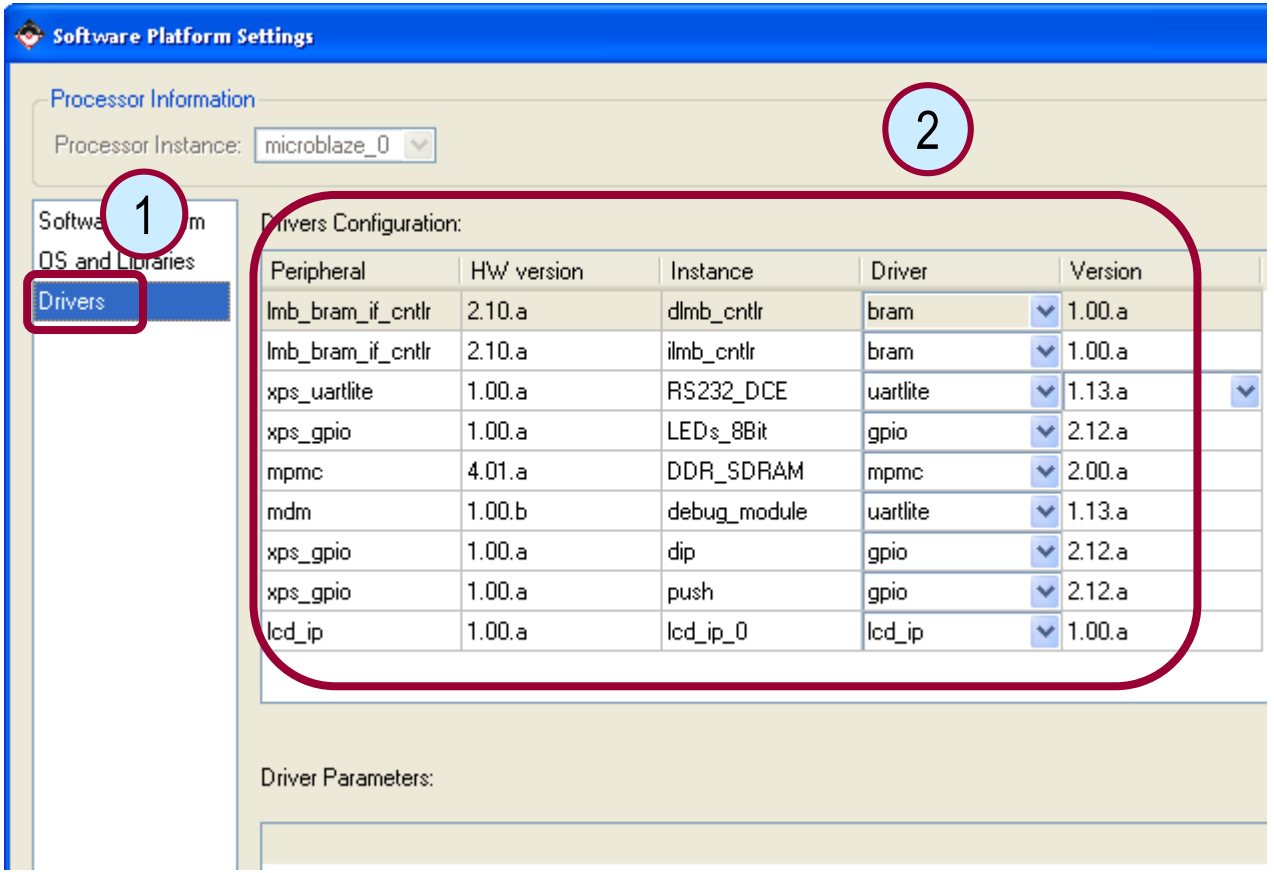
Software Platform Settings(2)

1. Select OS and Libraries panel
2. Select processor instance
3. Set stdin and stdout devices as well as assign fpu, malloc, and profiling related parameters
4. Configure selected libraries parameters



Software Platform Settings(3)

1. Select Drivers panel
2. Select drivers and version for each device in the design

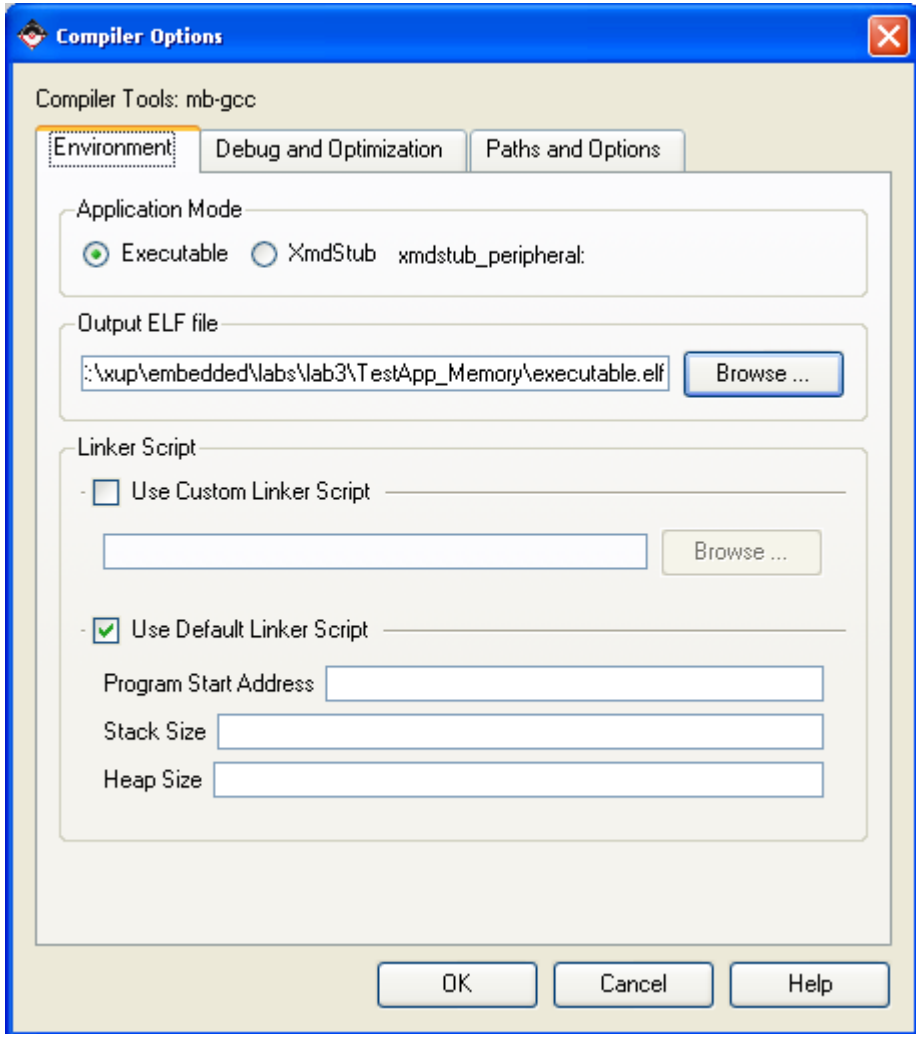


Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - **Compiler Settings**
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

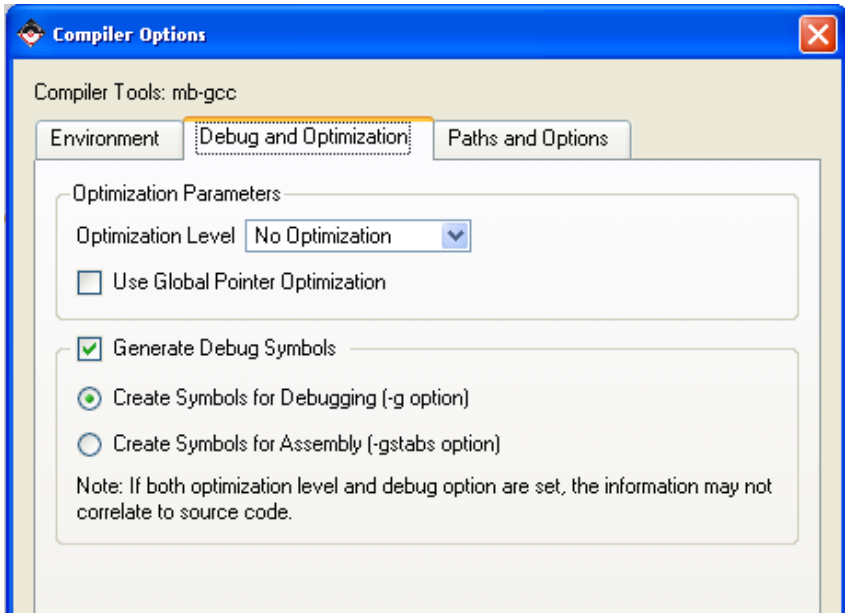
Compiler Settings

- Compiler settings can be assigned by double-clicking Compiler Options entry under an application in the Application tab
- Environment tab
  - Application Mode
    - Executable
    - XmdStub (MicroBlaze™ processor only)
  - Use Custom Linker Script
    - If checked then provide the path to the linker script
  - Use default Linker Script
    - Program Start Address
    - Stack Size
    - Heap Size



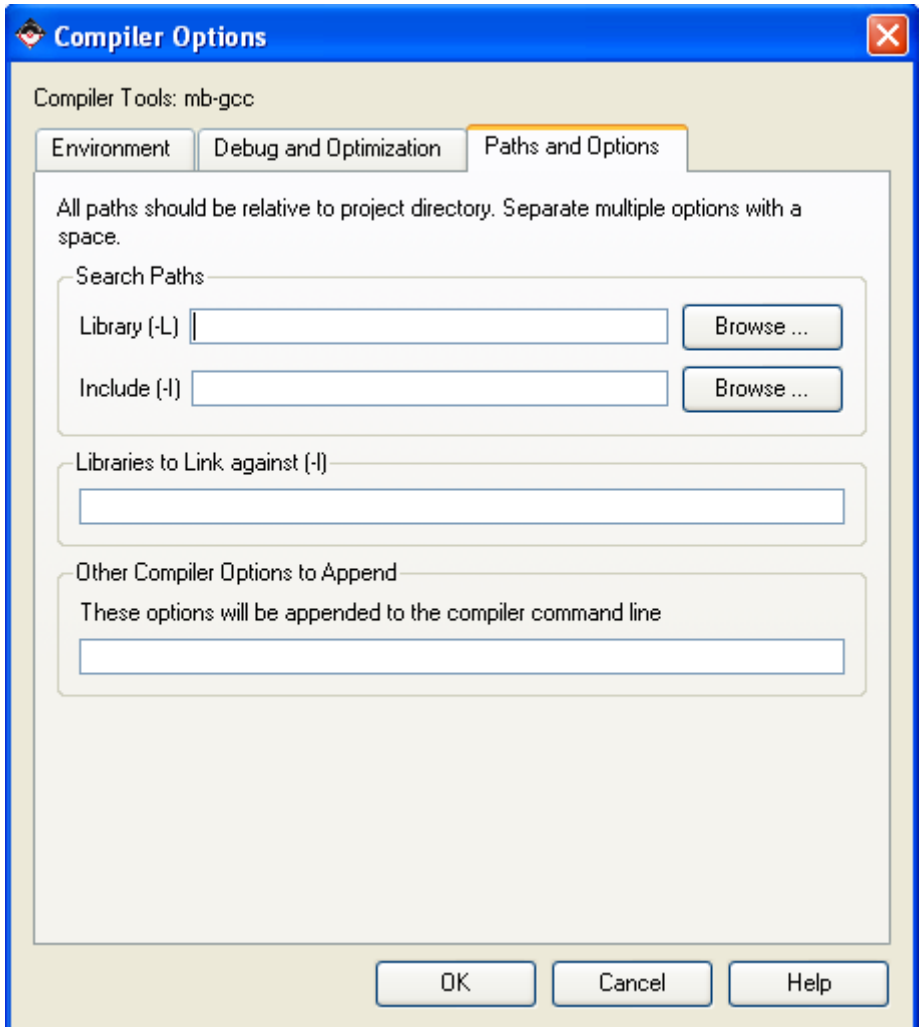
Compiler Settings

- Debug and Optimization tab
  - Optimization Parameters
    - Optimization Level: 0 to 3
      - While debugging your code, level 0 (no optimization) is recommended
      - Levels 1 and above will cause code rearrangement
    - Use Global Pointer Optimization
      - Compiler stores all variables up to 8 bytes of size in a special area that can be accessed by registers r2/r13 (called small data anchors) and a 16-bit offset
  - Generate Debug Symbols
    - Checking this option allows the generation of debugging information based on the option selected
    - Create symbol for debugging (-g option)
    - Create symbols for assembly (-gstabs option)



Compiler Settings

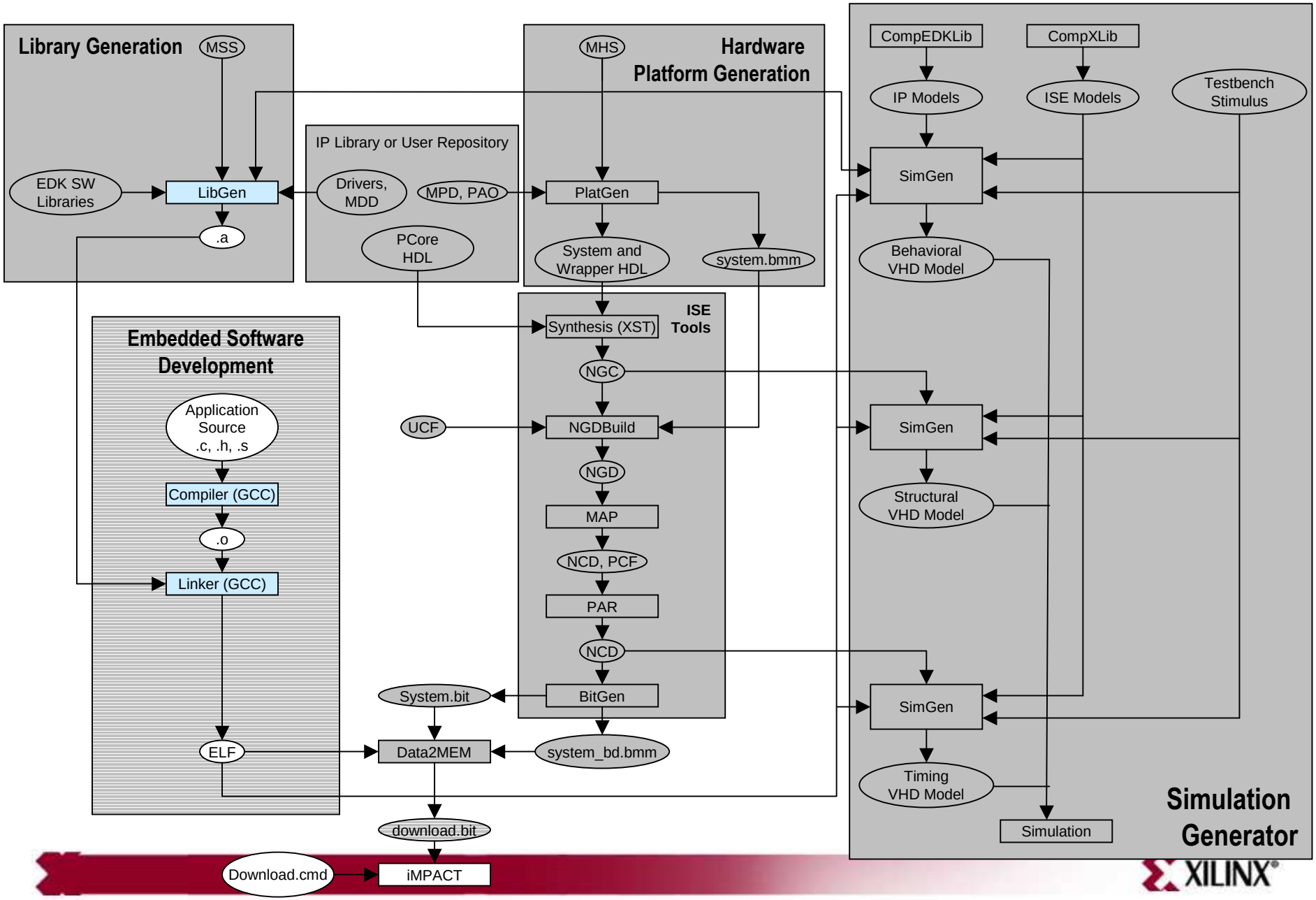
- Paths and Options tab
  - Search Paths
    - Library (-L)
    - Include (-I)
  - Libraries to Link against
    - List user libraries to be used
  - Other compiler options to append
    - For example: -g
    - See GNU docs for more options



Outline

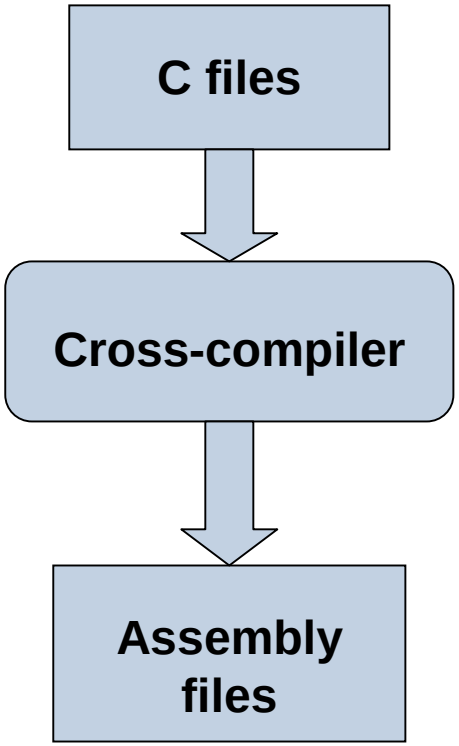
- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- **GNU Development Tools: GCC, AS, LD, Binutils**
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Software Development with GNU Tools



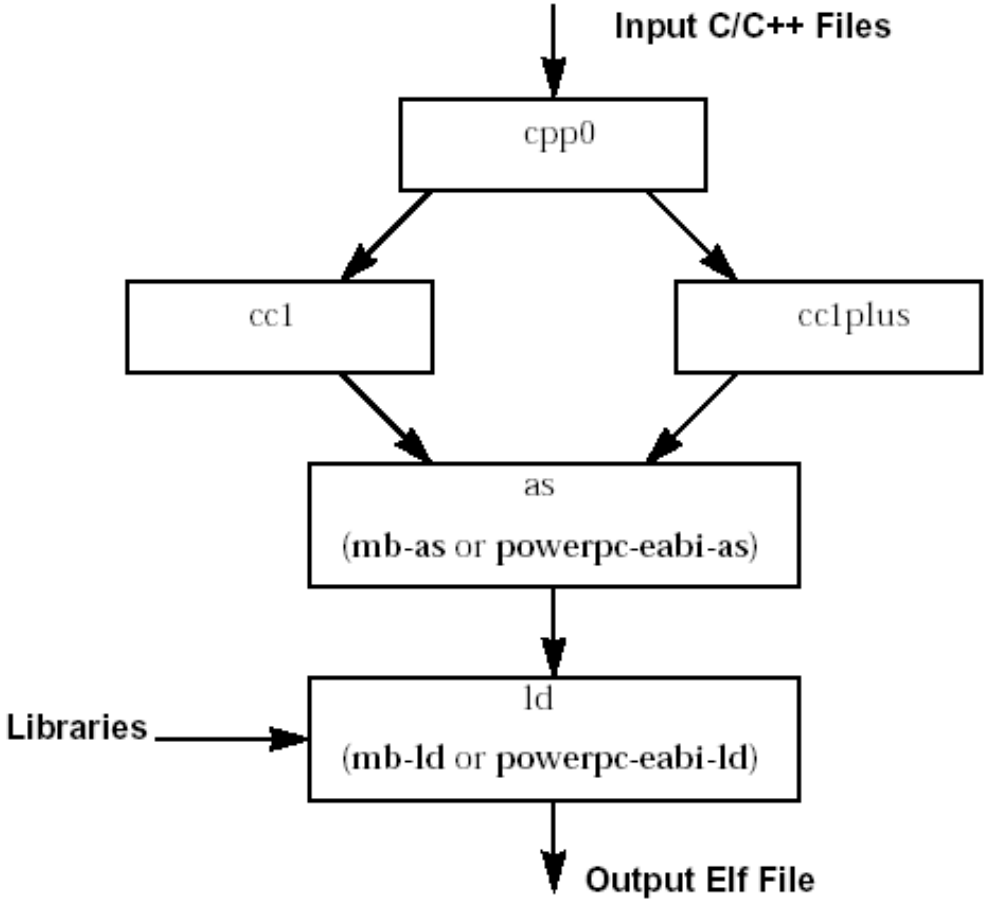
GNU Tools: GCC

- GCC translates C source code into assembly language
- GCC also functions as the user interface to the GNU assembler and to the GNU linker, calling the assembler and the linker with the appropriate parameters
- Supported cross-compilers:
  - GNU GCC (mb-gcc)
- Command line only; uses the settings set through the GUI



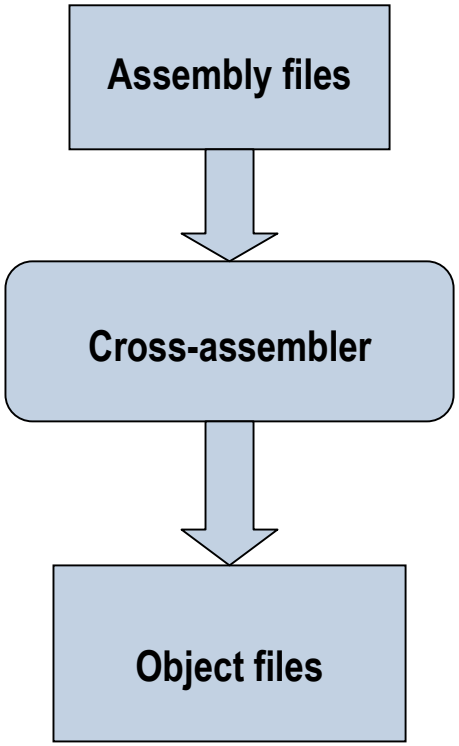
GNU Tools

- Calls four different executable
  - Preprocessor (cpp0)
    - Replaces all macros with definitions defined in the sourc and header files
  - Language specific c-compiler
    - cc1 C-programming language
    - cc1plus C++ language
  - Assembler
    - mb-as
  - Linker and loader
    - mb-ld



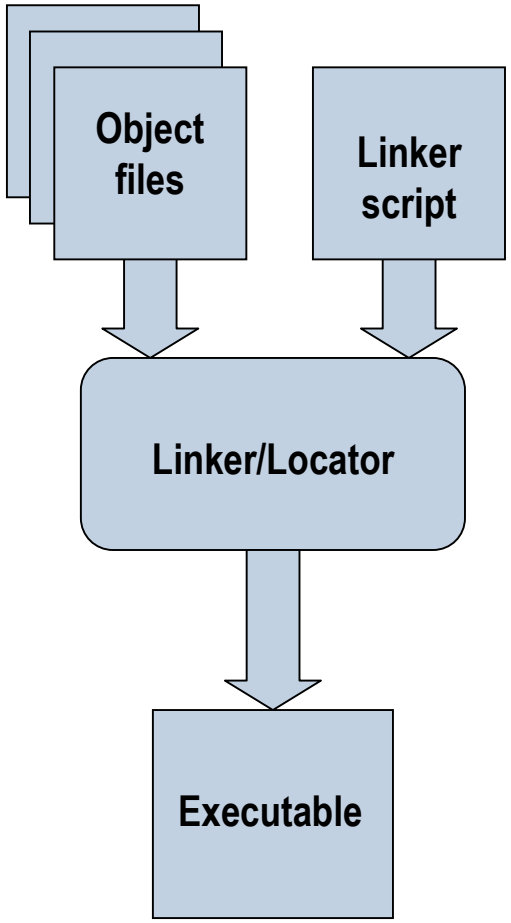
GNU Tools: AS

- Input: Assembly language files
  - File extension: **.s**
- Output: Object code
  - File extension: **.o**
  - Contains
    - Assembled piece of code
    - Constant data
    - External references
    - Debugging information
- Typically, the compiler automatically calls the assembler
- Use the -Wa switch if the source files are assembly only and use gcc



GNU Tools: LD

- Linker
- Inputs:
  - Several object files
  - Archived object files (library)
  - Linker script (mapfile)
- Output:
  - Executable image (.ELF)



GNU Utilities

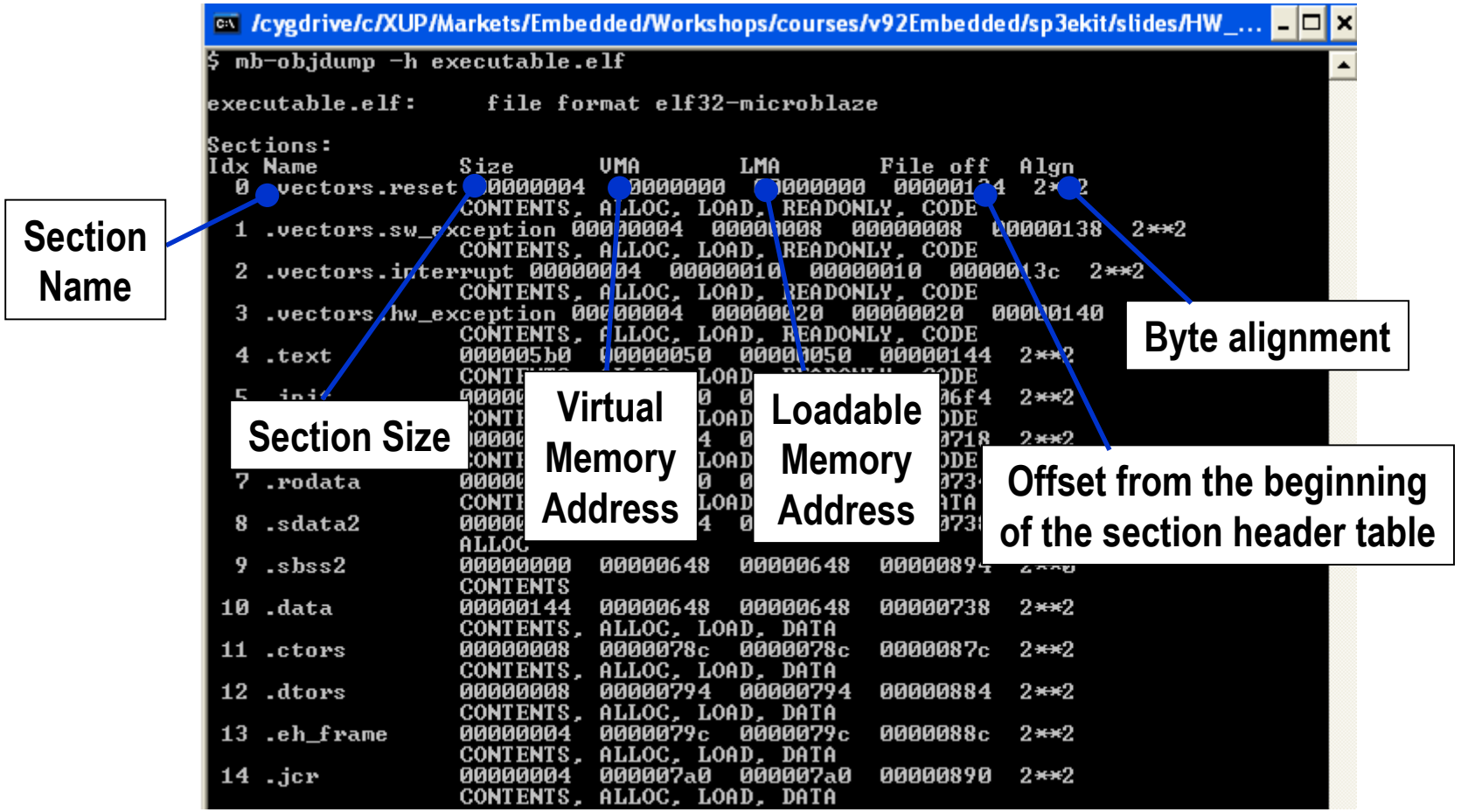
See Embedded System Tools Reference Manual for Complete List of utilities

- AR Archiver
  - Create, modify, and extract from libraries
  - Used in EDK to combine the object files of the Board Support Package (BSP) in a library
  - Used in EDK to extract object files from different libraries
- Object Dump
  - Display information from object files and executables
    - Header information, memory map
    - Data
    - Disassemble code

MicroBlaze Object Dump

Display summary information from the section headers

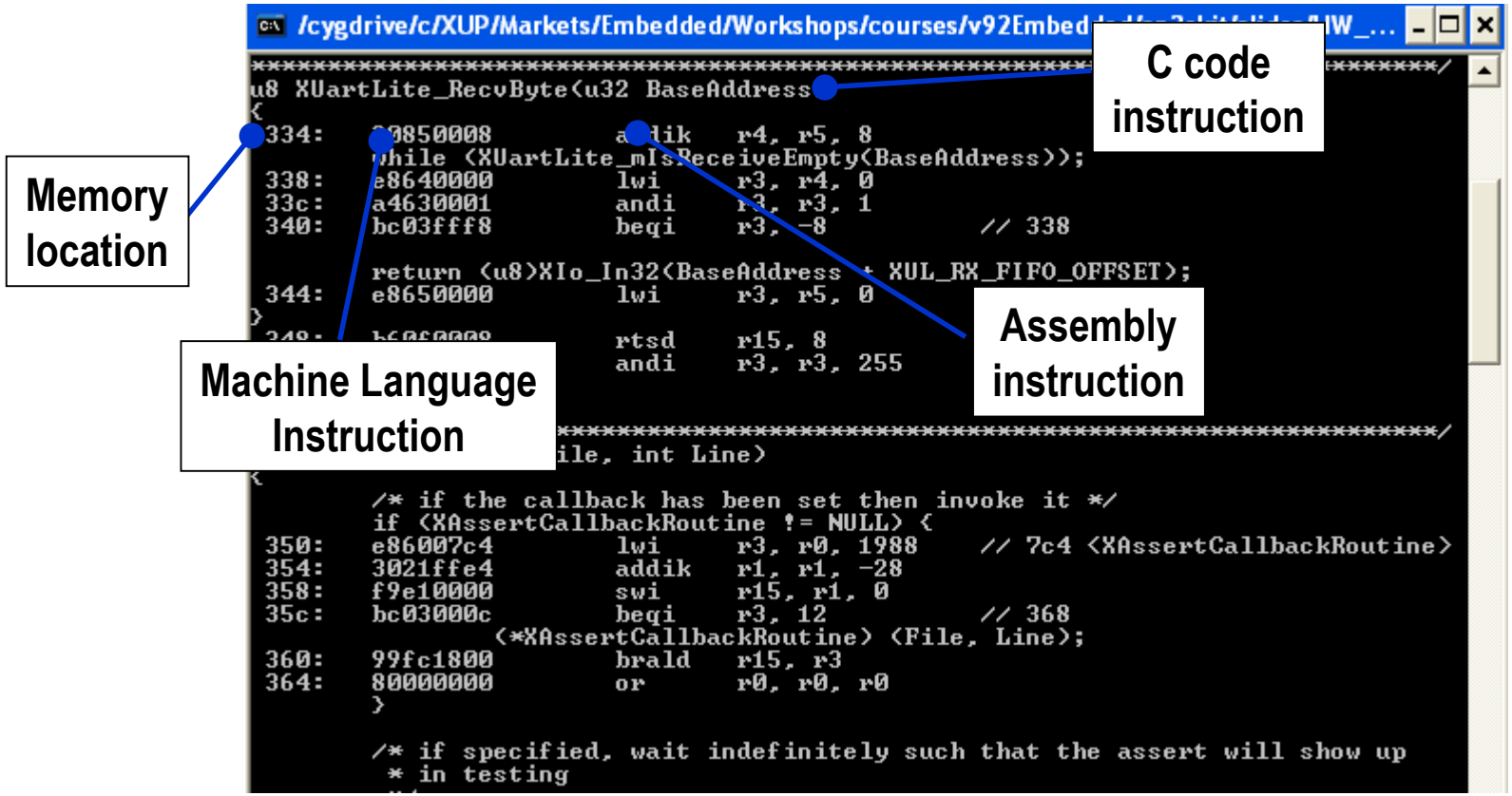
mb-objdump –h executable.elf



MicroBlaze Object Dump

Dumping the source and assembly code

mb-objdump –S executable.elf



Outline

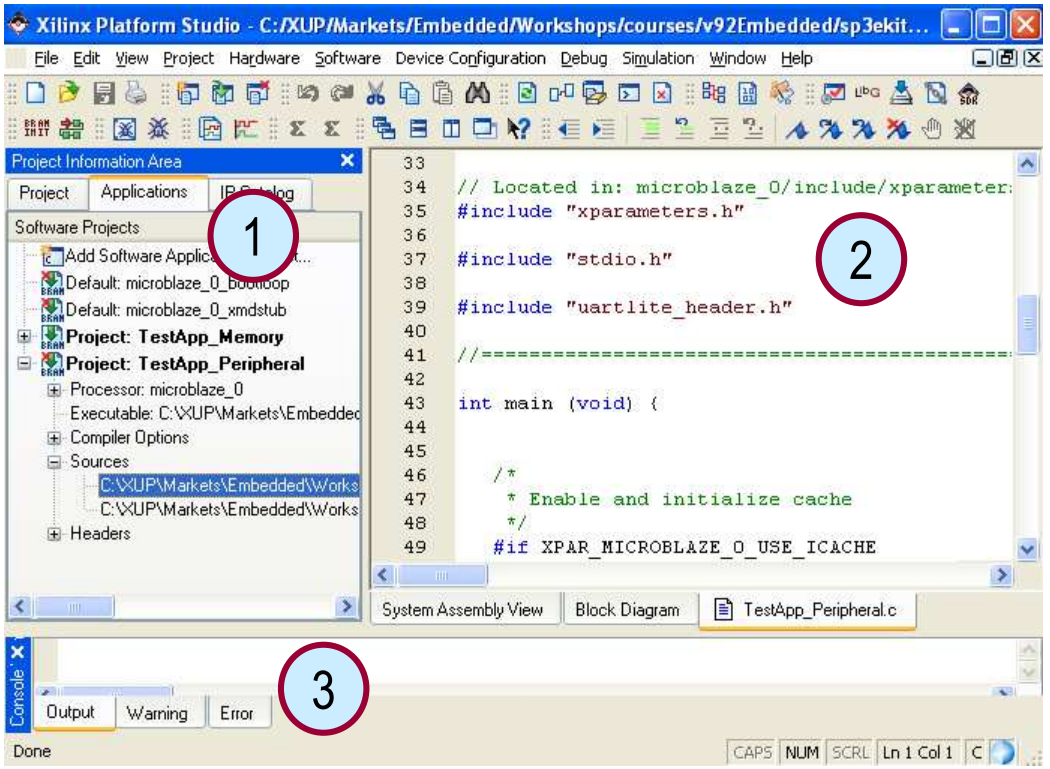
- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- **Development Environments**
  - **XPS**
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Software Development Environment: XPS

1. Lists Software Projects
   (Specifies processor instance, groups source code according to instance, xparameters.h, source and header files)

2. Standard text editor for creating c/c++ applications

3. Console displays status, errors, and warnings



Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- **Development Environments**
  - XPS
  - **SDK**
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

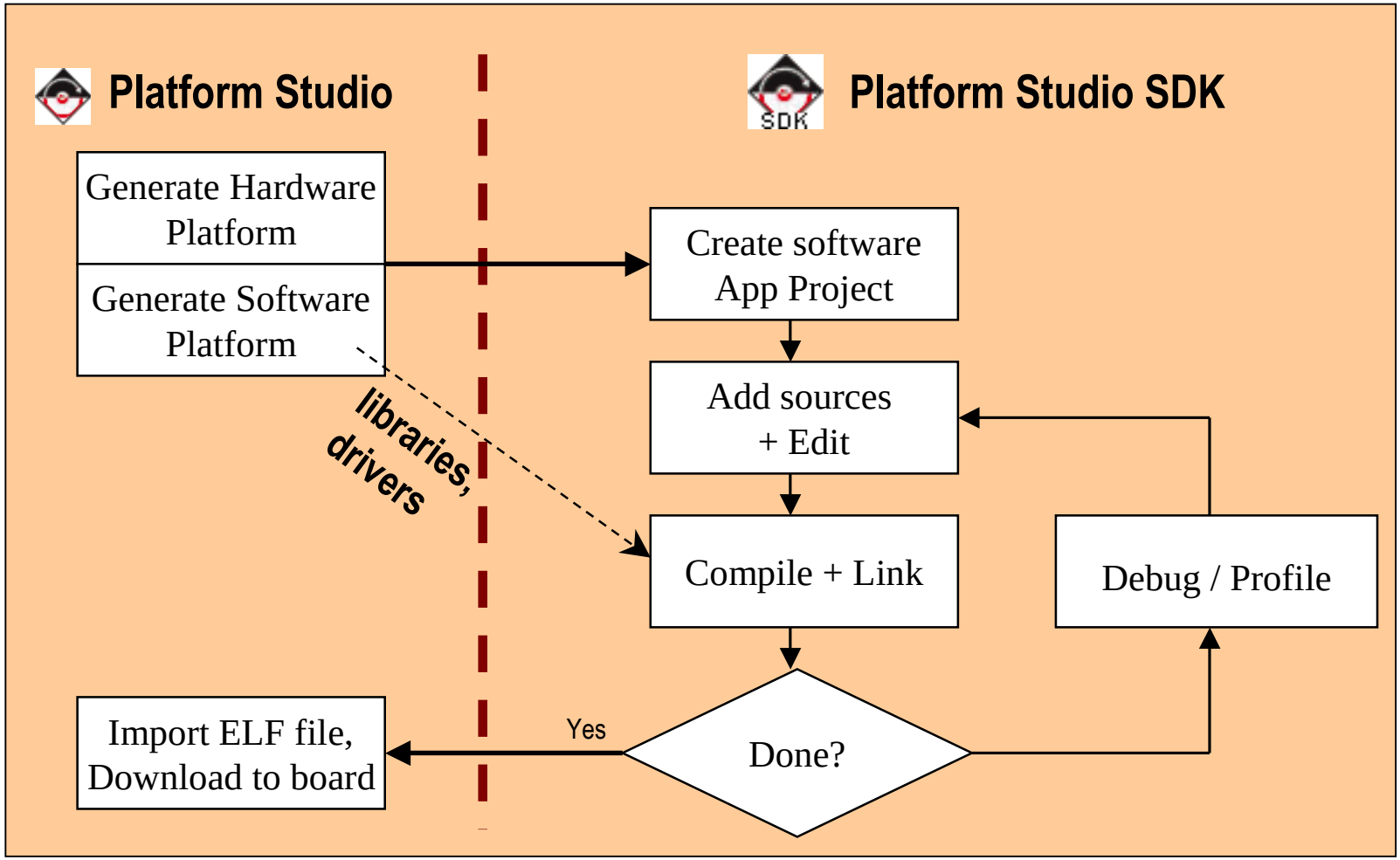
Software Development Environment: SDK

- Java-based application development environment
- Based on the open-source effort by the Eclipse Consortium\
- Feature-rich C/C++ code editor and compilation environment
- Project management
- Application build configuration and automatic Makefile generation
- Error Navigation
- Well-integrated environment for seamless debugging of embedded targets
- Source code version control

Eclipse/CDT Frameworks

- Builder framework
  - Compiles and Links Source files
  - Default Build options are specified when application is created: Choice of Debug, Release, Profile configurations
  - User can custom build options later when developing application
  - Build types: Standard Make, Managed Make
- Launch framework
  - Specifies what action needs to be taken: Run (+ Profile) application or Debug application
  - In SDK, this is akin to the Target Connection settings
- Debug framework
  - Launches debugger (gdb), loads application and begins debug session
  - Debug views show information about state of debug session
  - Hides ugliness of debug details
- Search framework
  - Helps development of application
- Help System
  - Online help system; context-sensitive

SDK Application Development Flow



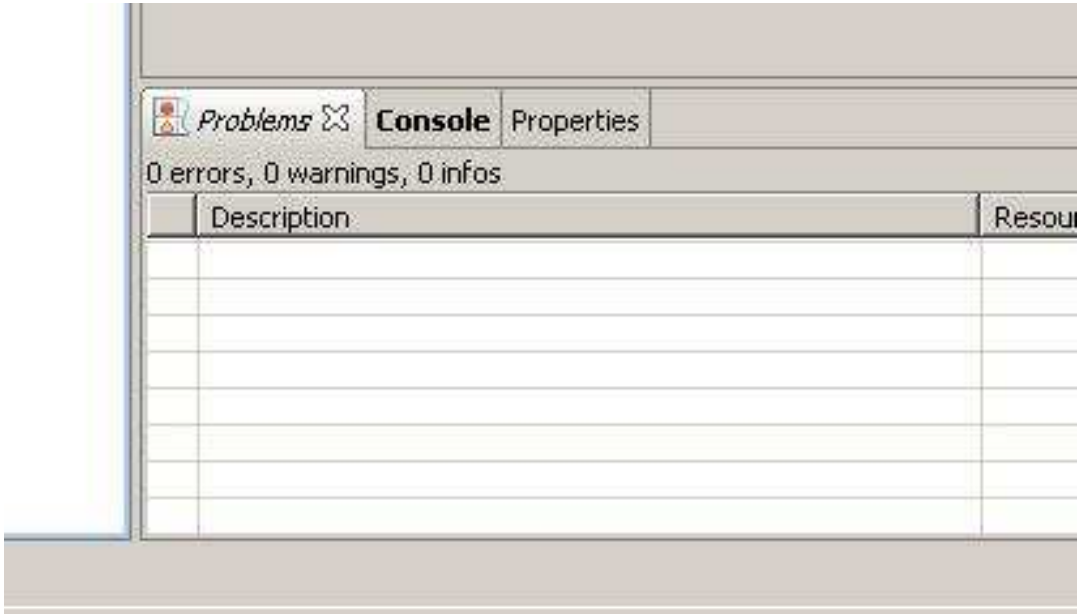
Libraries can be generate/updated from SDK

Workspaces and Perspectives

- Workspace
  - Location to store preferences & internal info about Projects
  - Transparent to SDK users
  - In SDK, source files not stored under Workspace
- Views, Editors
  - Basic User interface element
- Perspectives
  - Collection of functionally related views
  - Layout of views in a perspective can be customized according to user preference

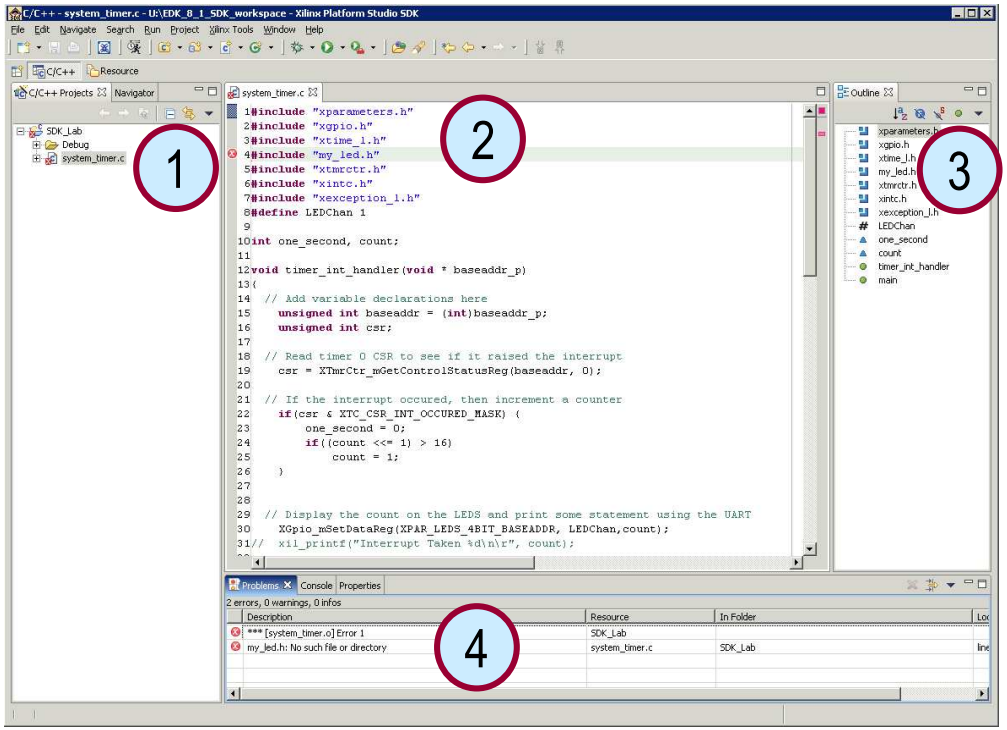
Views

- **Eclipse Platform views:** Navigator view, Tasks view, Problems view
- **Debug views:** Stack view, Variables view
- **C/C++ views:** Projects view, Outline view



C/C++ Perspective

1. C/C++ project outline displays the elements of a project with file decorators (icons) for easy identification
2. C/C++ editor for integrated software creation
3. Code outline displays elements of the software file under development with file decorators (icons) for easy identification
4. Problems, Console, Properties view lists output information associated with the software development flow



Opening Perspectives and Views

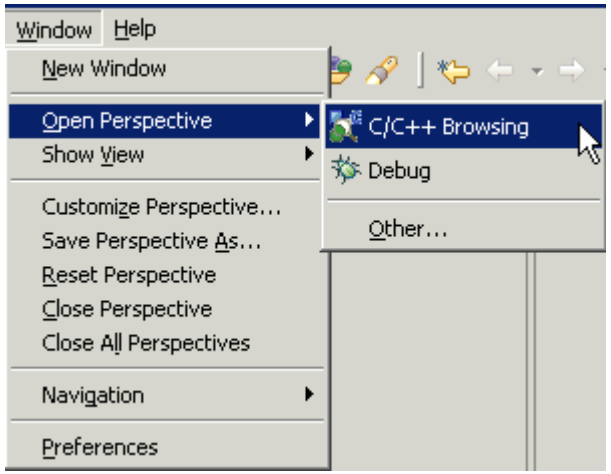
- To open a Perspective, use

  Window → Open Perspective

- To open a view, use

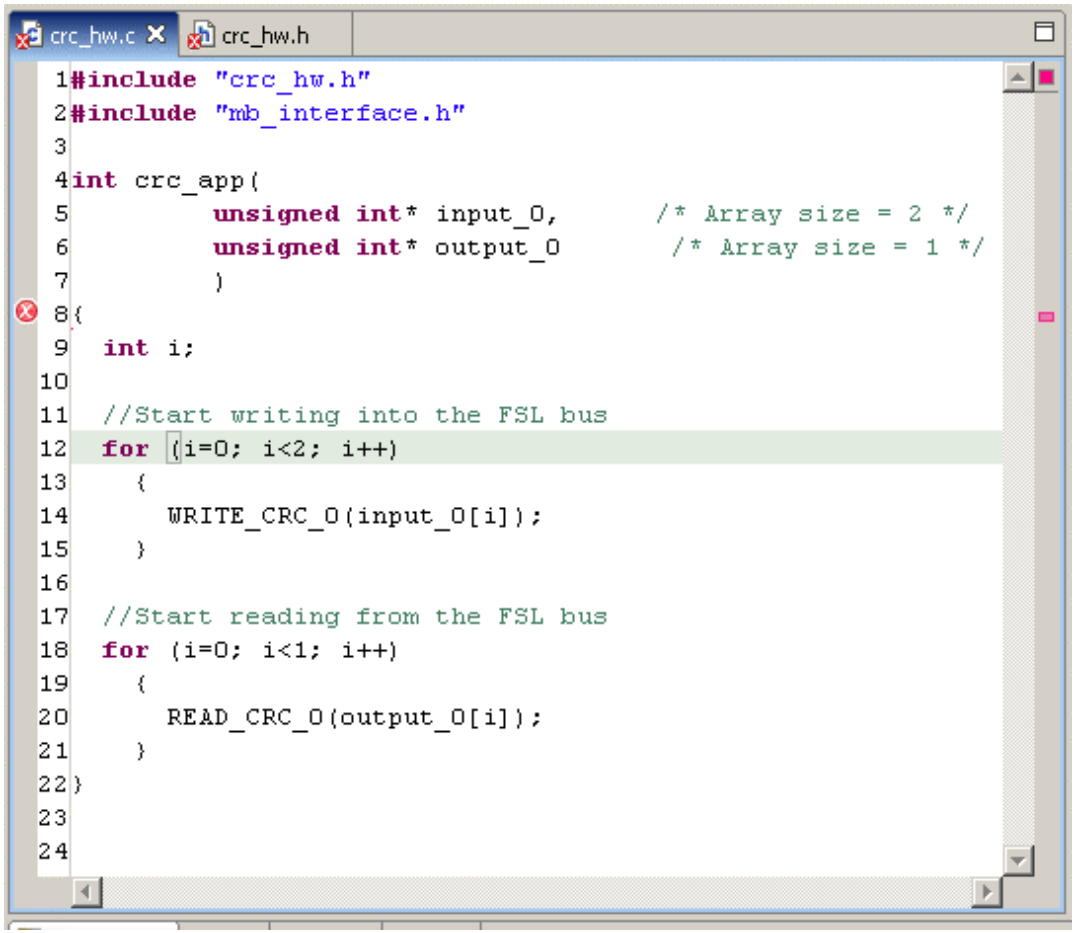
  Window → Show View

  If the view is already present in the current perspective, the view is highlighted



Editors

- bracket matching
- syntax coloring
- content assist
- refactoring
- keyboard shortcuts



Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- **Device Drivers**
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Device Drivers

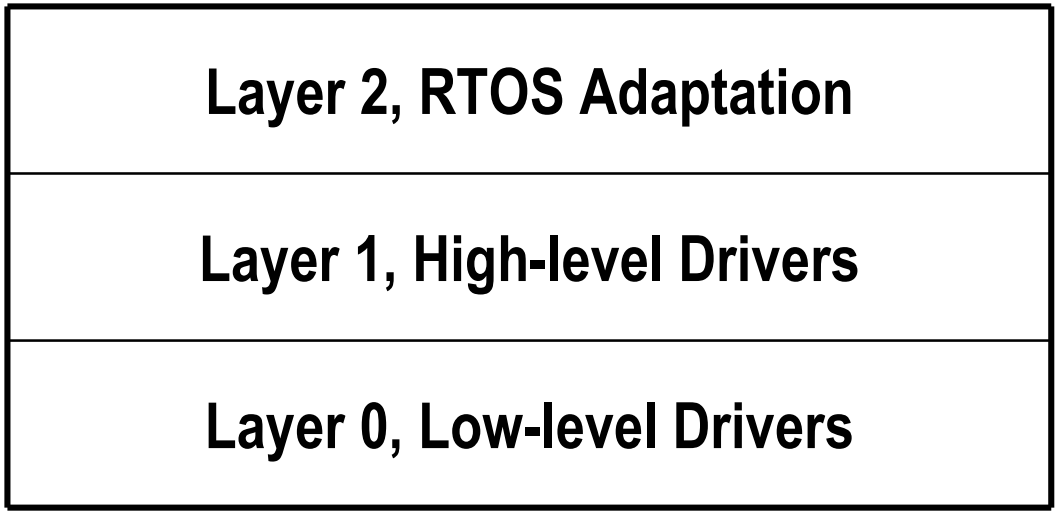
- The Xilinx device drivers are designed to meet the following objectives:
  - Provide maximum portability
    - The device drivers are provided as ANSI C source code
  - Support FPGA configurability
    - Supports multiple instances of the device without code duplication for each instance, while at the same time managing unique characteristics on a per-instance basis
  - Support simple and complex use cases
    - A layered device driver architecture provides both
      - Simple device drivers with minimal memory footprints
      - Full-featured device drivers with larger memory footprints
  - Ease of use and maintenance
    - Xilinx uses coding standards and provides well-documented source code for developers

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- **Device Drivers**
  - **Level 0, Level 1**
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Drivers: Level 0 / Level 1

- The layered architecture provides seamless integration with…
  - (Layer 2) RTOS application layer
  - (Layer 1) High-level device drivers that are full-featured and portable across operating systems and processors
  - (Layer 0) Low-level drivers for simple use cases



Drivers: Level 0

- Consists of low-level device drivers
- Implemented as macros and functions that are designed to allow a developer to create a small system
- Characteristics:
  - Small memory footprint
  - Little to no error checking is performed
  - Supports primary device features only
  - No support of device configuration parameters
  - Supports multiple instances of a device with base address input to the API
  - Polled I/O only
  - Blocking function calls
  - Header files have "_l" in their names (for example, xuartlite_l.h)

Drivers: Level 1

- Consists of high-level device drivers
- Implemented as macros and functions and designed to allow a developer to utilize all of the features of a device
- Characteristics:
  - Abstract API that isolates the API from hardware device changes
  - Supports device configuration parameters
  - Supports multiple instances of a device
  - Polled and interrupt driven I/O
  - Non-blocking function calls to aid complex applications
  - May have a large memory footprint
  - Typically, provides buffer interfaces for data transfers as opposed to byte interfaces
  - Header files *do not* have "_l" in their names (for example, xuartlite.h)

Comparison Example

- **Uartlite Level 1**
  - **XStatus XUartLite_Initialize** (**XUartLite** *InstancePtr, **Xuint16** DeviceId)
  - void **XUartLite_ResetFifos** (**XUartLite** *InstancePtr)
  - unsigned int **XUartLite_Send** (**XUartLite** *InstancePtr, **Xuint8** *DataBufferPtr, unsigned int NumBytes)
  - unsigned int **XUartLite_Recv** (**XUartLite** *InstancePtr, **Xuint8** *DataBufferPtr, unsigned int NumBytes)
  - **Xboolean XUartLite_IsSending** (**XUartLite** *InstancePtr)
  - void **XUartLite_GetStats** (**XUartLite** *InstancePtr, **XUartLite_Stats** *StatsPtr)
  - void **XUartLite_ClearStats** (**XUartLite** *InstancePtr)
  - **XStatus XUartLite_SelfTest** (**XUartLite** *InstancePtr)
  - void **XUartLite_EnableInterrupt** (**XUartLite** *InstancePtr)
  - void **XUartLite_DisableInterrupt** (**XUartLite** *InstancePtr)
  - void **XUartLite_SetRecvHandler** (**XUartLite** *InstancePtr, **XUartLite_Handler** FuncPtr, void *CallBackRef)
  - void **XUartLite_SetSendHandler** (**XUartLite** *InstancePtr, **XUartLite_Handler** FuncPtr, void *CallBackRef)
  - void **XUartLite_InterruptHandler** (**XUartLite** *InstancePtr)
- **Uartlite Level 0**
  - void **XUartLite_SendByte** (**Xuint32** BaseAddress, **Xuint8** Data)
  - **Xuint8 XUartLite_RecvByte** (**Xuint32** BaseAddress)

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environment
  - XPS
  - SDK
- **Device Drivers**
  - Level 0, Level 1
  - **MicroBlaze Processor: Interrupts**
  - Integration in EDK
- Libraries
- BSP
  - Boot Files and Sequence

Interrupt Management

With Interrupt Controller

- The interrupt controller is required if more than one interrupting device is present
  - Connect peripheral's interrupt requesting signals to the **Intr** port of the interrupt controller in the MHS file

    e.g., PORT Intr = RS232_Interrupt & interrupt_push & interrupt_timer
  - Connect interrupt controller output **intc** to a processor interrupt pin

    e.g., PORT Irq = interrupt_req

  - Define an external requesting signal, if needed, in the global ports section of the MHS file

    e.g., PORT interrupt_in1 = interrupt_in1, DIR = IN, LEVEL = low, SIGIS = Interrupt
  - Connect the external interrupt signal to the Intr port of the interrupt controller

Interrupt Management

Without Interrupt Controller

- The interrupt controller is not required when only one interrupting device is present
  - The interrupt signal of the peripheral (or the external interrupt signal) must be connected to the interrupt input of the MicroBlaze™ processor in the MHS file

- Software interface for the interrupt
  - Define the signal in MSS file to associate them to peripherals

    e.g., PARAMETER int_handler = uart_int_handler, int_port = Interrupt
  - Write an interrupt handler routine to service the request
  - The base address of the peripheral instance can be accessed as XPAR_*INSTANCE_NAME*_BASEADDR

MicroBlaze Interrupts

- One INTERRUPT port on the MicroBlaze™ processor
- MicroBlaze processor functions
  - void microblaze_enable_interrupts(void)
    - This function enables interrupts on the MicroBlaze processor
    - When the MicroBlaze processor starts up, interrupts are disabled. Interrupts must be explicitly turned on by using this function
  - void microblaze_disable_interrupts(void)
    - This function disables interrupts on the MicroBlaze processor. This function may be called when entering a critical section of code where a context switch is undesirable

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - **Integration in EDK**
- Libraries
- BSP
  - Boot Files and Sequence

Integration in EDK

- When the interrupt generating device is connected to the processor interrupt pin, either through an interrupt controller or directly, the interrupt handler function must be developed (You must explicitly write code to set up the interrupt mechanism)
- The interrupt handler must be registered explicitly in code

```
71 int main() {
72
73   int count_mod_3;
74
75    // Enable MicroBlaze Interrupts
76   microblaze_enable_interrupts();
77
78   /* Register the Timer interrupt handler in the vector table */
79   XIntc_RegisterHandler(XPAR_XPS_INTC_0_BASEADDR,
80                         XPAR_XPS_INTC_0_DELAY_INTERRUPT_INTR,
81                         (XInterruptHandler) timer_int_handler,
82                         (void *)XPAR_DELAY_BASEADDR);
83
84   /* Initialize and set the direction of the GPIO connected to LEDs */
85   XGpio_Initialize(&gpio, XPAR_LEDS_8BIT_DEVICE_ID);
86   XGpio_SetDataDirection(&gpio,LEDChan, 0);
```

Registering an Interrupt Handler

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- **Libraries**
- BSP
  - Boot Files and Sequence

Libraries

- Xilinx provides three libraries
  - Math library **(libm)**
    - The math library is an improvement over the newlib math library
    - The -lm option is used for libm functions
  - Standard C language support **(libc)**
    - The functions of this library are automatically available
  - Xilinx C drivers and libraries (**libxil)**
    - Xilinx file support functions: **Fatfs**
    - Xilinx memory file system: **Mfs**
    - Xilinx networking support: **lwlp**
    - Xilinx flash memory support: **Flash**

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- **BSP**
  - Boot Files and Sequence

What is a BSP?

- Board Support Package (BSP):
  - Lowest layer of software modules used to access processor specific functions
    - Interrupt and Exception Handling
    - Instruction and Data Cache Handling
    - Fast Simplex Link interface macros
    - Program Profiling
  - Allows you to use IP peripheral-device drivers
    - GPIO, IIC controller, PCI controller, UART
  - Offers glue functionality to link code against standard libraries
    - Time, sleep
    - Files
    - Memory
  - Standalone BSP (no operating system)
    - Libgen generates libxil.a library

Hardware IP Device Drivers

- Driver
  - Provides an interface for the software to communicate with the hardware
  - Designed to be portable across processor architectures and operating systems
- Delivery format
  - Delivered as source code, allowing it to be built and optimized
  - Minimized assembly language
  - C programming language

Outline

- Introduction
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- GNU Development Tools: GCC, AS, LD, Binutils
- Development Environments
  - XPS
  - SDK
- Device Drivers
  - Level 0, Level 1
  - MicroBlaze Processor: Interrupts
  - Integration in EDK
- Libraries
- BSP
  - **Boot Files and Sequence**

Boot Files

- The compiler includes pre-compiled startup and end files in the final link command when forming an executable
- Startup Files setup the language and the platform environment before your application code executes
- The following actions are typically performed
  - Setup any reset, interrupt, and exception vectors as required
  - Setup stack pointer, small-data anchors, and other registers
  - Clear the BSS memory regions to zero
  - Invoke language initialization functions, such as c++ constructors
  - Initialize the hardware subsystem (ie. initialize profiling timers if the program is to be profiled)
  - Setup arguments for the main procedure and invoke it
  - Invoke language cleanup functions, such as c++ destructors
  - De-Initialize the hardware sub-system (ie. clean up the profiling subsystem if the program was profiled)

See Embedded System Tools reference manual for details

First Stage Initialization Files

- crt0.S
  - used for programs which are to be executed in standalone mode, without the use of any boot loader
- crt1.S
  - used when the application is debugged in a software-intrusive manner
- crt2.S
  - Used when the executable is loaded using a boot loader
- crt3.S
  - Employed when the executable does not use any vectors and wishes to reduce code size

Second Stage Initialization Files

- crtinit
  - Default second stage C startup file
- pgcrtinit
  - Used during profiling
- sim-crtinit
  - Used when the –mno-clearbss switch is used in the compiler
- sim-pgcrtinit
  - Used during profiling in conjunction with the –mno-clearbss switch

crt0.s

- Application entry point at label **_start**
- _start
  - Set up any reset, interrupt, and exception vectors as required
  - Transfers control to crtinit (see next slide)
  - On returning from _critinit, it ends the program by infinitely looping in the _exit label

crtinit

- Clear the BSS (.bss and .sbss sections) memory regions to zero
- Invokes _program_init: language initialization functions, such as C++ constructors
- Invokes “constructor” functions (_init): Initializes interrupt handler and the hardware sub-system
- Set up arguments for main() and invokes main()
- Invokes “destructor” functions (_fini)
- Invokes _program_clean and returns

Skills Check

Knowledge Check

- What GNU GCC option is used to specify that debugging information should be placed in the executable?
- What is included in a BSP?
- What are some of the differences between a Level 0 and a Level 1 driver?

Answers

- What GNU GCC option is used to specify that debugging information should be placed in the executable?
  - -g
- What is included in a BSP?
  - IP drivers
  - Processor functions
  - Library functions
- What are some of the differences between a Level 0 and a Level 1 driver?
  - Size
  - Functionality
  - Ease of use

Knowledge Check

- List libraries supported and their functionality

- How many interrupt pins are present on MicroBlaze?

Answers

- List libraries supported and their functionality
  - Fatfs – provides file support functions
  - MFS – provides memory file system support functions
  - lwip – provides networking support functions including handling of multiple connections
  - Flash – provides read/write/erase types of functions for Intel parallel flash devices so user can program flash memory during run-time
- How many interrupt pins are present on MicroBlaze?
  - One

Where Can I Learn More?

- Tool documentation
  - ***Embedded System Tools Guide → Microprocessor Software Specifications***
  - ***Embedded System Tools Guide → Microprocessor Driver Definition***
  - ***Embedded System Tools Guide → Microprocessor Library Definition***
  - ***EDK OS and Libraries Reference Guide → LibXil File, Net, and Kernel***
  - *Processor IP Reference Guide*
  - *Xilinx Drivers*
- Processor documentation
  - *MicroBlaze Processor Reference Guide*
- Support website
  - EDK Website: www.xilinx.com/edk