Software Development - Basic

MicroBlaze
14.2 Version

Objectives

- **After completing this module, you will be able to:**
  - Understand the basic concepts of the Eclipse IDE in SDK
  - List SDK features
  - Identify the GNU tools functionality
  - List steps in creating a software application

Outline

- ***Introduction***
- **SDK Development Environment**
- **SDK Project Creation**
- **GNU Development Tools: GCC, AS, LD, Binutils**
- **Software Settings**
  - Software Platform Settings
  - Compiler Settings
- **Summary**

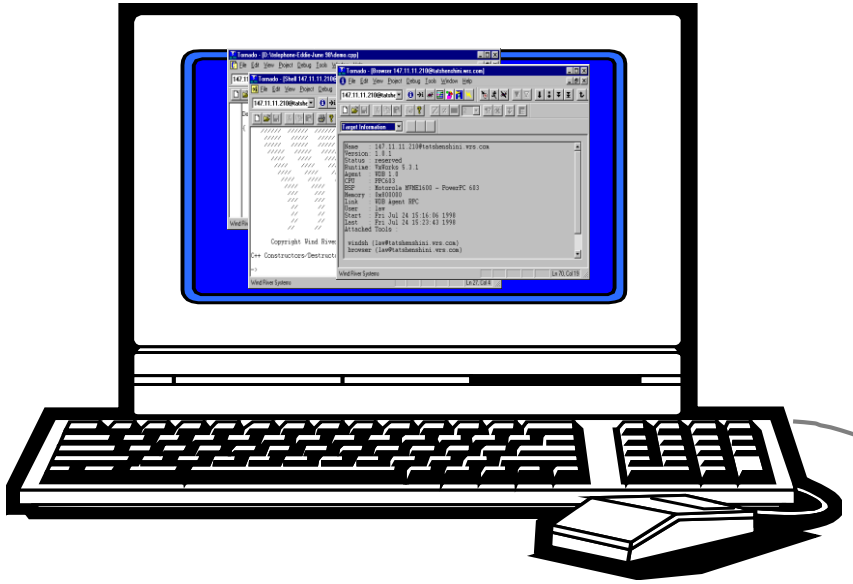
Desktop versus Embedded

- **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)**

Host Computer



Target Computer

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
- *SDK Development Environment*
- SDK Project Creation
- GNU Development Tools: GCC, AS, LD, Binutils
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- Summary

Eclipse/CDT Frameworks

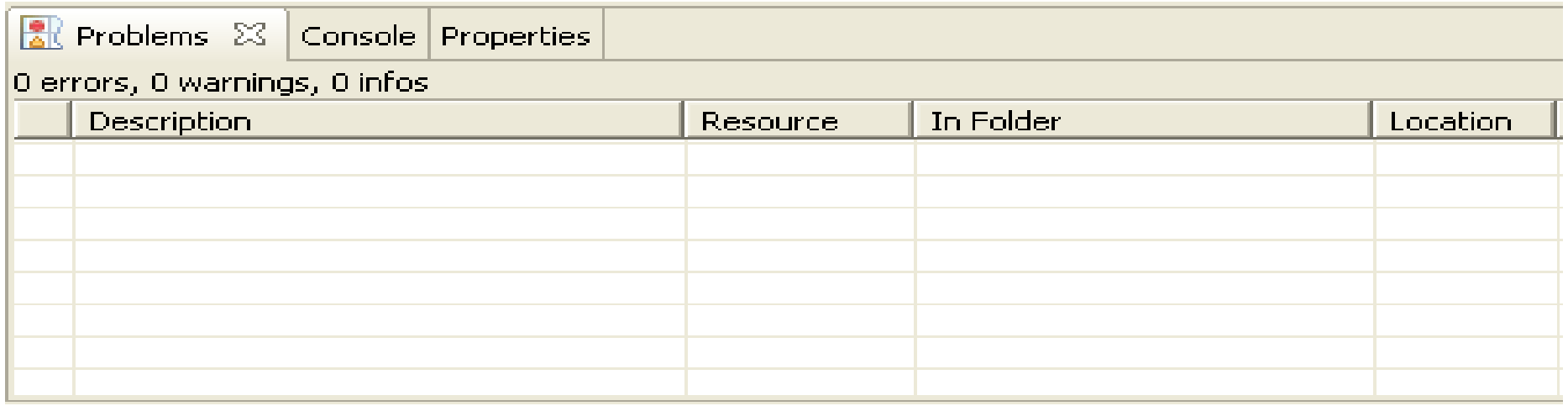
- **Builder framework**
  - Compiles and Links Source files
  - Default Build options are specified when application is created: Choice of Debug, Release, Profile configurations
  - User can custom build options later when developing application
  - Build types: Standard Make, Managed Make
- **Launch 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
- **Search framework**
  - Helps development of application
- **Help System**
  - Online help system; context-sensitive

Workspaces and Perspectives

- **Workspace**
  - Location to store preferences & internal info about projects
  - Transparent to users
  - 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

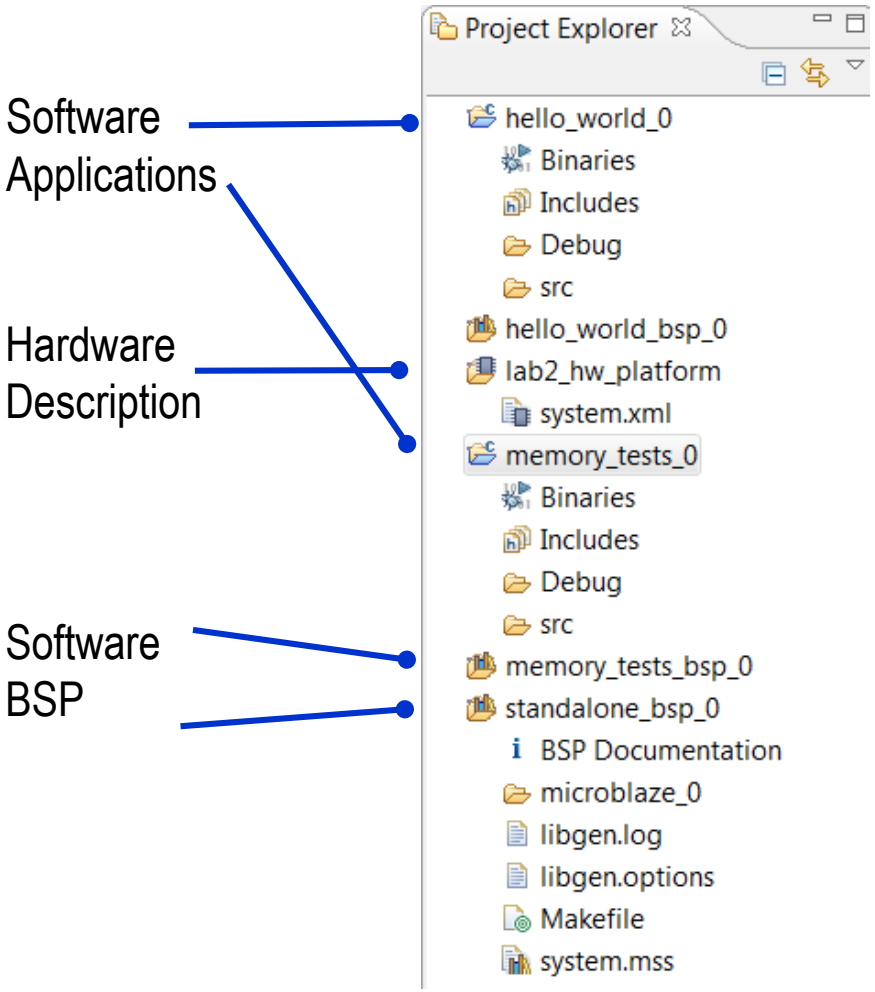
Views

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



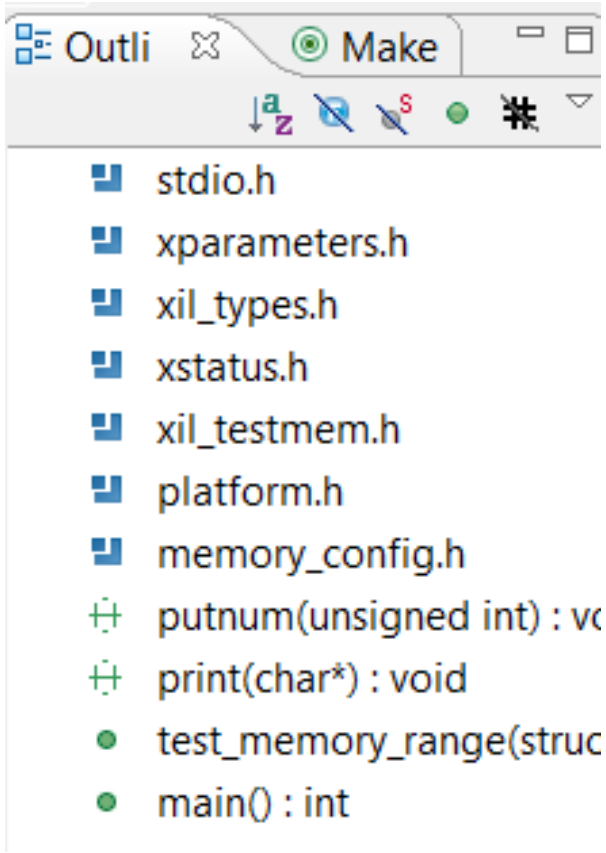
C/C++ Project View

- Hierarchical list of the workspace projects in a hierarchical format
- Double-click to open a file
- Right-click the project to access its properties



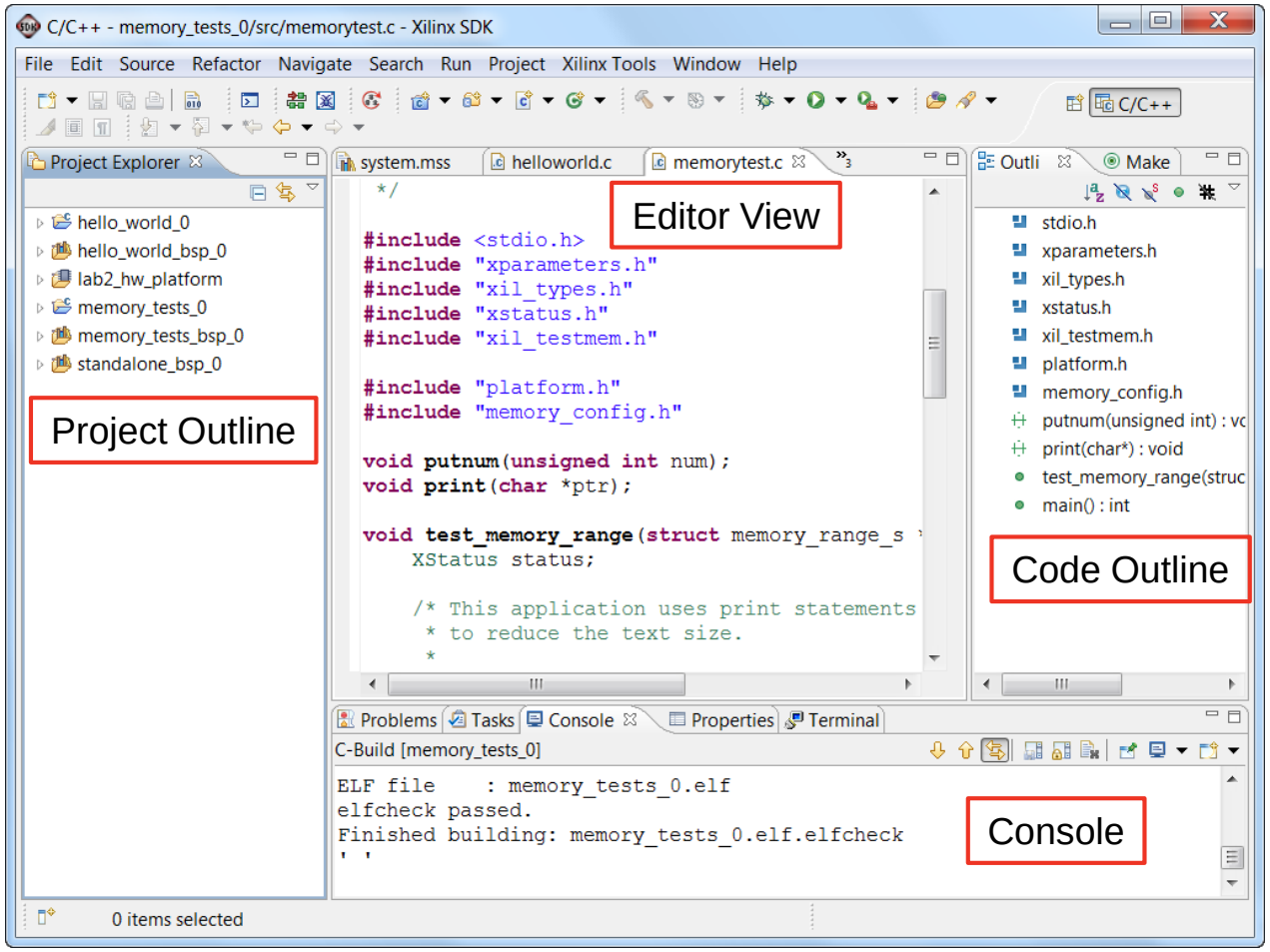
Outline View

- **Displays an outline of the structured file that is currently open in the editor**
- **The contents of the outline view are editor specific**
- **Content type is indicated by the icon**
- **For a C source, icons represent**
  - *#define* statements
  - Include files
  - Function calls
  - Declarations
- **Selecting a symbol will navigate to the same in the editor window**



C/C++ Perspective

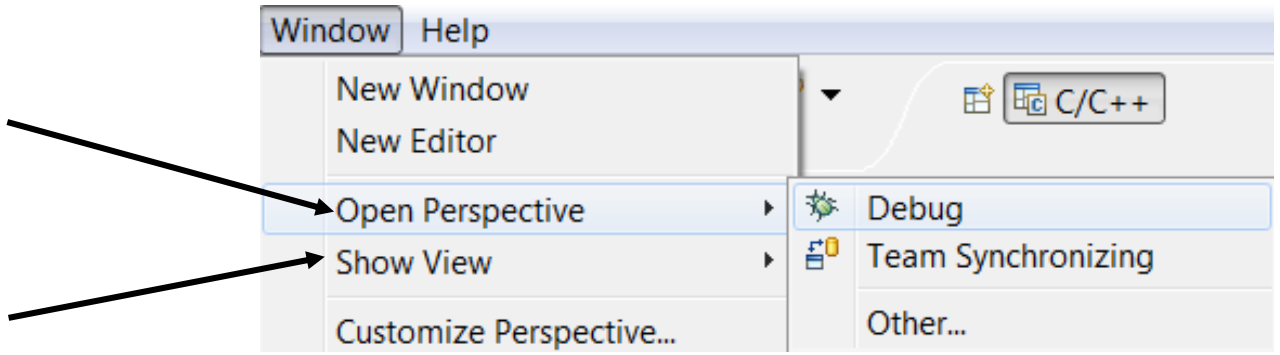
- C/C++ project outline displays the elements of a project with file decorators (icons) for easy identification
- C/C++ editor for integrated software creation
- Code outline displays elements of the software file under development with file decorators (icons) for easy identification
- Problems, Console,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



XILINX ALL PROGRAMMABLE.

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



Editor

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

memorytest.c

```c
#include <stdio.h>
#include "xparameters.h"
#include "xil_types.h"
#include "xstatus.h"
#include "xil_testmem.h"

#include "platform.h"
#include "memory_config.h"

void putnum(unsigned int num);
void print(char *ptr);

void test_memory_range(struct memory_range_s *
    XStatus status;

    /* This application uses print statements
     * to reduce the text size.
     *
     * The default linker script generated for
     * heap memory allocated. This implies tha
```

Outline

- Introduction
- SDK Development Environment
- *SDK Project Creation*
- GNU Development Tools: GCC, AS, LD, Binutils
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- Summary

Launching SDK

- **Launch SDK from XPS project (recommended flow)**
  - In XPS, **Project > Export Hardware Design to SDK …**
  - A hardware image XML file is first generated
  - A hardware platform specification project is then automatically created
    - The software application then can be developed and associated
- **Create a board support package for a user application**
  - If using standard project template of SDK then this step may be skipped as it is automatically created by SDK
- **Create a software application**
  - Several standard application templates available
  - An Empty Application project may be selected for non-standard application
    - A board support package must be associated during the empty application project creation
    - Source files may be imported or created in the empty application project

Creating a Board Support Package

- **BSP must be created for the non-standard application project**
  - In SDK, File > New > Xilinx Board Support Package
  - Select appropriate OS support
  - Xilinx provides two platforms
    - Standalone
    - Xilkernel
  - Third-party operating systems are supported with the appropriate BSP selection
  - Select required libraries support
- **The Board Support Package provides software services based on the processor and peripherals that make up the processor system**

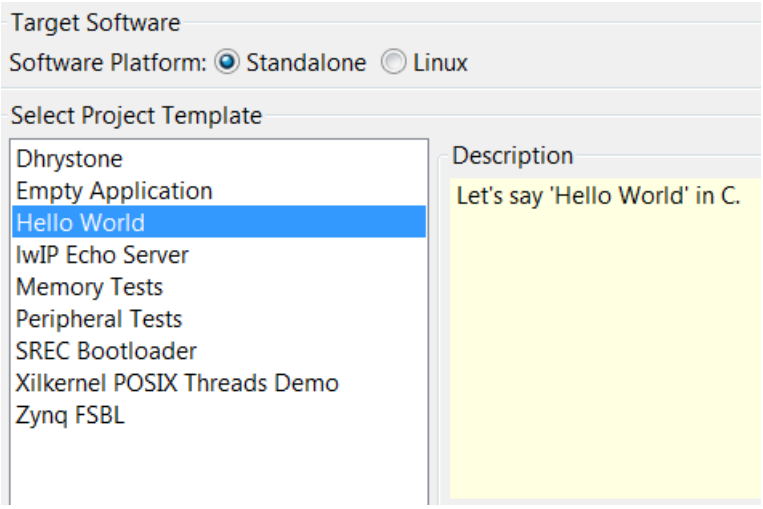
Board Support Package OS

xilkernel standalone	Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Name	Version	Description
lwip140	1.02.a	lwIP TCP/IP Stack library: lwIP v1.4.0, Xilinx...
xilfatfs	1.00.a	Provides read/write routines to access file...
xilflash	3.02.a	Xilinx Flash library for Intel/AMD CFI com...
xilisf	3.00.a	Xilinx In-system and Serial Flash Library
xilmfs	1.00.a	Xilinx Memory File System

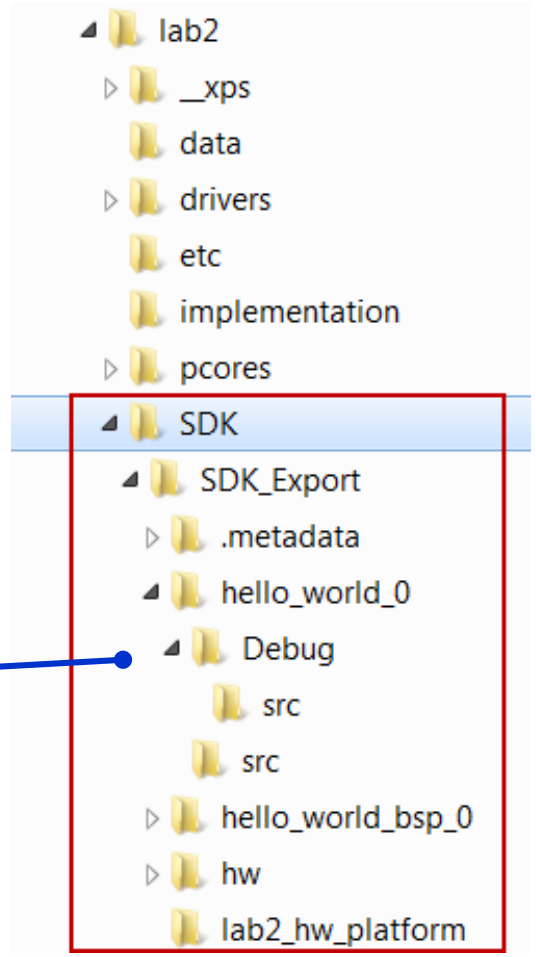
Creating a Software Application Project

- **SDK supports multiple software application projects**
- **A software project is attached to a BSP project**
- **Sample applications are provided**
  - Great for quick test of hardware
  - Starting point to base your own application on
- **Typically an Empty Application is opened to begin a non-standard project**



Directory Structure

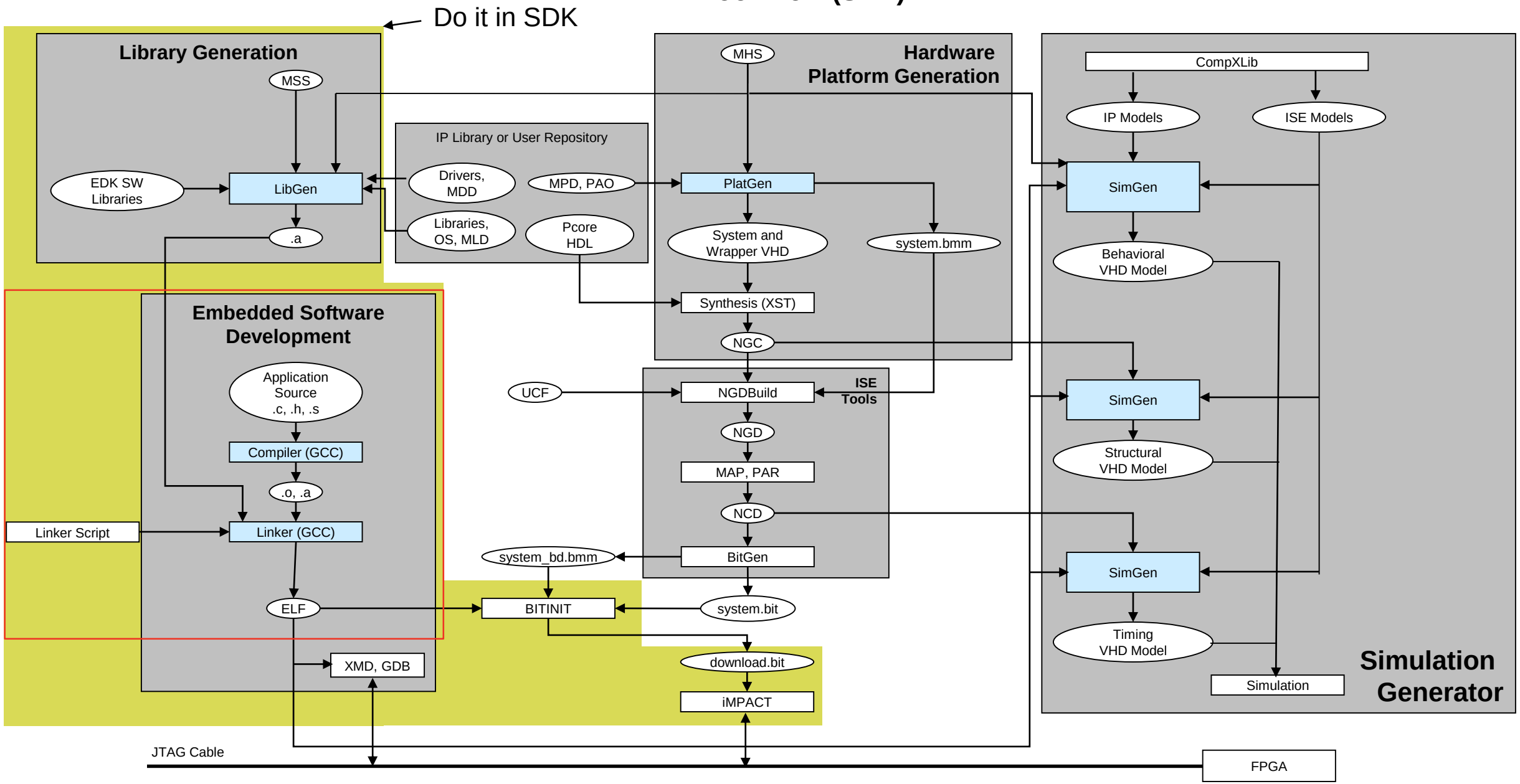
- **SDK projects are place in the application directory that was specified when SDK was launched**
- **Each project may have multiple directories for system files and configurations**
- **Configurations are property tool option permutations of the software application. Each configuration has project properties set depending on needs. An ELF file is generated for each**
  - Release configuration
  - Debug configuration
  - Profile configuration
- **A Debug configuration is created by default**



Outline

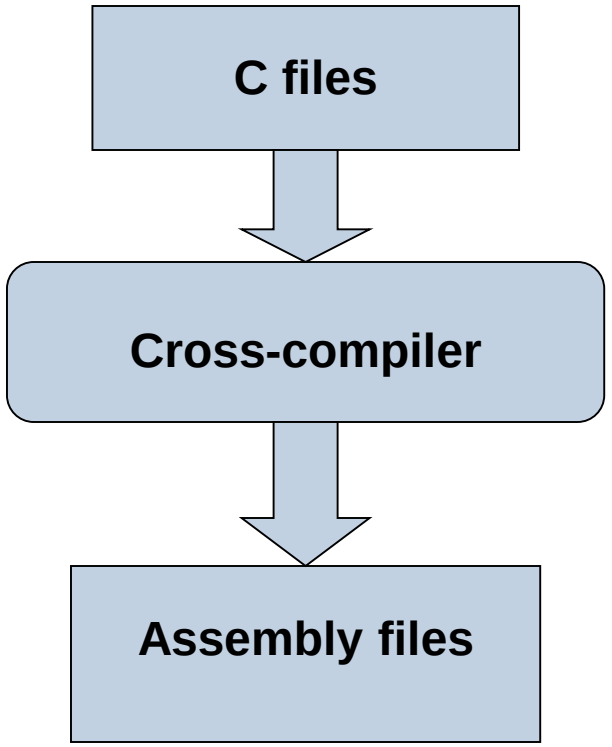
- Introduction
- SDK Development Environment
- SDK Project Creation
- *GNU Development Tools: GCC, AS, LD, Binutils*
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- Summary

EDK Tool Flow (SDK)



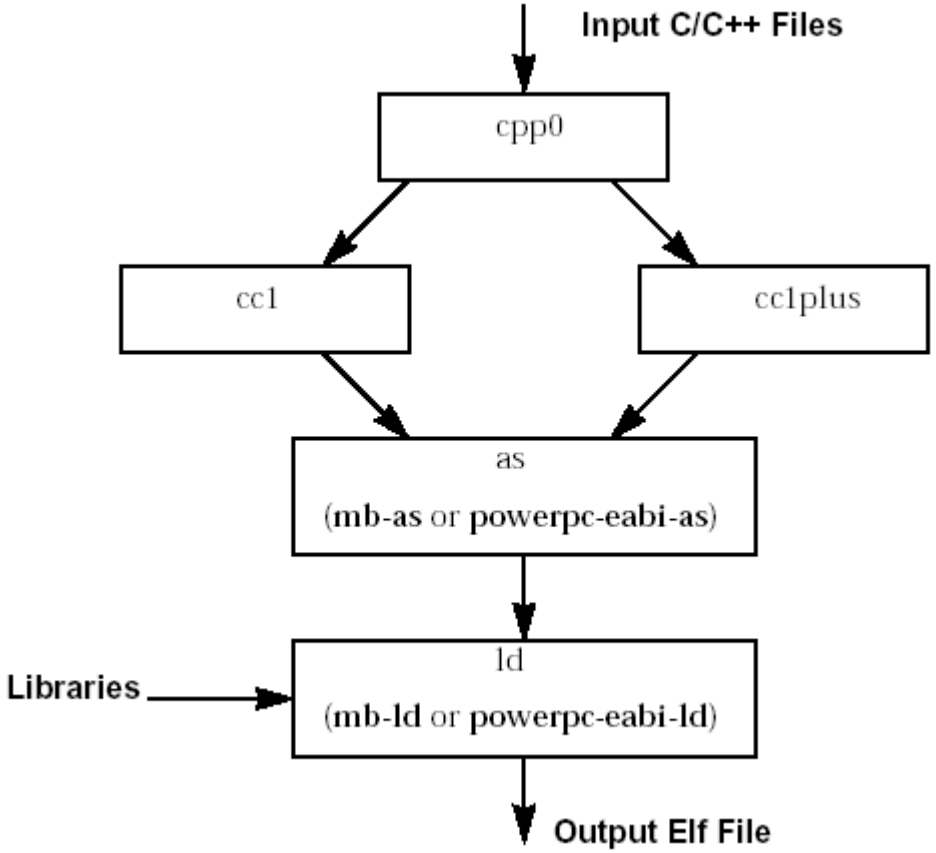
GNU Tools: GCC

- 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



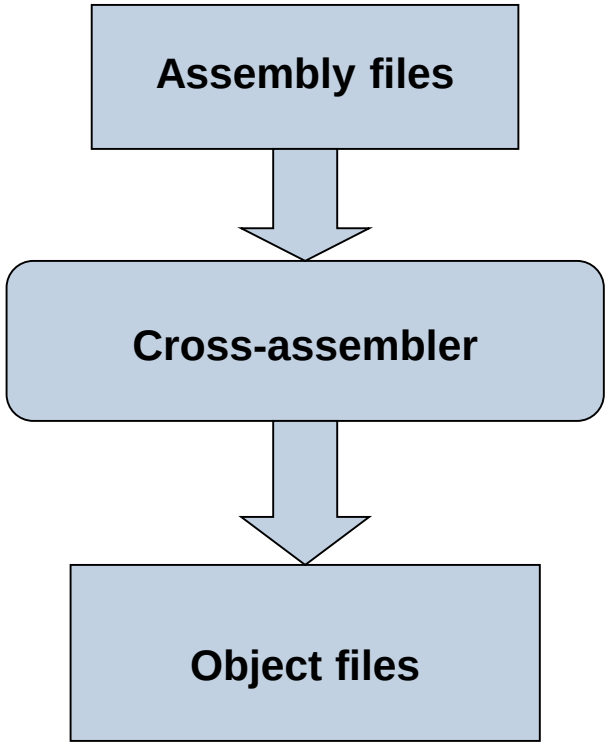
GNU Tools: GCC

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



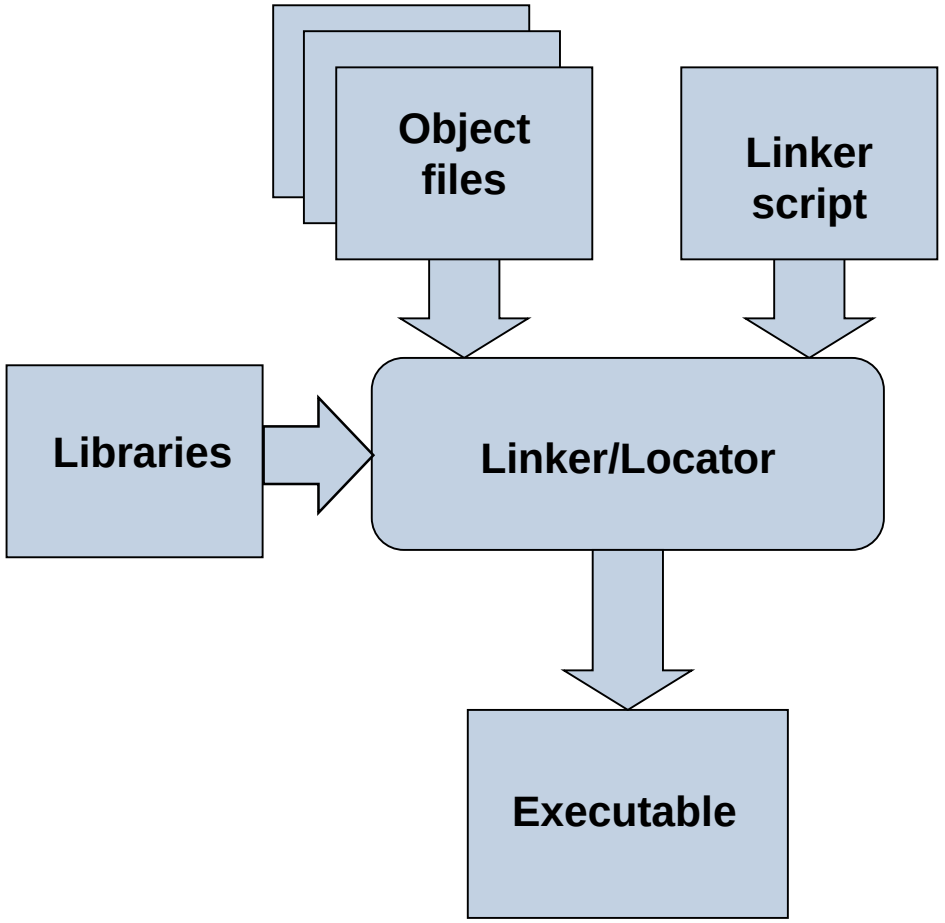
GNU Tools: AS

- **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 want to use the gcc**



GNU Tools: LD

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



GNU 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

```
/cygdrive/c/xup/embedded/labs/lab3/TestApp_Memory_microblaze_0
$ mb-objdump -h executable.elf

executable.elf:     file format elf32-microblaze

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .vectors.reset 00000004  00000000  00000000  000000d4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .vectors.sw_exception 00000004  00000008  00000008  000000d8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .vectors.interrupt 00000004  00000010  00000010  000000dc  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .vectors.hw_exception 00000004  00000020  00000020  000000e0  2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  4 .text         000015b8  00000050  00000050  000000e4  2**2
                  CONTEN                LOAD,
  5               000000                8   00          169c  2**2
                  CONTEN                LOAD,          ODE
  6               000000                2c  00          16c0
                  CONTEN                LOAD,          ODE
  7 .ctors        000000                48  00          16dc
                  CONTEN                LOAD,
$ _
                  CONTENTS, ALLOC, LOAD, DATA
  9 .rodata       0000043e  00001658  00001658  000016ec  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .sbss2        00000000  00001a98  00001a98  00001c5c  2**0
                  CONTENTS
 11 .data         00000128  00001a98  00001a98  00001b2c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 12 .eh_frame     00000004  00001bc0  00001bc0  00001c54  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 13 .jcr          00000004  00001bc4  00001bc4  00001c58  2**2
                  CONTENTS, ALLOC, LOAD, DATA
```

Section Name

Section Size

Virtual Memory Address

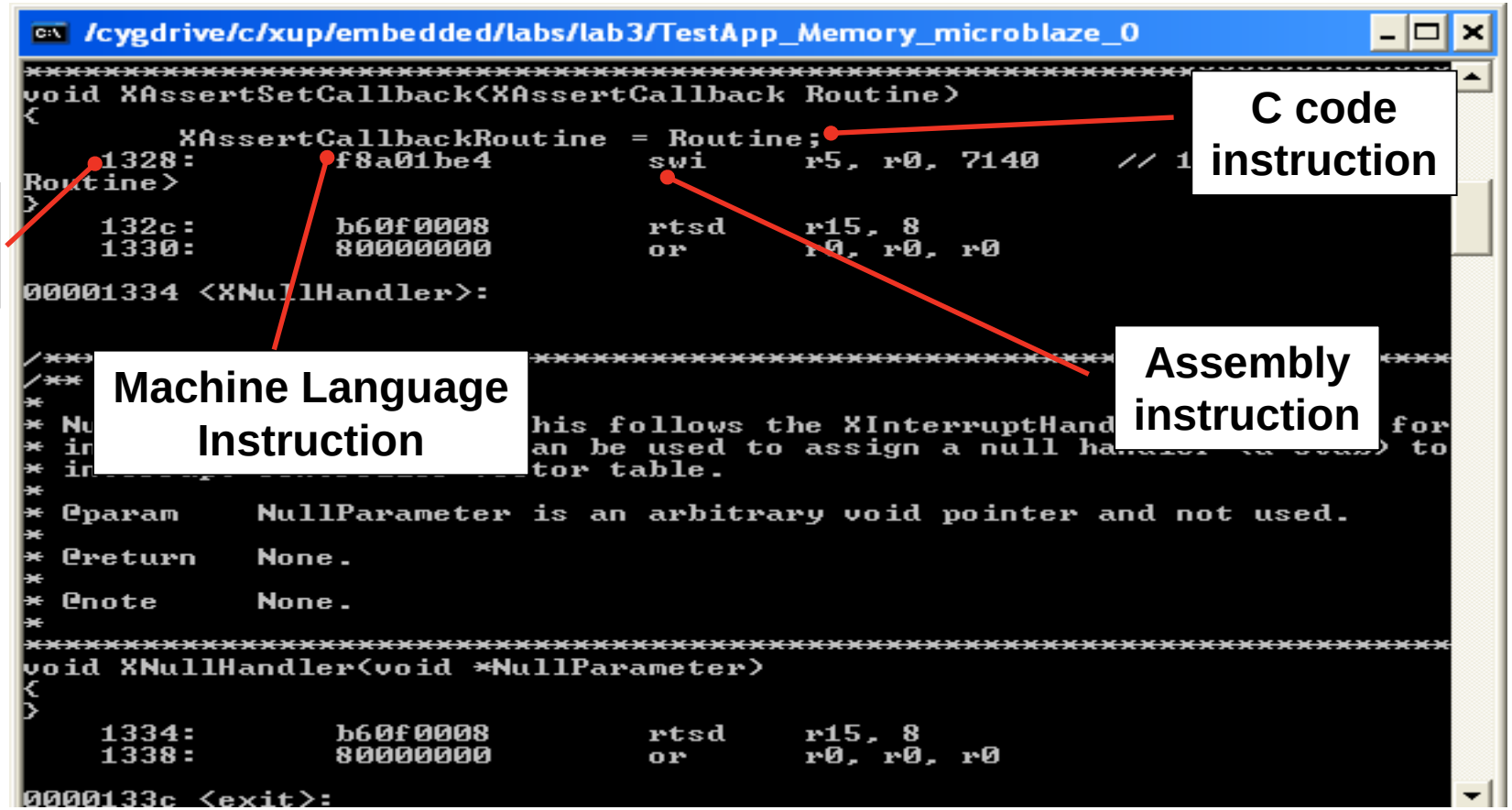
Loadable Memory Address

Offset from the beginning of the section header table

Byte alignment

MicroBlaze Object Dump
Dumping the source and assembly code

mb-objdump –S executable.elf

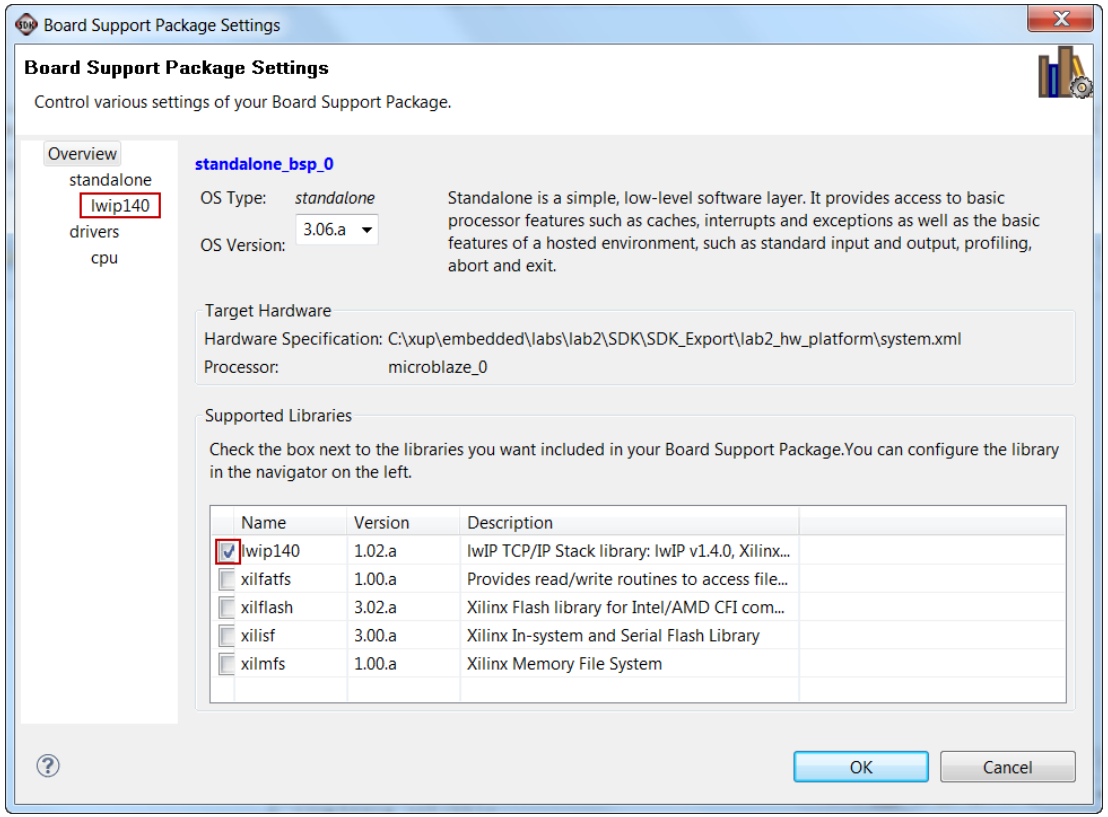


Outline

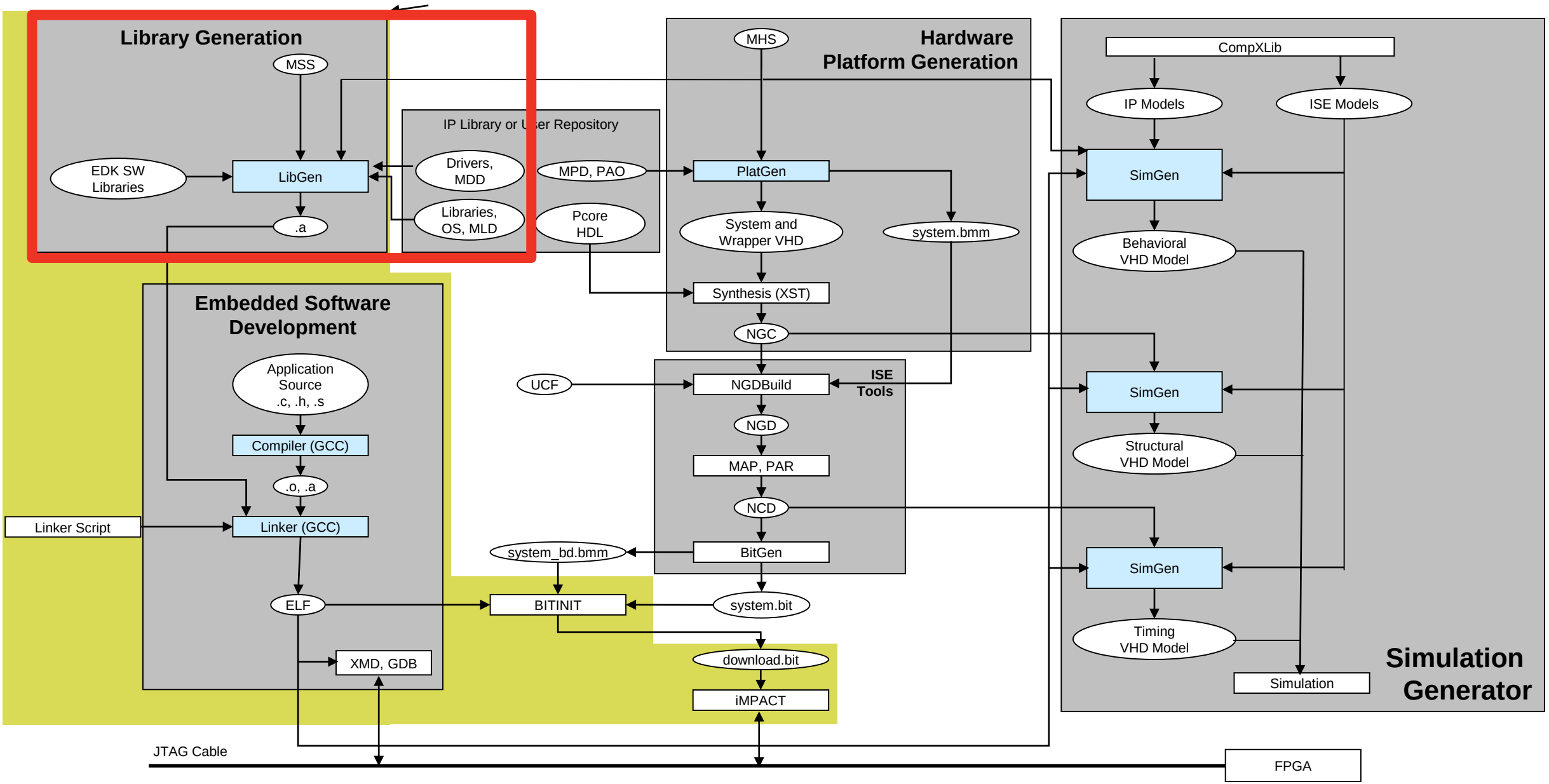
- Introduction
- SDK Development Environment
- SDK Project Creation
- GNU Development Tools: GCC, AS, LD, Binutils
- ***Software Settings***
  - *Software Platform Settings*
  - Compiler Settings
- Summary

Accessing Software Platform Properties

- Select the created board support package in the Project Explorer view
- Xilinx Tools > Board Support Package Settings
- Sets all of the software BSP related options in the design
- Has multiple forms selection
  - Overview
  - Standalone
  - Drivers
  - CPU
- As individual Standalone services are selected a configurable menu selection item will appear



EDK Tool Flow (SDK)



Library Generation Flow (in SDK)

- **Library Generator – LibGen**
  - Input files → MSS
  - Output files → libc.a, libXil.a, libm.a

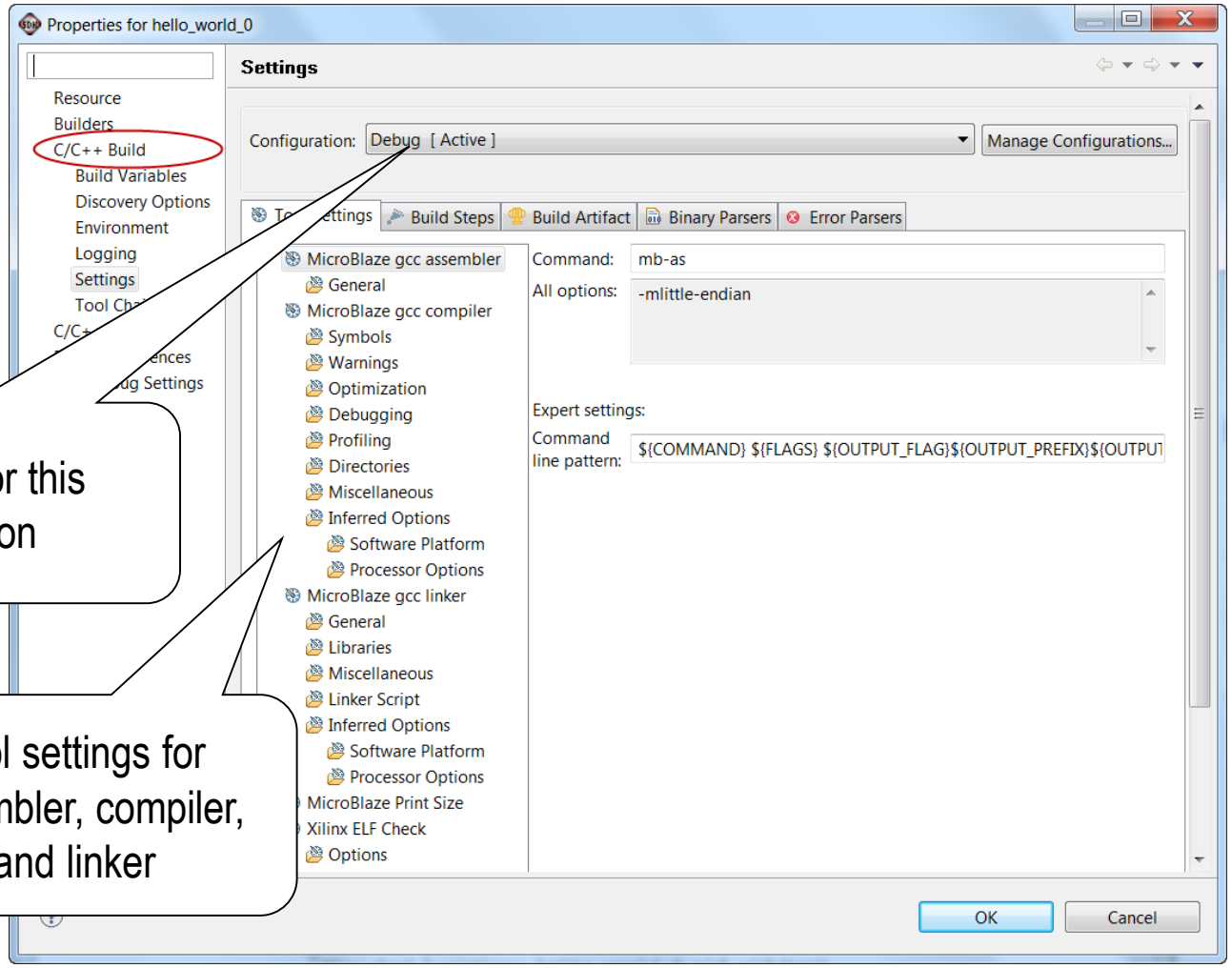
  - LibGen is generally the first tool run to configure libraries and device drivers
    - The MSS file defines the drivers associated with peripherals, standard input/output devices, and other related software features
  - LibGen configures libraries and drivers with this information and produces an archive of object files:
    - libc.a - Standard C library
    - libXil.a - Xilinx library
    - libm.a - Math functions library

Outline

- Introduction
- SDK Development Environment
- SDK Project Creation
- GNU Development Tools: GCC, AS, LD, Binutils
- ***Software Settings***
  - Software Platform Settings
  - *Compiler Settings*
- Summary

C/C++ Build Settings

- Right-click the top level of an application project and select C/C++ Build Settings
- Most-accessed properties are in the C/C++ Build panel Settings tab
- Each configuration has its own properties

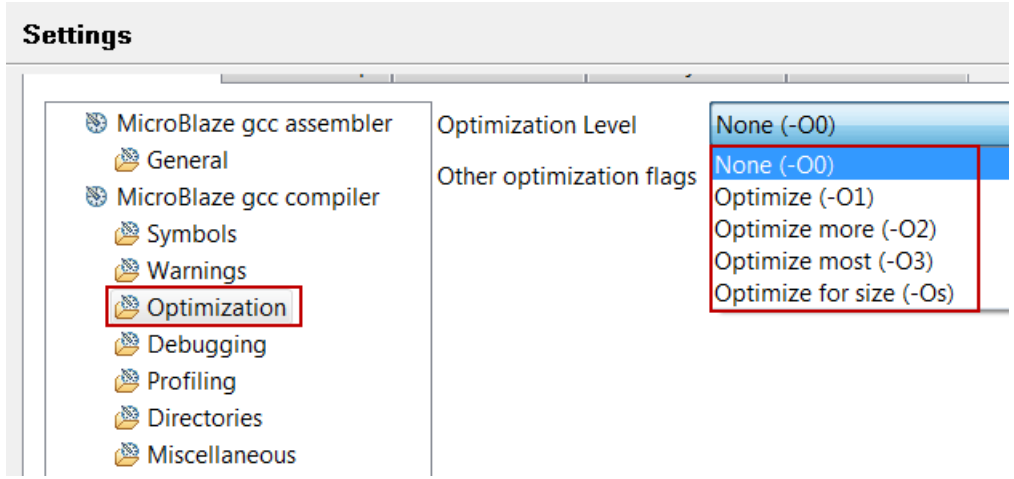


Set options for this configuration

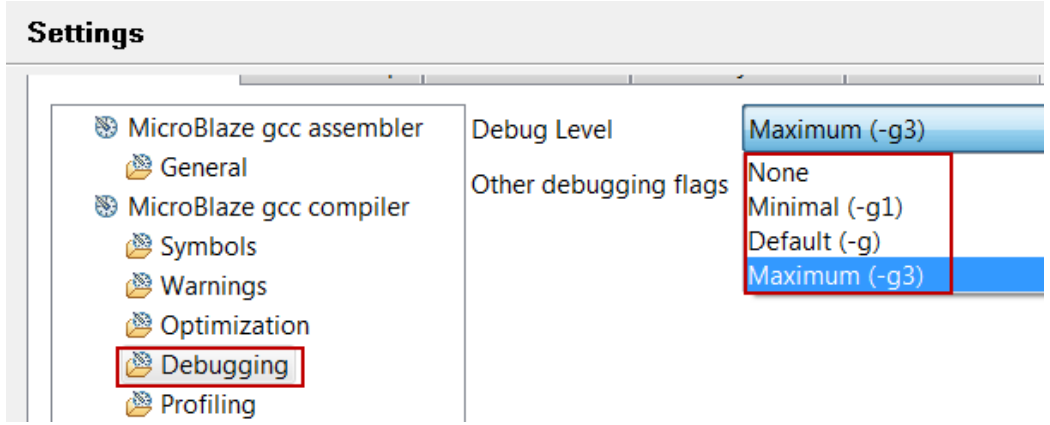
Tool settings for assembler, compiler, and linker

Debug/Optimization Properties

- **Compiler optimization level**
  - None
  - Low
  - Medium
  - High
  - Size Optimized



- **Enable debug symbols in executable**
  - Necessary for debugging
  - Set optimization level to none if possible

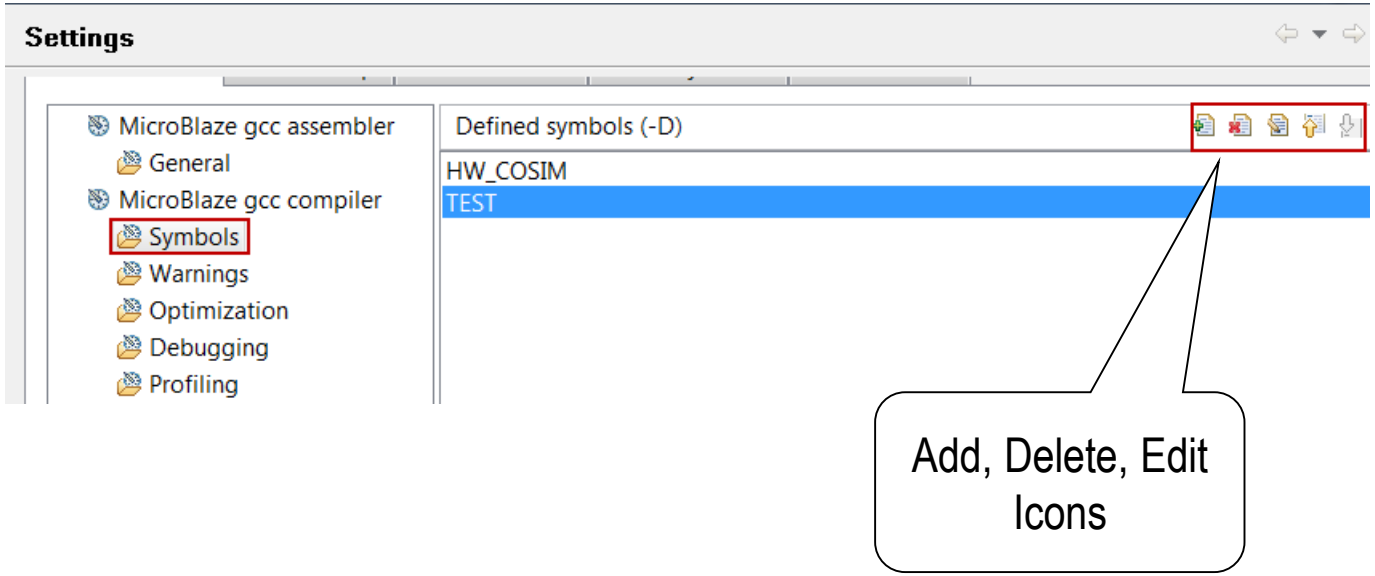


Miscellaneous Compiler Properties

- **Define symbols for conditional compiling**
  - Add
  - Delete
  - Edit
- **References C source**

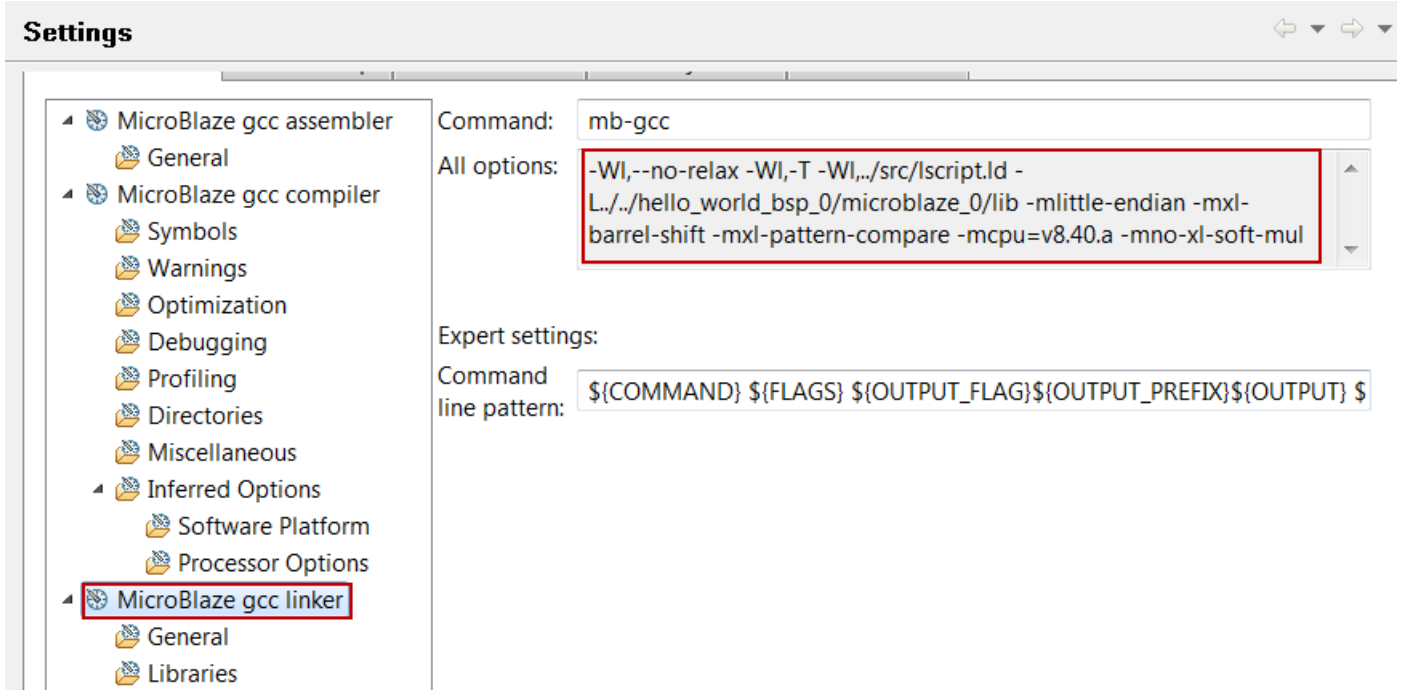
```
#ifdef symbol
conditional statements
#endif
```

- **Passed to compiler as *–D* option**
- **Other compiler options are available**



Linker Properties

- The Root panel displays properties for the selected configuration
- Shown are the linker options for the Debug configuration
- Default settings are fine for simple applications



Outline

- Introduction
- SDK Development Environment
- SDK Project Creation
- GNU Development Tools: GCC, AS, LD, Binutils
- Software Settings
  - Software Platform Settings
  - Compiler Settings
- *Summary*

Summary

- Software development for an embedded system in FPGA imposes unique challenges due to unique hardware platform
- SDK provides many rich perspectives which enable ease of accessing information through related views
- GNU tools are used for compiling C/C++ source files, linking, creating executable output, and debugging
- Software platform settings allow inclusion of software library support
- Compiler settings provide switches including compiling, linking, debugging, and profiling