AMD
XILINX

Software Development and Debug

2021.2

Objectives

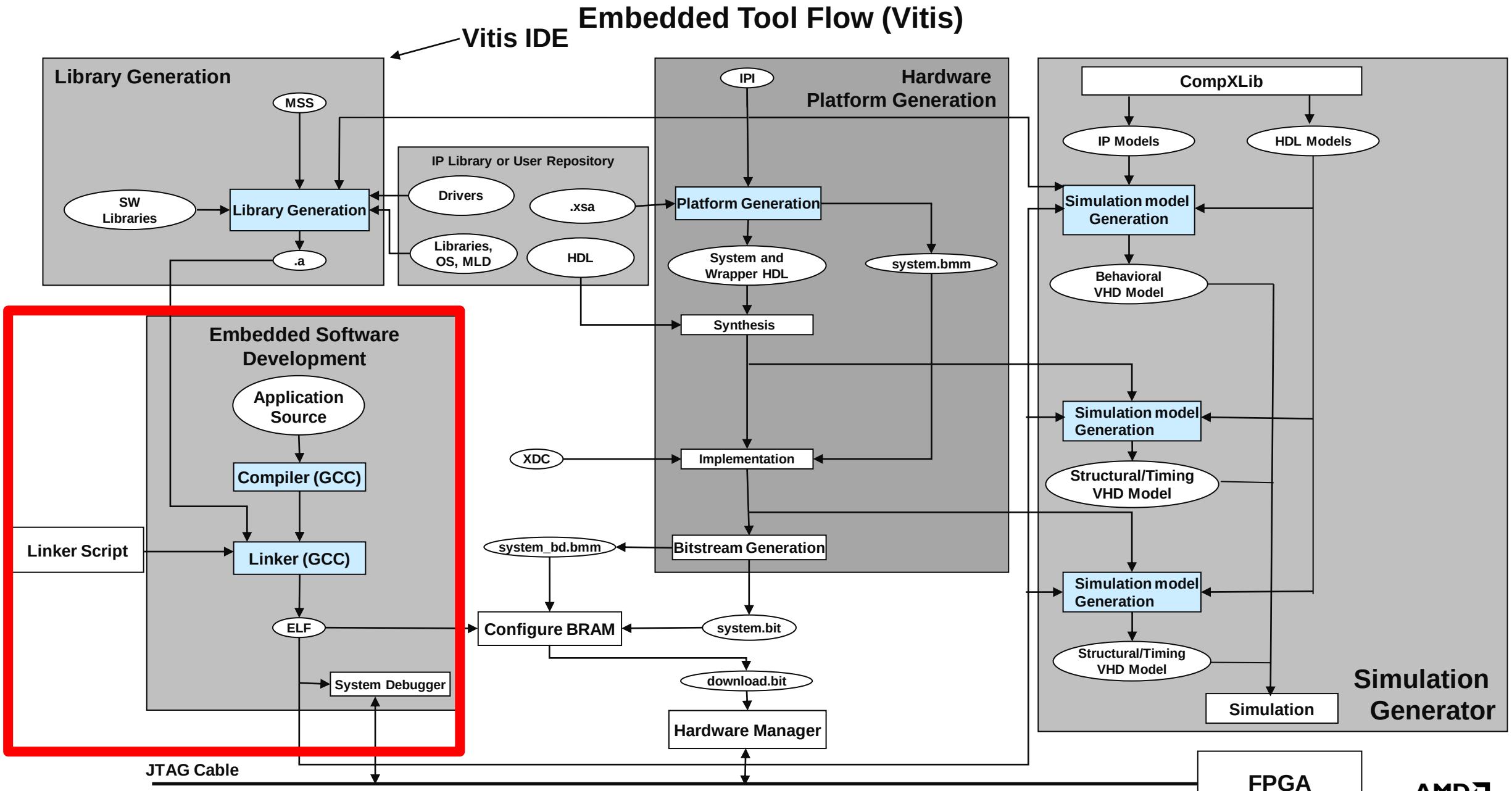
- **After completing this module, you will be able to:**
  - Identify the GNU tools functionality
  - State when address management is needed
  - Describe the object file sections
  - Describe what a linker script does
  - Describe Debug functionality and Eclipse Target Communications Framework

Outline

- GNU Development Tools: GCC, AS, LD, Binutils
- Address Management
- Software Settings
  - *Software Platform Settings*
  - *Compiler Settings*
  - *Linker Script*
- Software Debug in Vitis
- Summary

GNU Development Tools: GCC, AS, LD, Binutils

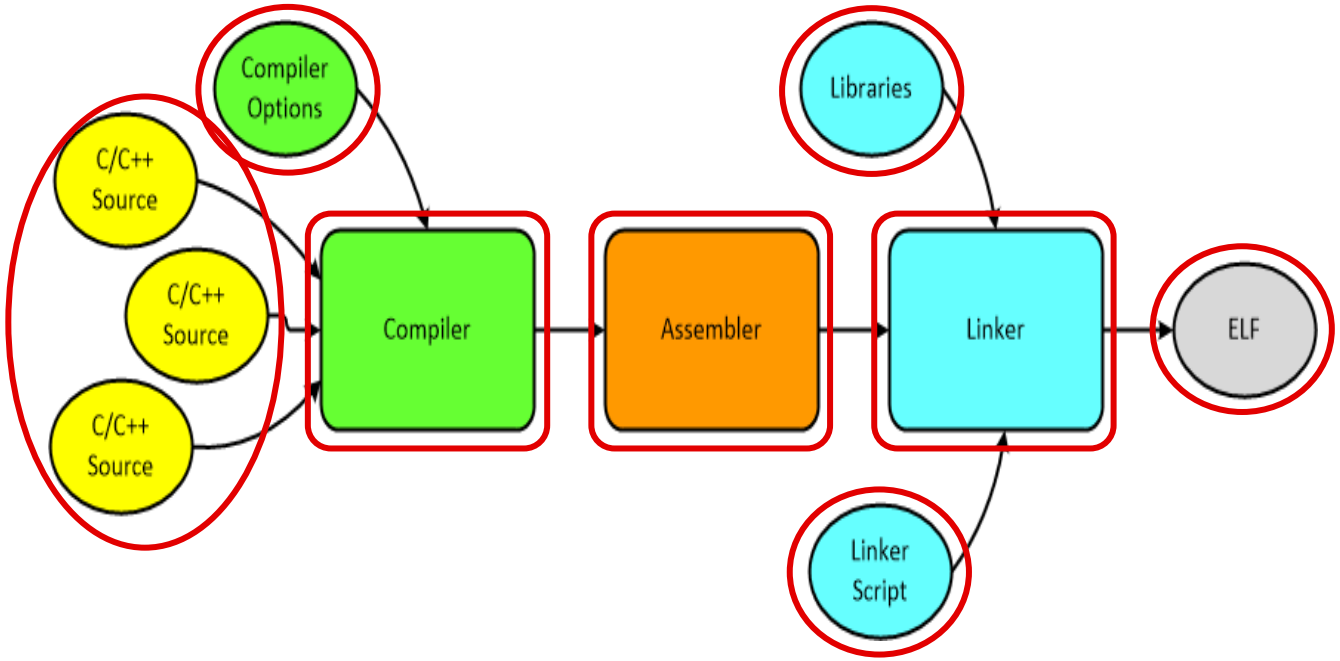
Embedded Tool Flow (Vitis)



AMD XILINX

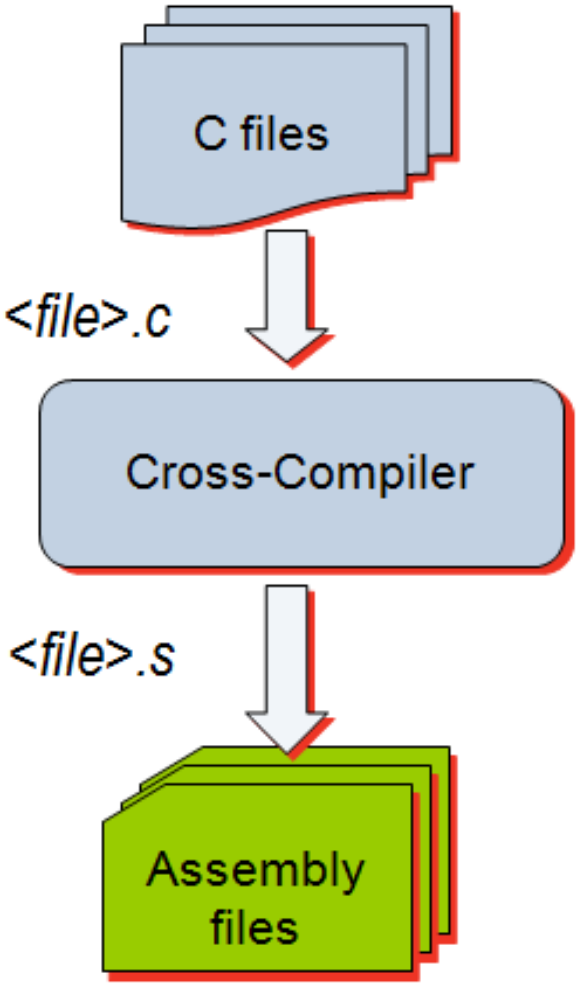
Controlling Compilation

- Eclipse development tool does not contain a toolchain
- Vitis IDE includes the GNU toolchain
- User can replace toolchain
- All toolchains have a well-defined path through the tools
- All toolchains require certain files for each stage



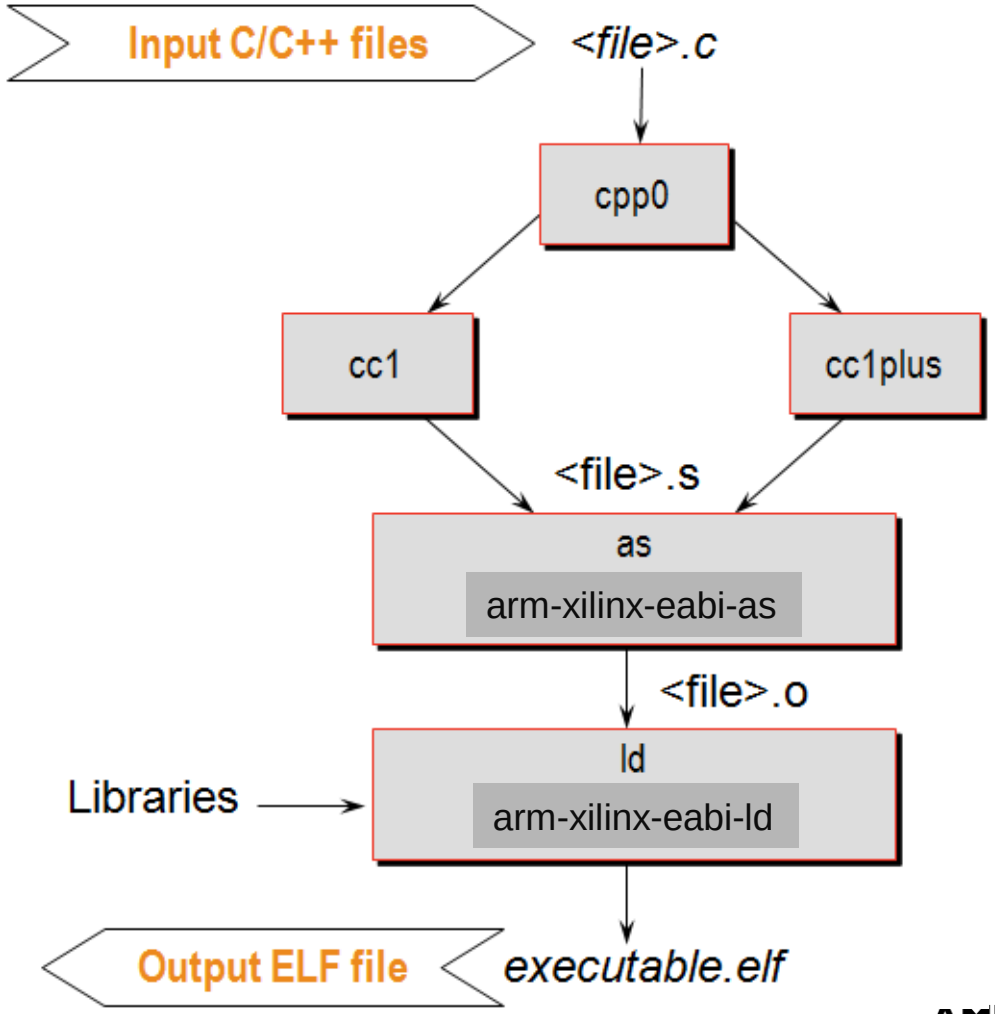
GNU Tools: GCC

- 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 (arm-xilinx-eabi-gcc)
- Command line only; uses the settings set through the GUI



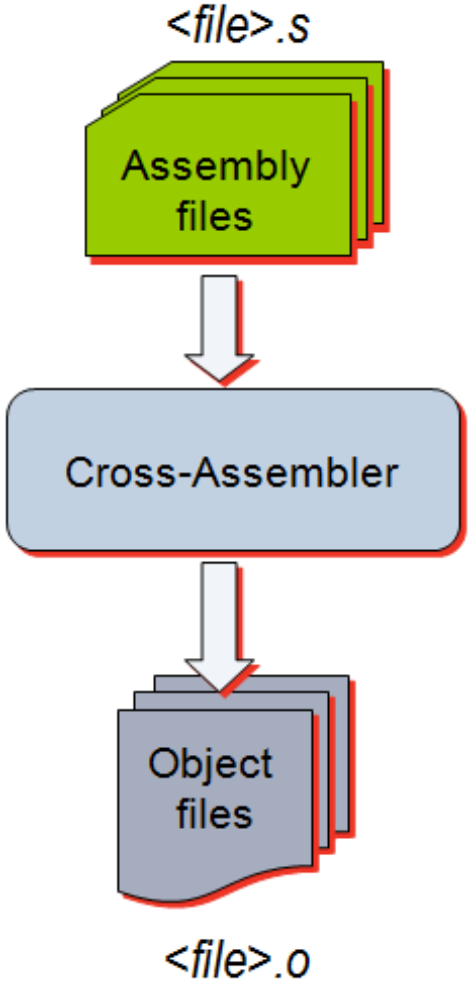
GNU Tools: GCC

- 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
    - arm-xilinx-eabi-as
  - Linker
    - arm-xilinx-eabi-ld



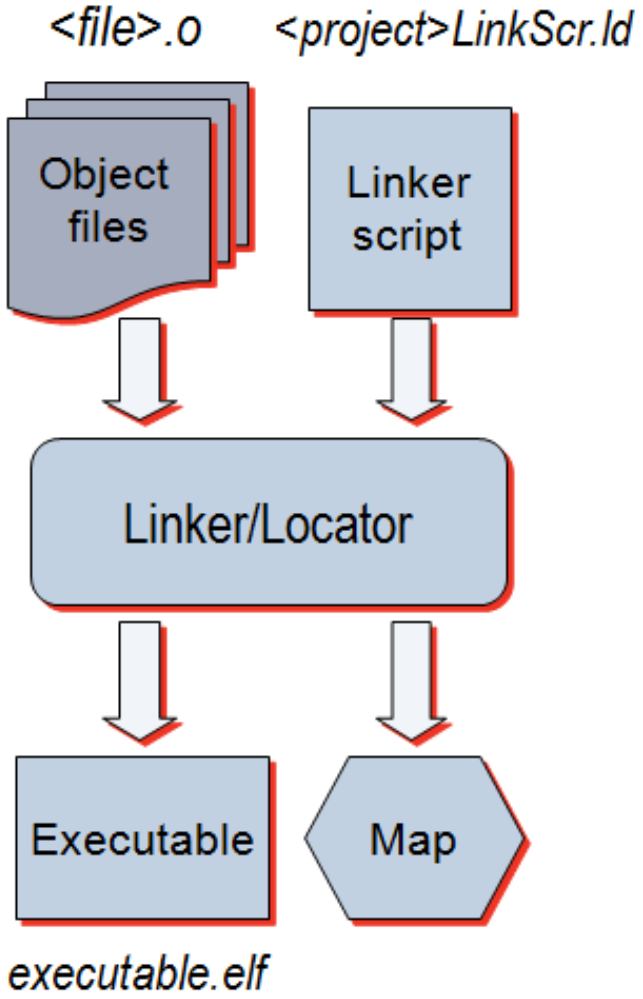
GNU Tools: AS

- 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

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



Object File Sections

- What is an object file?
  - An object file is an assembled piece of code
    - Machine language:
      li r31,0 = 0x3BE0 0000
  - Constant data
  - There may be references to external objects that are defined elsewhere
  - This file may contain debugging information

Object File Sections

Sectional layout of an object or an executable file

.text	Text section
.rodata	Read-only data section
.sdata2	Small read-only data section (less than eight bytes)
.sbss2	Small read-only uninitialized data section
.data	Read-write data section
.sdata	Small read-write data section
.sbss	Small uninitialized data section
.bss	Uninitialized data section

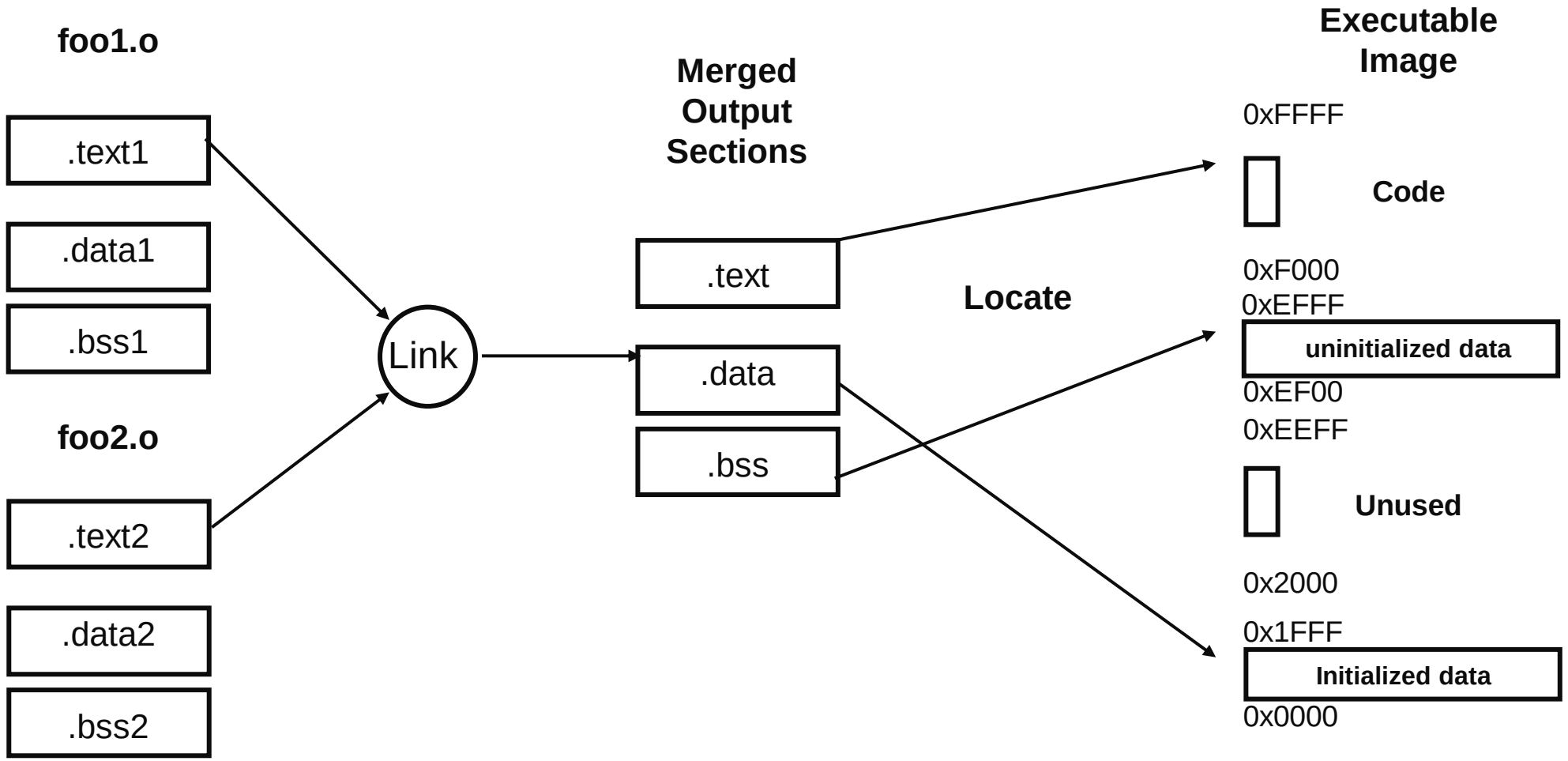
Sections Example

```
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 */
...

}
```

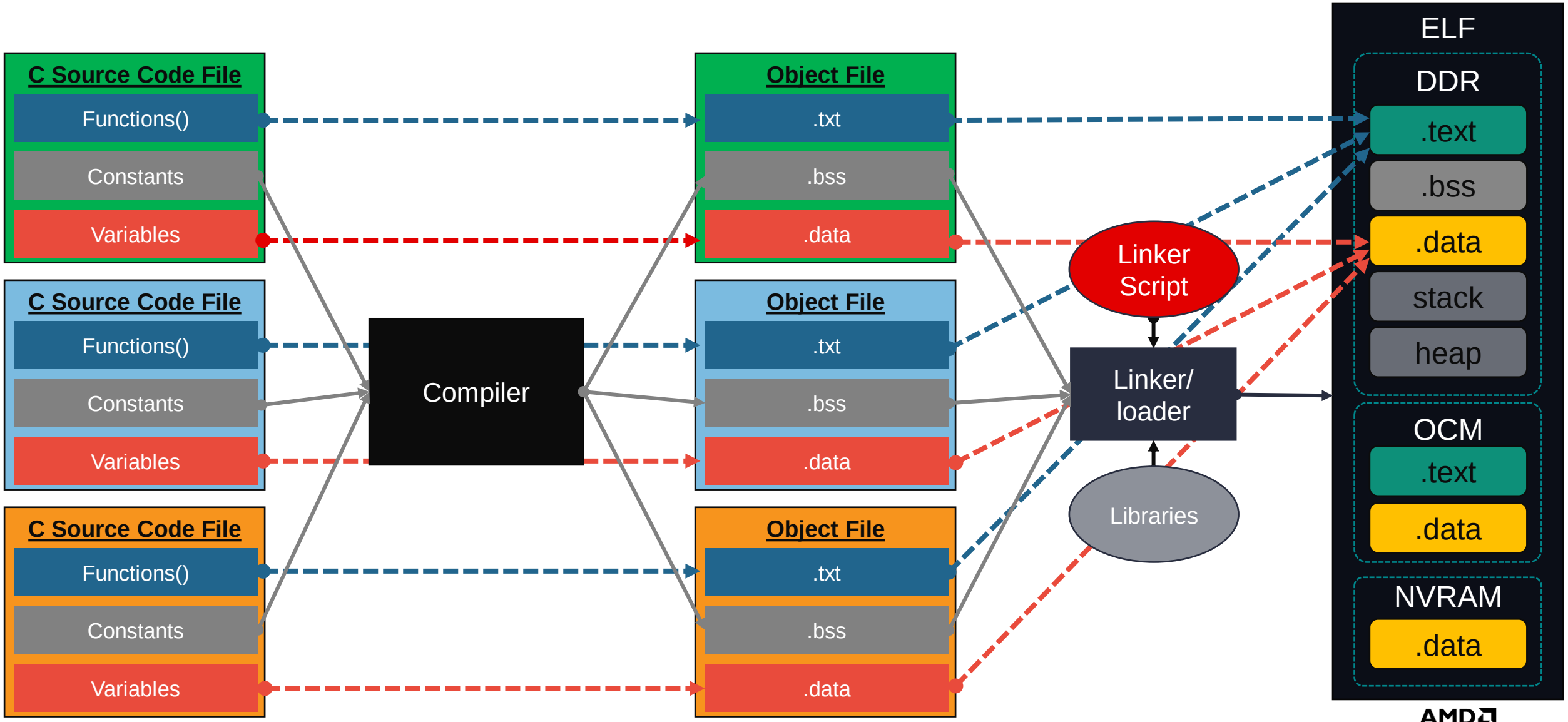
Linker Script

- **Linker script controls the linking process**
  - Map the code and data to a specified memory space
  - Set the entry point to the executable
  - Reserve space for the stack
- **Required if the design contains a discontinuous memory space**

Linker and Locator Flows



Linker and Locator Flows



AMD XILINX

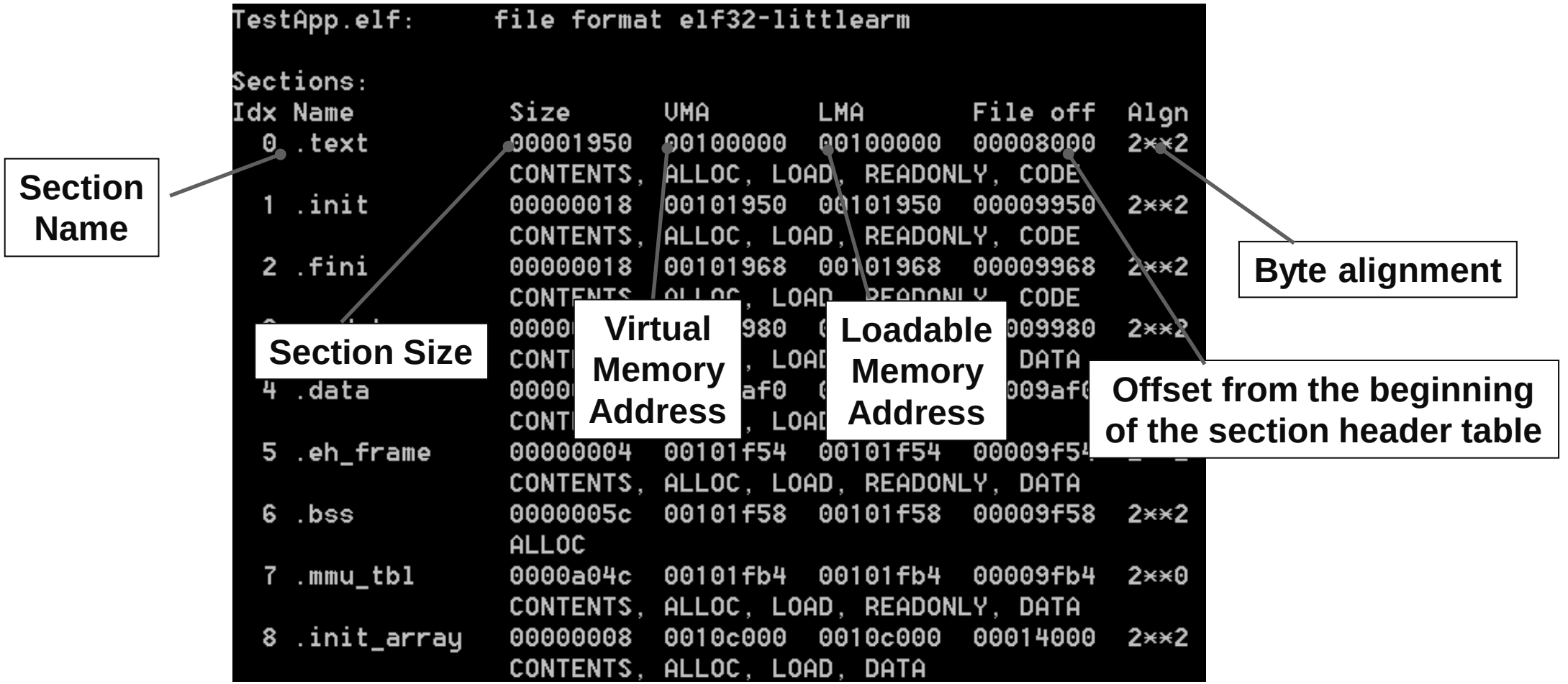
GNU Utilities

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

Object Dump

Display summary information from the section headers

arm-xilinx-eabi-objdump –h executable.elf

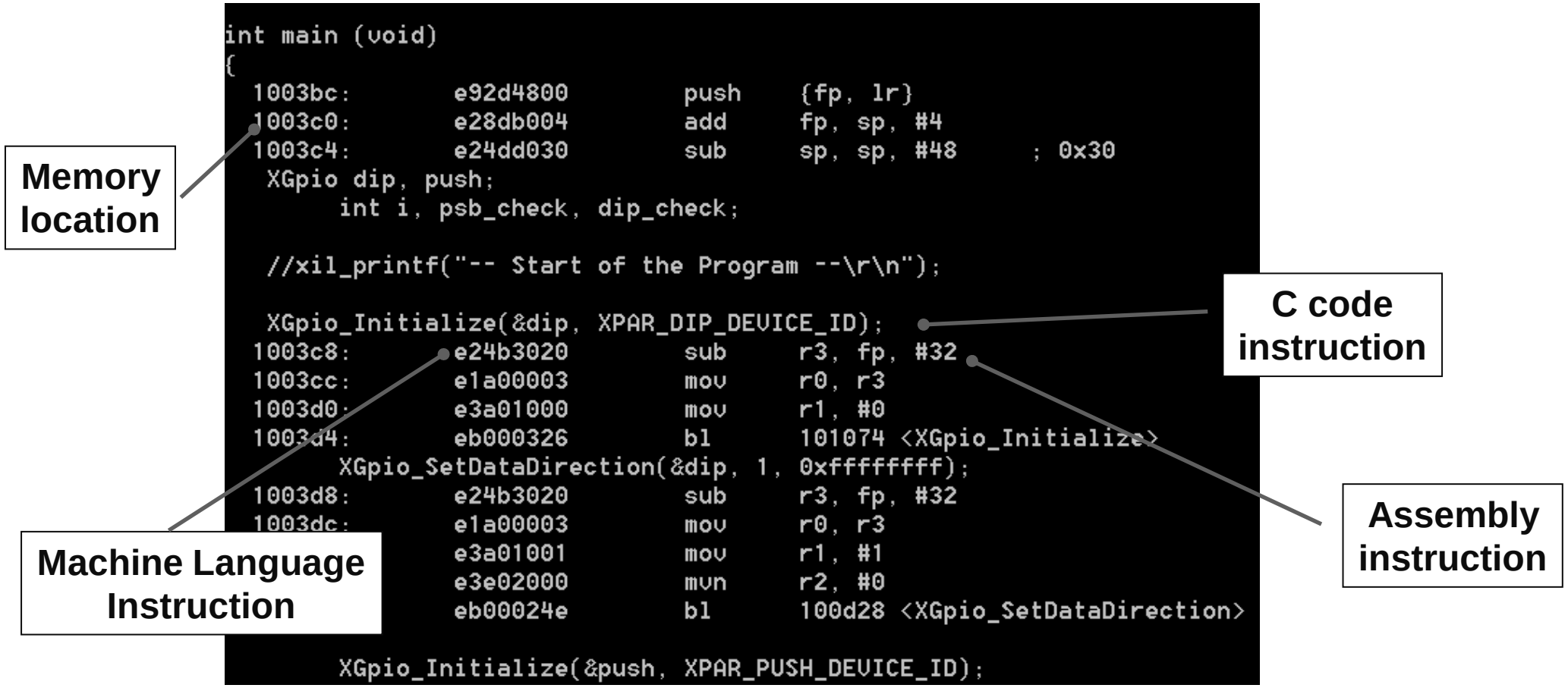


AMD XILINX

Object Dump

Dumping the source and assembly code

arm-xilinx-eabi-objdump –S executable.elf



Address Management

Address Management

- Embedded processor design requires you to manage the following:
  - Address map for the peripherals
  - Location of the application code in the memory space
    - Block RAM
    - External memory (Flash, DDR3, SRAM)
- Memory requirements for your programs are based on the following:
  - The amount of memory required for storing the instructions
  - The amount of memory required for storing the data associated with the program

Standard ARM Programming Model

- Processing system and programmable logic look the same
  - AMBA® and AXI interfaces
  - Memory-mapped I/O
  - Register access
- Consistency for PS and PL = ease of use
- Memory map usage: total of 4 GB
  - 1 GB: DDR RAM
  - 2 GB: dedicated to PL peripherals
  - 1 GB: PS peripherals, OCM, external flash

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

Programmer's View of Programmable Logic

- Programmable logic (PL) memory map
  - 2 GB total space
    - 1 GB for each AXI master: GP0, and GP1
  - Accessible from any processing system (PS) master
    - Either Cortex-A9 CPU
    - PS DMA engine
    - PS peripheral DMA engine
      - Ethernet
      - USB
      - SD/SDIO

Custom Peripheral

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

Code Snippet

```
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);
}
```

Software Settings

Minimal Required Services

- C language standard services
  - C language construct services
  - *stdin* and *stdout*
  - Math library
  - *malloc*
- Processor support requires these services
  - Interrupt
  - Cache
  - Language environment support

Operating Systems

- Operating systems are a collection of software routines that comprise a unified and standard set of system services
- The Standalone domain is used when no operating system is desired
  - Provides a minimal amount of processor and library services as previously illustrated
  - Can be considered a minimal, non-standard operating system
  - Installed as a software platform
- Variety of third-party operating systems are available
  - Linux – many flavors
  - RTOS – real-time operating system; also has many flavors; Free RTOS (an option for the Cortex™-A9 processor)
  - XilKernel – provided by Xilinx; small and simple; only for MicroBlaze
- Operating system domains are added and become part of the Vitis Platform

What an Operating System Provides?

- Operating system services
  - GUI support
  - TCP/IP services**
  - Task management
  - Resource management**
  - Familiar programming services and tasks
  - Easy connection to already written applications
  - Ability to reload and change applications
  - Full file system services**

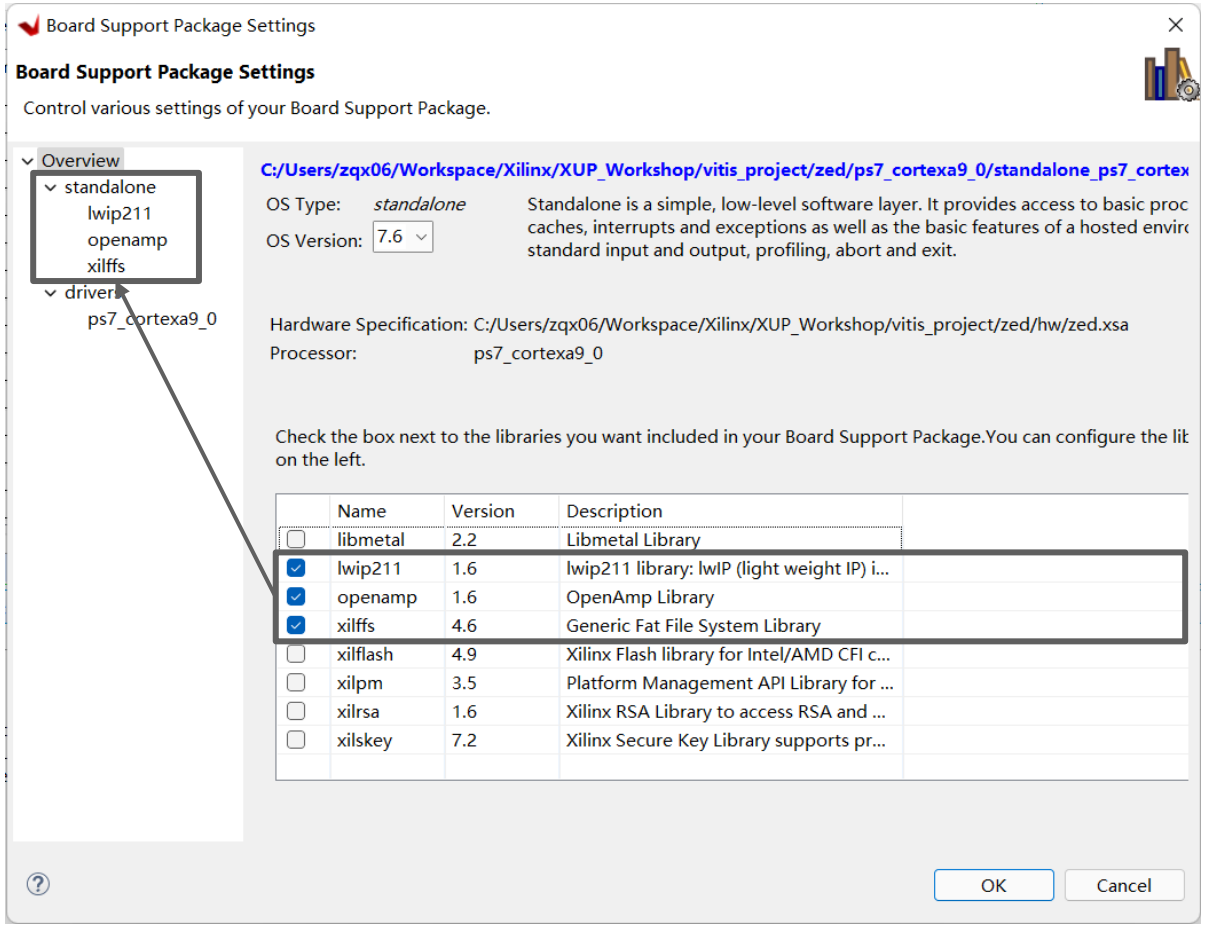
** Also available as additions to the Standalone domain

Do I Need an Operating System?

- **The Standalone domain includes the previously discussed items**
- **Design considerations for systems using the Standalone domain**
  - All services needed are included in the platform
  - The application is static—it never changes
  - The application fits in block RAM (MicroBlaze™ processor), OCM RAM (Zynq™ AP SoC), or DDR memory
  - The application is single-task based
  - Interrupts may or may not be used

Accessing Software Platform Properties

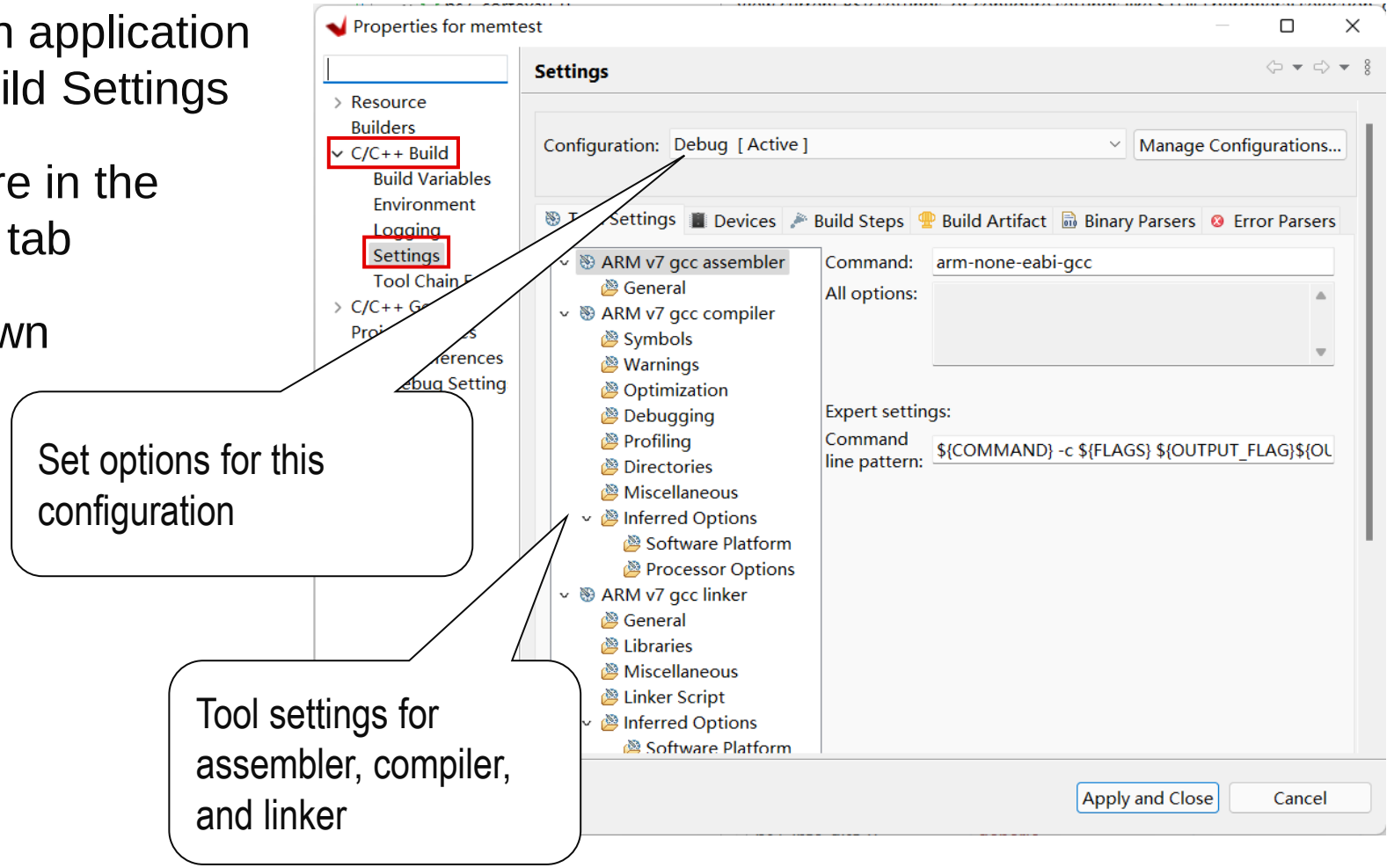
- Double click the platform.spr in the Project Explorer view
- Click on Modify BSP Settings
- Sets all of the software BSP related options in the design
- Has multiple forms selection
  - Overview
  - Standalone
  - Drivers
  - CPU
- As individual Standalone services are selected a configurable menu selection item will appear



AMD XILINX

C/C++ Build Settings

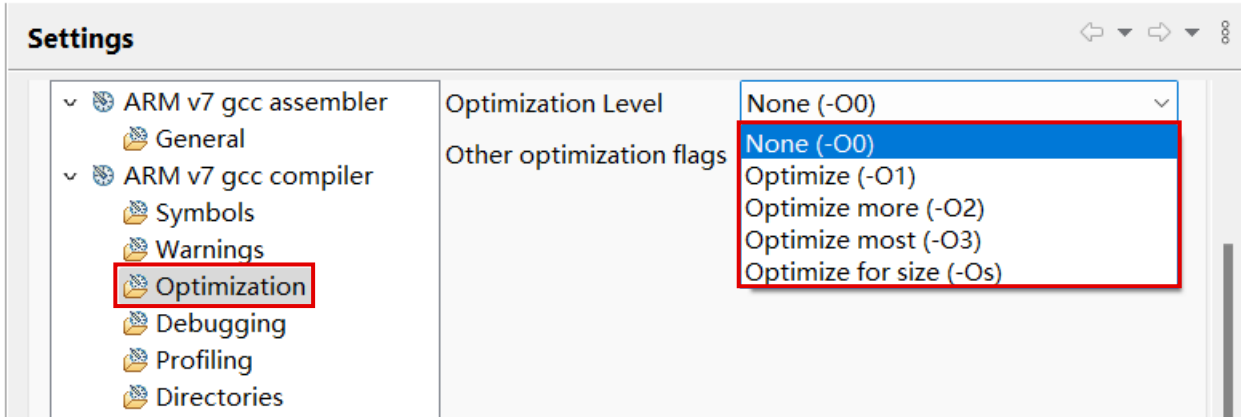
- Right-click the top level of an application project and select C/C++ Build Settings
- Most-accessed properties are in the C/C++ Build panel **Settings** tab
- Each configuration has its own properties

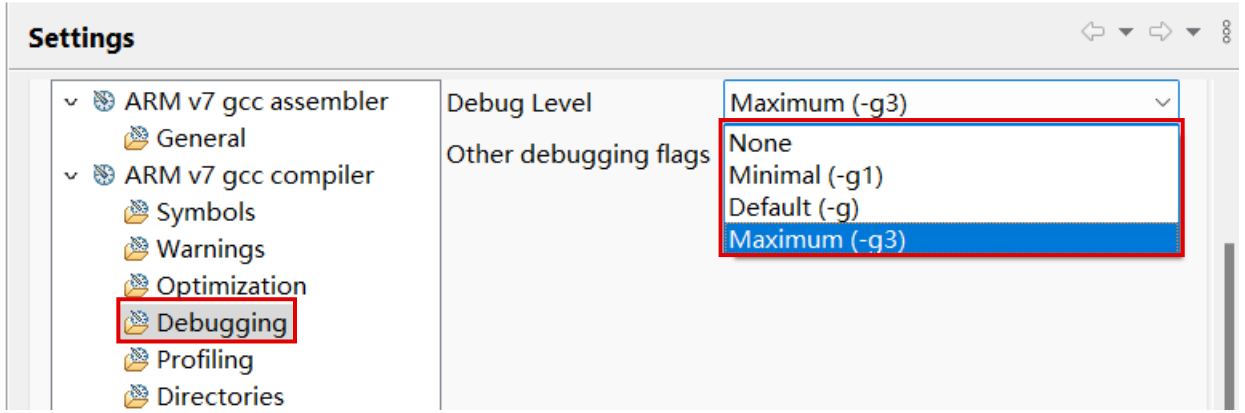


AMD XILINX

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



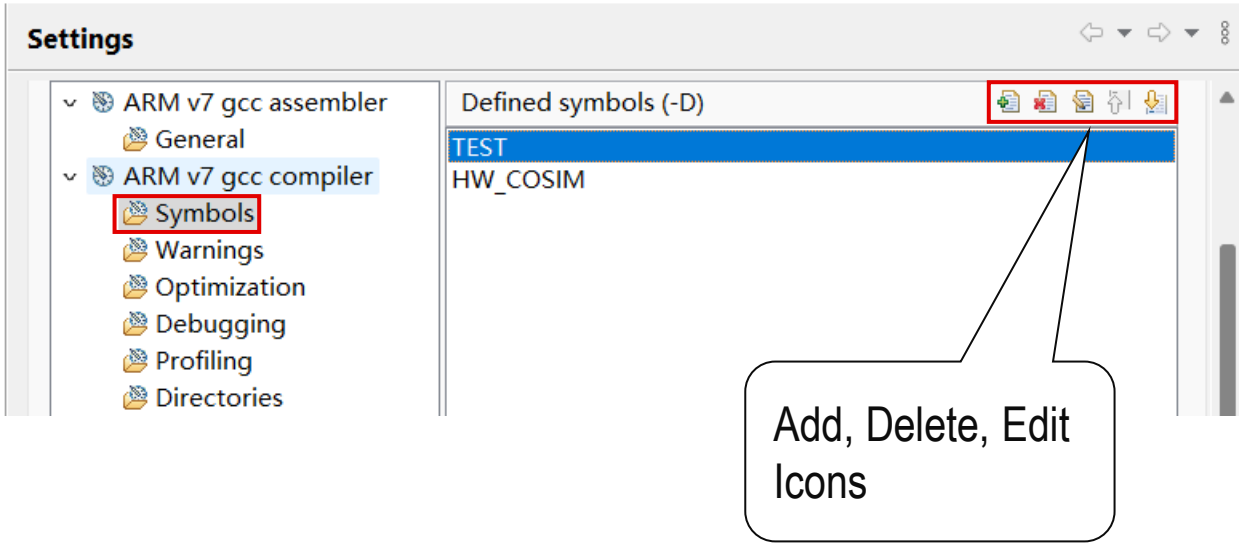


Miscellaneous Compiler Properties

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

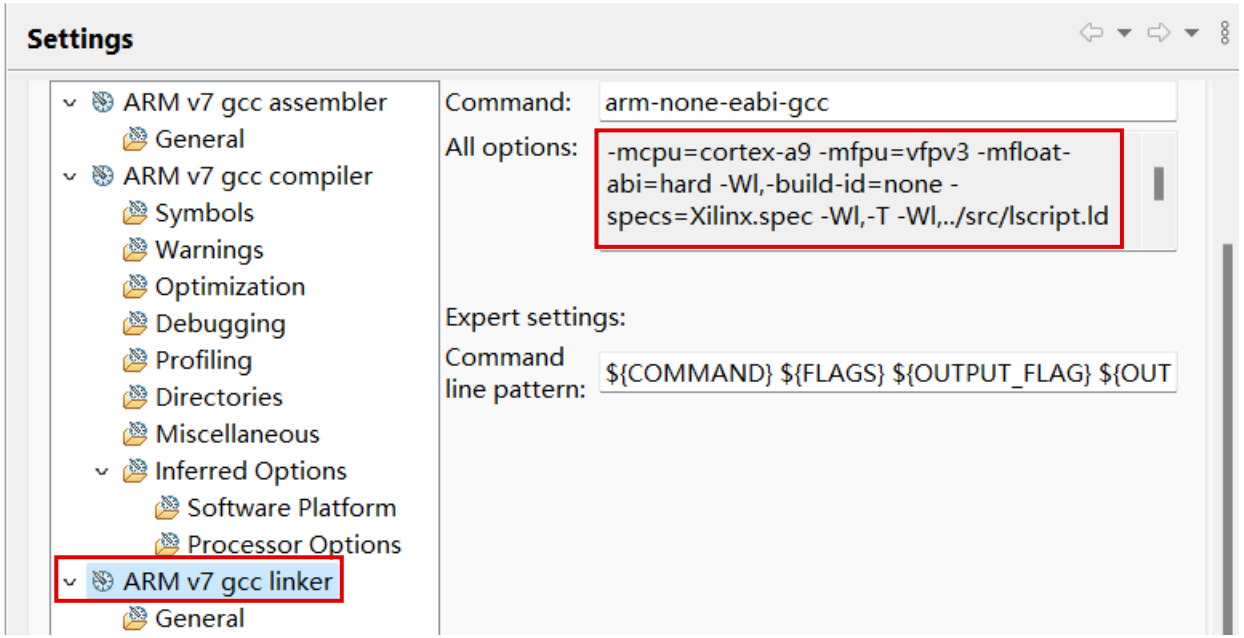
```
#ifdef symbol
conditional statements
#endif
```

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



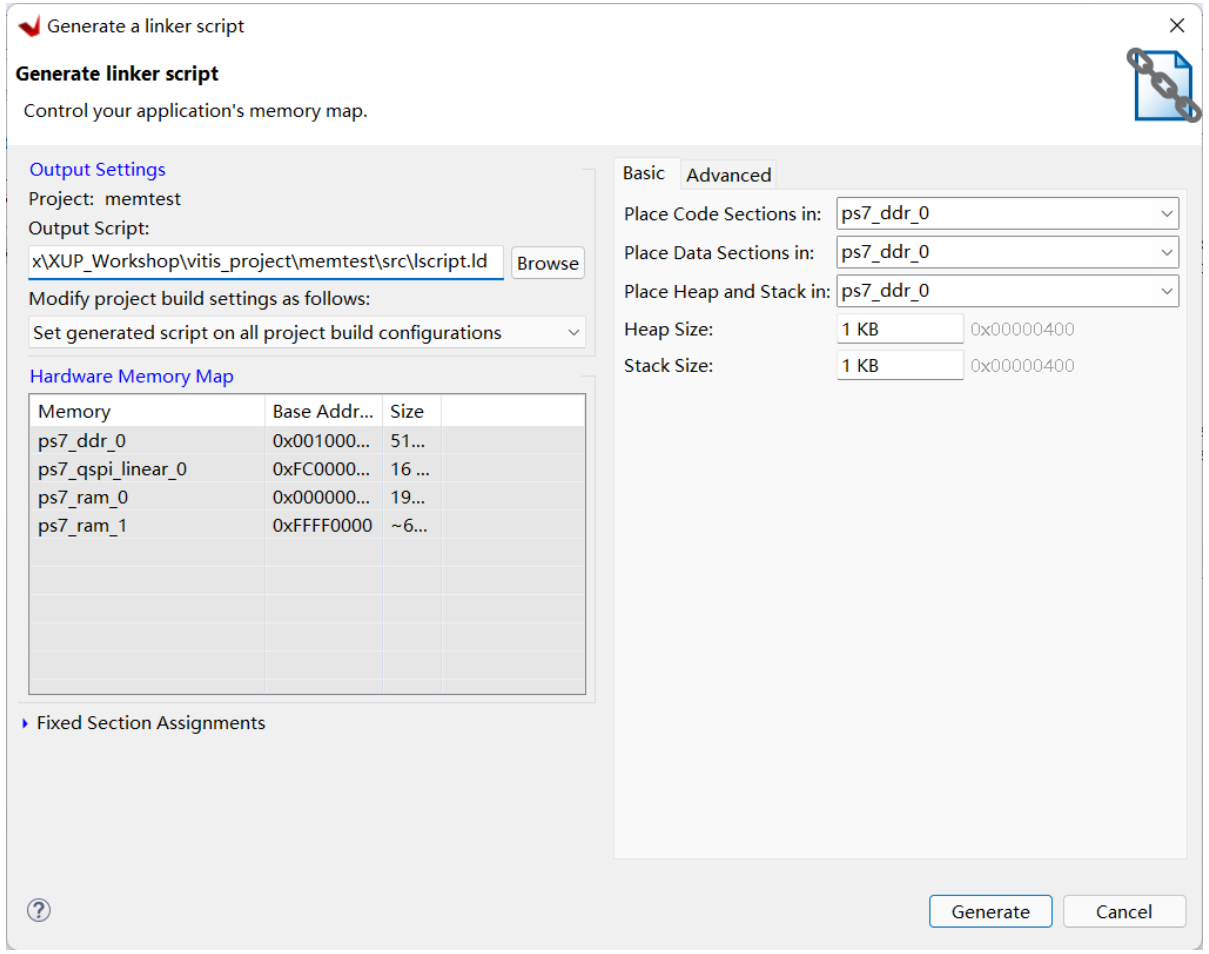
Linker Properties

- 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



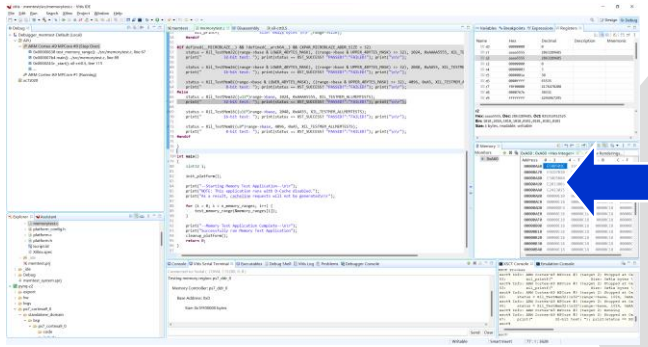
Linker Script Generator GUI

- Table-based GUI allows you to define the memory space for code and data sections
- Launch from **Xilinx > Generate Linker Script**, or from the Explorer view, **right-click on project > Generate Linker Script**
- The tool will create a new linker script (the old script is backed up)

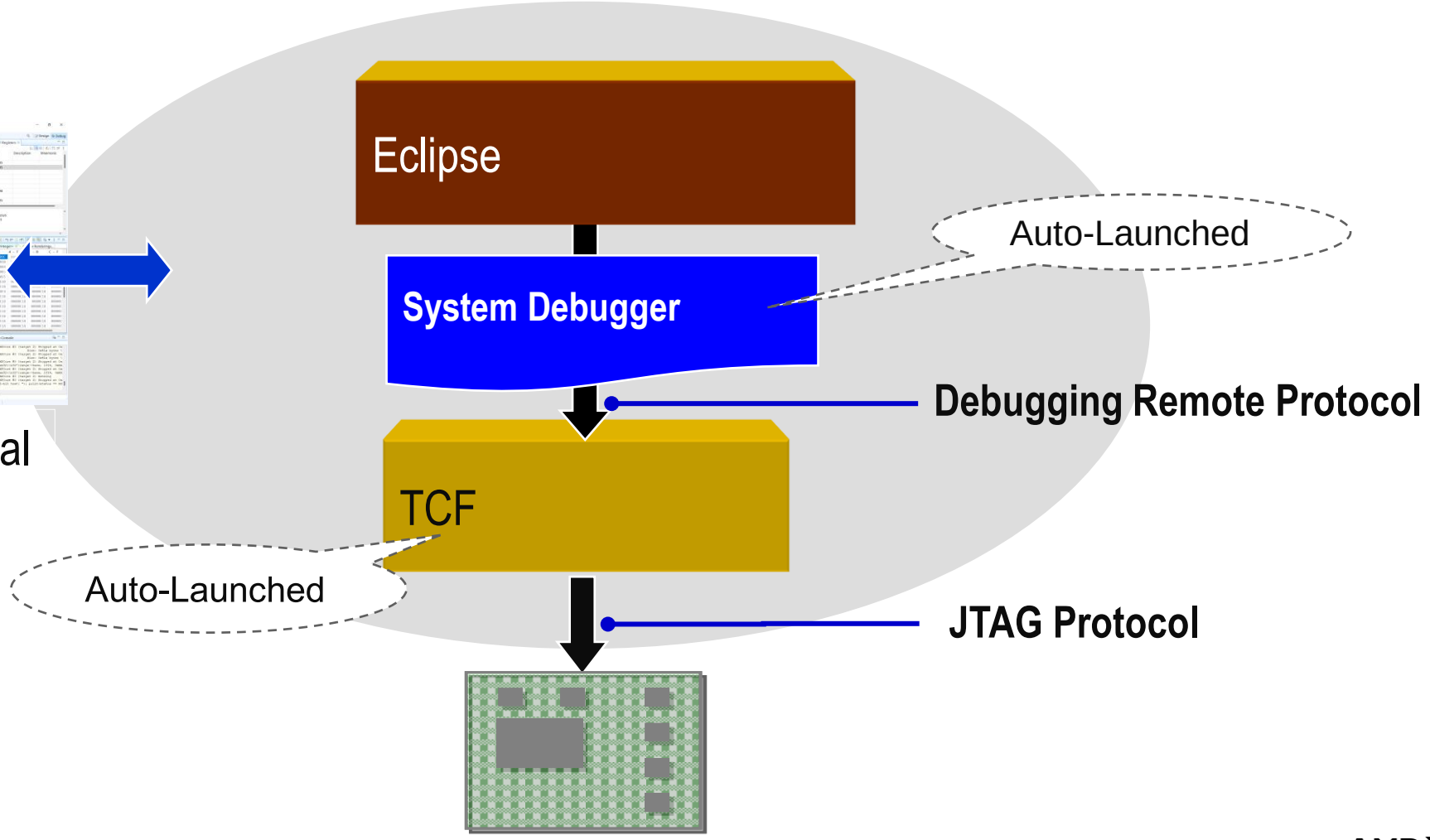


Software Debug in Vitis

Debugging Using Vitis IDE (TCF)



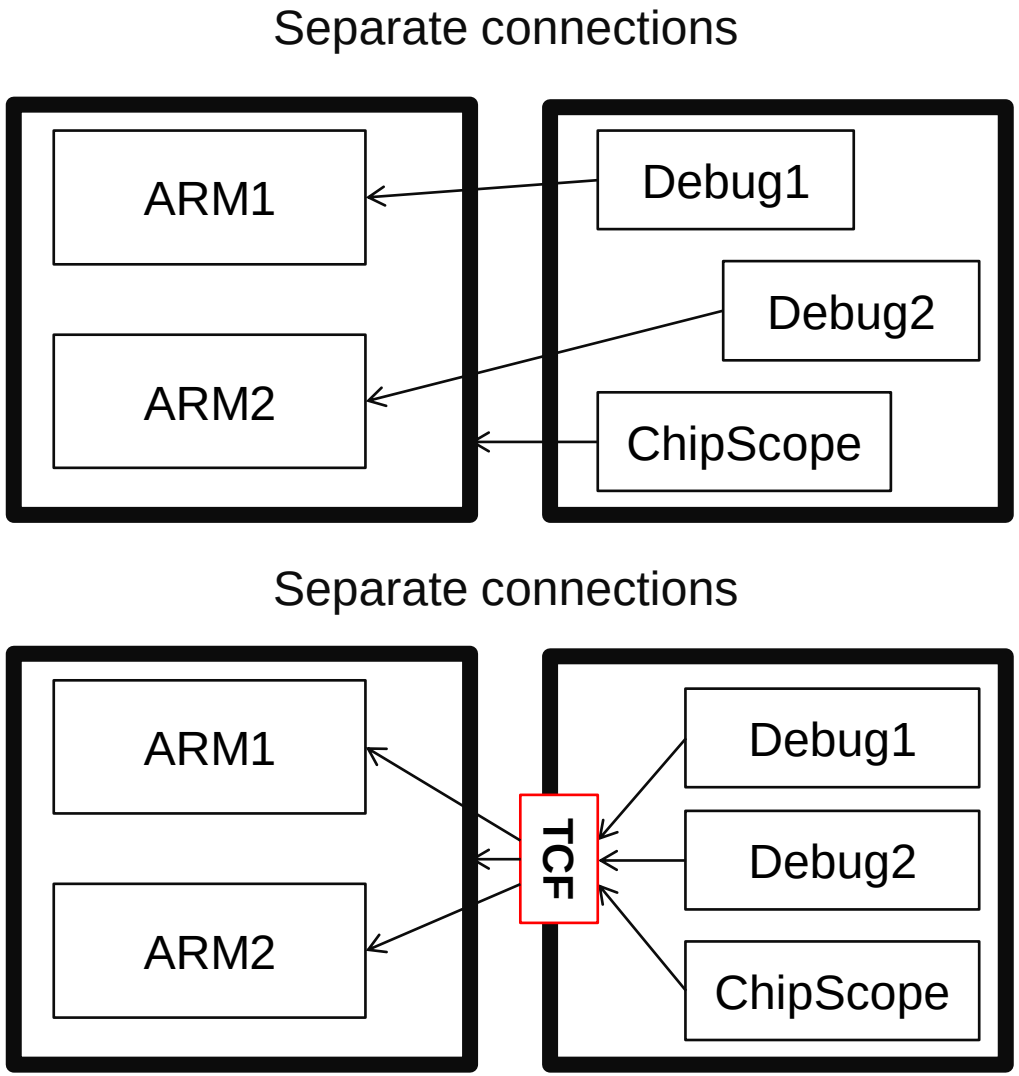
Xilinx Custom Graphical Debug Interface



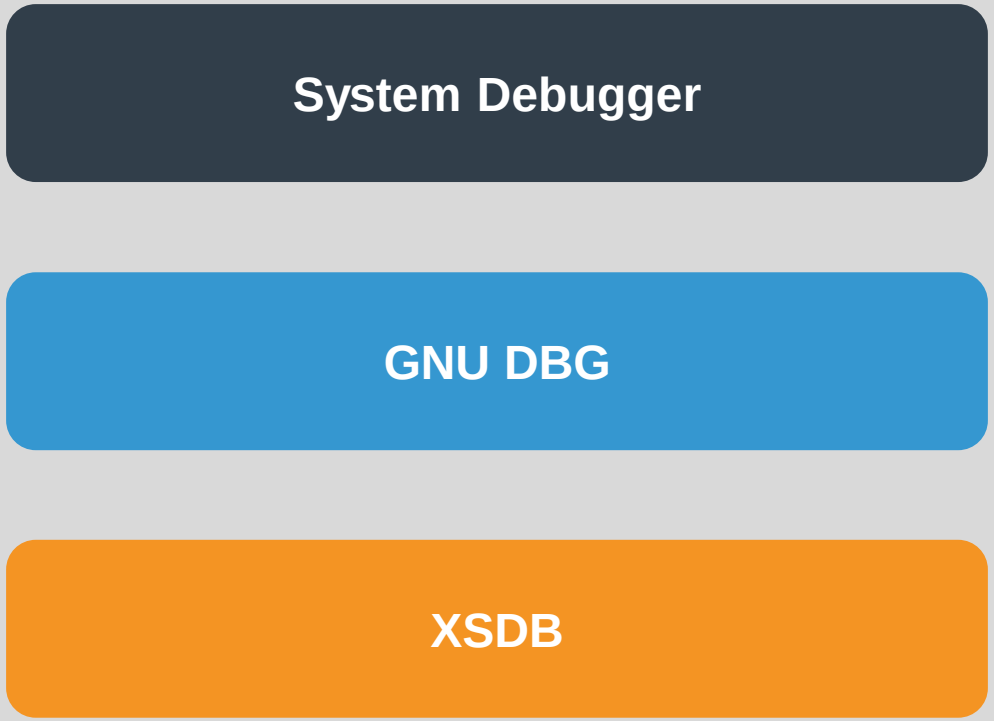
AMD XILINX

Eclipse Target Communication Framework

- Open, extensible network protocol
- Allows services to transparently plug in
- All communication links can share the same protocol
- Transport-agnostic channel abstraction
  - (No specific transport layer. E.g. TCP/IP, Serial Line, SSH tunnel)

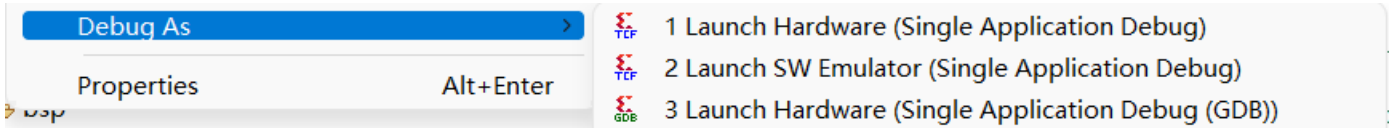


Software Debugging Support



System Debugger

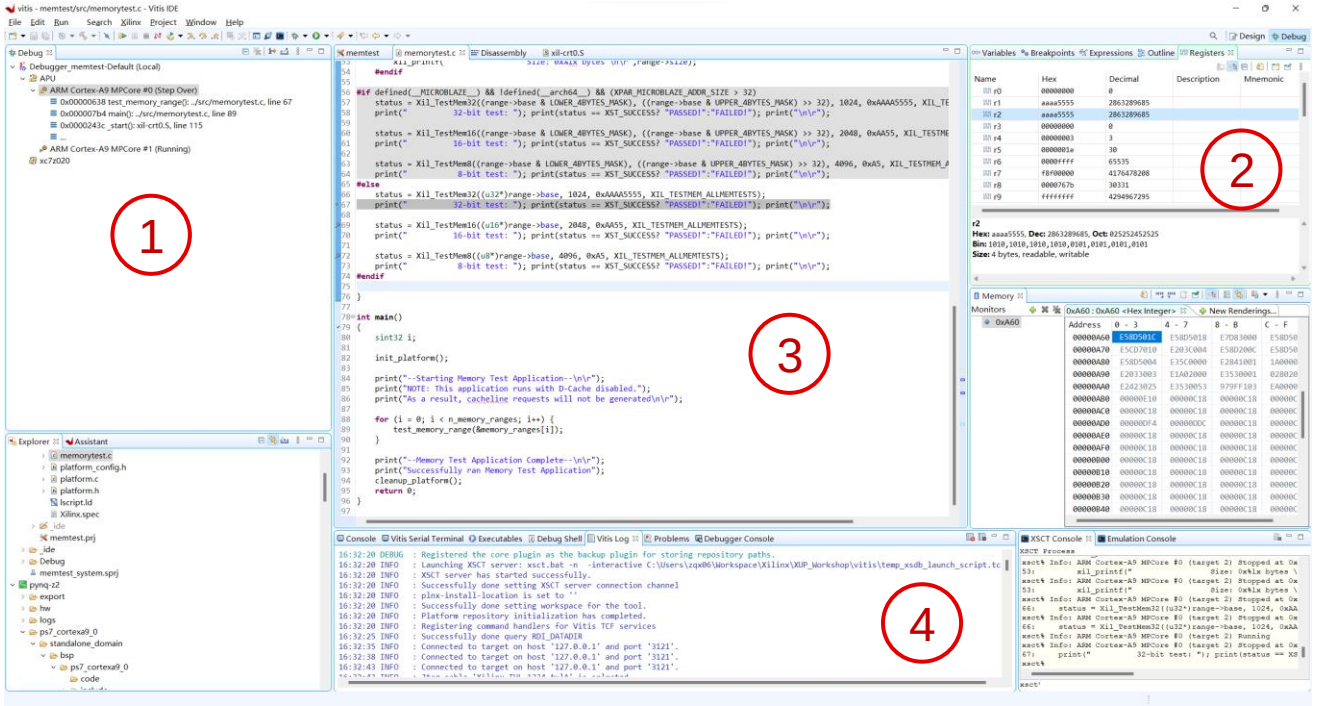
- Eclipse Target Communication Framework
  - Extensible network protocol for communicating with embedded systems
- Single Configuration per target
  - (Not per tool like gdb)



- Homogenous, and heterogeneous, SMP and AMP support
- Neon Support
- True multicore debug through a single JTAG
- Faster than GDB/XSDB

Vitis Debug Perspective

1. Stack frame for target threads
2. Outline, Variables, breakpoints, and registers views
   a. Disassembly view can be added using
   Window > Show View > Disassembly
3. C/C++ editor
4. Console, Vitis Log, and Memory views



AMD XILINX

Debug GUI

- Run-time control

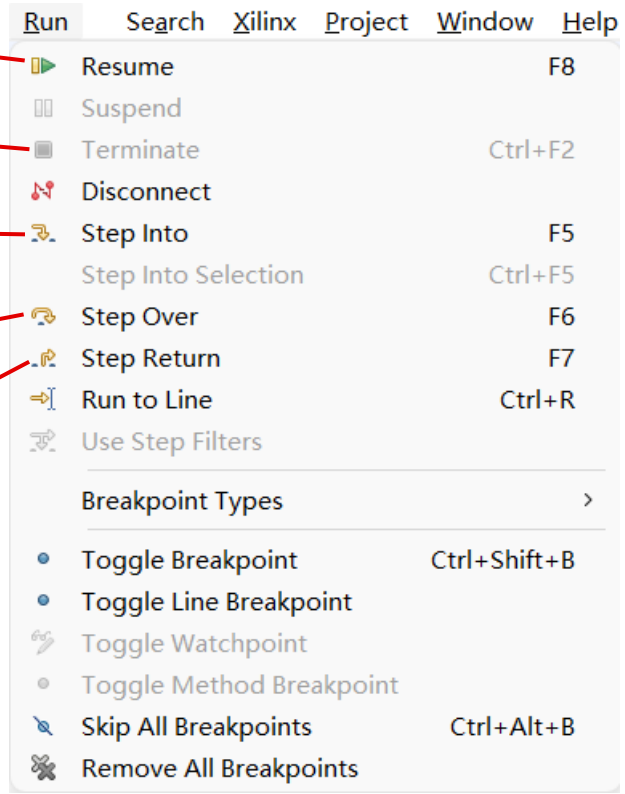
Continue to next breakpoint

Stop debug session

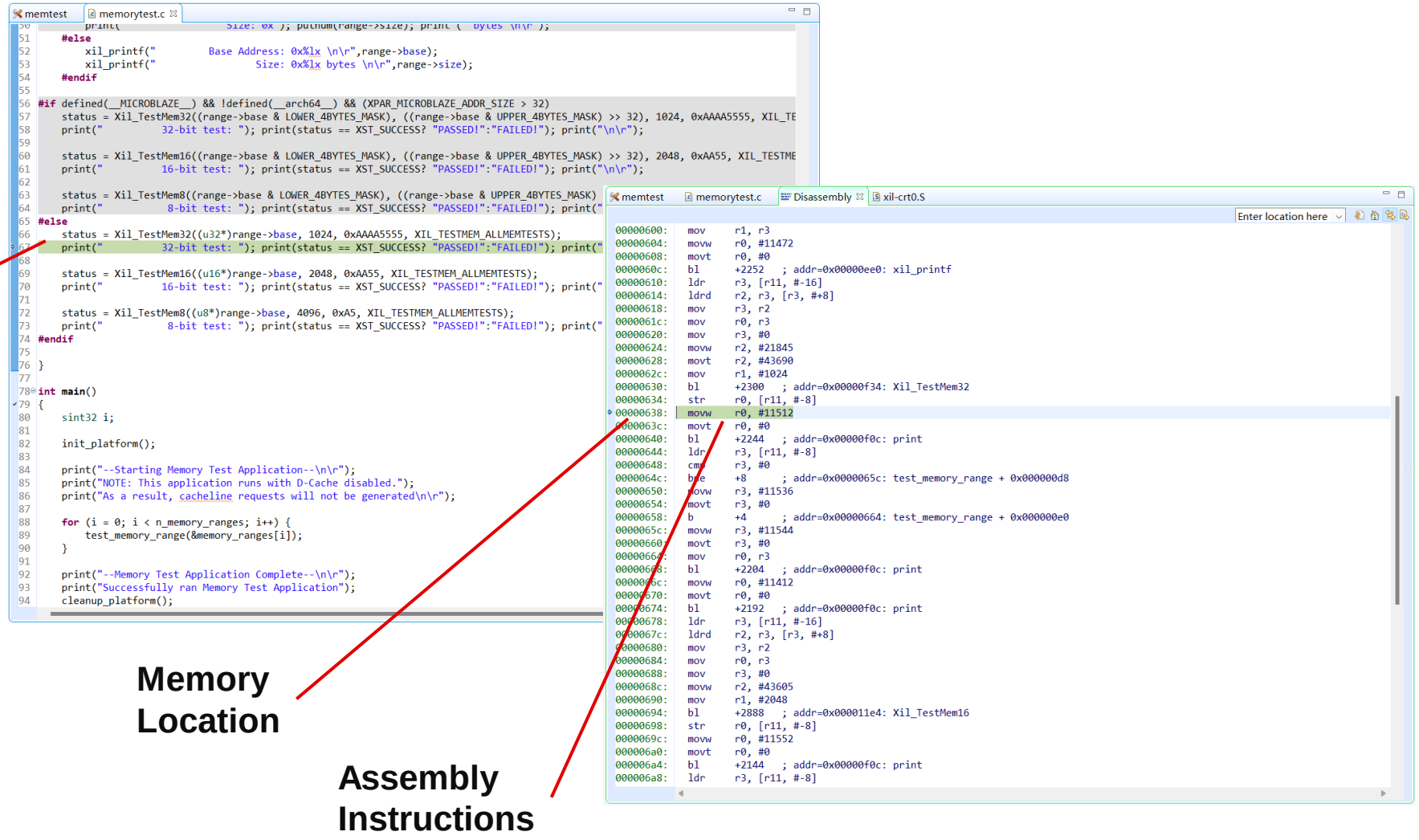
Step by source lines
(steps into functions)

Next source line
(steps over functions)

Step return from function



Debug

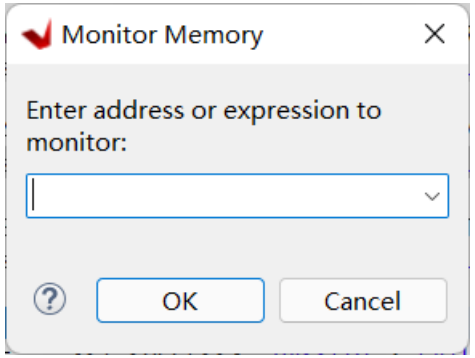


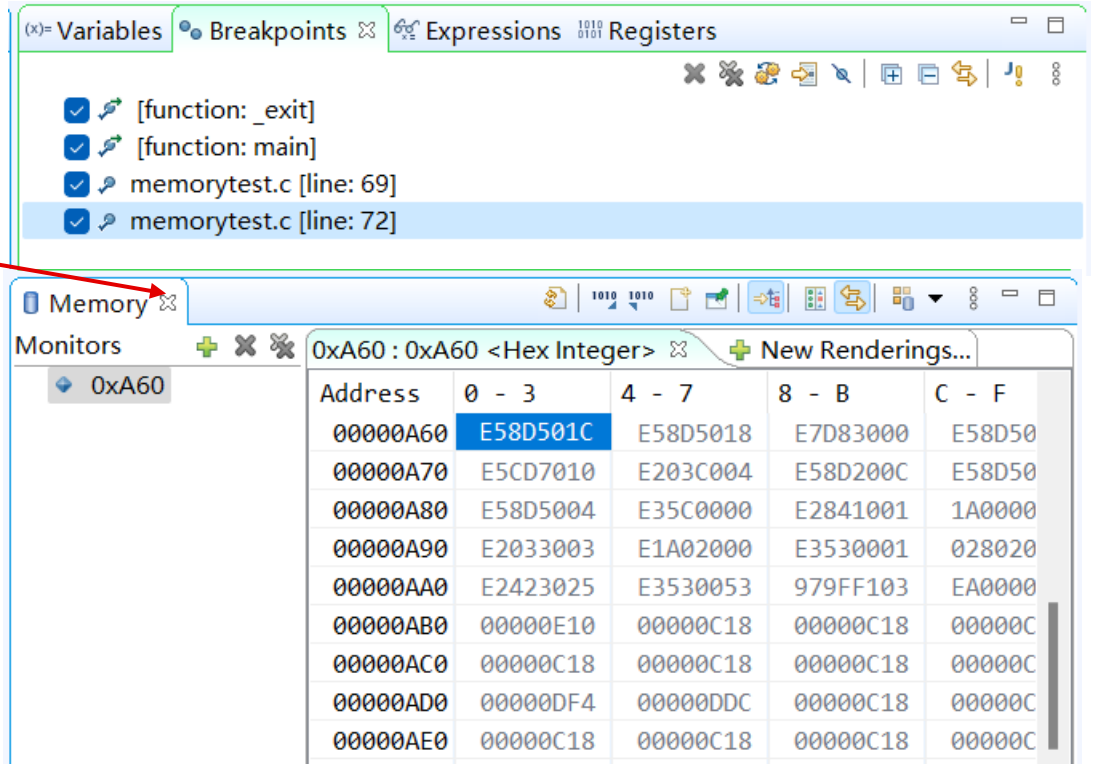
AMD XILINX

Debug Functionality

- Breakpoints can be enabled or disabled
- To change any memory value, click a memory field

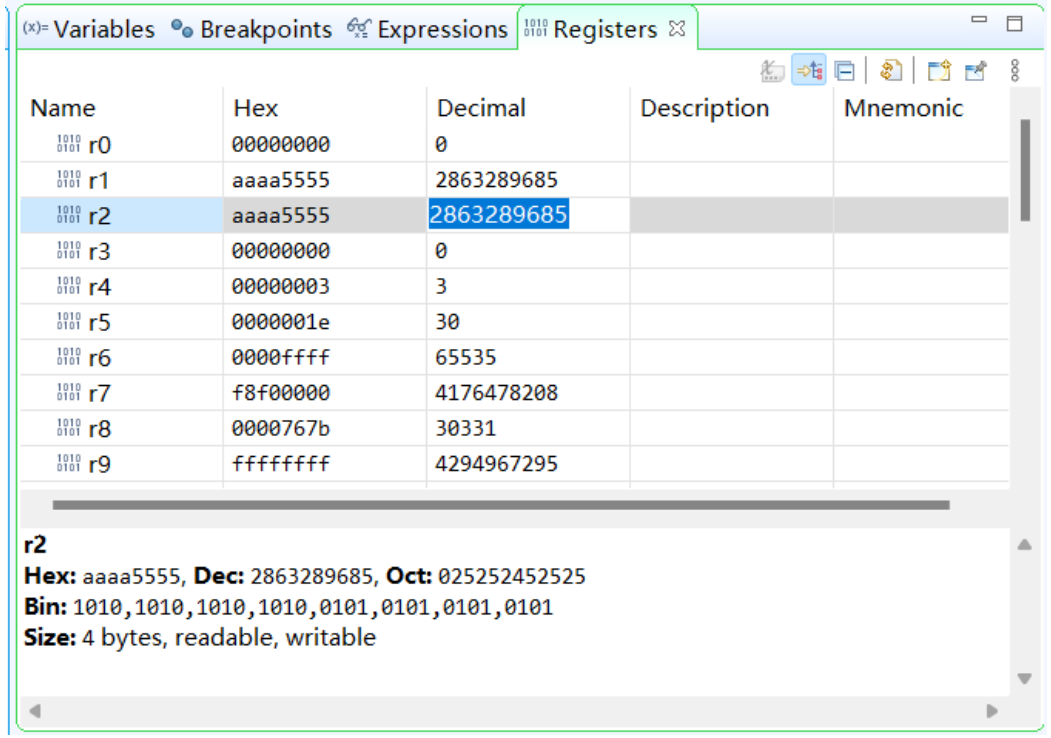
Add a memory monitor





Debug Functionality

- Yellow represents registers that have changed (useful when following assembly code)
- To change any value, click to edit



Summary

Summary

- Standalone APIs and Drivers
  - Different level of drivers
- Debugging is an integral part of embedded systems development
- System Debugger/TCF
  - Multicore Debug, shared connection
- Vitis provides environment, perspective, and underlying tools to enable seamless software debugging

Thank You

Disclaimer and Attribution

AMD
XILINX