



Software Development and Debug

2021.2

Objectives

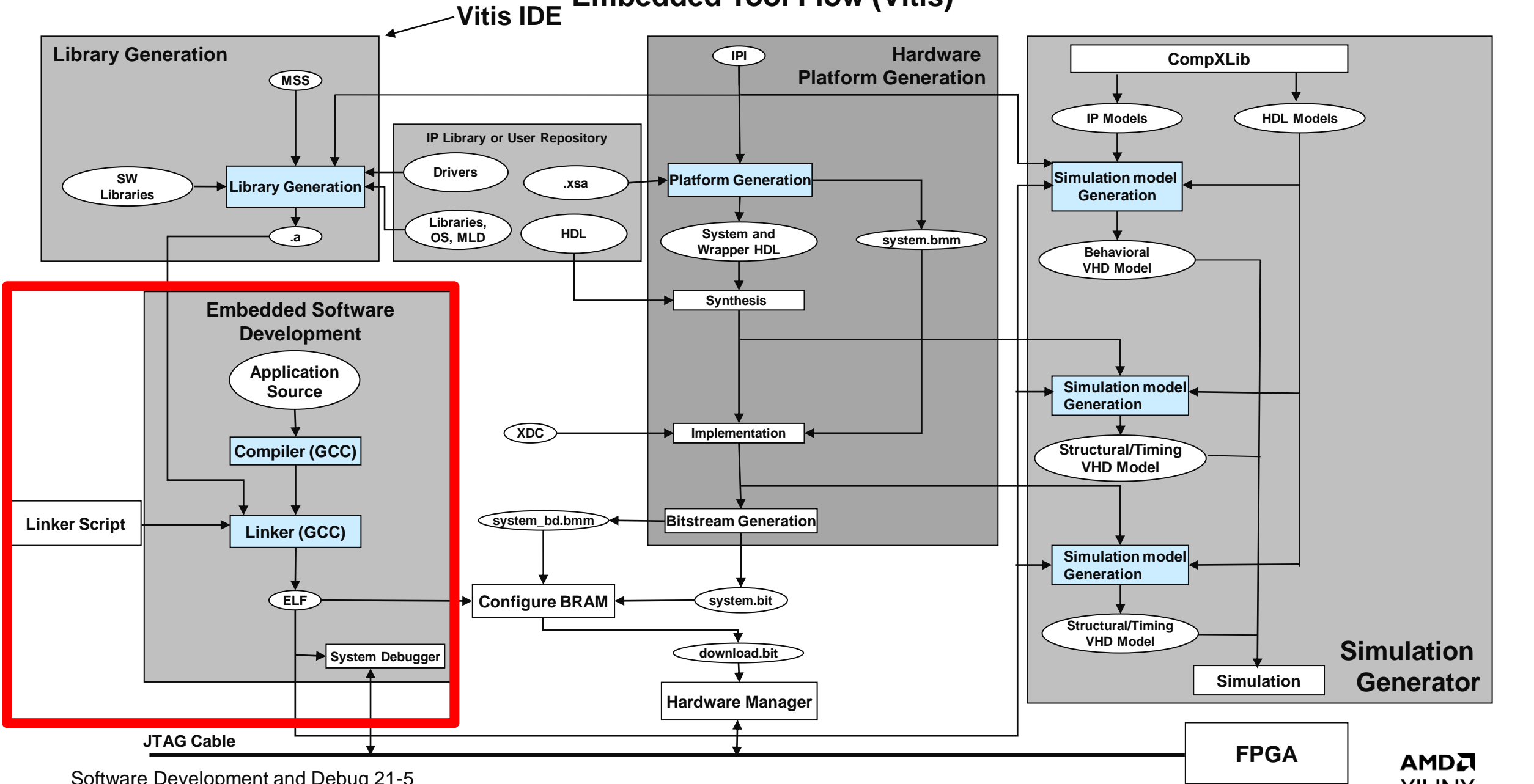
- ▶ **After completing this module, you will be able to:**
 - Identify the GNU tools functionality
 - State when address management is needed
 - Describe the object file sections
 - Describe what a linker script does
 - Describe Debug functionality and Eclipse Target Communications Framework

Outline

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

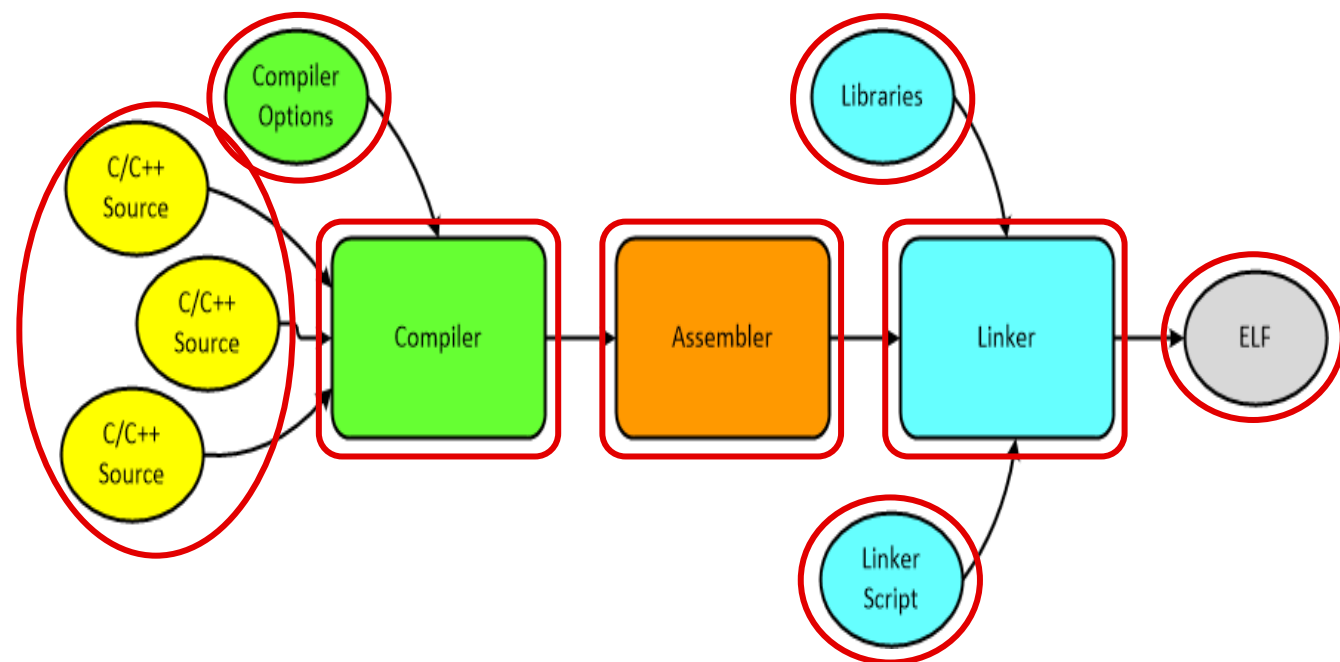
GNU Development Tools: GCC, AS, LD, Binutils

Embedded Tool Flow (Vitis)



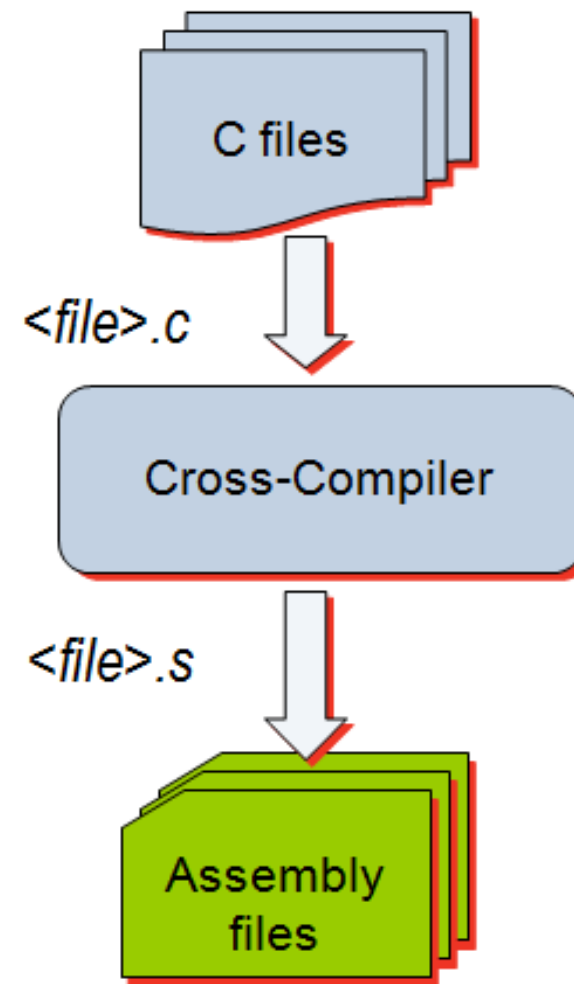
Controlling Compilation

- ▶ 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



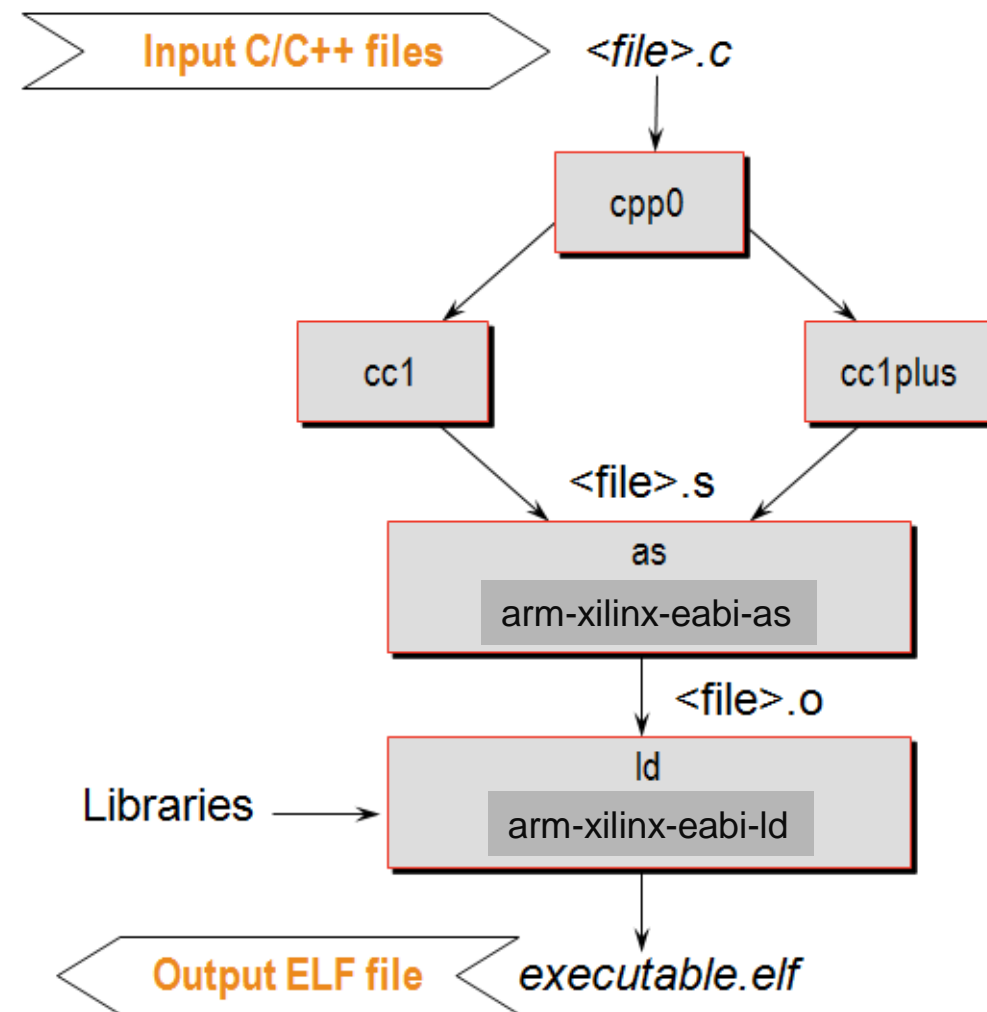
GNU Tools: GCC

- ▶ 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



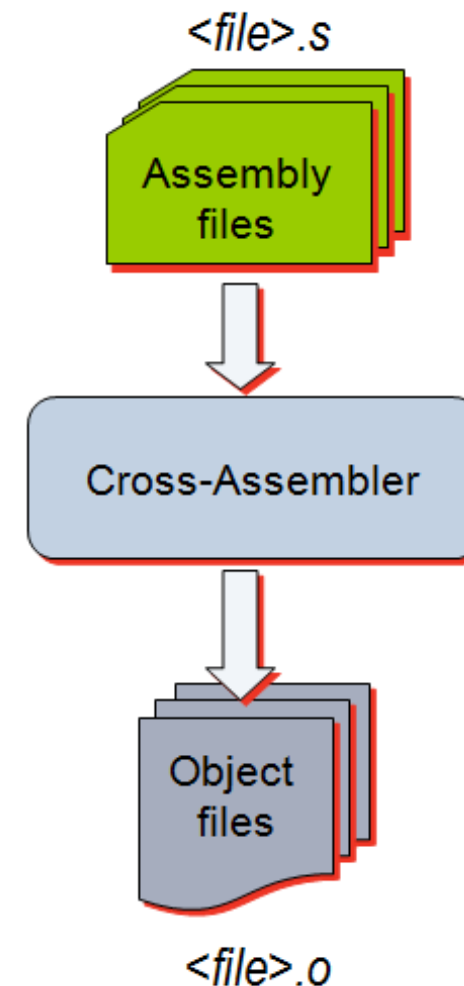
GNU Tools: GCC

- ▶ 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



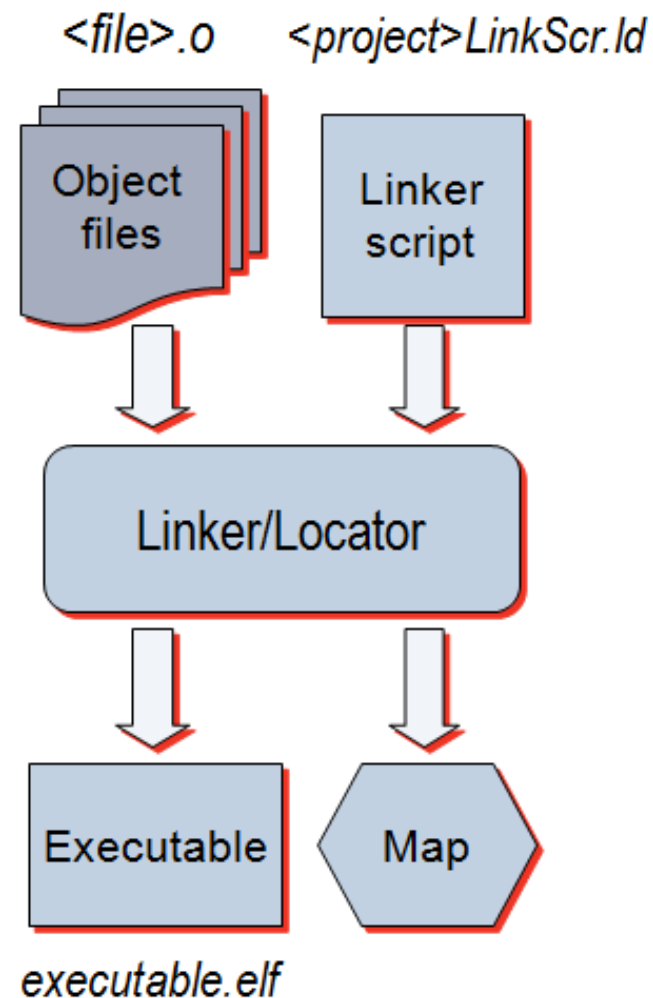
GNU Tools: AS

- ▶ 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(){

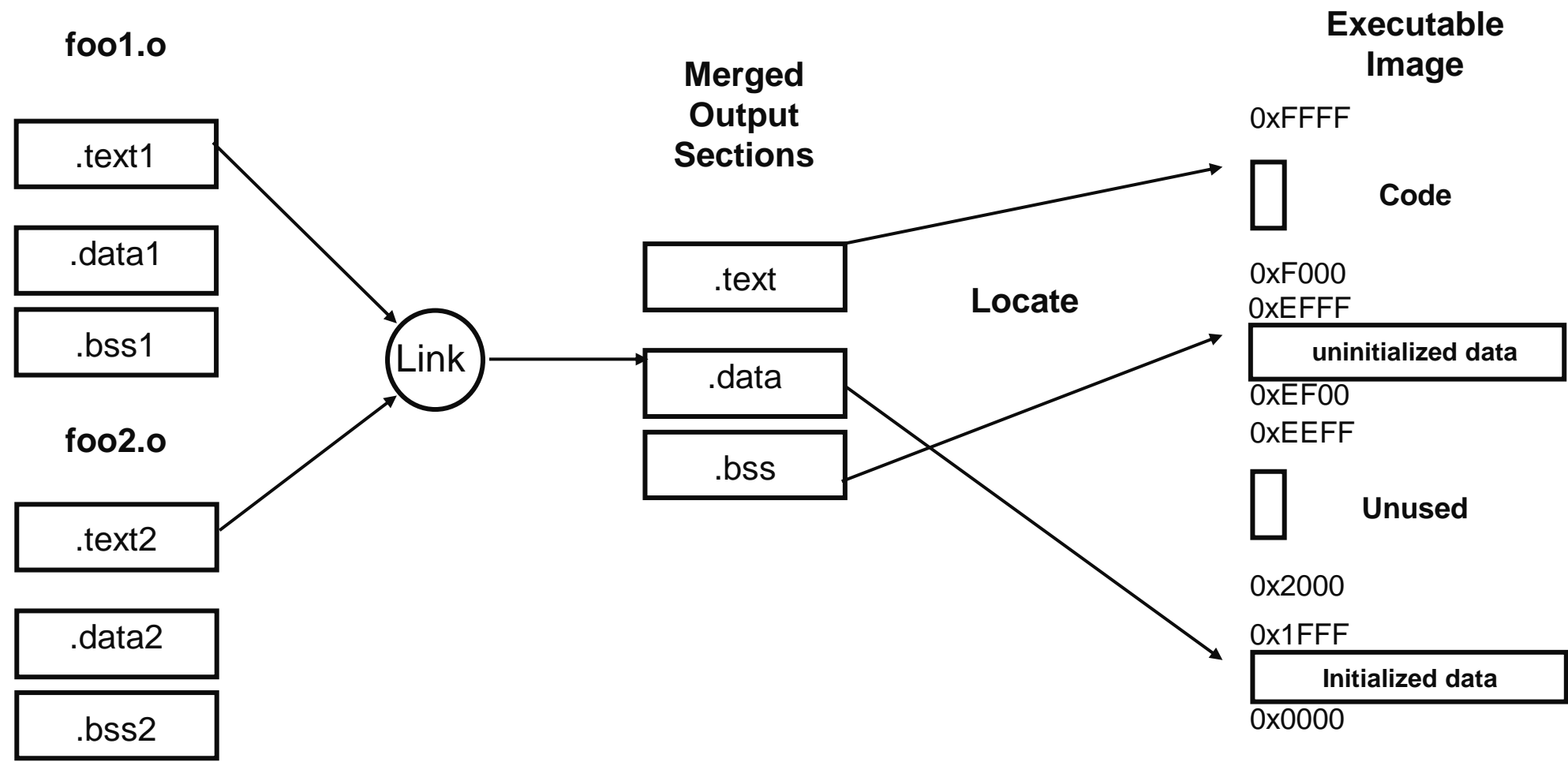
    ...
    I = I + 10;  /* TEXT */
    ...

}
```

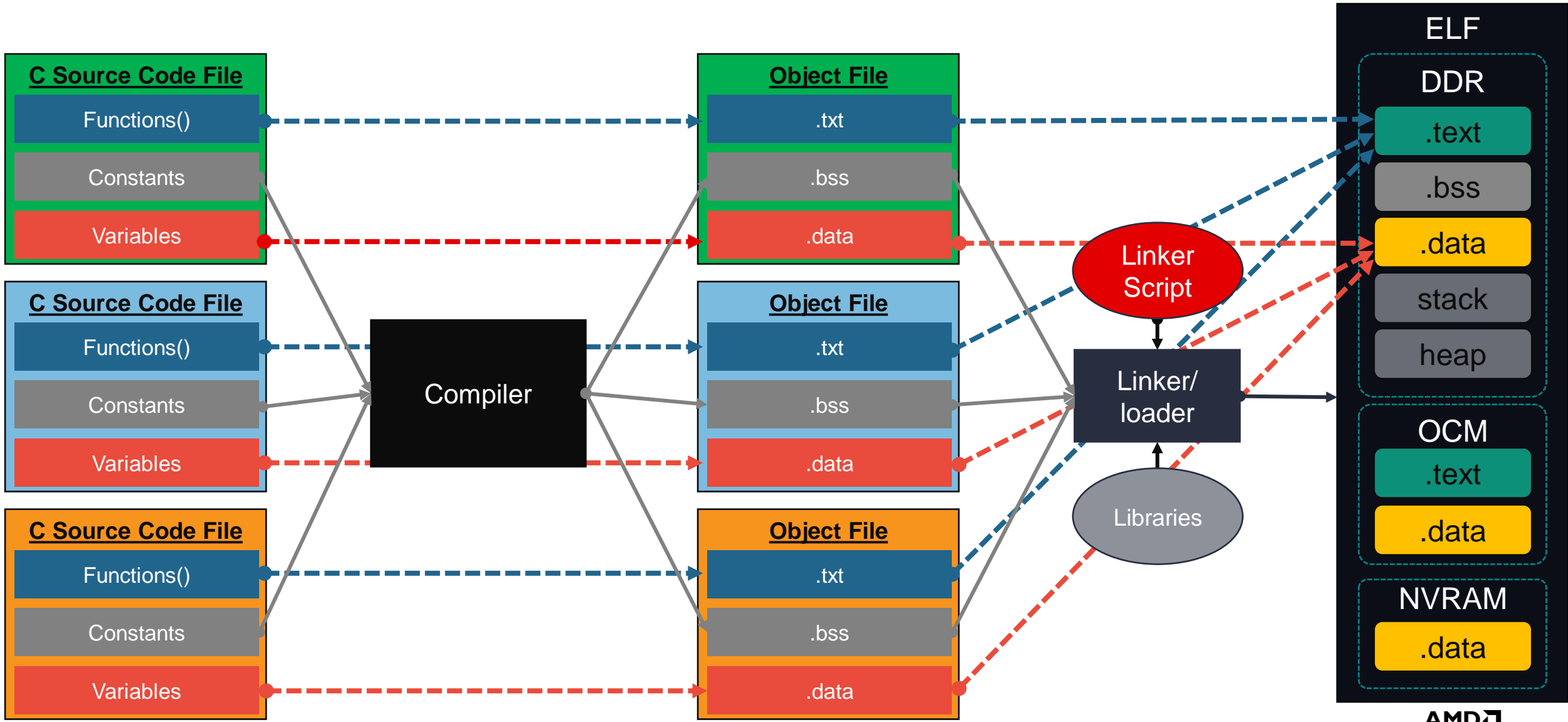
Linker Script

- ▶ **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



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
```

TestApp.elf: file format elf32-littlearm

Sections:

Idx	Name	Size	UMA	LMA	File off	Algn
0	.text	00001950	00100000	00100000	00008000	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.init	00000018	00101950	00101950	00009950	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
2	.fini	00000018	00101968	00101968	00009968	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
3	.rodata	00000980	00101980	00101980	00009980	2**2
	CONTENTS, ALLOC, LOAD, DATA					
4	.data	00000af0	001019af0	001019af0	000099af0	2**2
	CONTENTS, ALLOC, LOAD, DATA					
5	.eh_frame	00000004	00101f54	00101f54	00009f54	2**2
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
6	.bss	0000005c	00101f58	00101f58	00009f58	2**2
	ALLOC					
7	.mmu_tbl	0000a04c	00101fb4	00101fb4	00009fb4	2**0
	CONTENTS, ALLOC, LOAD, READONLY, DATA					
8	.init_array	00000008	0010c000	0010c000	00014000	2**2
	CONTENTS, ALLOC, LOAD, DATA					

Section Name

Section Size

Virtual Memory Address

Loadable Memory Address

Byte alignment

Offset from the beginning of the section header table

Object Dump

Dumping the source and assembly code

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

Memory location

Machine Language Instruction

C code instruction

Assembly instruction

```
int main (void)
{
    1003bc:      e92d4800      push    {fp, lr}
    1003c0:      e28db004      add     fp, sp, #4
    1003c4:      e24dd030      sub     sp, sp, #48      ; 0x30
    XGpio dip, push;
        int i, psb_check, dip_check;

    //xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
    1003c8:      e24b3020      sub     r3, fp, #32
    1003cc:      e1a00003      mov     r0, r3
    1003d0:      e3a01000      mov     r1, #0
    1003d4:      eb000326      bl      101074 <XGpio_Initialize>
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);
    1003d8:      e24b3020      sub     r3, fp, #32
    1003dc:      e1a00003      mov     r0, r3
        e3a01001      mov     r1, #1
        e3e02000      mvn     r2, #0
        eb00024e      bl      100d28 <XGpio_SetDataDirection>

    XGpio_Initialize(&push, XPAR_PUSH_DEVICE_ID);
```

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**

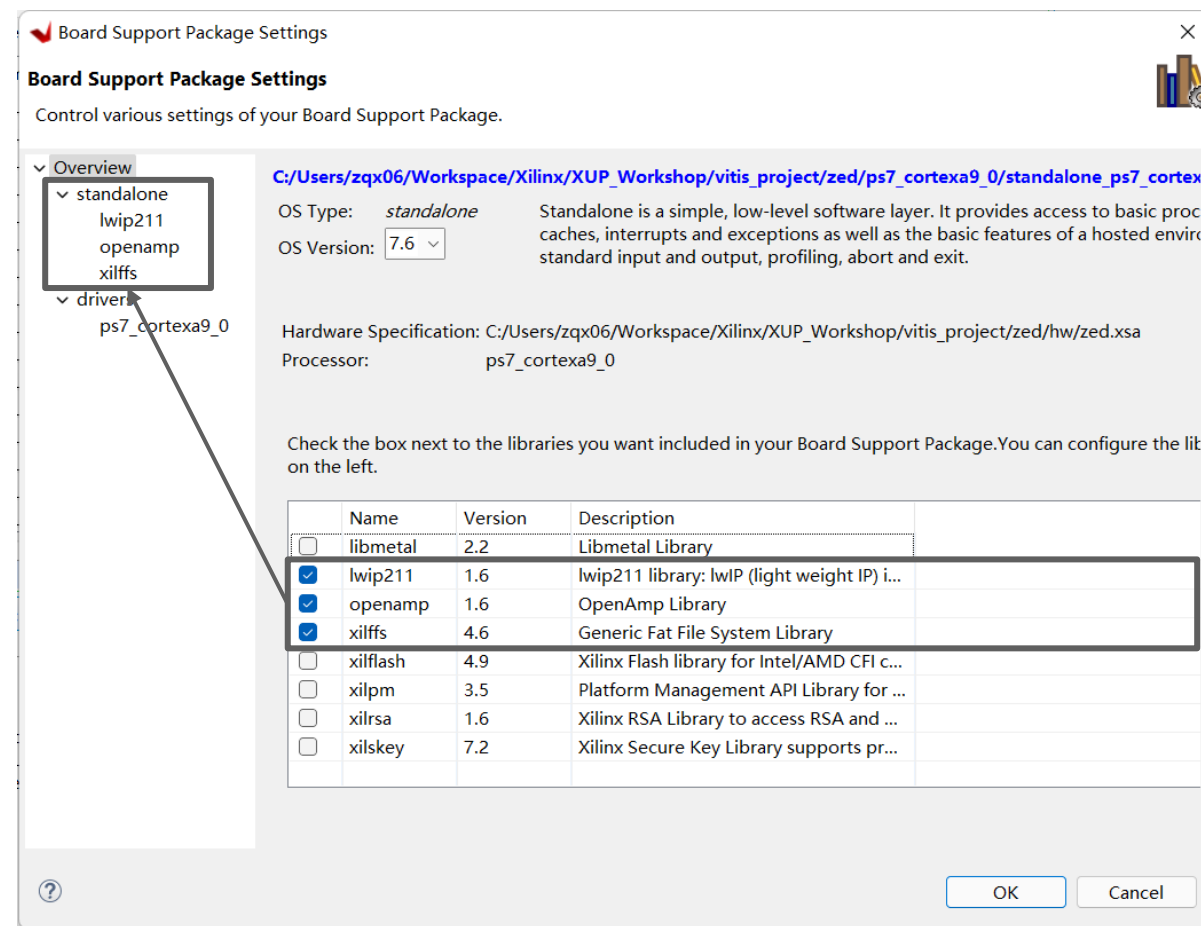
** Also available as additions to the Standalone domain

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

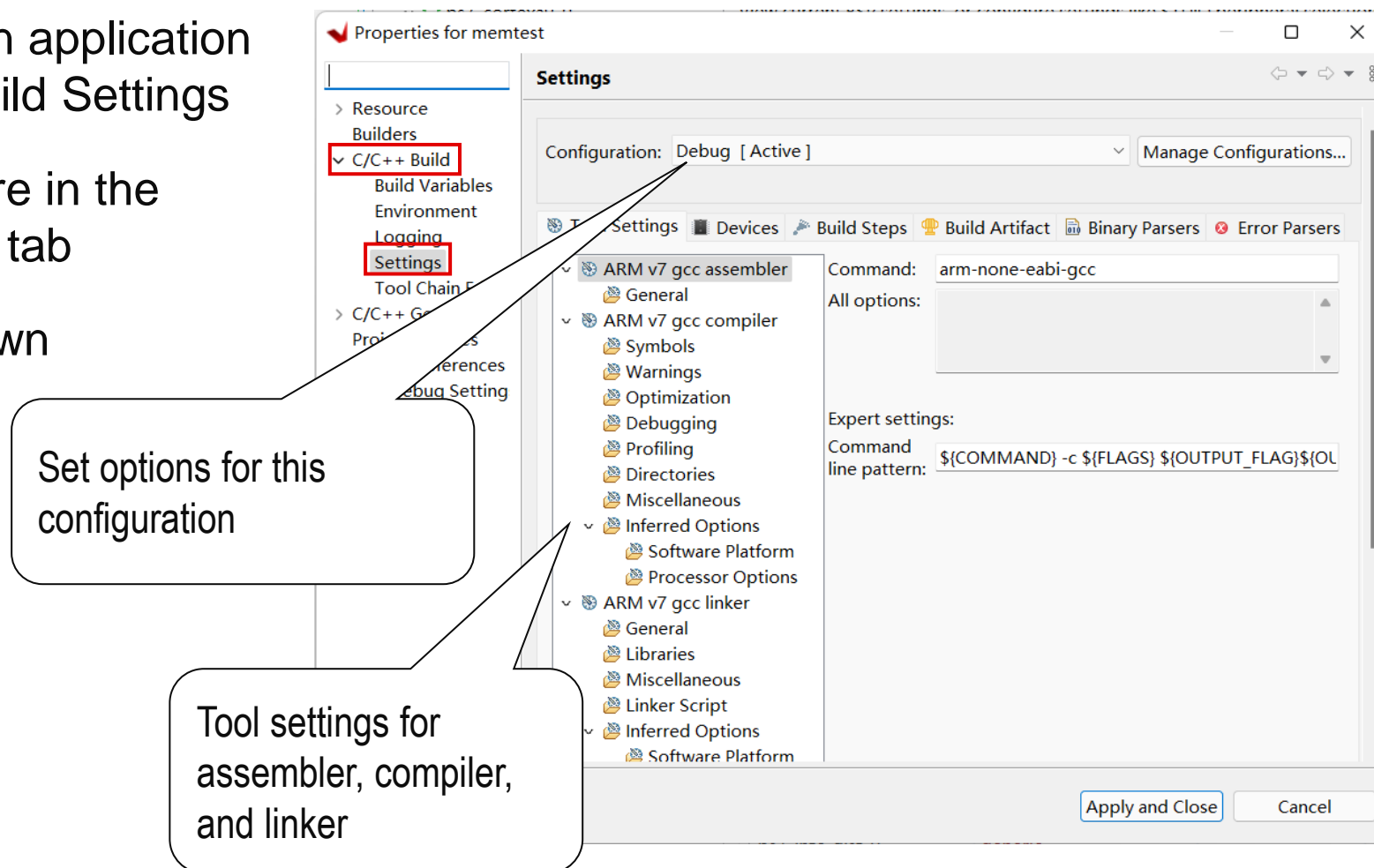
Accessing Software Platform Properties

- ▶ 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



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



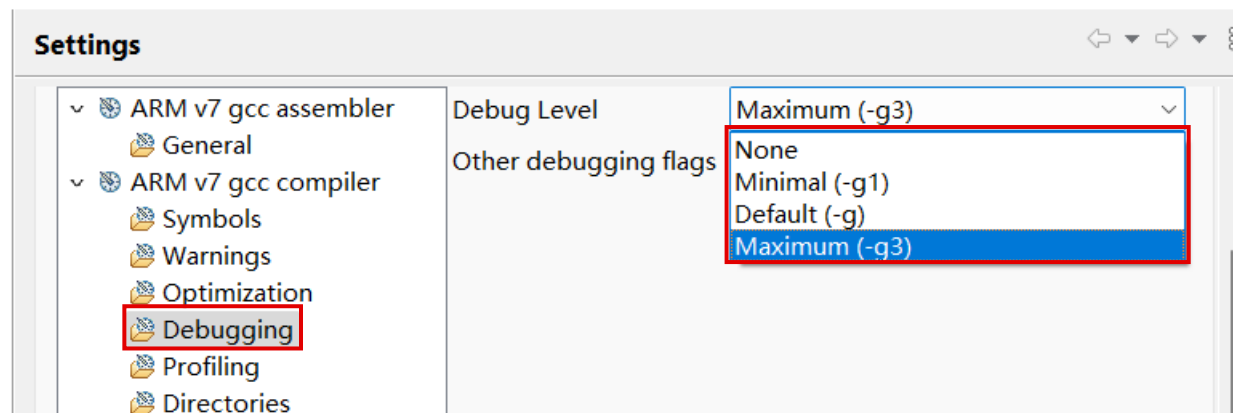
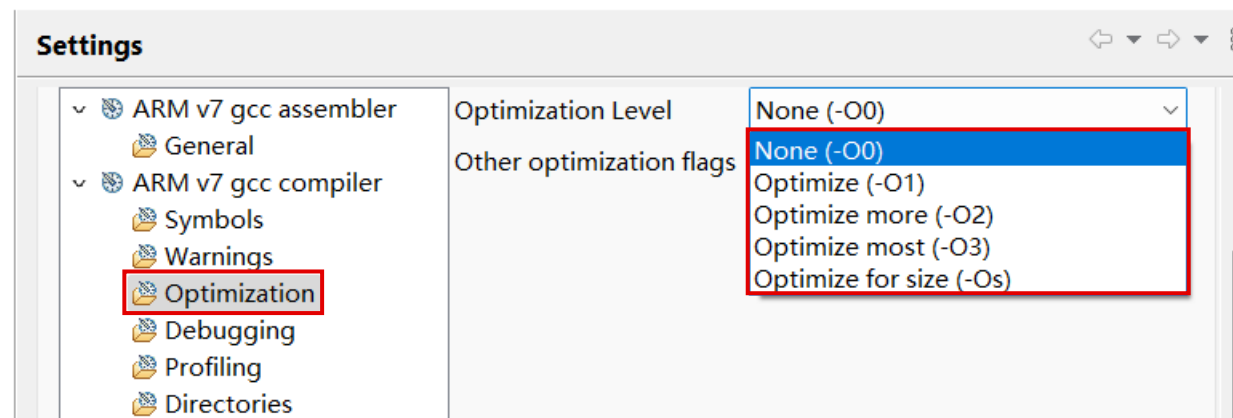
Debug/Optimization Properties

► 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

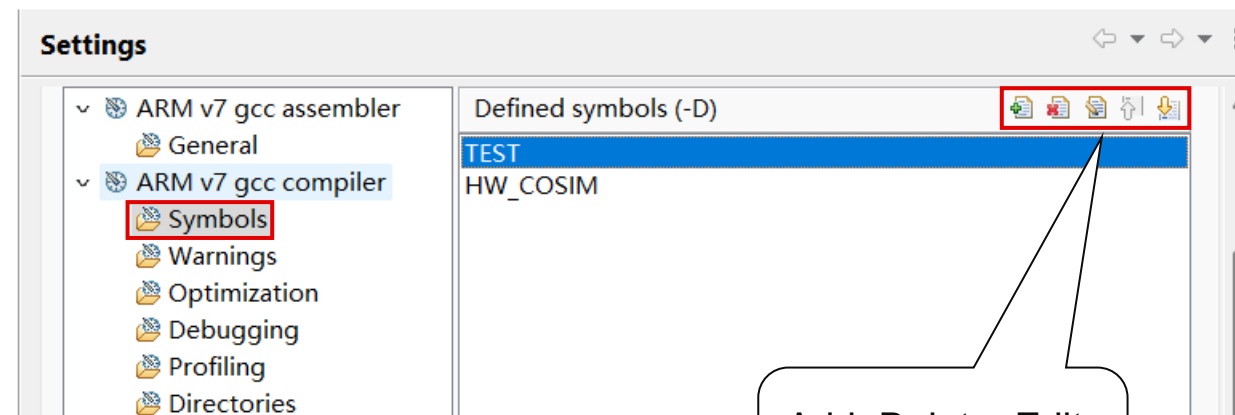
- ▶ Define symbols for conditional compiling

- Add
- Delete
- Edit

- ▶ References C source

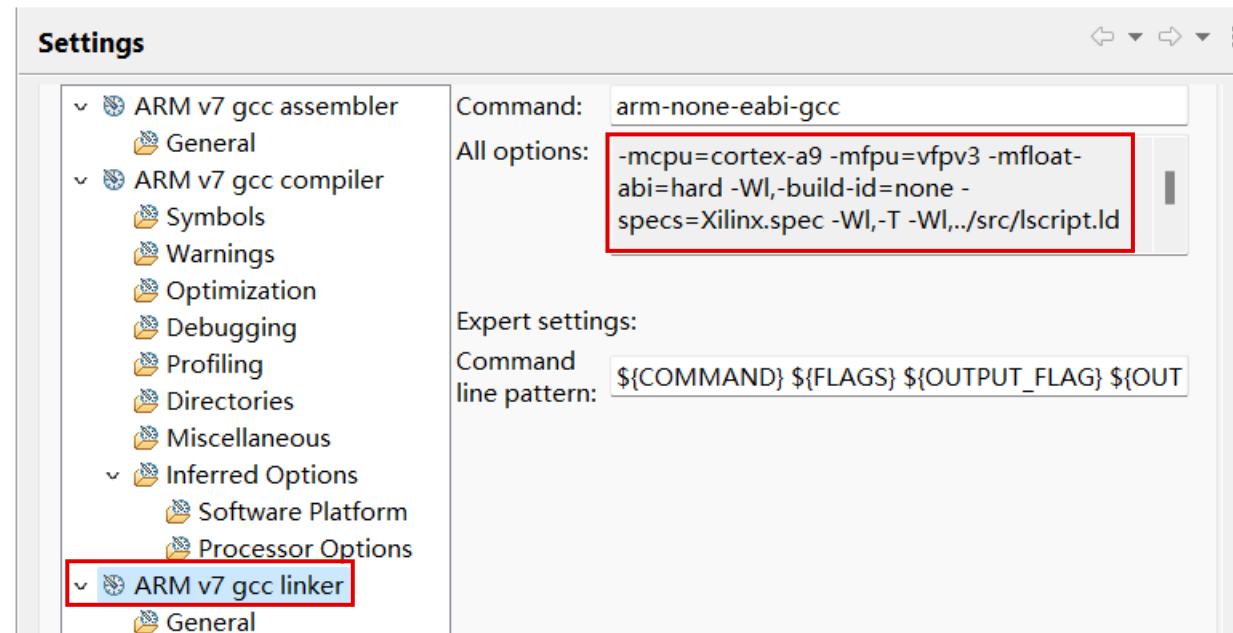
```
#ifdef symbol  
conditional statements  
#endif
```

- ▶ Passed to compiler as `-D` option
- ▶ Other compiler options are available



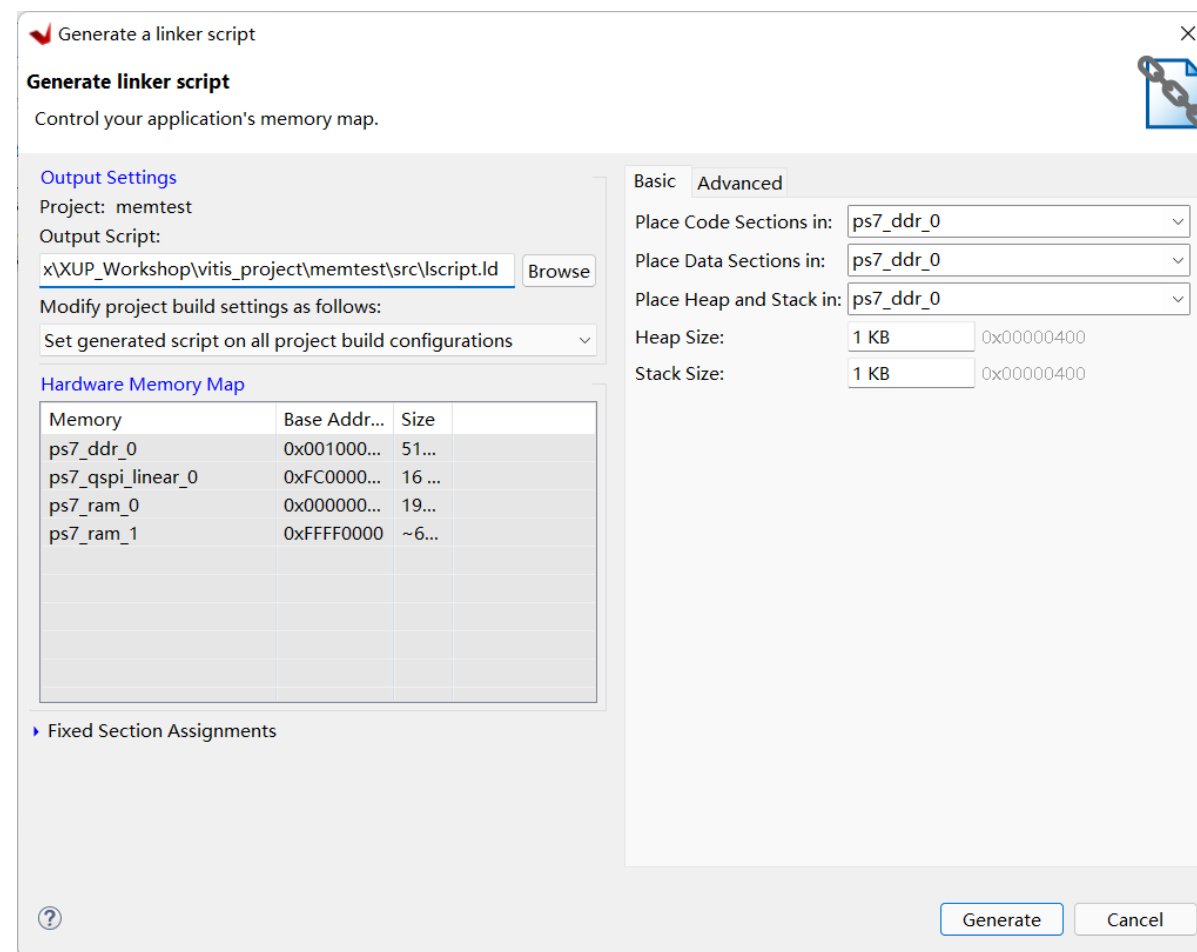
Linker Properties

- ▶ 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



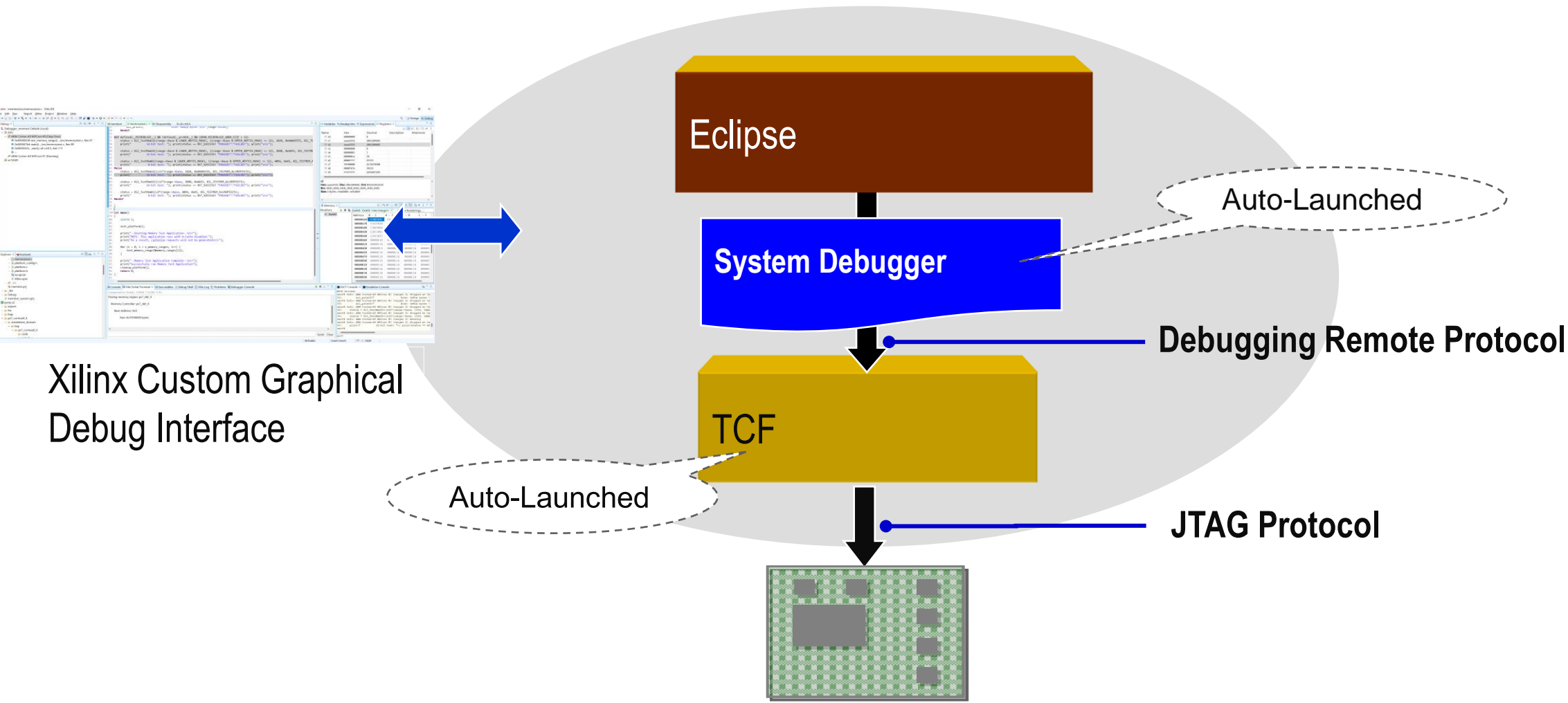
Linker Script Generator GUI

- ▶ 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

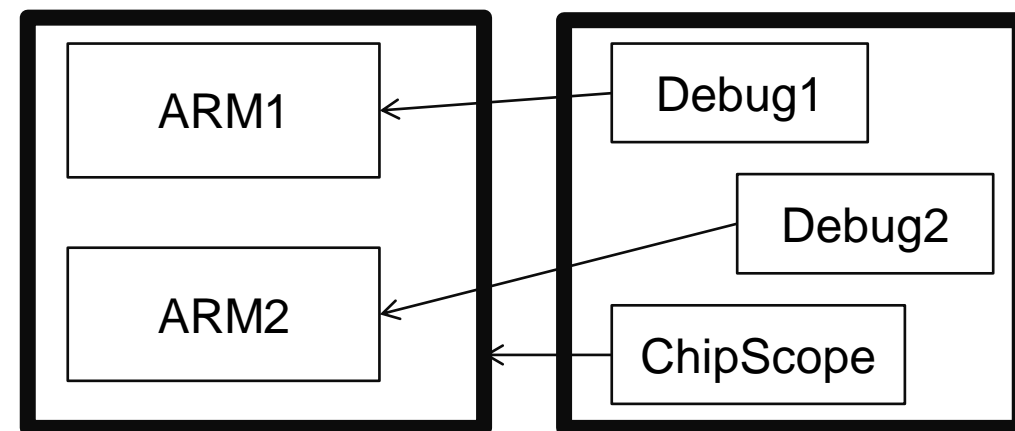
Debugging Using Vitis IDE (TCF)



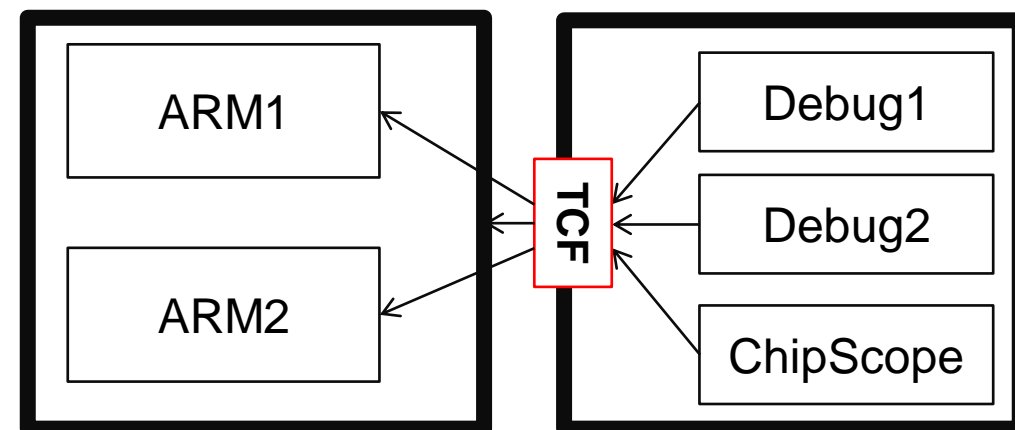
Eclipse Target Communication Framework

- ▶ 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)

Separate connections



Separate connections



Software Debugging Support

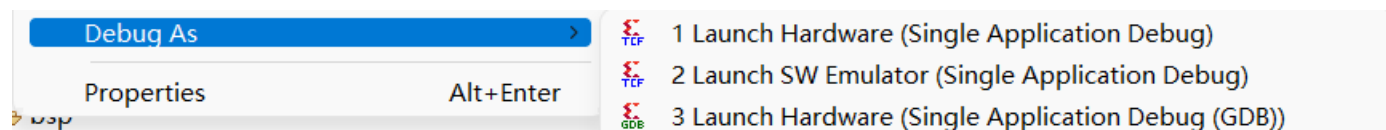
System Debugger

GNU DBG

XSDB

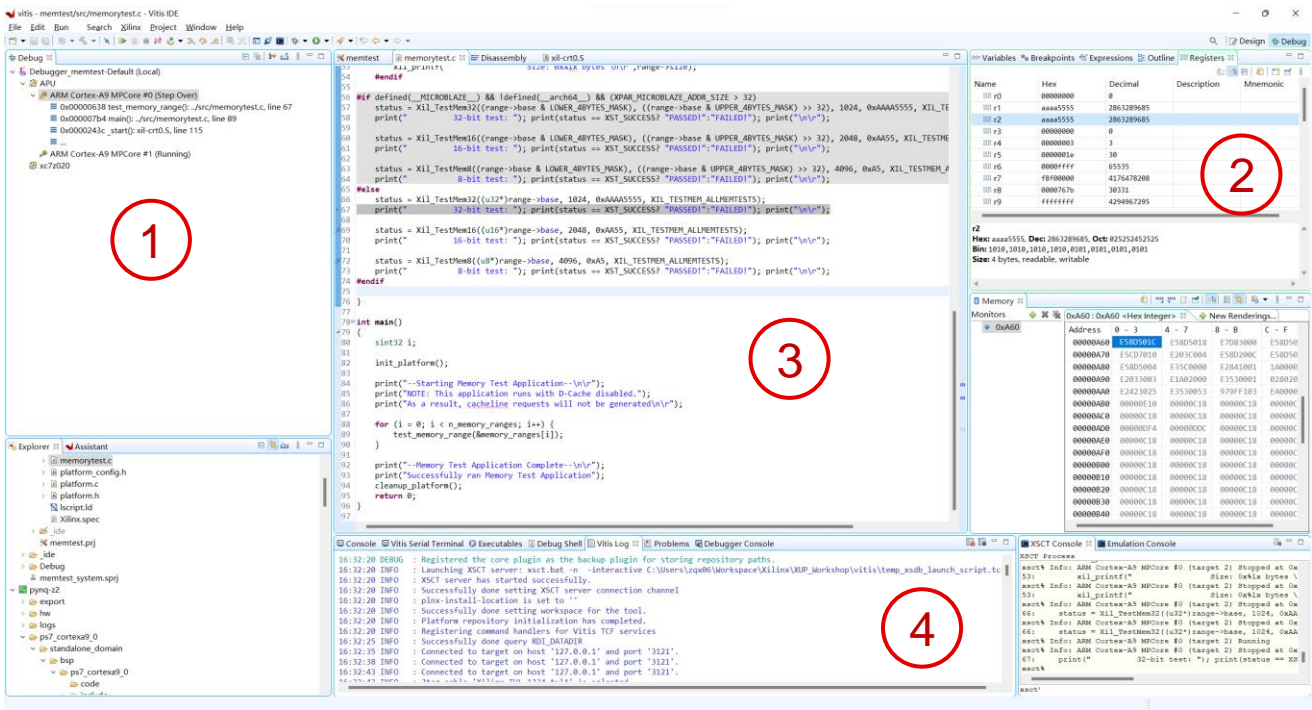
System Debugger

- ▶ 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



Debug GUI

► Run-time control

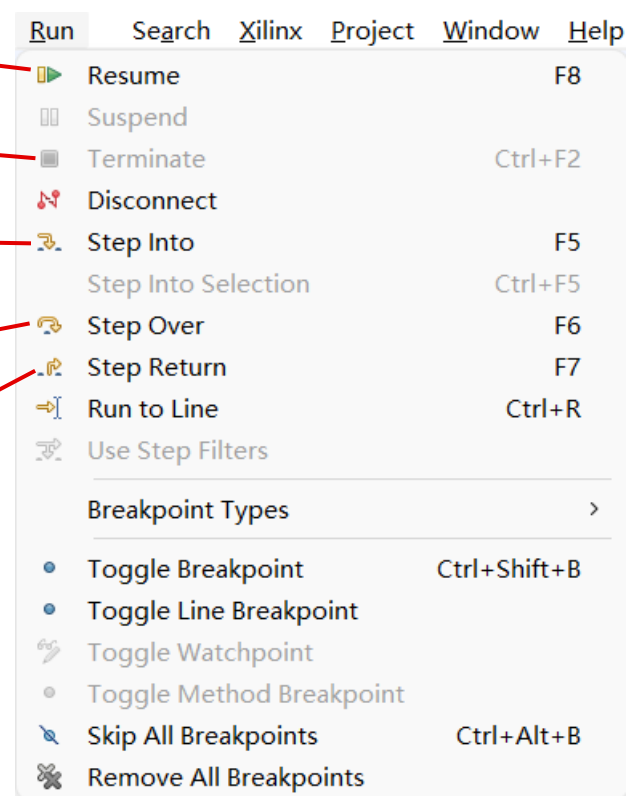
Continue to next breakpoint

Stop debug session

Step by source lines
(steps into functions)

Next source line
(steps over functions)

Step return from function



Debug

C Code

```
memtest | memorytest.c
50 printf("Size: 0x%x; paddrum(range->size); printf(" bytes: 0x%x\n",
51 #else
52 xil_printf("Base Address: 0x%x\n", range->base);
53 xil_printf("Size: 0x%x bytes\n", range->size);
54 #endif
55
56 #if defined(__MICROBLAZE__) && !defined(__arch64__) && (XPAR_MICROBLAZE_ADDR_SIZE > 32)
57 status = Xil_TestMem32((range->base & LOWER_4BYTES_MASK), ((range->base & UPPER_4BYTES_MASK) >> 32), 1024, 0xAAAA5555, XIL_TESTMEM_ALLMEMTESTS);
58 printf("32-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
59
60 status = Xil_TestMem16((range->base & LOWER_4BYTES_MASK), ((range->base & UPPER_4BYTES_MASK) >> 32), 2048, 0xAA55, XIL_TESTMEM_ALLMEMTESTS);
61 printf("16-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
62
63 status = Xil_TestMem8((range->base & LOWER_4BYTES_MASK), ((range->base & UPPER_4BYTES_MASK) >> 32), 4096, 0xA5, XIL_TESTMEM_ALLMEMTESTS);
64 printf("8-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
65 #else
66 status = Xil_TestMem32((u32*)range->base, 1024, 0xAAAA5555, XIL_TESTMEM_ALLMEMTESTS);
67 printf("32-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
68
69 status = Xil_TestMem16((u16*)range->base, 2048, 0xAA55, XIL_TESTMEM_ALLMEMTESTS);
70 printf("16-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
71
72 status = Xil_TestMem8((u8*)range->base, 4096, 0xA5, XIL_TESTMEM_ALLMEMTESTS);
73 printf("8-bit test: "); print(status == XST_SUCCESS? "PASSED!": "FAILED!"); print("\n\r");
74 #endif
75 }
76
77 int main()
78 {
79     sint32 i;
80
81     init_platform();
82
83     print("--Starting Memory Test Application--\n\r");
84     print("NOTE: This application runs with D-Cache disabled.");
85     print("As a result, cacheline requests will not be generated\n\r");
86
87     for (i = 0; i < n_memory_ranges; i++) {
88         test_memory_range(&memory_ranges[i]);
89     }
90
91     print("--Memory Test Application Complete--\n\r");
92     print("Successfully ran Memory Test Application");
93     cleanup_platform();
94 }
```

Memory Location

Assembly Instructions

```
memtest | memorytest.c | Disassembly | xil-crt0.S
00000600: mov r1, r3
00000604: movw r0, #11472
00000608: movt r0, #0
0000060c: bl +2252 ; addr=0x00000ee0: xil_printf
00000610: ldr r3, [r11, #-16]
00000614: ldrd r2, r3, [r3, #+8]
00000618: mov r3, r2
0000061c: mov r0, r3
00000620: mov r3, #0
00000624: movw r2, #21845
00000628: movt r2, #43690
0000062c: mov r1, #1024
00000630: bl +2300 ; addr=0x00000f34: Xil_TestMem32
00000634: str r0, [r11, #-8]
00000638: movw r0, #11512
0000063c: movt r0, #0
00000640: bl +2244 ; addr=0x00000f0c: print
00000644: ldr r3, [r11, #-8]
00000648: cmpr r3, #0
0000064c: bne +8 ; addr=0x0000065c: test_memory_range + 0x000000d8
00000650: movw r3, #11536
00000654: movt r3, #0
00000658: b +4 ; addr=0x00000664: test_memory_range + 0x000000e0
0000065c: movw r3, #11544
00000660: movt r3, #0
00000664: mov r0, r3
00000668: bl +2204 ; addr=0x00000f0c: print
0000066c: movw r0, #11412
00000670: movt r0, #0
00000674: bl +2192 ; addr=0x00000f0c: print
00000678: ldr r3, [r11, #-16]
0000067c: ldrd r2, r3, [r3, #+8]
00000680: mov r3, r2
00000684: mov r0, r3
00000688: mov r3, #0
0000068c: movw r2, #43605
00000690: mov r1, #2048
00000694: bl +2888 ; addr=0x000011e4: Xil_TestMem16
00000698: str r0, [r11, #-8]
0000069c: movw r0, #11552
000006a0: movt r0, #0
000006a4: bl +2144 ; addr=0x00000f0c: print
000006a8: ldr r3, [r11, #-8]
```



Debug Functionality

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

Add a memory monitor

Monitor Memory

Enter address or expression to monitor:

?

OK

Cancel

(x)= Variables Breakpoints Expressions Registers

☒ [function: _exit]

☒ [function: main]

☒ memorytest.c [line: 69]

☒ memorytest.c [line: 72]

Memory

Monitors

0xA60

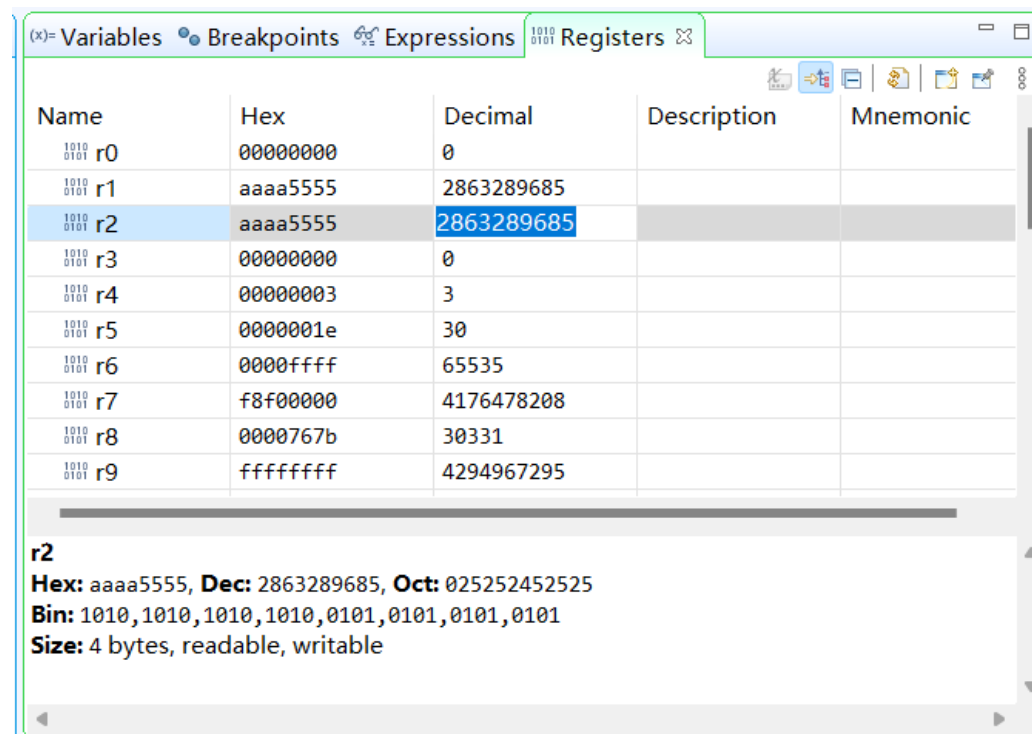
0xA60 : 0xA60 <Hex Integer>

+ New Renderings...

Address	0 - 3	4 - 7	8 - B	C - F
00000A60	E58D501C	E58D5018	E7D83000	E58D50
00000A70	E5CD7010	E203C004	E58D200C	E58D50
00000A80	E58D5004	E35C0000	E2841001	1A0000
00000A90	E2033003	E1A02000	E3530001	028020
00000AA0	E2423025	E3530053	979FF103	EA0000
00000AB0	00000E10	00000C18	00000C18	00000C
00000AC0	00000C18	00000C18	00000C18	00000C
00000AD0	00000DF4	00000DDC	00000C18	00000C
00000AE0	00000C18	00000C18	00000C18	00000C

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

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

© Copyright 2022 Advanced Micro Devices, Inc. All rights reserved. Xilinx, the Xilinx logo, AMD, the AMD Arrow logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Vitis, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

