System Debug

Objectives

After completing this module, you will be able to:

- Describe GNU Debugger (GDB) functionality
- Describe Xilinx Microprocessor Debugger (XMD) functionality
- Describe the integration of XMD and GDB with SDK

Outline

- **Debugging Tools**
- Software Debug Environments
  - XPS
  - SDK
- Simultaneous HW/SW Debugging

Introduction

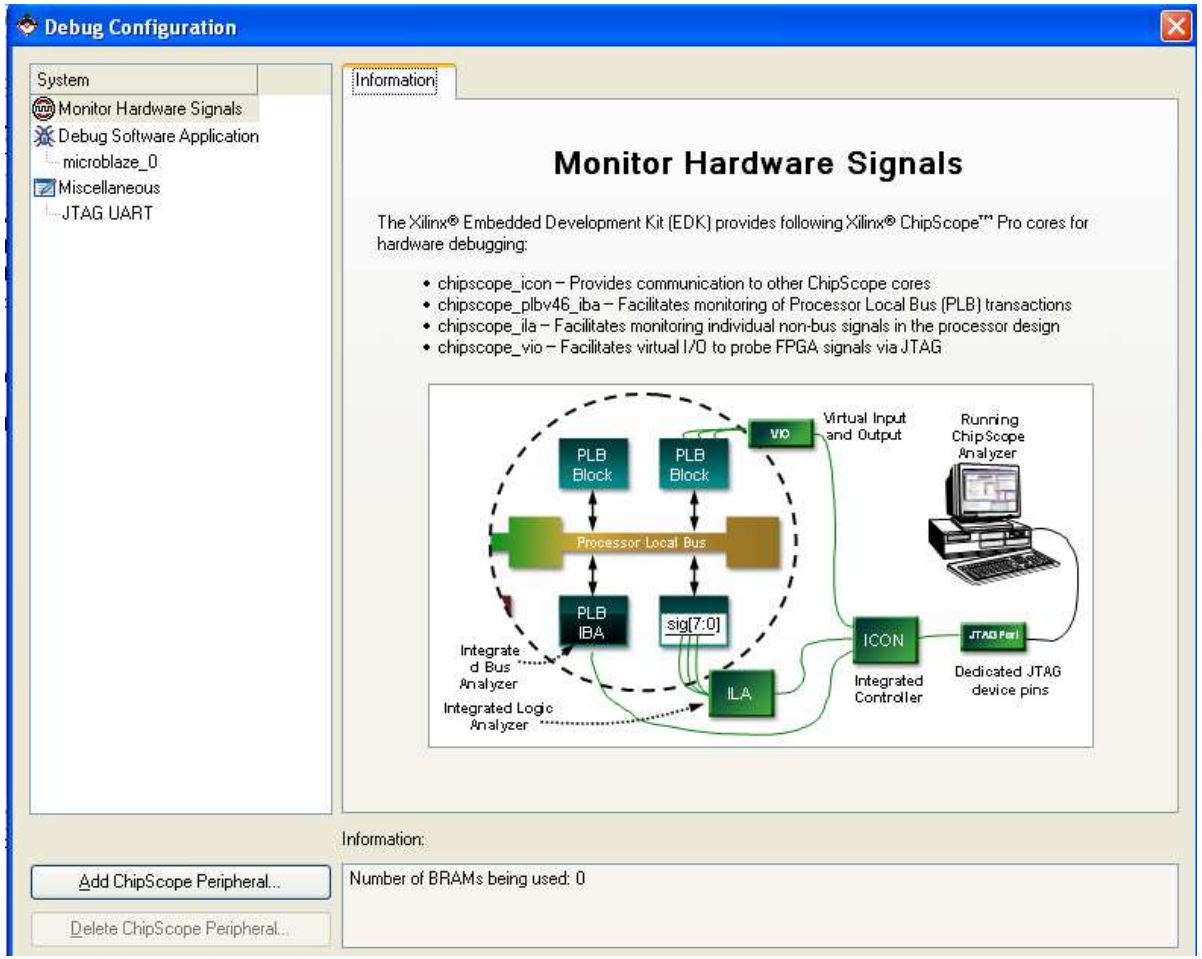
- Debugging is an integral part of embedded systems development
- The debugging process is defined as testing, stabilizing, localizing, and correcting errors
- Two methods of debugging:
  - Hardware debugging via a logic probe, logic analyzer, in-circuit emulator, or background debugger
  - Software debugging via a debugging instrument
    - A software debugging instrument is source code that is added to the program for the purpose of debugging
- Debugging types:
  - Functional debugging
  - Performance debugging

Hardware Debugging Support

- ChipScope™ Pro tool cores are available to a Xilinx Platform Studio design
  - PLB IBA (Integrated Bus Analyzer)
  - OPB IBA
  - ILA (Integrated Logic Analyzer)
  - VIO (Virtual I/O)
- Enables co-debug of software with GNU gdb and hardware with ChipScope Analyzer

Debug Configuration Wizard

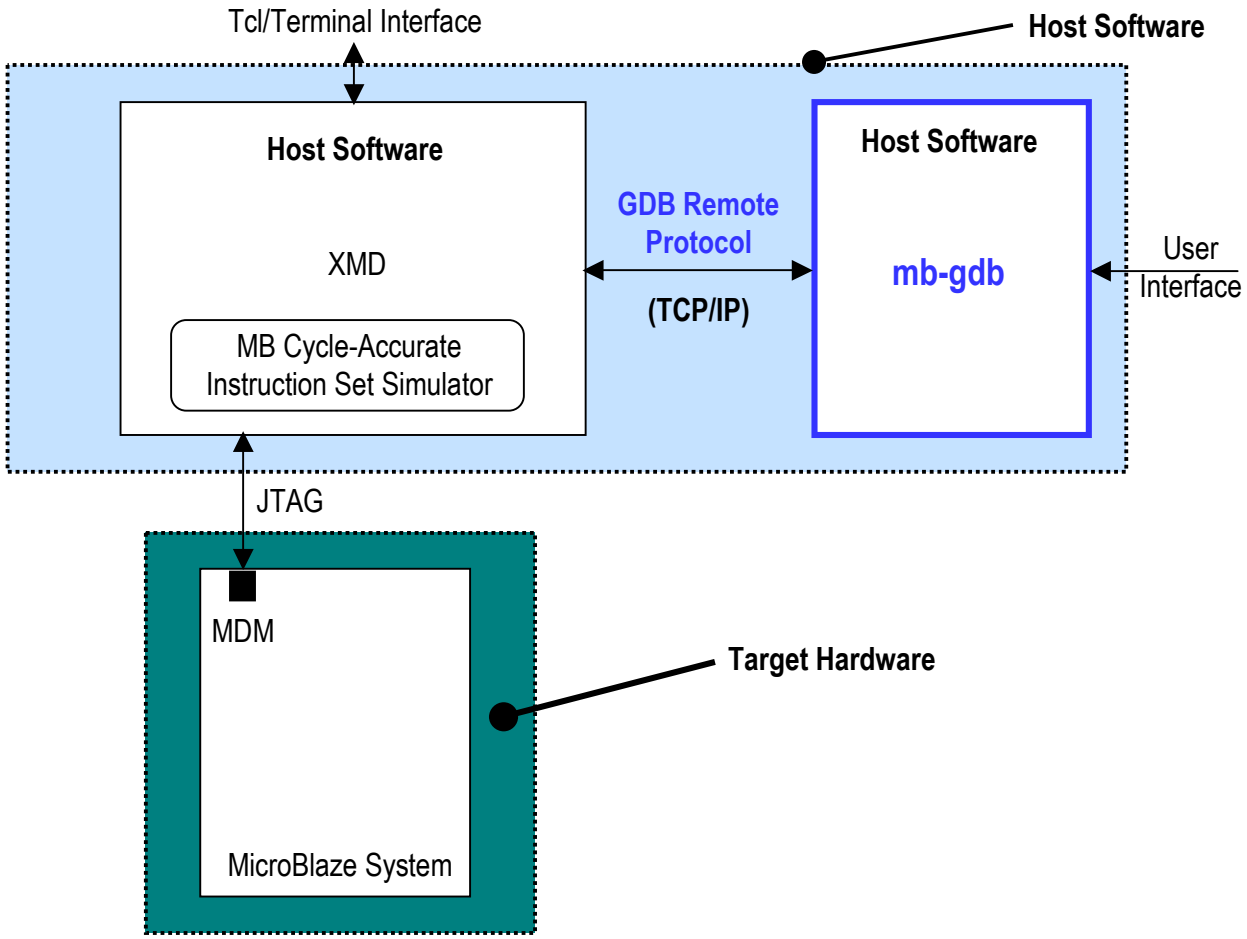
Easily add chipscope cores to an EDK design



Software Debugging Support

- EDK supports software debugging via:
  - GNU Debugger (GDB)
    - Software debugger that runs on PC
  - Microprocessor Debug Module (MDM)
    - Debug interface in MicroBlaze system
  - Xilinx Microprocessor Debugger (XMD)
    - Facilitates an interface between the GNU tools and the MicroBlaze MDM

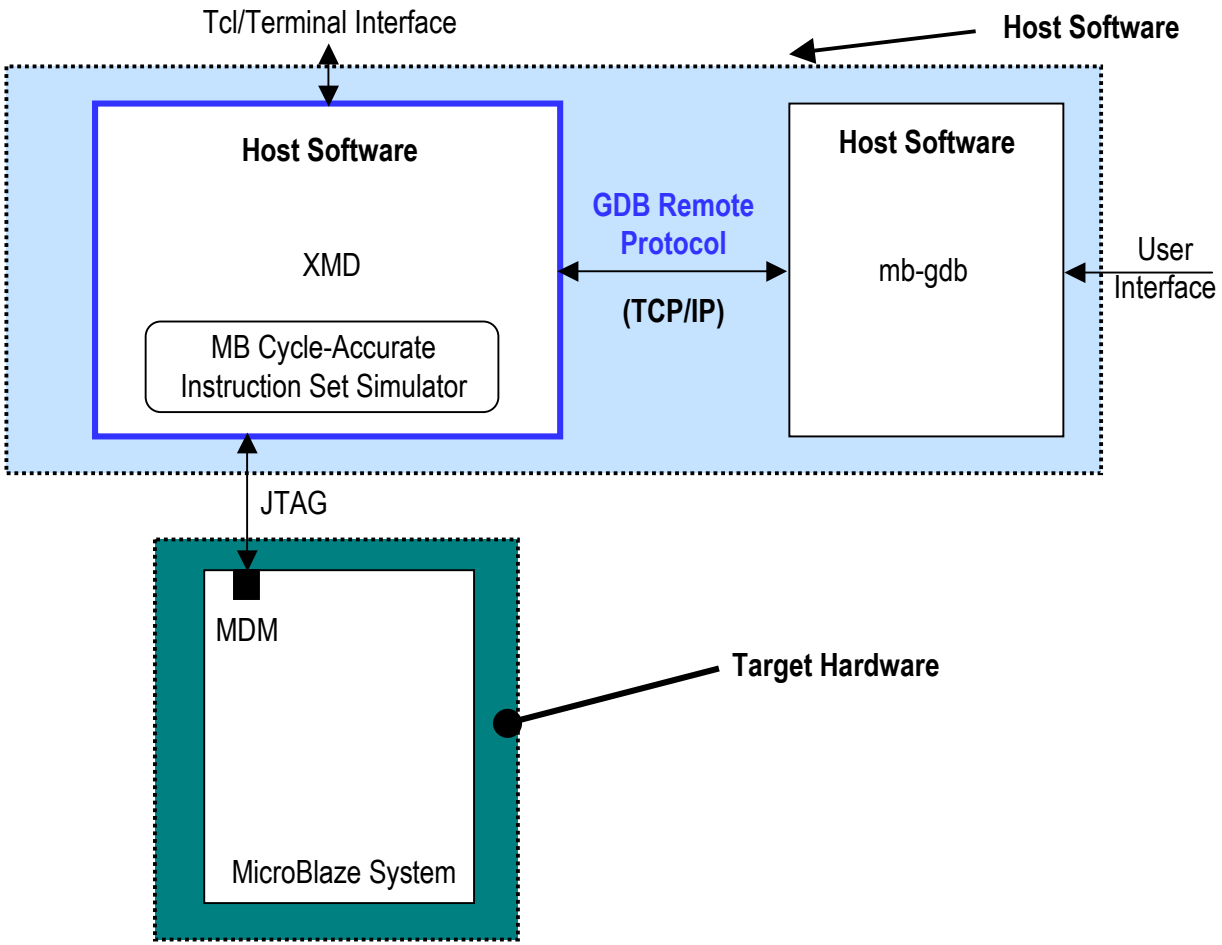
GDB Functionality



GDB Functionality

- GDB is a source-level debugger that helps you debug your program
  - Start your program
  - Set breakpoints (make your program stop on specified conditions)
  - Examine what has happened, when your program encounters breakpoints
    - Registers
    - Memory
    - Stack
    - Variables
    - Expressions
  - Change things in your program so that you can experiment with correcting the effects of one bug and go on to another
- You can use GDB to debug programs written in C and C++

XMD Functionality



XMD Functionality

- Xilinx Microprocessor Debug (XMD) engine
  - A program that facilitates a unified GDB interface
  - A Tool command language (Tcl) interface
- XMD supports debugging user programs on different targets:
  - Cycle-accurate MicroBlaze™ processor instruction set simulator
  - MicroBlaze systems running **xmdstub** on a hardware board
  - MicroBlaze systems using the MDM peripheral
- **mb-gdb** communicates with **xmd** by using the Remote TCP protocol and controlling the corresponding targets
- GDB can connect to **xmd** on the same computer or on a remote computer on the Internet

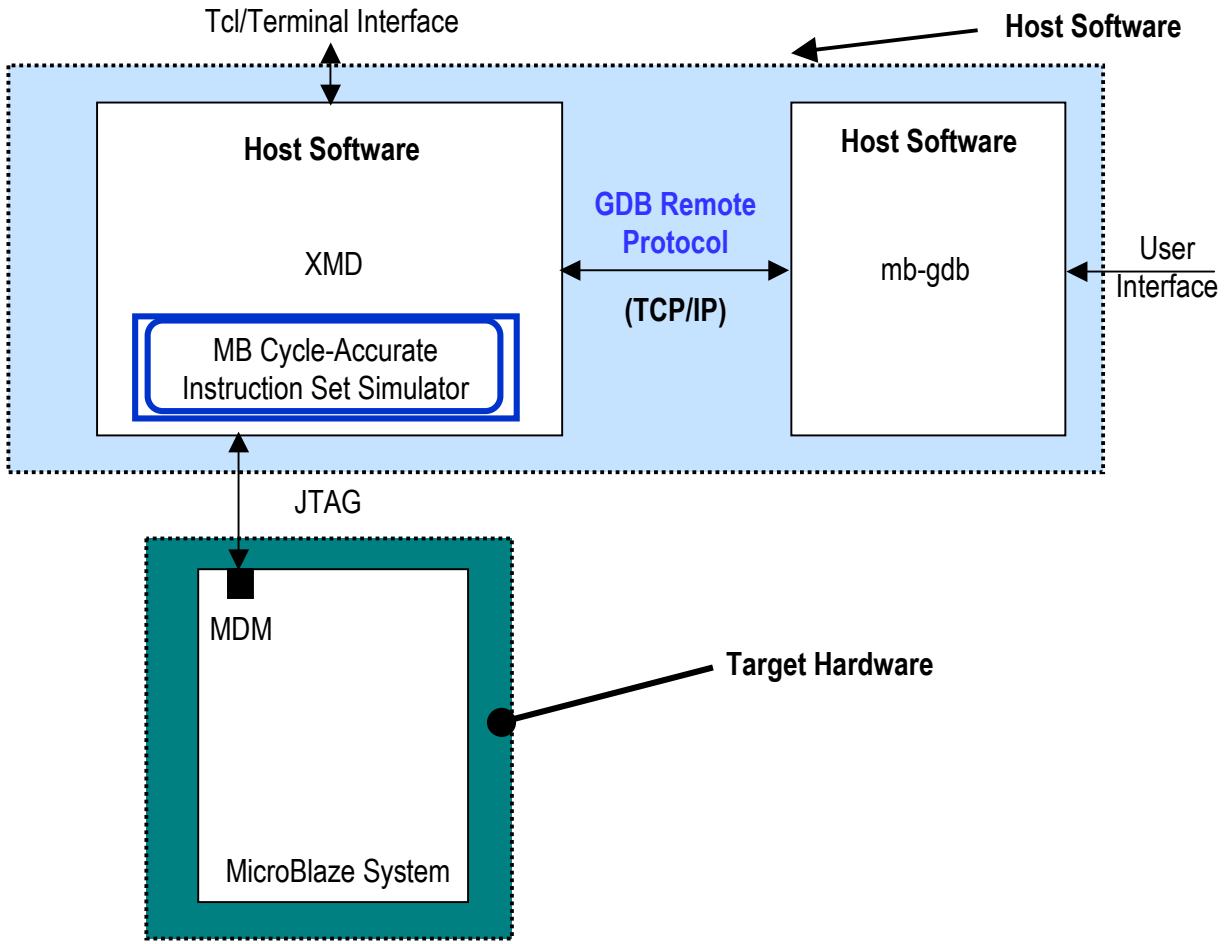
XMD Options

- mbconnect <**sim**|**stub**|**mdm**> [*options*]
- Simulator target options
  - **-memsize** *size*
- xmdstub target options
  - **-comm <serial|jtag>**
  - **-posit** *device position*
  - **-chain** *device count <list of BSDL files>*
  - **-port** *serial port*
  - **-baud** *baud rate*

XMD Tcl Interface

- **x?:** lists all Tcl commands
- **xrmem *target addr* [*num*]:** Reads num bytes or 1 byte from the memory address addr
- **xwmem *target addr value*:** Writes an 8-bit byte *value* at the specified memory *addr*
- **xrreg *target* [*reg*]:** Reads all registers or only register number **reg**
- **xwreg *target reg value*:** Writes a 32-bit *value* into register number *reg*
- **xdownload *target* [-data] *filename* [*addr*]:** Downloads the given ELF or data file (with -data option) onto the memory of the current target
- **xcontinue *target* [*addr*]:** Continues execution from the current PC or from the optional address argument
- **xstep *target*:** Single steps one MicroBlaze™ processor instruction. If the PC is at an IMM instruction, the next instruction is executed as well

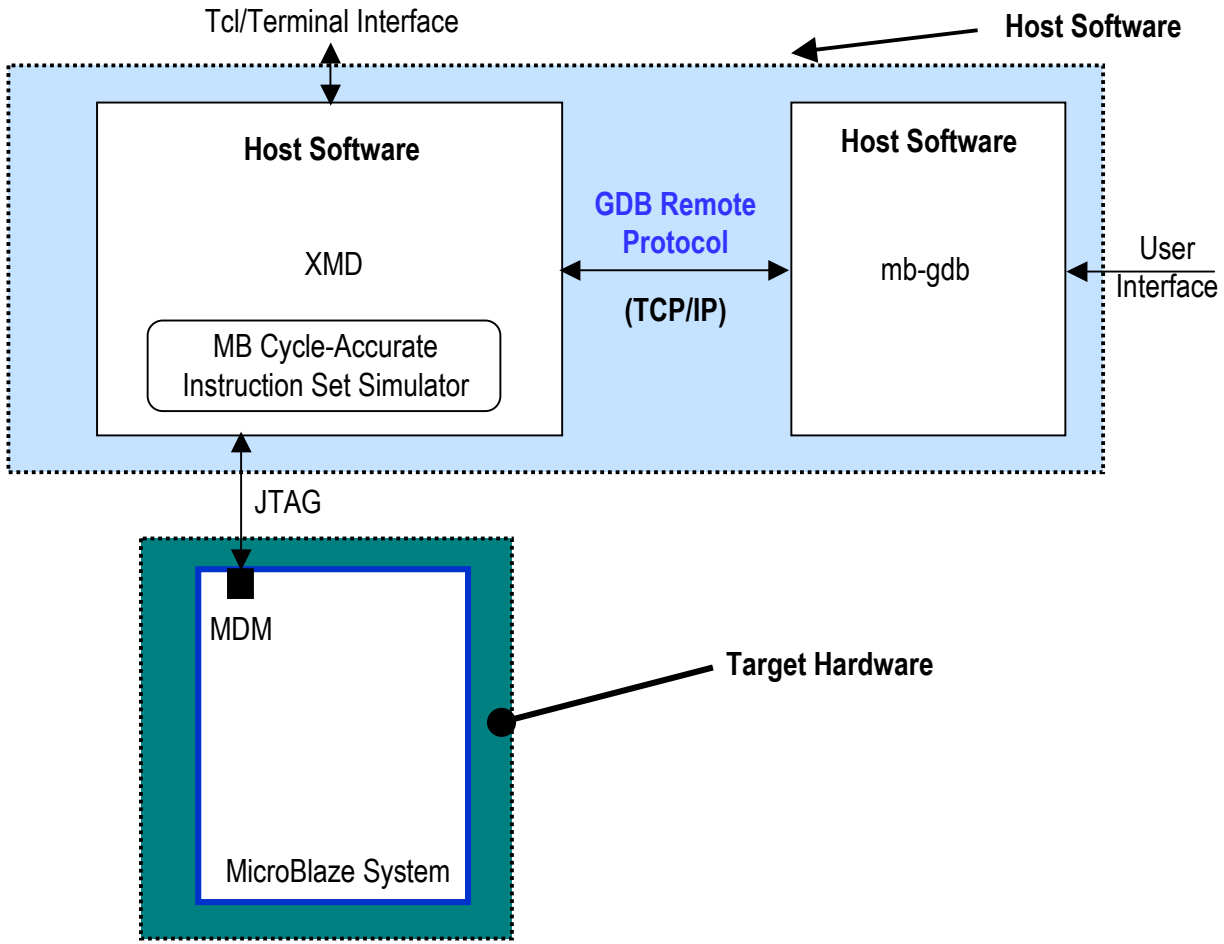
MicroBlaze Simulator Target



MicroBlaze Simulator Target

- **mb-gdb** and **xmd** can be used to debug programs on the cycle-accurate simulator built into XMD
- Simulator target requirements
  - Programs should be compiled for debugging and should be linked with the startup code in crt0.o
  - Programs can have a maximum size of 64 KB only
  - Does not support the simulation of OPB peripherals
- Sample session of XMD and GDB
  - **XMD%** mbconnect sim

Hardware Target



Hardware Target

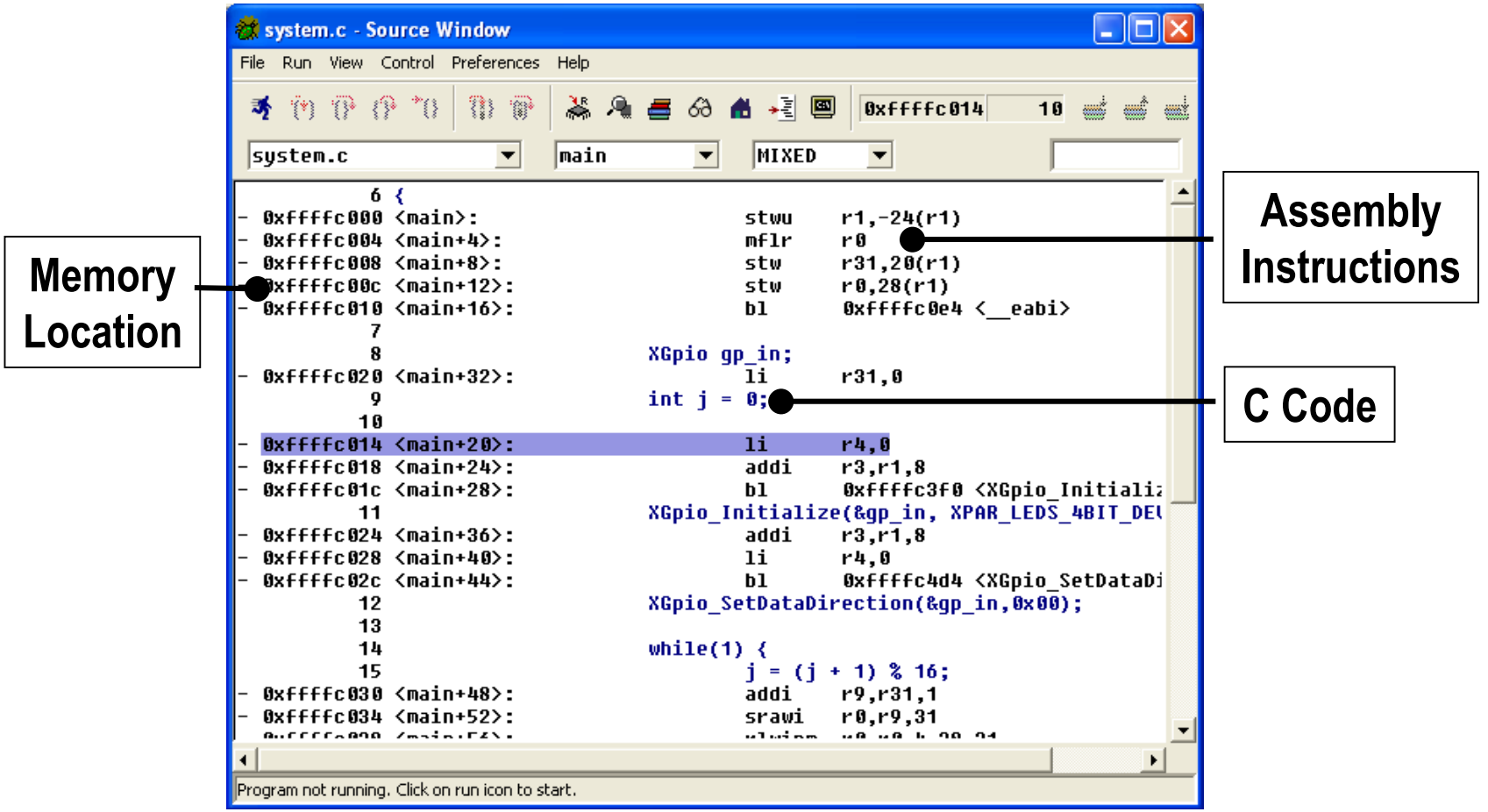
- MicroBlaze™ processor MDM
  - hardware MDM debug peripheral
- The MDM target supports non-intrusive debugging by using:
  - Hardware breakpoints
  - Hardware single step
  - This removes the need to run **xmdstub**
  - This removes the requirement to have large memory
- Sample session of XMD and GDB
  - **XMD% mbconnect** mdm

Outline

- Debugging Tools
- **Software Debug Environments**
  - **XPS**
  - SDK
- Simultaneous HW/SW Debug

XPS Debug Environment

GDB provides graphical interface for viewing source coded



XPS Debug Environment

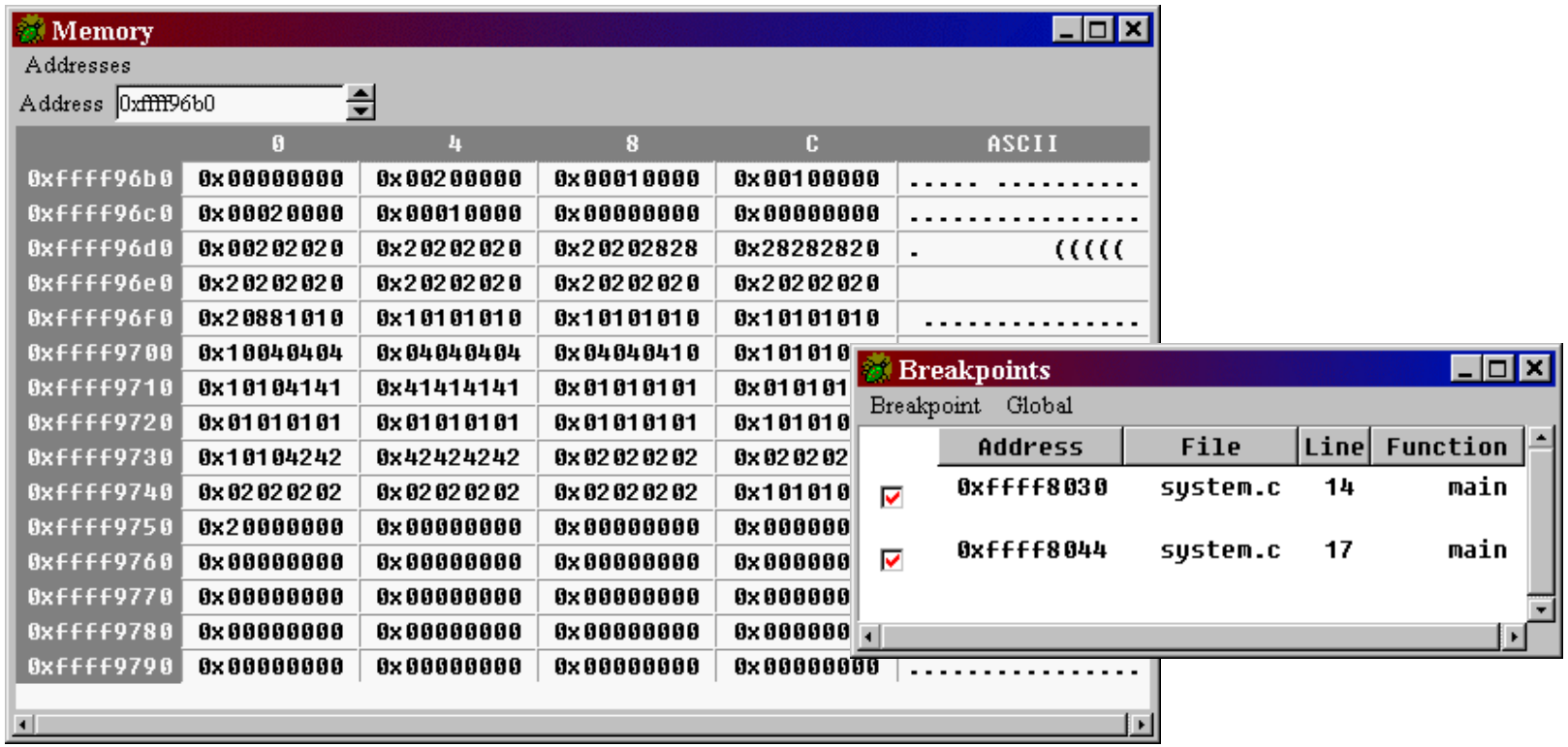
GDB Run-Time Control

- S: Step by source lines (Step into functions) 
- SI: Step by machine instruction 
- C: Continue to next breakpoint 
- N: Next source line (Steps over functions) 
- NI: Next machine instruction 
- F: Finish (Ignores all breakpoints) 

XPS Debug Environment

GDB Functionality

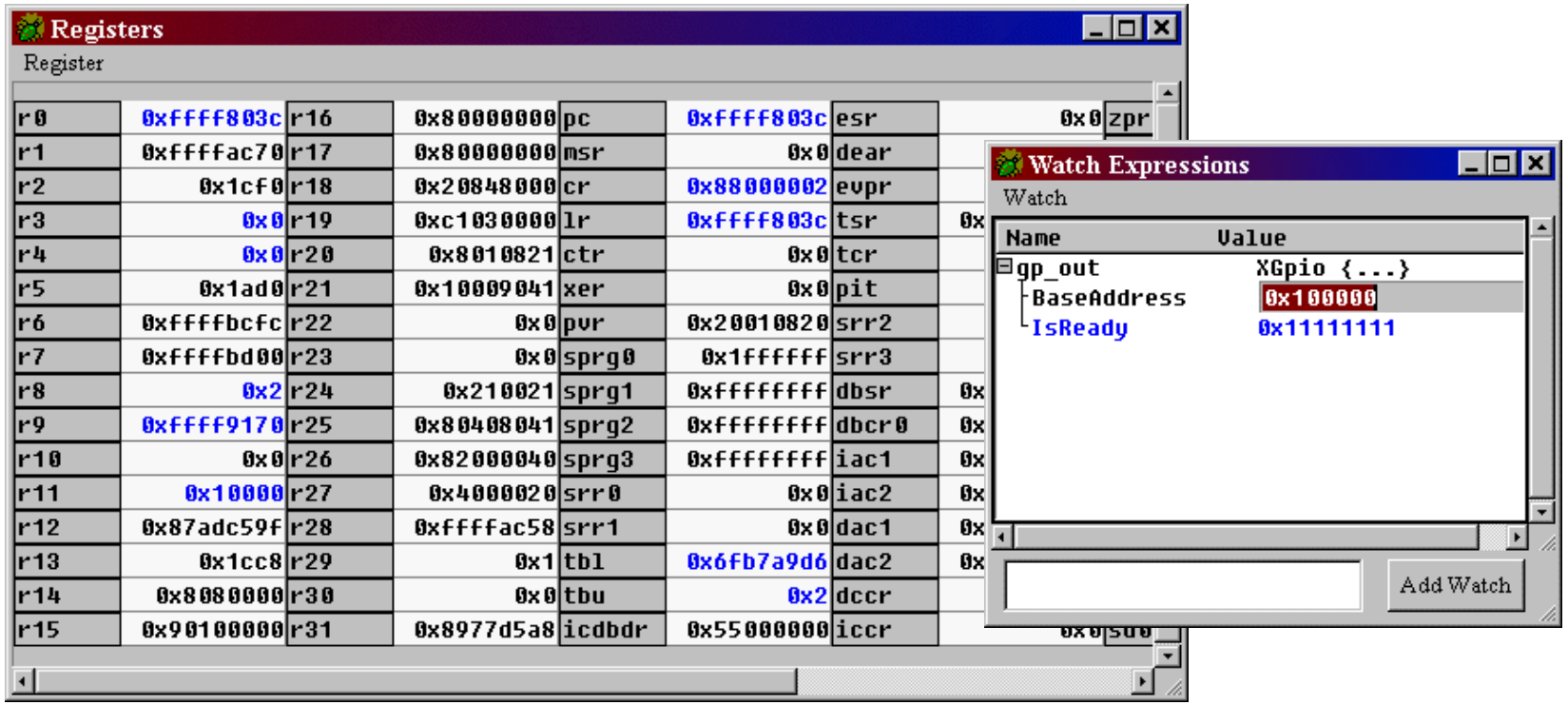
- Breakpoints can be enabled or disabled
- To change any memory value, simply double-click in a memory field



XPS Debug Environment

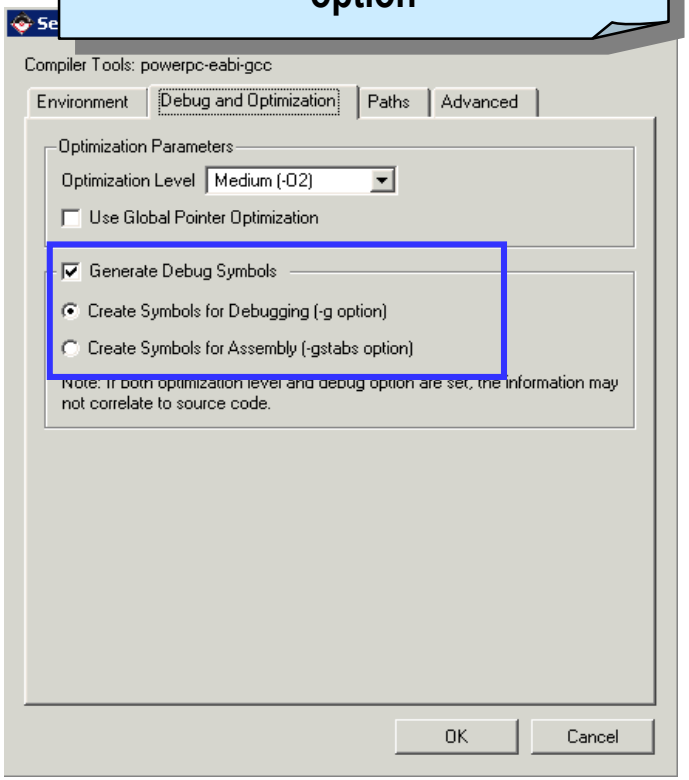
GDB Functionality

- Blue represents registers that have changed
- To change any value, double-click in a field

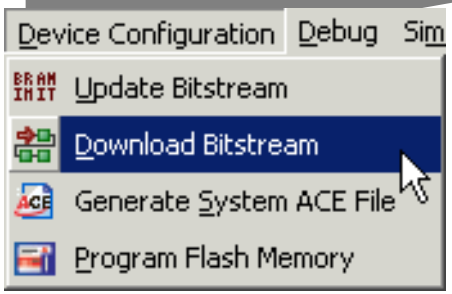


A Debugging Sample

1 Compile with the debugging option



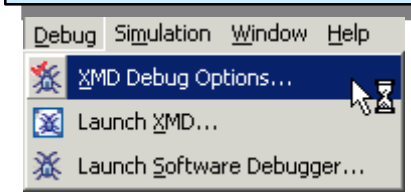
2 Download the bitstream



This will go through the necessary steps, generate a bitstream file, and download the file

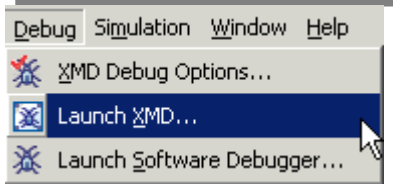
Start XMD

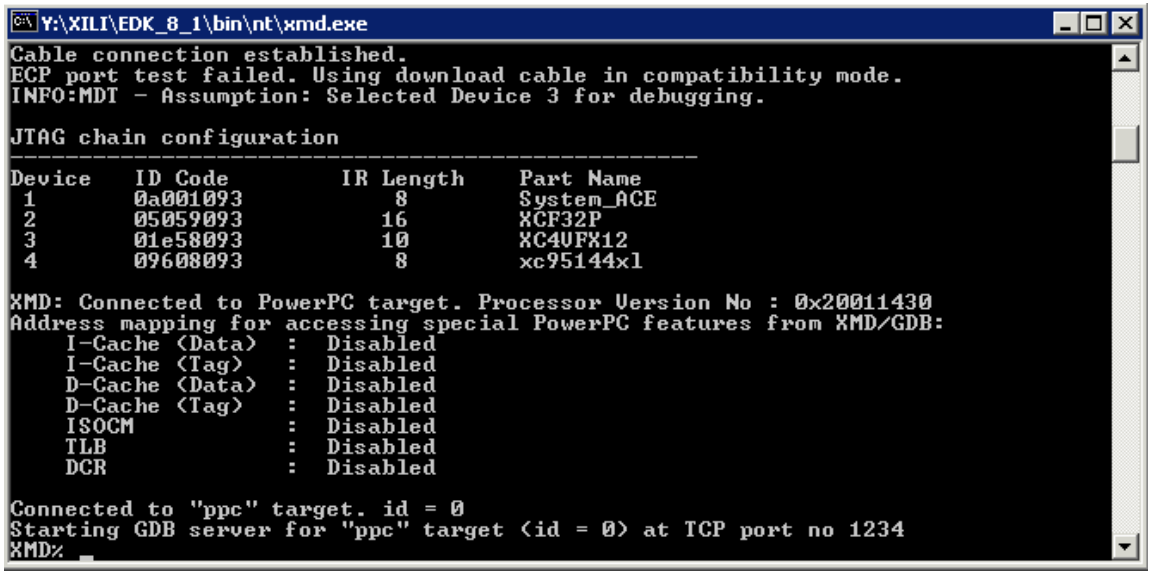
3 **Set XMD Debug Options**



Set connection type and JTAG properties

4 **Start the XMD shell**



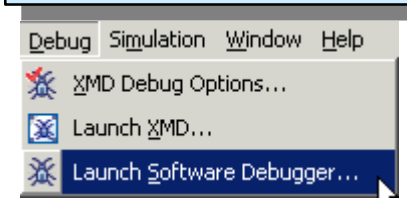


This opens a connection with the hardware, indicating whether the connecting ports and caches are enabled or not

Start Software Debugger

5

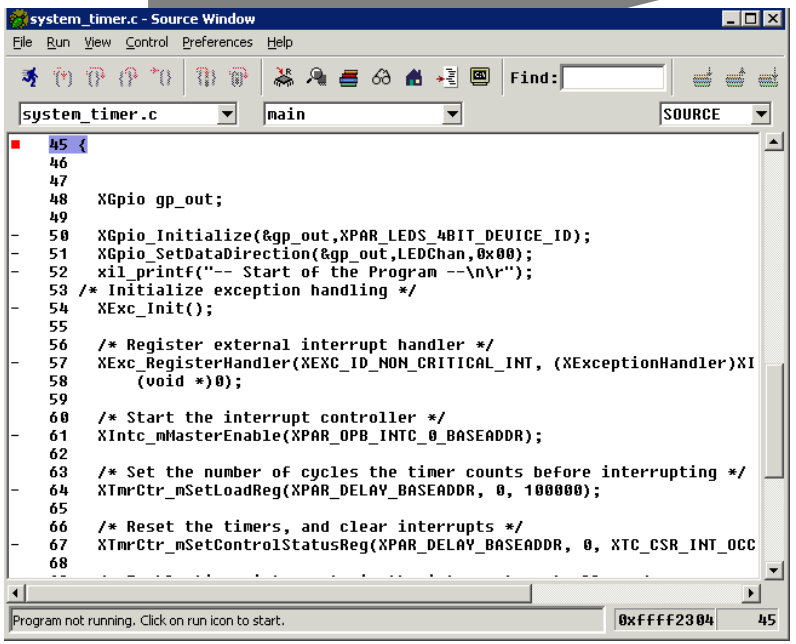
Start Software Debugger



If there are more than one application in the project then the tools will provide choice to select an application

6

A source code window displays

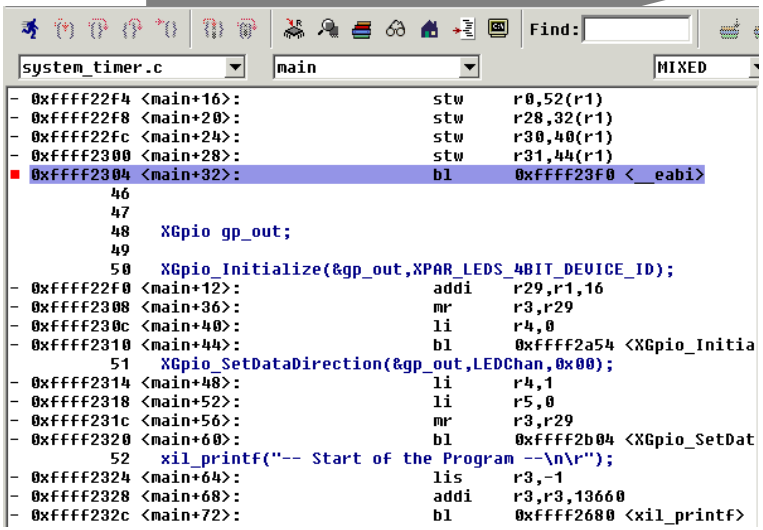


Change the code window display from SOURCE to MIXED to show C and assembly code

Software Debugger Connect

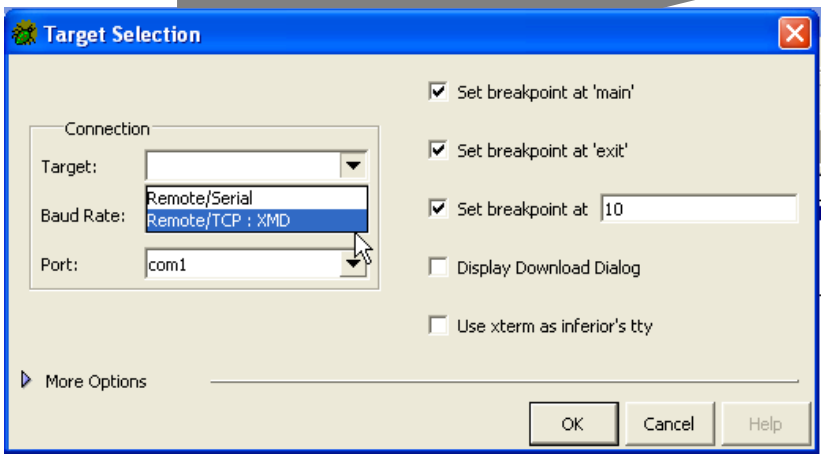
7

A window displaying C and assembly code



8

Select the target as Remote/TCP: XMD



Enter the port number that was displayed when connecting to the target

Debug Program

9 Set any breakpoints as necessary

```
0xffffc030 <main+48>:    addi    r9,r31,1
0xffffc034 <main+52>:    srawi   r0,r9,31
0xffffc038 <main+56>:    rlwinm  r0,r0,4,28,31
0xffffc03c <main+60>:    add     r0,r9,r0
0xffffc040 <main+64>:    rlwinm  r0,r0,0,0,27
0xffffc044 <main+68>:    subf    r31,r0,r9
```

10 Click the Run button

11 When a breakpoint is encountered, the debugger stops

```
0xffffc034 <main+52>:    srawi   r0,r9,31
0xffffc038 <main+56>:    rlwinm  r0,r0,4,28,31
0xffffc03c <main+60>:    add     r0,r9,r0
0xffffc040 <main+64>:    rlwinm  r0,r0,0,0,27
0xffffc044 <main+68>:    subf    r31,r0,r9
```

12 View the necessary windows

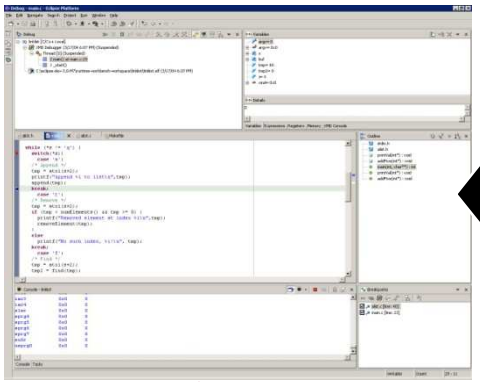
r0	0							zpr	0x0
r1	0xf							pid	0x0
r2	0x4858	r18	0xff237527	cr	0x93000000	evpr	0xcf7b0000	sgr	0xffffffff
r3	0xffffdb88	r19	0x38c2770f	lr	0xffffc05c	tsr	0xfc000000	dcwr	0x0
r4	0x1	r20	0x38f93cc0	ctr	0x0	tcr	0x0	ccr0	0x700000
r5	0x4b84	r21	0x5d8b3553	xer	0xc0000057	pit	0x0	dbcr1	0x0
r6	0x79e62141	r22	0x704291ee	pvr	0x20010820	srr2	0xc8afdbc	dvc1	0xbadfda0e
r7	0x0	r23	0x65b65d3f	sprg0	0x800d2110	srr3	0x0	dvc2	0xbdfbee77
r8	0x79e62141	r24	0x8e890241	sprg1	0xf31bc7b0	dbsr	0x10100000	iac3	0xd3708d8c
r9	0x16	r25	0x9ac88899	sprg2	0xf5820ff6	dbcr0	0x81000000	iac4	0x10980098
r10	0x16	r26	0x3f21bc16	sprg3	0xc0681370	iac1	0x51aa5620	sler	0x0
r11	0x16	r27	0xbbae8a08	srr0	0xc0000	iac2	0xdddac8c	sprg4	0xff9059c5
r12	0x79e62141	r28	0xffffdb88	srr1	0x0	dac1	0x9f0a9ecf	sprg5	0x8c21fb81
r13	0x4ba4	r29	0x1	tbl	0x9ff58281	dac2	0x73ec793	sprg6	0x16c8e6fe
r14	0xc8b00cf3	r30	0x0	tbu	0x16	dccr	0x0	sprg7	0xe3fdfffb
r15	0xfeeaff39	r31	0x1	icdbdr	0x55000000	iccr	0x0	su0r	0x0

Exit the debugger by typing *quit* in the console window

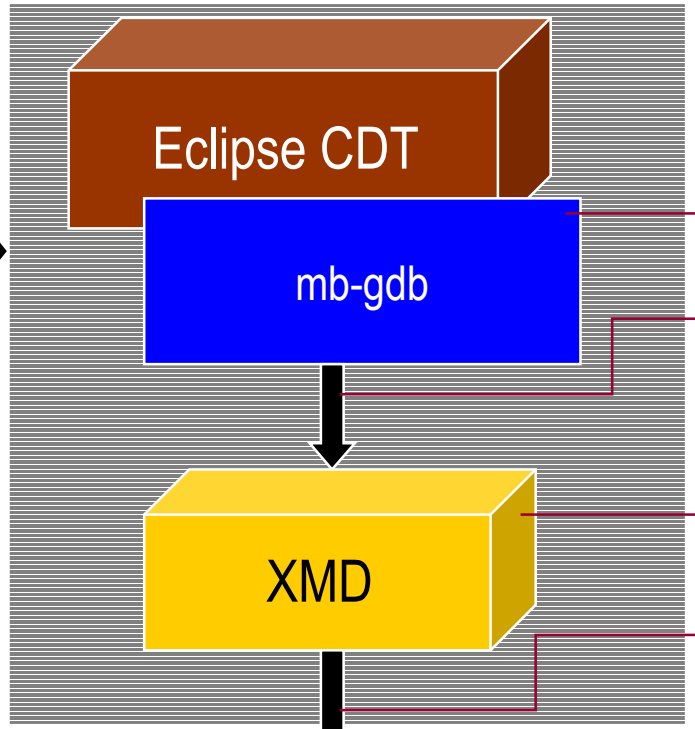
Outline

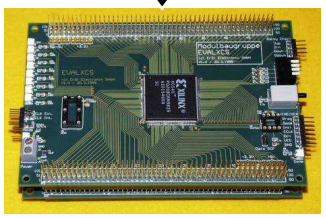
- Debugging Tools
- Software Debug Environments
  - XPS
  - **SDK**
- Simultaneous HW/SW Debug

Debugging Using SDK



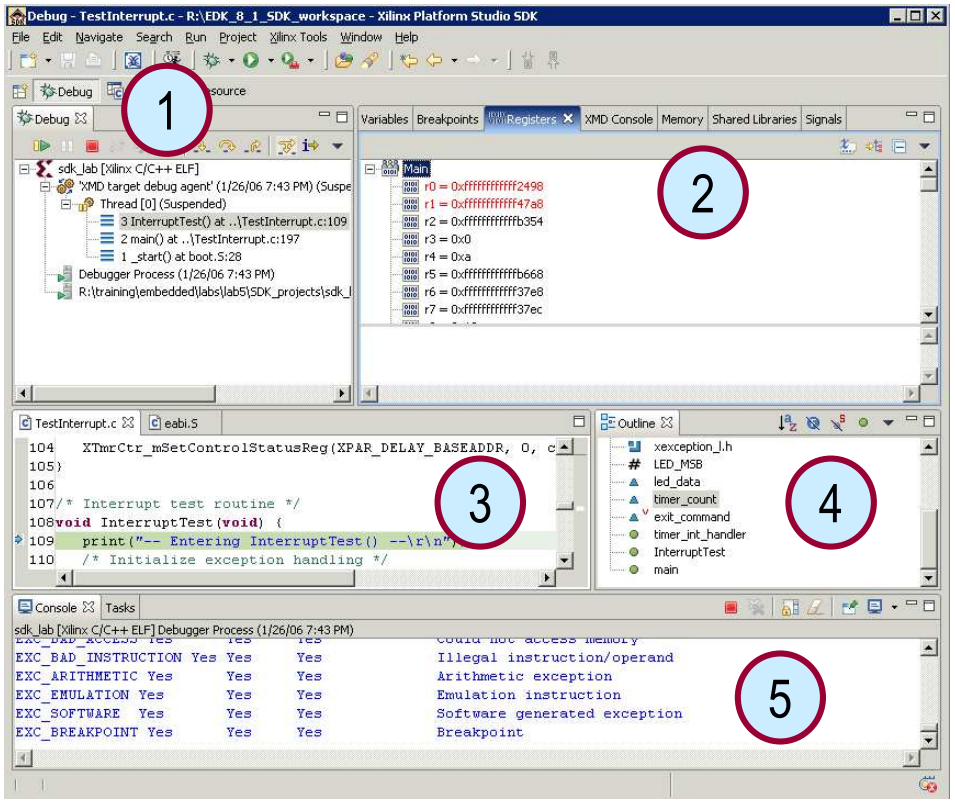
Xilinx custom graphical debug interface





SDK Debug Perspective

1. The stack frame for target threads that you are debugging. Each thread in your program is represented as a node in the tree
2. Variables, Breakpoints, and Registers views allow for viewing and real-time interaction with the view contents for more powerful debugging potential
3. C/C++ editor highlights the location of the execution pointer, along with allowing the setting of breakpoints
4. Code outline and disassembly view provide compiler level insight to what is occurring in the running source
5. Console view lists output information



Debugging in XPS vs SDK

Debugging in XPS

- Download bitstream from XPS
- Launch XMD
- Provide Target Connection Options
- Launch GDB (Insight GUI)
- Set GDB Server connection port in GDB
- Download program
- Begin Debugging

Debugging in SDK

- Download bitstream from XPS/SDK
- ~~Launch XMD~~
- Provide Target Connection Options
- ~~Launch GDB (Insight GUI)~~
- ~~Set GDB Server connection port in GDB~~
- ~~Download program~~
- Begin Debugging

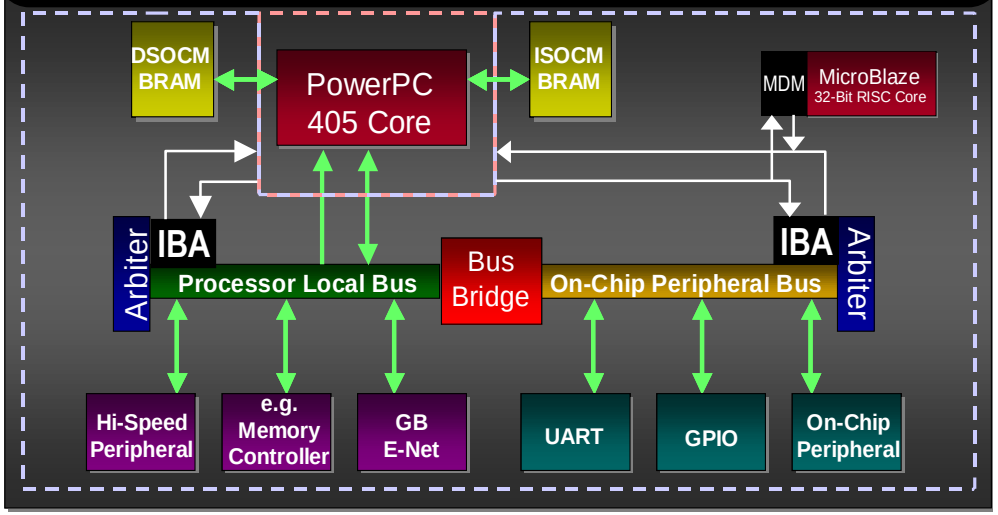
Outline

- Debugging Tools
- Debug Environments
  - XPS
  - SDK
- **Simultaneous HW/SW Debug**

Simultaneous HW/SW Debug

- ChipScope™ Pro IBA & ILA cores in target
- ChipScope Pro Analyzer on host
- GDB debugger on host
- XMD supports simultaneous access over Xilinx parallel cables
- IBA cores available for PLB/OPB
  - Monitors bus transactions
- ILA cores available for IP
  - Monitors signals

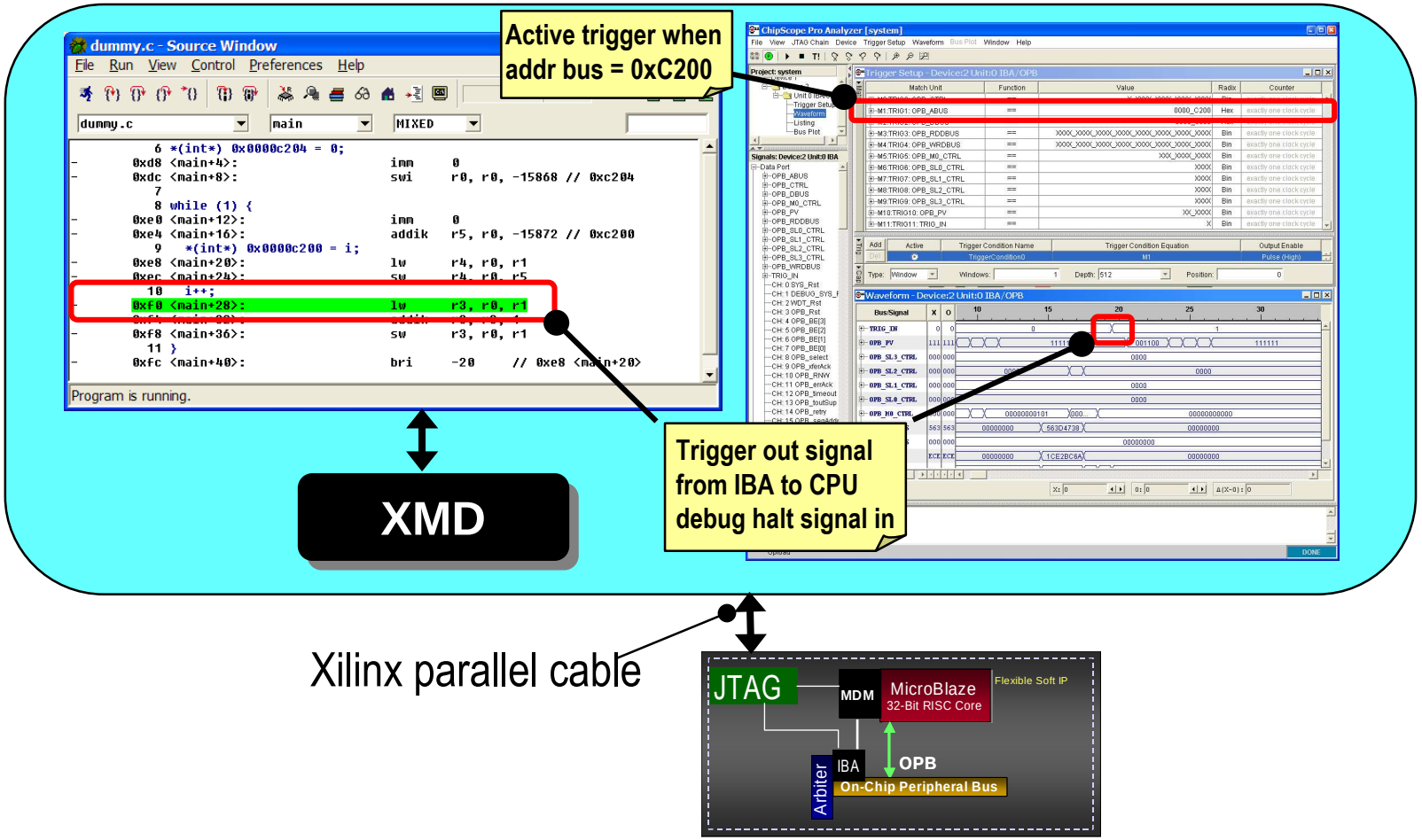
Minimal *skid-by* as cross-triggering is done on chip between IBA cores and PPC and MicroBlaze™ debug interfaces



Set breakpoint in GDB: when hit → triggers the ChipScope tool

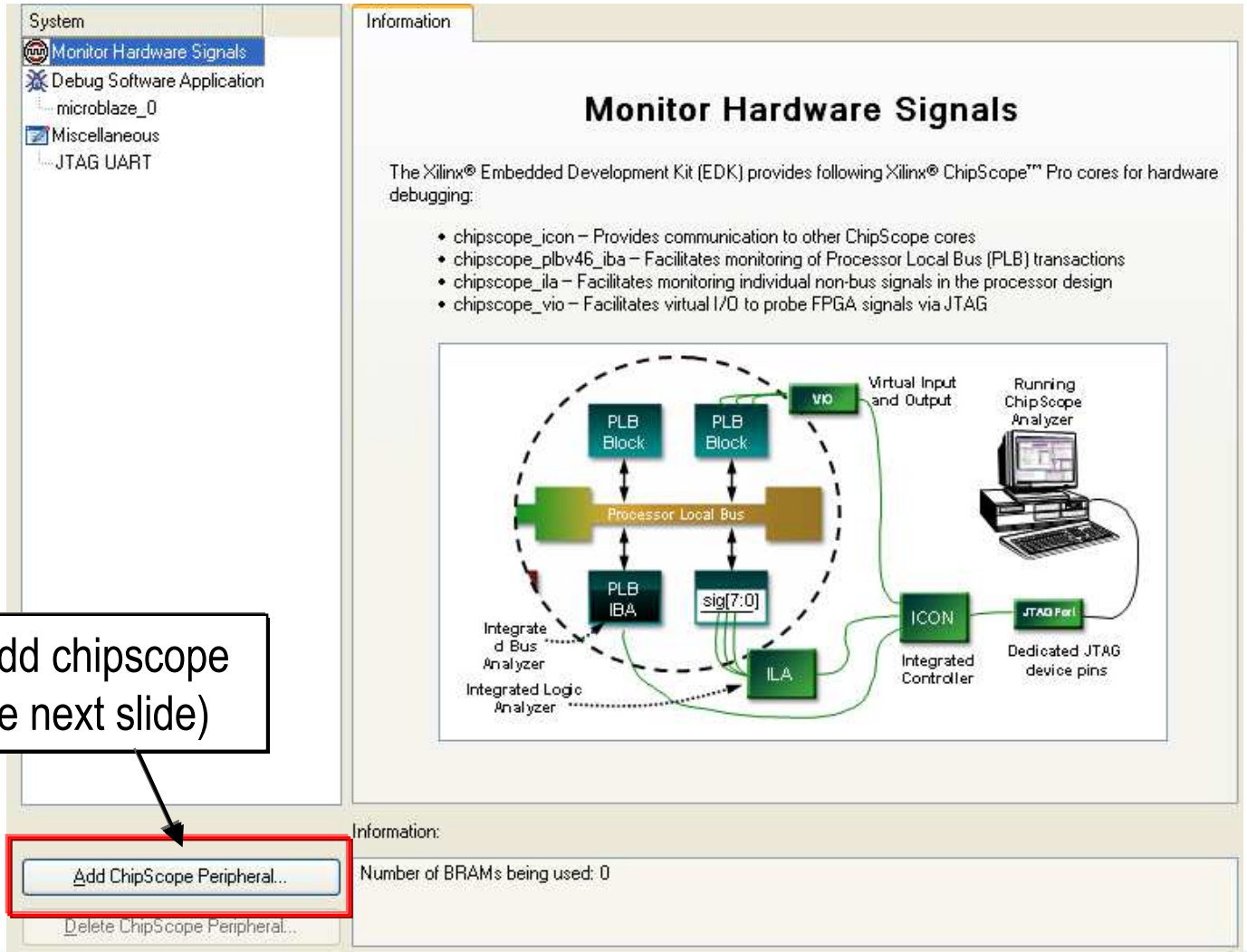
Set trigger in ChipScope: when hit → halts CPU and debugger stops

Simultaneous HW/SW Debug



Insert Chipscope Cores in Design

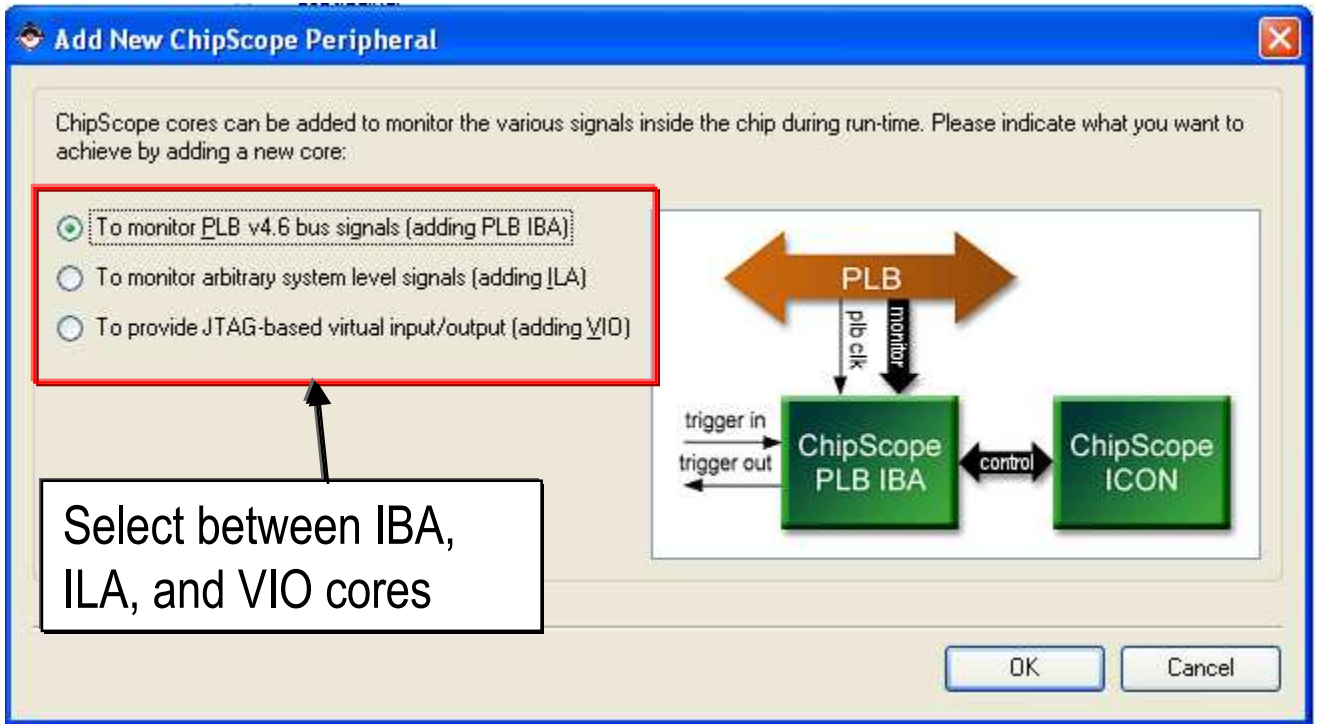
Access the Debug Configuration wizard from within XPS



Click to add chipscope cores (see next slide)

Insert Chipscope Cores into Design

Select and configure Chipscope Pro cores



Knowledge Check

- What tool is used to connect the GNU Debugger to the hardware target?
- What environments are available for debugging Xilinx embedded software applications?
- Into what parts of the design do debuggers provide visibility?

Answers

- What tool is used to connect the GNU Debugger to the hardware target?
  - XMD
- What environments are available for debugging Xilinx embedded software applications?
  - XPS
  - SDK
- Into what parts of the design do debuggers provide visibility?
  - Registers
  - Memory
  - Stack
  - Variables
  - Expressions

Where Can I Learn More?

- Tool documentation
  - ***Embedded System Tools Guide → GNU Compiler Tools***
  - ***Embedded System Tools Guide → GNU Debugger***
  - ***Embedded System Tools Guide → Xilinx Microprocessor Debugger***
  - ***Xilinx Platform Studio SDK Online Documentation***
- Support Website
  - EDK Website: www.xilinx.com/edk