



# 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

The screenshot shows the 'Debug Configuration' window with the 'Monitor Hardware Signals' option selected. The 'Information' tab is active, displaying the following text:

**Monitor Hardware Signals**

The Xilinx® Embedded Development Kit (EDK) provides following Xilinx® ChipScope™ Pro cores for hardware debugging:

- chipscope\_icon – Provides communication to other ChipScope cores
- chipscope\_plbv46\_iba – Facilitates monitoring of Processor Local Bus (PLB) transactions
- chipscope\_ila – Facilitates monitoring individual non-bus signals in the processor design
- chipscope\_vio – Facilitates virtual I/O to probe FPGA signals via JTAG

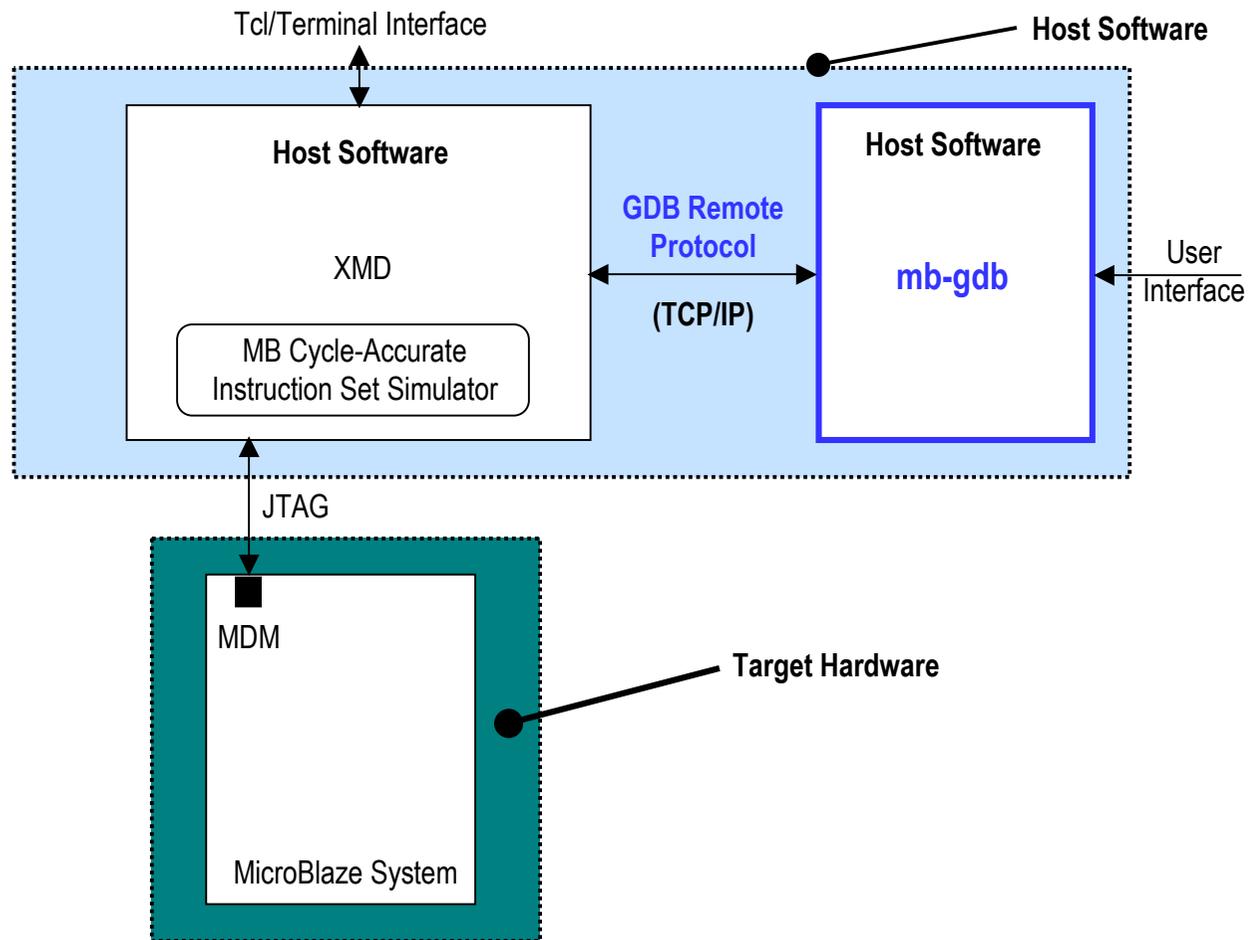
The diagram illustrates the internal architecture. A central 'Processor Local Bus' connects to 'PLB Block' and 'PLB IBA' components. An 'Integrated Bus Analyzer' is connected to the PLB IBA. A 'sig[7:0]' signal is shown, connected to an 'Integrated Logic Analyzer' (ILA). An 'I/O' block is connected to the PLB Block and a 'Virtual Input and Output' block. An 'Integrated Controller' (ICON) is connected to the PLB Block and a 'JTAG File' block. A 'Running ChipScope Analyzer' is shown connected to the JTAG File block via 'Dedicated JTAG device pins'.

Information:  
Number of BRAMs being used: 0

# 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

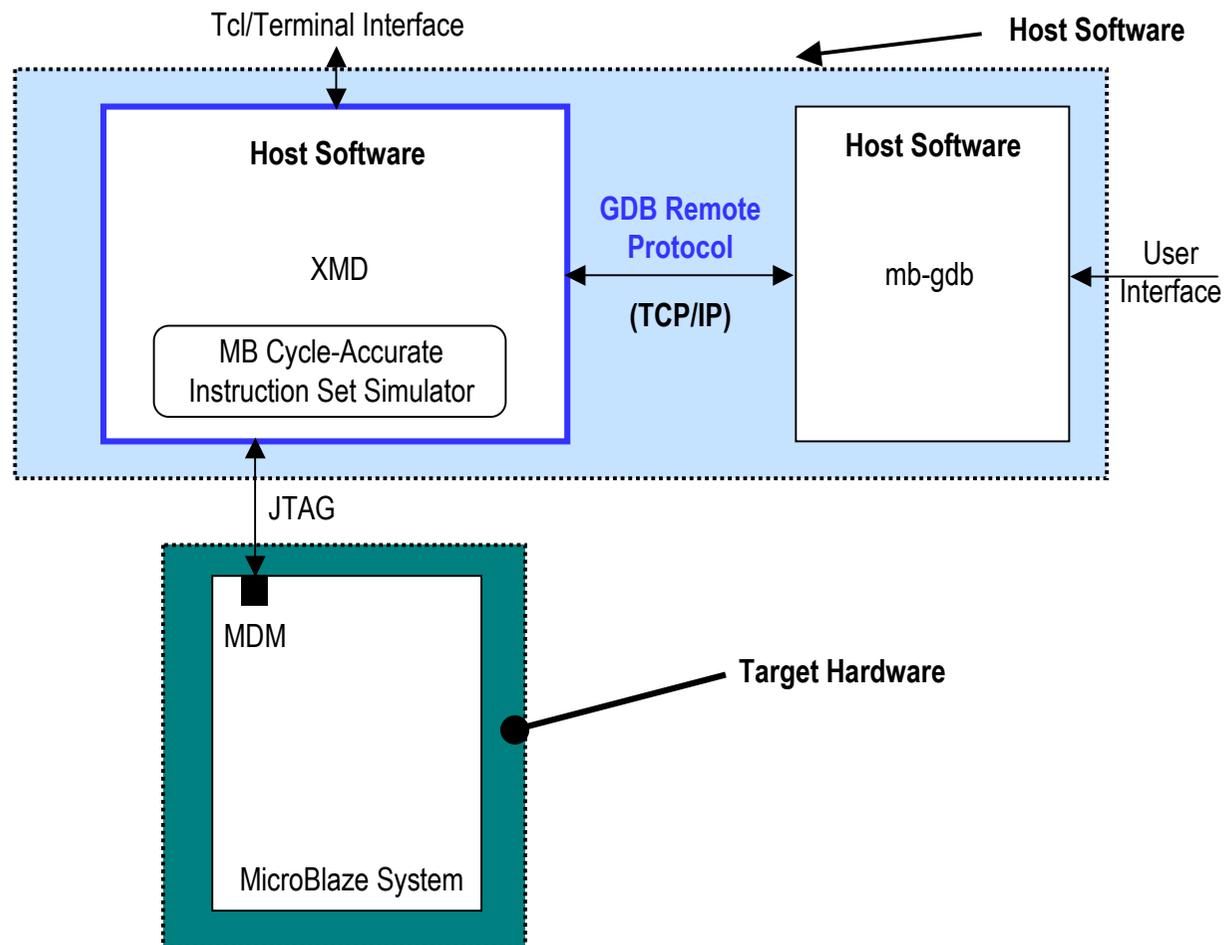
# GDB Functionality



# 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

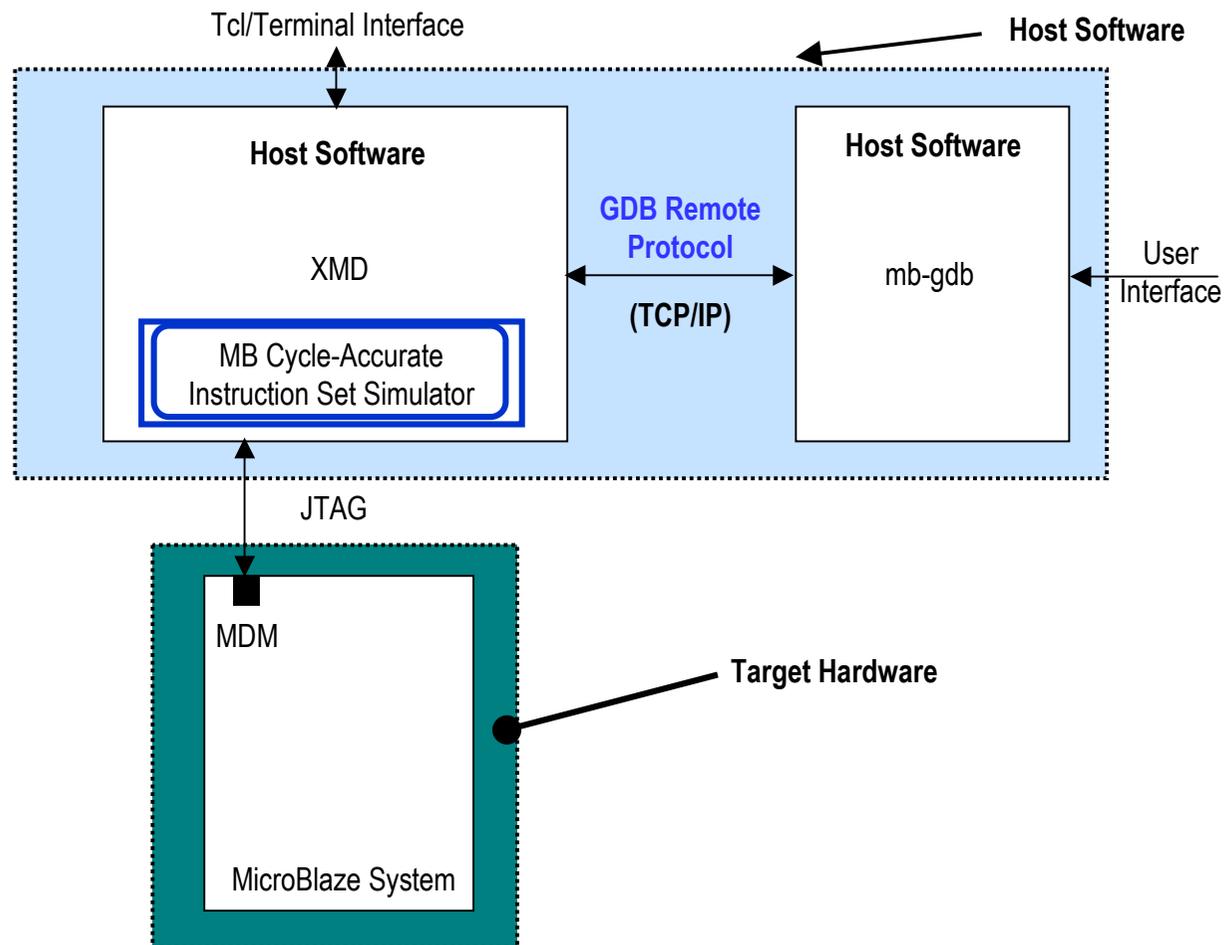
# XMD Options

- 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

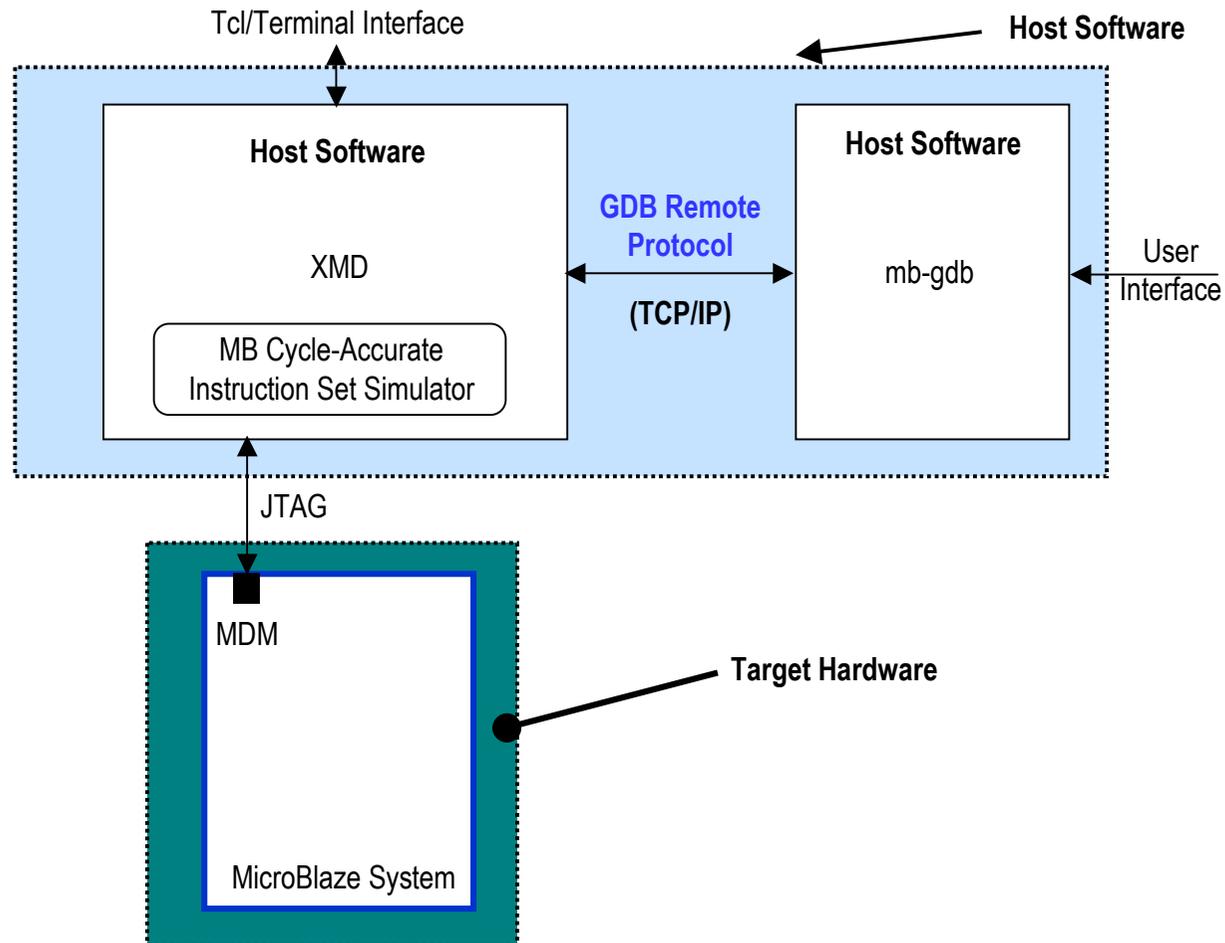
# MicroBlaze Simulator Target



# 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 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

The screenshot displays two windows from the XPS Debug Environment. The 'Memory' window shows a table of memory addresses and their corresponding values in hexadecimal and ASCII. The 'Breakpoints' window shows a list of active breakpoints.

**Memory Window:**

Address	0	4	8	C	ASCII
0xffff96b0	0x00000000	0x00200000	0x00010000	0x00100000	.....
0xffff96c0	0x00020000	0x00010000	0x00000000	0x00000000	.....
0xffff96d0	0x00202020	0x20202020	0x20202828	0x28282820	. (((((
0xffff96e0	0x20202020	0x20202020	0x20202020	0x20202020	
0xffff96f0	0x20881010	0x10101010	0x10101010	0x10101010	.....
0xffff9700	0x10040404	0x04040404	0x04040410	0x10101010	
0xffff9710	0x10104141	0x41414141	0x01010101	0x01010101	
0xffff9720	0x01010101	0x01010101	0x01010101	0x10101010	
0xffff9730	0x10104242	0x42424242	0x02020202	0x02020202	
0xffff9740	0x02020202	0x02020202	0x02020202	0x10101010	
0xffff9750	0x20000000	0x00000000	0x00000000	0x00000000	
0xffff9760	0x00000000	0x00000000	0x00000000	0x00000000	
0xffff9770	0x00000000	0x00000000	0x00000000	0x00000000	
0xffff9780	0x00000000	0x00000000	0x00000000	0x00000000	
0xffff9790	0x00000000	0x00000000	0x00000000	0x00000000	.....

**Breakpoints Window:**

Breakpoint	Global	Address	File	Line	Function
<input checked="" type="checkbox"/>		0xffff8030	system.c	14	main
<input checked="" type="checkbox"/>		0xffff8044	system.c	17	main



# XPS Debug Environment

## GDB Functionality

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

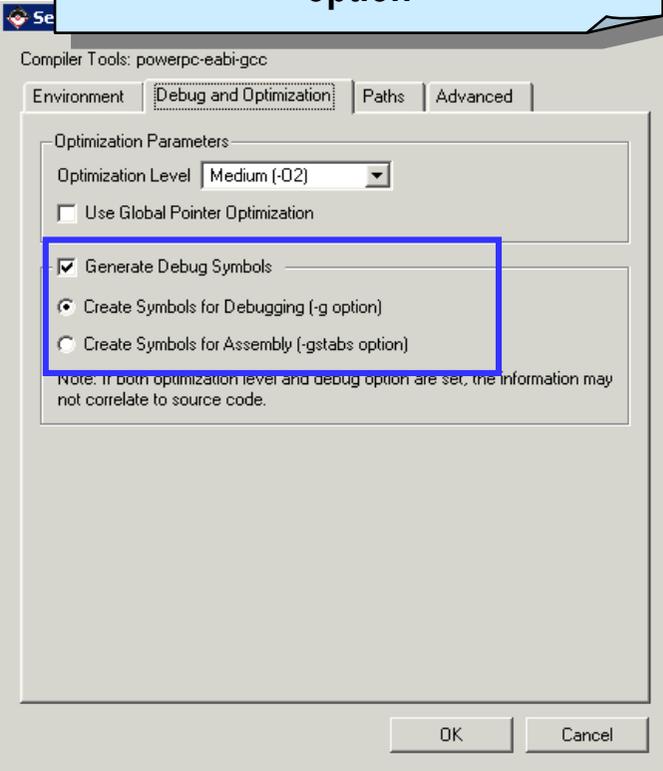
The screenshot displays the XPS Debug Environment interface. The 'Registers' window shows a list of registers with their names, addresses, and values. The 'Watch Expressions' window shows a tree view of watched expressions with their names and values.

Register	Value	Register	Value	Register	Value	Register	Value
r0	0xffff803c	r16	0x80000000	pc	0xffff803c	esr	0x0
r1	0xffffac70	r17	0x80000000	msr	0x0	dear	
r2	0x1cf0	r18	0x20848000	cr	0x88000002	evpr	
r3	0x0	r19	0xc1030000	lr	0xffff803c	tsr	0x0
r4	0x0	r20	0x8010821	ctr	0x0	tcr	0x0
r5	0x1ad0	r21	0x10009041	xer	0x0	pit	0x0
r6	0xffffbcbfc	r22	0x0	pvr	0x20010820	srr2	0x0
r7	0xffffbd00	r23	0x0	sprg0	0x1fffffff	srr3	0x0
r8	0x2	r24	0x210021	sprg1	0xffffffff	dbcr0	0x0
r9	0xffff9170	r25	0x80408041	sprg2	0xffffffff	dbcr1	0x0
r10	0x0	r26	0x82000040	sprg3	0xffffffff	iac1	0x0
r11	0x10000	r27	0x4000020	srr0	0x0	iac2	0x0
r12	0x87adc59f	r28	0xffffac58	srr1	0x0	dac1	0x0
r13	0x1cc8	r29	0x1	tbl	0x6fb7a9d6	dac2	0x0
r14	0x80800000	r30	0x0	tbu	0x2	dccr	0x0
r15	0x90100000	r31	0x8977d5a8	icbdr	0x55000000	iccr	0x0

Name	Value
gp_out	XGpio {...}
BaseAddress	0x100000
IsReady	0x11111111

# A Debugging Sample

**1** Compile with the debugging option



Compiler Tools: powerpc-eabi-gcc

Environment **Debug and Optimization** Paths Advanced

Optimization Parameters

Optimization Level Medium (-O2)

Use Global Pointer Optimization

Generate Debug Symbols

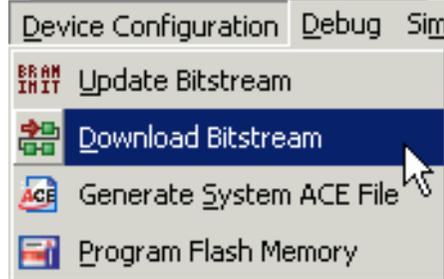
Create Symbols for Debugging (-g option)

Create Symbols for Assembly (-gstabs option)

Note: If both optimization level and debug option are set, the information may not correlate to source code.

OK Cancel

**2** Download the bitstream



Device Configuration **Debug** Sim

Update Bitstream

Download Bitstream

Generate System ACE File

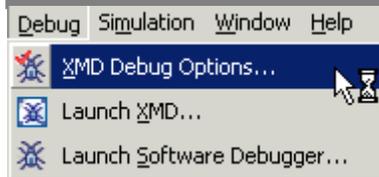
Program Flash Memory

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

# Start XMD

3

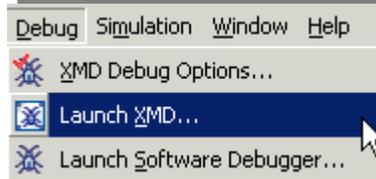
## Set XMD Debug Options



Set connection type and JTAG properties

4

## Start the XMD shell



```
Y:\XILI\EDK_8_1\bin\nt\xmd.exe
Cable connection established.
ECP port test failed. Using download cable in compatibility mode.
INFO:MDT - Assumption: Selected Device 3 for debugging.

JTAG chain configuration
-----
Device  ID Code      IR Length  Part Name
 1      0a001093         8      System_ACE
 2      05059093        16      XCF32P
 3      01e58093        10      XC4UFX12
 4      09608093         8      xc95144x1

XMD: Connected to PowerPC target. Processor Version No : 0x20011430
Address mapping for accessing special PowerPC features from XMD/GDB:
I-Cache <Data>  : Disabled
I-Cache <Tag>   : Disabled
D-Cache <Data>  : Disabled
D-Cache <Tag>   : Disabled
ISOCM          : Disabled
TLB            : Disabled
DCR            : Disabled

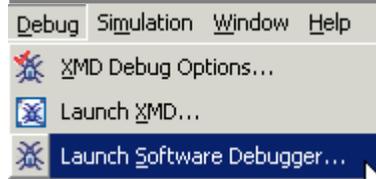
Connected to "ppc" target. id = 0
Starting GDB server for "ppc" target (id = 0) at TCP port no 1234
XMD%_
```

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

# Start Software Debugger

5

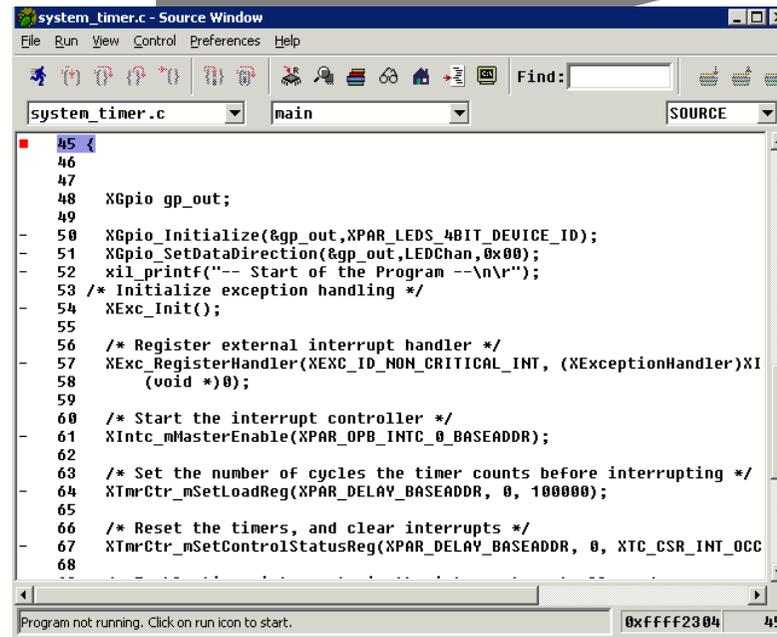
Start Software Debugger



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

A screenshot of a source code window titled 'system\_timer.c - Source Window'. The window shows C code for a timer application. The code includes comments and function calls like XGpio\_initialize, XExc\_Init, and XIntc\_mMasterEnable. The status bar at the bottom indicates 'Program not running. Click on run icon to start.' and shows memory address '0xFFFF2304' and line number '45'.

```
45  
46  
47  
48 XGpio gp_out;  
49  
50 XGpio_initialize(&gp_out, XPAR_LEDS_4BIT_DEVICE_ID);  
51 XGpio_SetDataDirection(&gp_out, LEDChan, 0x00);  
52 xil_printf("-- Start of the Program --\n\r");  
53 /* Initialize exception handling */  
54 XExc_Init();  
55  
56 /* Register external interrupt handler */  
57 XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT, (XExceptionHandler)XI  
58 (void *)0);  
59  
60 /* Start the interrupt controller */  
61 XIntc_mMasterEnable(XPAR_OPB_INTC_0_BASEADDR);  
62  
63 /* Set the number of cycles the timer counts before interrupting */  
64 XTrmCtr_mSetLoadReg(XPAR_DELAY_BASEADDR, 0, 100000);  
65  
66 /* Reset the timers, and clear interrupts */  
67 XTrmCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0, XTC_CSR_INT_OCC  
68
```

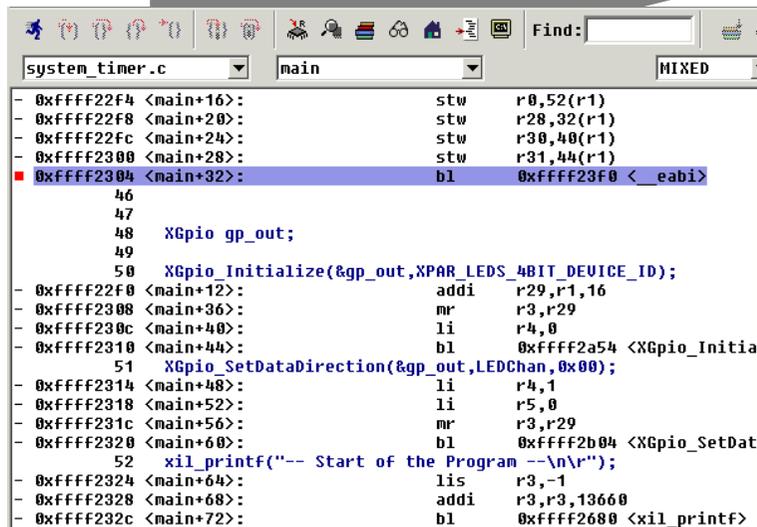
Change the code window display from SOURCE to MIXED to show C and assembly code



# Software Debugger Connect

7

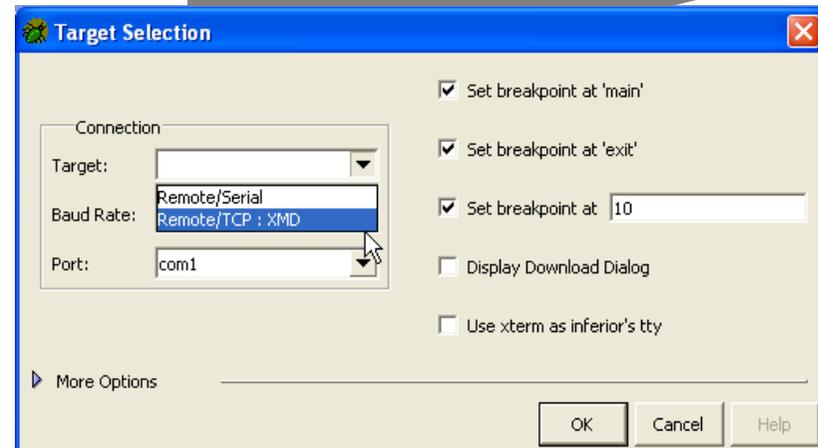
A window displaying  
C and assembly code



```
system_tiner.c  main  MIXED
0xffff22f4 <main+16>: stw  r0,52(r1)
0xffff22f8 <main+20>: stw  r28,32(r1)
0xffff22fc <main+24>: stw  r30,40(r1)
0xffff2300 <main+28>: stw  r31,44(r1)
0xffff2304 <main+32>: b1   0xffff23f0 <_eabi>
46
47
48  XGpio_gp_out;
49
50  XGpio_Initialize(&gp_out,XPBAR_LEDS_4BIT_DEVICE_ID);
0xffff22f0 <main+12>: addi r29,r1,16
0xffff2308 <main+36>: mr   r3,r29
0xffff230c <main+40>: li   r4,0
0xffff2310 <main+44>: b1   0xffff2a54 <XGpio_Initia
51  XGpio_SetDataDirection(&gp_out,LEDChan,0x00);
0xffff2314 <main+48>: li   r4,1
0xffff2318 <main+52>: li   r5,0
0xffff231c <main+56>: mr   r3,r29
0xffff2320 <main+60>: b1   0xffff2b04 <XGpio_SetDat
52  xil_printf("-- Start of the Program --\n\r");
0xffff2324 <main+64>: lis  r3,-1
0xffff2328 <main+68>: addi r3,r3,13660
0xffff232c <main+72>: b1   0xffff2680 <xil_printf>
```

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

```

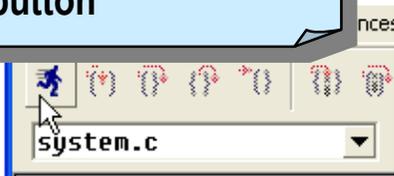
0xfffffc030 <main+48>:      addi    r9,r31,1
- 0xfffffc034 <main+52>:      srawi   r0,r9,31
- 0xfffffc038 <main+56>:      rlwinm  r0,r0,4,28,31
0xfffffc03c <main+60>:      add     r0,r9,r0
- 0xfffffc040 <main+64>:      rlwinm  r0,r0,0,0,27
- 0xfffffc044 <main+68>:      subf   r31,r0,r9
    
```

**11** When a breakpoint is encountered, the debugger stops

```

- 0xfffffc034 <main+52>:      srawi   r0,r9,31
- 0xfffffc038 <main+56>:      rlwinm  r0,r0,4,28,31
- 0xfffffc03c <main+60>:      add     r0,r9,r0
0xfffffc040 <main+64>:      rlwinm  r0,r0,0,0,27
- 0xfffffc044 <main+68>:      subf   r31,r0,r9
    
```

**10** Click the Run button



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

**12** View the necessary windows

Reg	Value	Reg	Value	Reg	Value	Reg	Value
r0	0x0	r18	0xff237527	cr	0x93000000	evpr	0xcf7b0000
r1	0xf	r19	0x38c2770f	lr	0xffffc05c	tsr	0xfc000000
r2	0x4858	r20	0x38f93cc0	ctr	0x0	tcr	0x0
r3	0xffffdb88	r21	0x5d8b3553	xer	0xc0000057	pit	0x0
r4	0x1	r22	0x704291ee	pur	0x20010820	srr2	0xc8afdbc
r5	0x4b04	r23	0x65b65d3f	sprg0	0x800d2110	srr3	0x0
r6	0x79e62141	r24	0x8e890241	sprg1	0xf31bc7b0	dbsr	0x10100000
r7	0x0	r25	0x9ac88899	sprg2	0xf5820ff6	dbcrr0	0x81000000
r8	0x79e62141	r26	0x3f21bc16	sprg3	0xc0681370	iac1	0x51aa5620
r9	0x16	r27	0xbbae8a08	srr0	0xc0000	iac2	0xdddac8c
r10	0x16	r28	0xffffdb88	srr1	0x0	dac1	0x9f0a9ecf
r11	0x16	r29	0x1	tb1	0x9ff58281	dac2	0x73ec793
r12	0x79e62141	r30	0x0	tbu	0x16	dccr	0x0
r13	0x4ba4	r31	0x1	icbdr	0x55000000	iccr	0x0
r14	0xc8b00cf3					su0r	0x0
r15	0xf0eaff39						



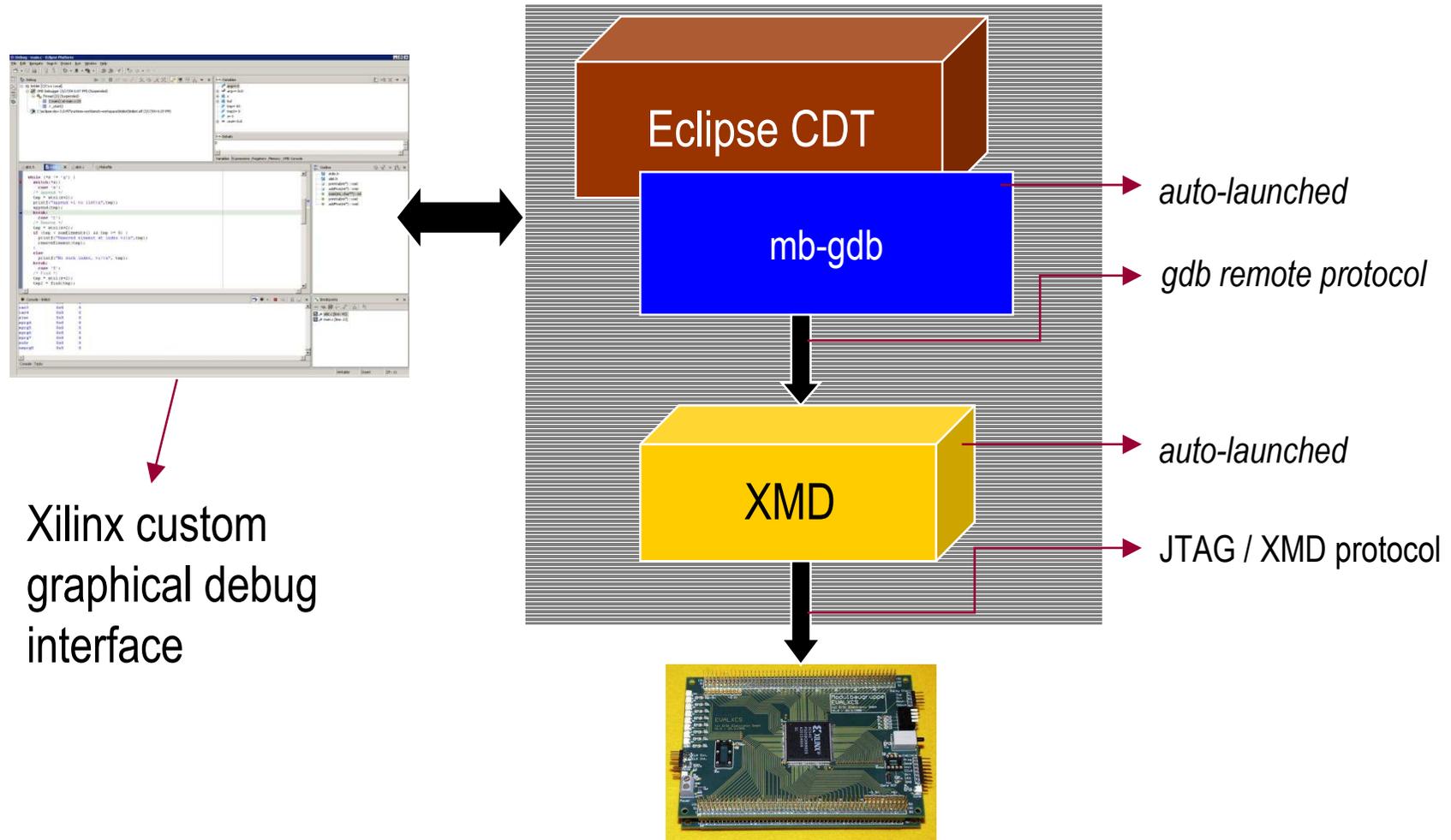
# Outline



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



# Debugging Using SDK



Xilinx custom graphical debug interface

*auto-launched*

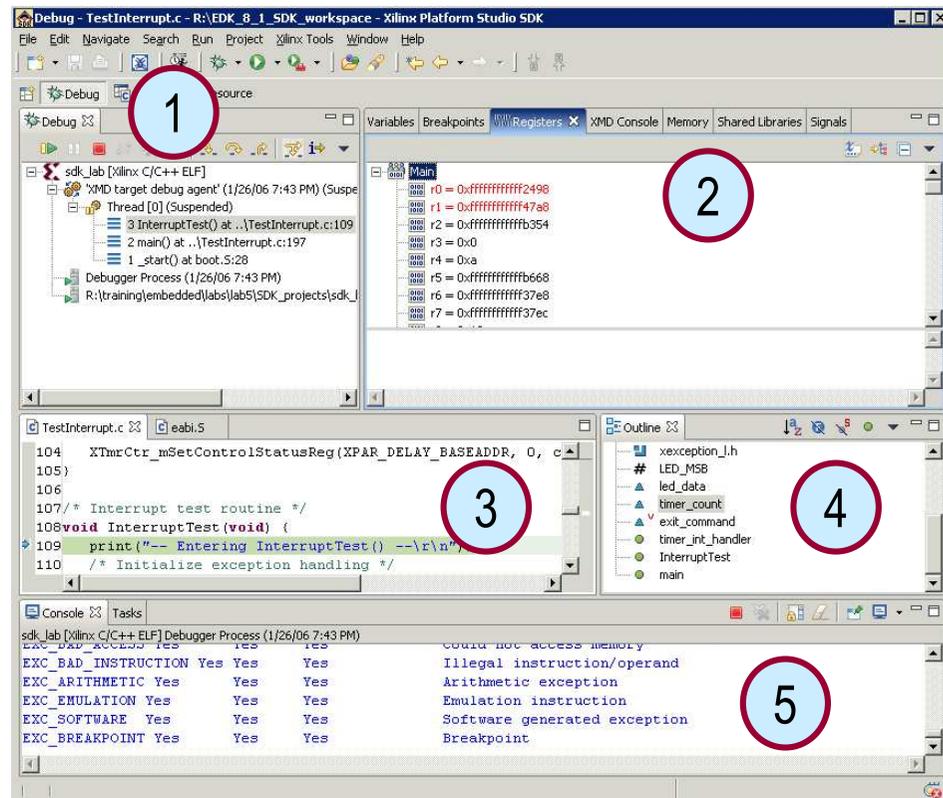
*gdb remote protocol*

*auto-launched*

*JTAG / XMD protocol*

# 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

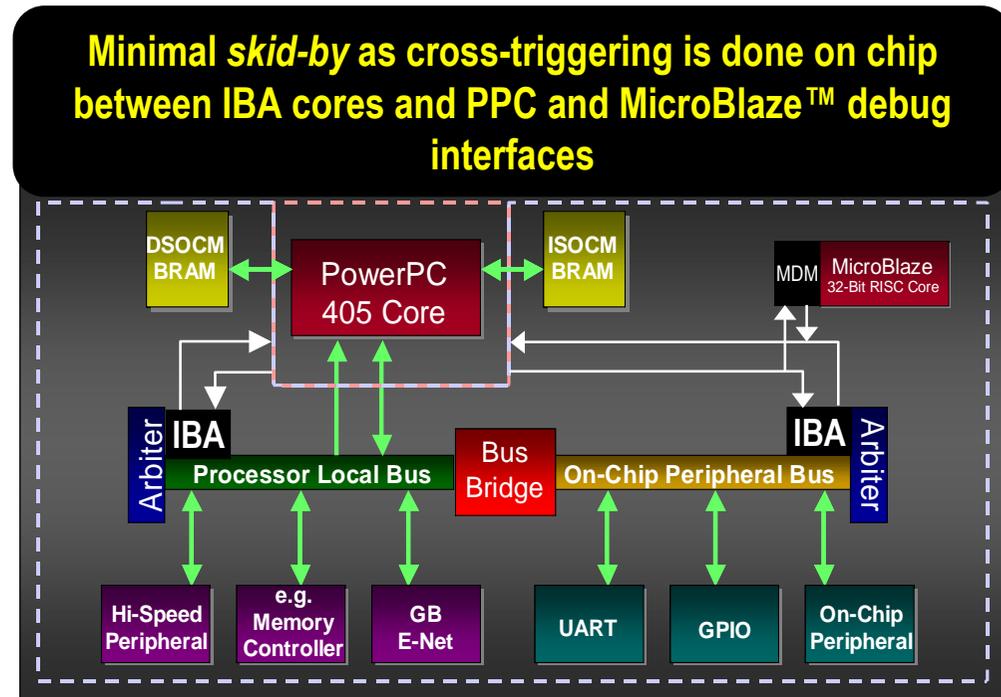


- **Simultaneous HW/SW Debug**



# 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



**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

The screenshot shows the ChipScope Pro Analyzer interface. On the left, the 'Source Window' displays the C code for 'dummy.c'. The line `10 i++;` is highlighted in green, and the instruction `lwr r3, r0, r1` is highlighted in red. A yellow callout box points to this line with the text 'Active trigger when addr bus = 0xC200'. On the right, the 'Trigger Setup' window shows a table of triggers. The first row is selected and highlighted in red:

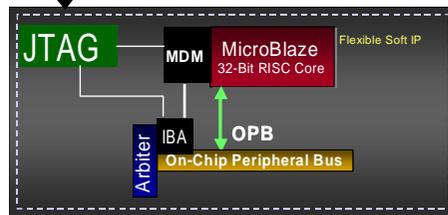
Match Unit	Function	Value	Radix	Counter
M1:TRIO1:OPB_ABUS		0000_C200	Hex	0

Below the table, the 'Waveform' window shows a signal trace for 'OPB\_ABUS' with a red box around a '1' at time 10, indicating the trigger event.

**XMD**

Trigger out signal from IBA to CPU debug halt signal in

Xilinx parallel cable



# Insert Chipscope Cores in Design

Access the Debug Configuration wizard from within XPS

The screenshot shows the 'Monitor Hardware Signals' wizard in XPS. The left pane lists system components: 'Monitor Hardware Signals' (selected), 'Debug Software Application', 'microblaze\_0', 'Miscellaneous', and 'JTAG UART'. The right pane, titled 'Information', contains the following text:

### Monitor Hardware Signals

The Xilinx® Embedded Development Kit (EDK) provides following Xilinx® ChipScope™ Pro cores for hardware debugging:

- chipscope\_icon – Provides communication to other ChipScope cores
- chipscope\_plbv46\_iba – Facilitates monitoring of Processor Local Bus (PLB) transactions
- chipscope\_ila – Facilitates monitoring individual non-bus signals in the processor design
- chipscope\_vio – Facilitates virtual I/O to probe FPGA signals via JTAG

Below the text is a diagram illustrating the hardware configuration. It shows a 'Processor Local Bus' connecting to 'PLB Block', 'PLB IBA', and 'VIO' blocks. An 'Integrated Bus Analyzer' is connected to the PLB IBA, and an 'Integrated Logic Analyzer' is connected to a 'sig[7:0]' signal. An 'ICON Integrated Controller' is connected to the VIO block and a 'JTAG Port' (Dedicated JTAG device pins). A 'Running ChipScope Analyzer' is shown connected to the JTAG Port. The bottom pane of the wizard shows the 'Add ChipScope Peripheral...' button highlighted with a red box.

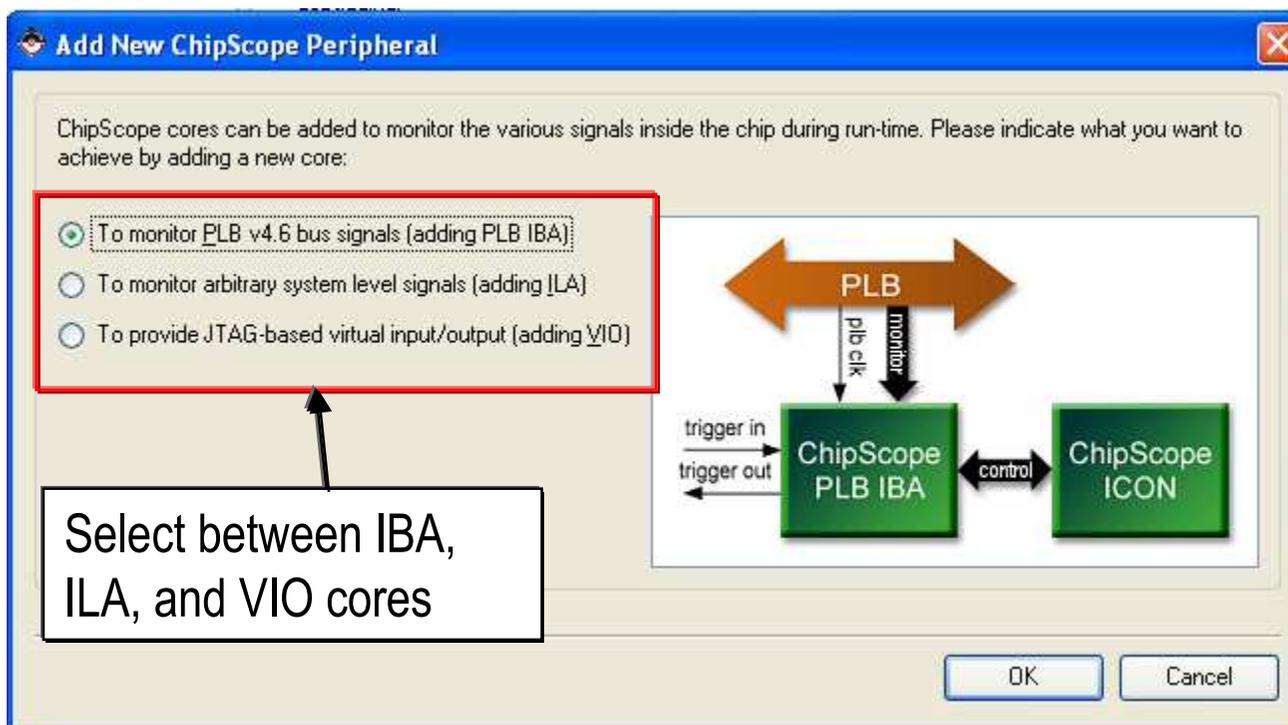
Click to add chipscope cores (see next slide)

Add ChipScope Peripheral...



# 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](http://www.xilinx.com/edk)