Embedded System Debug
Introduction

- Debugging is an integral part of embedded systems development
  - testing
  - Stabilizing, localizing, and correcting errors
- Debugging types
  - Functional debugging
  - Performance debugging
- Challenges
  - target system may be hard to observe;
  - target may be hard to control;
  - may be hard to generate realistic inputs;
  - setup sequence may be complex.
Methods of debugging

- Hardware debugging
  - via oscilloscope, 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
Hardware debugging tools

- Oscilloscope
  - Voltage vs. time over bus or I/O ports
  - Platform independent

- Logic analyzer
  - Captures and tracks multiple signals simultaneously and can graph them
  - Allow triggers on changes of states of signals or clock events
  - Not internal memory, registers, etc
  - Platform independent
Hardware debugging tools (cont)

• In-Circuit Emulator
  • Active device that can replace microprocessor in the system
  • Can operate at the full speed of the processor
  • Allow visibility and controls of internal memory, registers, variables, etc
  • Allow setting break points, single stepping, etc.
  • Processor dependent
  • Can be very expensive

• Others
  • JTAG, voltmeter, multimeter
Xilinx ChipScope

- Build a logic analyzer inside a chip
  - Provides device configuration, trigger setup, and trace display for ChipScope Pro cores.
- ChipScope™ Pro tool cores
  - Integrated Controller Pro (ICON)
    - Provides a communication path between the JTAG port and other cores
  - Integrated Logic Analyzer (ILA)
    - For monitoring internal signals
  - Integrated Bus Analyzer (IBA)
    - For monitoring bus interconnects
  - Virtual Input/Output (VIO):
    - For monitoring and driving internal FPGA signals
  - Others
Software debugging tools

- **Debugger**
  - May have special hardware in the embedded device that allows a *Host* to manage and query the device via a serial cable
  - Loading/tracing/stepping code on target
  - Can set breakpoints to stop software execution
  - Allow readings and writing to registers, RAM, I/O ports, etc.
  - Processor dependent
  - e.g. Xilinx MicroProcessor Debugger (XMD)

- **Instruction-level simulator**
  - Running on host
  - Simulating the execution of code
  - No actions of bus or I/O devices
  - Good visibility: register, memory, etc
  - e.g. GNU Debugger (GDB)

- **Hardware/software co-debug**
  - Xilinx XMD + ChipScope Pro
GNU Debugger

- Running on host
- 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 you can experiment with correcting the effects of one bug and go on to learn about another
- You can use GDB to debug programs written in C and C++
GDB

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

- Blue represents registers that have changed
- To change any value, double-click in a field
Xilinx MicroProcessor Debugger (XMD)

- **What**
  - A software utility that can help for debugging embedded software
  - GDB communicates with the hardware through XMD
XMD Functionality

- Xilinx Microprocessor Debug (XMD) engine
  - A program that facilitates a unified GNU Debugging (GDB) interface
  - A Tool command language (TCL) interface
  - GDB communicates with xmd by using the Remote TCP protocol and control the corresponding targets

- XMD supports debugging user programs on different targets
  - Cycle-accurate PowerPC™ processor instruction set simulator
  - PowerPC™ system on a hardware board
Simultaneous HW/SW Debug

- Xilinx XMD + ChipScope Pro
  - ChipScope Pro Analyzer on host
  - GDB debugger on host
- PLB/OPB IBA instantiation in XPS
  - Are treated like the peripheral cores

Set breakpoint in GDB – when hit → triggers ChipScope
Set trigger in ChipScope – when hit → halts CPU and debugger stops
Simultaneous HW/SW Debug

- Flexible Soft IP
- MicroBlaze 32-Bit RISC Core
- Arbiter
- OPB On-Chip Peripheral Bus
- MDM
- IBA
- JTAG

XMD

Active trigger when addr_bus = 0xC200

Trigger out signal from IBA to CPU debug halt signal

Xilinx Parallel Cable
Debugging Using SDK

- Eclipse CDT
  - powerpc-eabi-gdb (or) mb-gdb
  - gdb remote protocol
  - auto-launched

- XMD
  - auto-launched
  - JTAG / XMD protocol

Xilinx custom graphical debug interface
## Debugging in XPS vs SDK

<table>
<thead>
<tr>
<th>Debugging in XPS</th>
<th>Debugging in SDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Download bitstream from XPS</td>
<td>- Download bitstream from XPS</td>
</tr>
<tr>
<td>- Launch XMD</td>
<td>- <strong>Launch XMD</strong></td>
</tr>
<tr>
<td>- Provide Target Connection Options</td>
<td>- Provide Target Connection Options</td>
</tr>
<tr>
<td>- Launch GDB (Insight GUI)</td>
<td>- <strong>Launch GDB (Insight GUI)</strong></td>
</tr>
<tr>
<td>- Set GDB Server connection port in GDB</td>
<td>- Set GDB Server connection port in GDB</td>
</tr>
<tr>
<td>- Download program</td>
<td>- Download program</td>
</tr>
<tr>
<td>- Begin Debugging</td>
<td>- Begin Debugging</td>
</tr>
</tbody>
</table>
Summary

- Embedded Debug
  - Hardware debug
  - Software debug
  - Hardware/software co-debug