Debugging

SDx 2018.2

Objectives

- **After completing this module, you will be able to:**
  - Identify typical software debugging capabilities
  - List tools available for system debugging
  - Describe how the debugger communicates with hardware
  - Distinguish between hardware and software debugging
    - Identify IP which can be of assistance when debugging
  - Describe hardware/software debugging using QEMU emulation flow

Outline

- Introduction
- Hardware Debugging
- Software Debugging
- Summary
- Lab5 Intro

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 dedicated hardware and part of the silicon that is accessible via JTAG or dedicated part pins
    - Controls the processor as an intrusive debug unit that is disabled during normal operation
    - Some "hard" processors have this feature permanently available while "soft" processors may have this physically removed from the delivered product
- **Debugging types:**
  - Functional debugging
  - Performance debugging

Xilinx Solution for Debugging Embedded Designs

- **Hardware debugging tool**
  - Vivado logic analyzer for hardware; functionally replaces expensive external logic analyzer
- **Software debugging tools**
  - System Debugger
    - SDK Debugging perspective and inexpensive JTAG cable replace ICE
- **Built-in, cross-probing trigger capability for hardware and software debug coherency**
  - Vivado logic analyzer invoked through Vivado
  - SDK Debugging perspective tool accessed from within XSDK
- **A single JTAG connection can be used for**
  - Programming the programmable logic
  - Downloading application
  - Hardware and software debugging

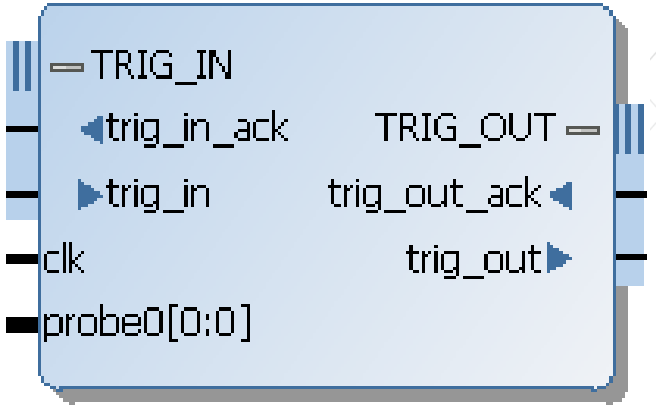
Hardware Debugging

Vivado Logic Analyzer System

- **Vivado logic analyzer tool cores provide full internal visibility to all soft IP**
  - Access to hard IP ports
  - Accesses all the internal signals and nodes within the programmable logic (ILA)
  - Stimulus can be applied using the Virtual I/O core (VIO)
- **Debugging occurs at, or near, system speeds**
  - Debug on-chip using the system clock
- **Minimize pins needed for debugging**
  - Access via the JTAG interface

ILA Core

- **Used for monitoring internal programmable logic signals for post-analysis**
- **Multiple configurable ILA trigger units**
  - Configurable trigger input widths and match types for use with different input signals types
- **Up to 64 probes through GUI**
  - Up to 1024 probes through tcl command
- **Sequential triggering**
- **Storage qualification**
- **Configurable cross triggering**
  - Trigger in and Trigger out interfaces
- **Pre- and post-trigger buffering (capture data before, during, and after trigger condition is met)**

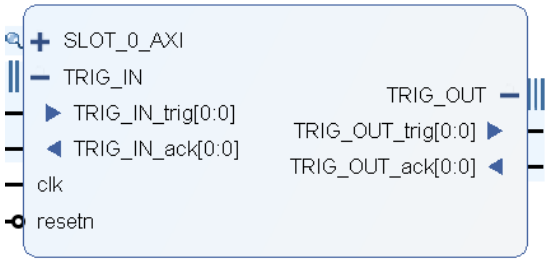


XILINX.

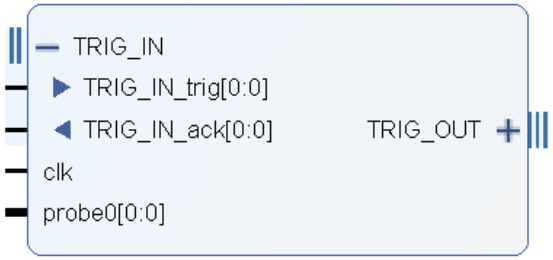
System ILA Core

- **Multiple probe ports, which can be combined into a single trigger condition**
- **Debugging of any debuggable interface including AXI4-MM and Stream**
- **User-selectable AXI4-MM channel debug and AXI Data/Address width selection**
- **Data and Trigger probe and interface type selection**
- **BRAM estimation**
- **AXI4-MM and AXI4-Stream Protocol checking**
- **Allows mixing of native and interface modes**

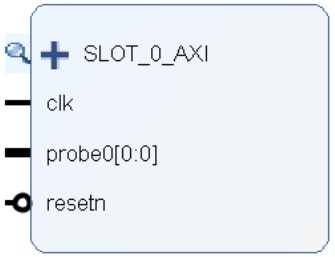
Interface with Cross-triggering



Native with Cross-triggering



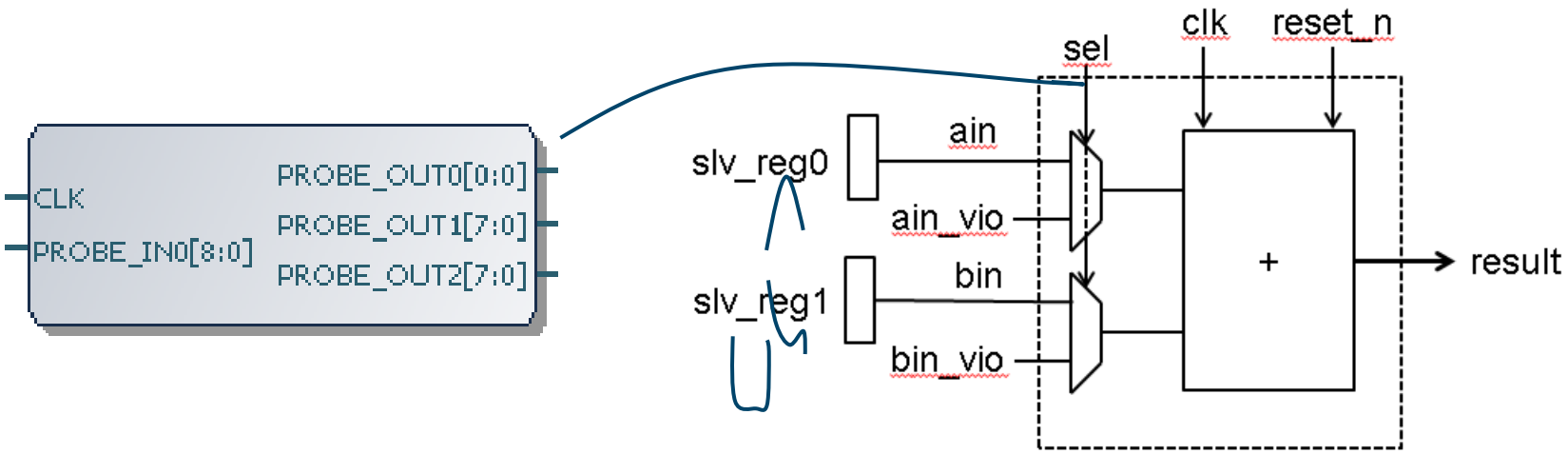
Native with no Cross-triggering



XILINX.

VIO Core

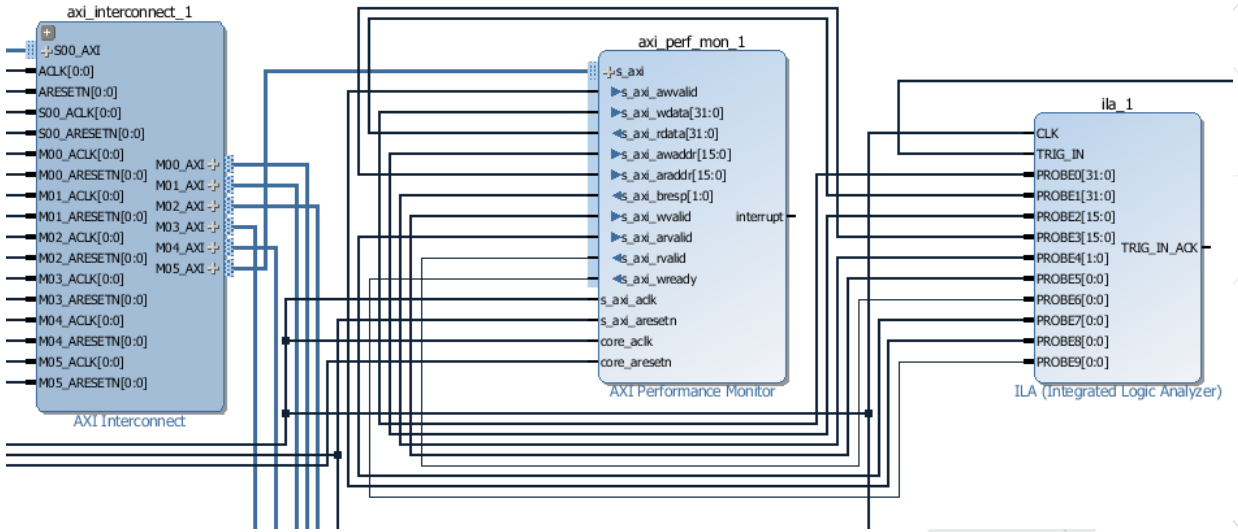
- Support for monitoring and driving internal programmable logic signals in real time
- Probe input unit
- Probe output unit



XILINX.

AXI Performance Monitor Core

- **Enables AXI system performance measurement for multiple slots**
  - AXI4 and AXI4-Stream
- **AXI Performance Monitor supports analyzing system behavior on AXI interfaces**
  - Event logging
    - Captures AXI events and external events
    - Time stamp between two successive events into streaming FIFO
  - Event counting
    - Measure events on AXI4/AXI4-Stream monitor slots or external event ports



XILINX.

Software Debugging

Software Debugging

- **Debuggers must be able to perform a basic set of functions**
  - Regardless of processor or platform
    - Control of the flow of execution
    - Execute only the next instruction (single step)
    - Step into a function/step out of a function
    - Step over a function (treat the function as a single quantity)
    - Stop on a specific line (breakpoint/conditional breakpoint)
    - Resume execution until done or breakpoint reached
    - Stop when a global variable is read or written (watchpoint)
  - Ability to examine and modify memory/variable and registers
  - Directly modify the program counter (PC) and control processor execution location
  - Load the target platform with the code

Software Debugging Support in XSDK

- **XSDK supports software debugging in SDSoC via:**
  - System Debugger (GUI) or XSDB (command-line)
    - Software debugger runs on PC
  - TCF(Target Communication Framework) debugger over digilent cable for ARM
    - Open source
    - Supports system level debugging
    - Improved performance

Xilinx System Debugger

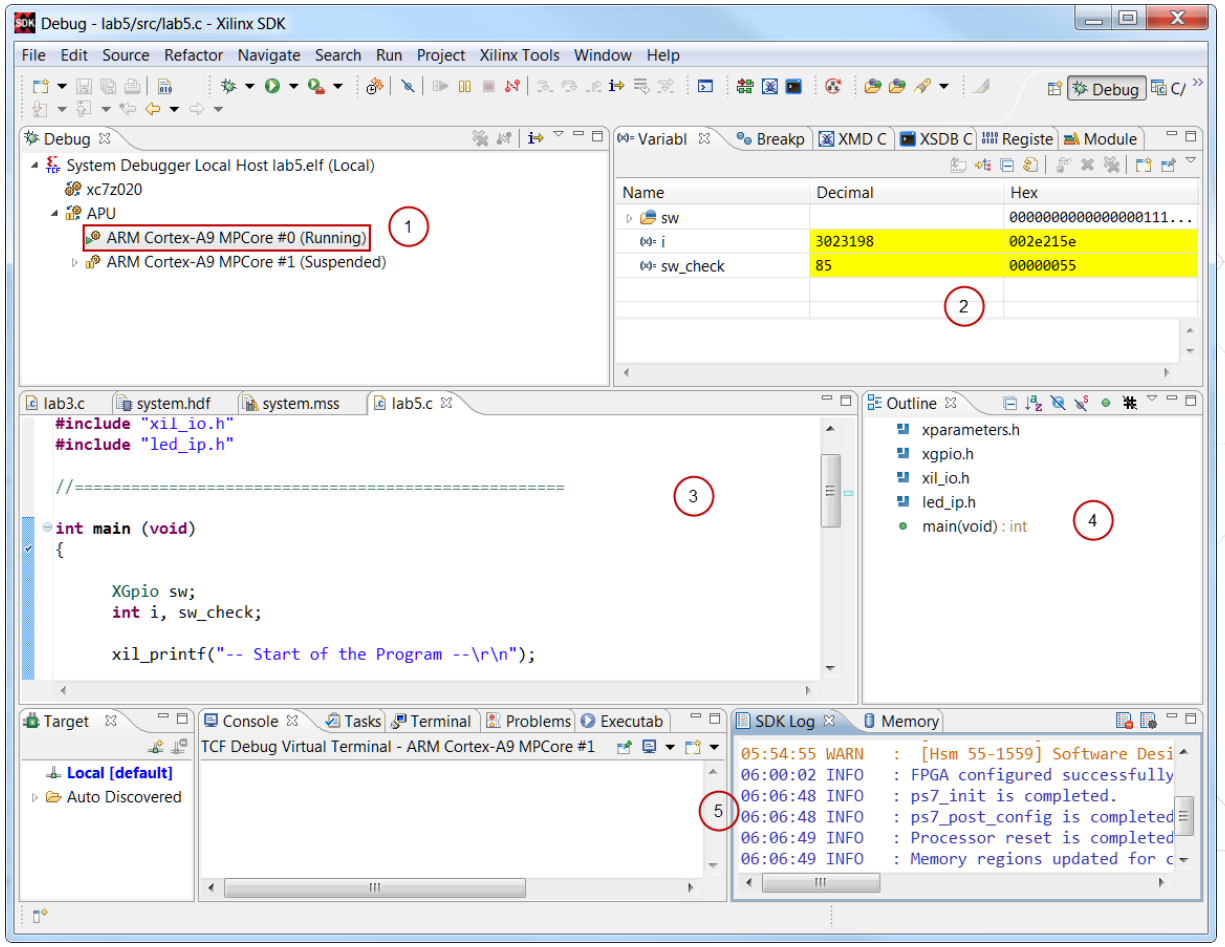
- **Xilinx System Debugger console (XSDB) provides a variety of user debug services**
  - Connection between your workstation (host) and designs being debugged (targets)
    - `connect arm hw`
  - Processor identification
    - `ta`
    - Lists available processors in the hardware and their status (running, suspended)
  - Program processors
  - Bitstream downloading
    - `fpga` <filename>
  - Low-level debug commands
    - `mrd, mwr, rrd …`
  - General Tcl interface and command interpreter

Configure the Programmable Logic

- **Before debugging session can be launched, the target programmable logic must be configured**
  - PL would have logic analyzer core(s) for hardware debugging
  - SDSoC Debug configuration will automatically do this

XSDK Debug Perspective

- **Stack frame for target threads**
- **Variables, breakpoints, and registers views**
- **C/C++ editor**
- **Code outline**
  - Disassembly view can be added using Window > Show View > Disassembly
- **Console, XSDK Log, and Memory views**



System Debugger

- **Source-level debugger GUI**
  - Runs through hw_server
  - Capable of remote debugging, as with XSDB, by connecting to remote hw_server instance
- **General process**
  - Start application
  - Set breakpoints/watchpoints
  - Examine state of registers and memory when breakpoint hit/application paused
  - Can change values in memory or registers
    - Allows user to correct for an error or create a difficult to achieve condition to test code
- **Only a single instance of System Debugger to be launched to debug multiple cores**
  - Independent control of each core
- **Supports C and C++**

Interacting With System Debugger

- **Code can be manually paused**
  - No guarantee where the PC is
  - Used when program is not behaving as expected and user wants to see where the code execution is occurring
- **Breakpoints/conditional breakpoints**
  - Set by user to cause the execution to stop at a specific line of C/C++ code
  - Conditional breakpoints also cause execution to stop at a location, but only when the conditions associated with that breakpoint are met
  - Useful when stopping when values are going out of bounds, or in the middle of a large loop
- **Once execution is suspended**
  - Memory, variables, and/or registers can be examined and modified
  - Control flow of execution
  - View assembly instructions related to the C/C++ source

Linux Debugging

- **Kernel debugging**
  - Linux kernel debugging through system debugger
- **Linux application debugging**
  - Linux application debugging using gdb
  - Linux application debugging using system debugger
    - PetaLinux comes with TCF agent
    - For other Linux distributions, need to include the TCF agent

Summary

Summary

- **Debugging is an integral part of embedded systems development**
- **Vivado and SDSoC SDK provides tools to facilitate hardware and software debugging**
  - Hardware debugging is done through using Vivado logic analyzer cores
  - Software debugging is performed using system debugger
- **SDSoC SDK provides environment, perspective, and underlying tools to enable seamless software debugging**
- **With software debugging and the Vivado logic analyzer supporting cross-probing enables simultaneous hardware and software debugging**
  - Use it to find and fix embedded system bugs faster
- **Full range of debugging capabilities is provided**
  - Control over execution using breakpoints, single-step, step-into, etc.
  - Visibility into various regions of memory and registers, including the ability to change values
- **QEMU provides system-level debugging capability**

Lab5 Intro

Lab5 Intro

- **Introduction**
  - This lab guides you through the steps involved in debugging software application in SDSoC. SDSoC supports debugging both Standalone as well as Linux application. SDSoC also provides the Dump/Restore Data File feature which can be used to dump the memory snapshot on a disk and restore the memory content from a pre-defined file.
- **Objectives**
  - Use the SDSoC environment to debug an Standalone application
  - Use the SDSoC environment to debug an Linux application

Adaptable.
Intelligent.

XILINX