EE382N-4
Advanced Microcontroller Systems

Debugging on the ZynqMP

Mark McDermott
Lev Shuhatovich

Portions of this lecture from Jack Gansel's "The Art of Designing Embedded Systems" and
https://web.engr.oregonstate.edu/~traylor/ece473/lectures/debug_techniques.pdf

What is debugging?

- Debugging is an integral part of embedded systems development
  - Differs greatly from debugging software or hardware alone.
- The debugging process is defined as testing, stabilizing, localizing, and correcting errors.
- System prerequisites:
  - Observability (o-scope, meter, blink an LED, print statement, trace)
  - Controllability (swap circuitry, push a button, initiate the fault)
- Two types of bugs:
  - Functional
  - Performance

Bugs don't go away on their own....

They come back to bite you!

Debugging Embedded Systems

- Requires a deep understanding of the hardware and its interactions with the software.
  - It can be helpful to have a detailed understanding of the system architecture and the functions of each component.
- Requires problem-solving skills and the ability to think critically.
  - It can also be helpful to have a good understanding of electrical engineering principles, as many hardware issues will involve electrical signals and circuits.
- Is made more challenging by the fact that many embedded systems are designed to operate in resource-constrained environments, with limited processing power, memory, and storage.
  - This can make it more difficult to use traditional debugging tools and techniques and may require the use of specialized tools or custom approaches.

Debugging Embedded Systems

- May be caused by problems with the software running on the system.
  - It is very helpful to have a good understanding of the software components of the system and how they interact with the hardware, as this can help you identify and fix issues that may be caused by software bugs.

- Often involves working with a variety of different tools and techniques and may require some trial and error.
  - It is important to be patient and persistent, as debugging can be a time-consuming frustrating process.

Testing and Debugging

- **Testing and debugging go together like peas in a pod:**
  - **Testing finds errors; debugging localizes and repairs them.**
  - **Together these form the "testing/debugging cycle": we test, then debug, then repeat.**
  - **Any debugging should be followed by a reapplication of all relevant tests, particularly regression tests. This avoids (more likely reduces) the introduction of new bugs when debugging.**
  - **Testing and debugging need not be done by the same people (and often should not be).**

Debugging Techniques: High Level View

- Make the bug repeatable
- Observe the behavior, getting information to isolate the apparent bug
- Create a suspect list (remain rational (and sober) during this step)
  - Have a seasoned engineer rank order the list
- Generate a hypothesis to pinpoint the bug
- Create an experiment to test the hypothesis
- Fix the bug
- Test the fix and then
- Run all regression tests to find issues that may be introduced by a new “fix”

Debugging Techniques

- Much of debugging is about asking the right questions
  - What changed last?
    - New cable, new hardware, re-soldered a wire, new code
  - Is this a problem or a symptom?
    - A part getting hot doesn't need more cooling
  - Is the power on? Is the input voltage correct?
    - Five volts from the USB port is never 5 volts
  - Are you compiling/editing the right file?
  - Is the hardware basically, OK? Run a smoke test test.
  - What assumption are you making that is wrong?
  - Explain the problem to someone else.

- Once you fix the bug….
  - prove the problem is gone, replace the bug, see it, then take it away again

Debugging Techniques

- **Debugging requires a correct attitude**
  - Assume that things are somewhat broken
  - Assume that a few small bugs may still exist
  - Trust the laws of physics. Your project depends on them.
  - You will witness something that cannot be happening; so why is it happening? Retrace assumptions.
  - Take a break. Walk around, get a coffee. Take a snooze.
- **Debugging requires discipline**
  - Resist unlikely sources of bugs:
    - Bad parts, compiler error, phase of moon
  - Stay focused on one problem at a time
  - Keep to the scientific method
  - When you find something that you can't explain, write it down.
  - Resist random o-scope probing
  - Don't take shots in the dark

Debugging Quotes

- ”...reason back from the state of the broken system to determine what could have caused this. Debugging involves backwards reasoning, like solving murder mysteries. Something impossible occurred, and the only solid information is that it really did occur. So, we must think backwards from the result to discover the reasons.”
  - Brian W. Kernighan, The Practice of Programming

- ”The single greatest asset that a designer can have is self-knowledge. Knowing when your thinking feels right and when you’re trying to fool yourself...knowing your strengths and weaknesses, prowess's and prejudices...learning when to ask questions and when to believe your answers.”
  - Jim Williams, Linear Technology

- ”When things are acting funny, measure the amount of funny.”
  - Bob Pease, National Semiconductor

Debugging (and a little Designing) wisdom

- Bugs that mysteriously go away will mysteriously reappear
- Disprove all theories
  - Trust but Verify
  - Truth can generally be found in one place: the code (C or Verilog)
- Firmware is not always wrong
  - But is mostly wrong
- The sooner you start to code (C or Verilog), the longer the program will take and the more bugs you will have
  - Plan, plan, plan, plan, plan, plan
  - Freeze the plan: Walking on water and developing software from a specification are easy if both are frozen.
- Better is the enemy of good enough
  - Hardware is not always perfect, but it can be good enough

Sources of HW bugs

- Incorrect Verilog
  - Flawed algorithm
  - Program logic (if/else, loop termination, select case, etc.)
  - Inferring State In Combinational Circuits
  - Incorrect use of blocking and nonblocking assignments
  - Using positional port naming and not implicit port naming
  - Lack of standardized port ordering -> always use [MSB:LSB]
- Synthesizing Verilog with wrong or non-existent constraints.
  - Timing constraints
  - Physical constraints
  - Power constraints
  - HINT: Constraint driven design is absolutely required
- Using intellectual property that you don't understand
  - Lack of Verification IP (i.e., verification patterns for the IP)

Verilog coding guidelines

Sources of SW bugs

- Compile time: syntax, spelling, static type mismatch.
  - Usually caught with compiler
- Design: flawed algorithm.
  - Incorrect outputs
- Program logic (if/else, loop termination, select case, etc.).
  - Incorrect outputs
- Memory problems: null pointers, array bounds, bad types, leaks.
  - Runtime exceptions
- Interface errors between modules, threads, programs (in particular, with shared resources: sockets, files, memory, etc.).
  - Runtime Exceptions
- Off-nominal conditions: failure of some part of software or underlying machinery (network, etc).
  - Incomplete functionality
- Deadlocks: multiple processes fighting for a resource.
  - Freeze ups, never ending processes

Boult, uccs.edu

What is a Debugger?

- "A software tool that is used to detect the source of program or script errors, by performing step-by-step execution of application code and viewing the content of code variables."

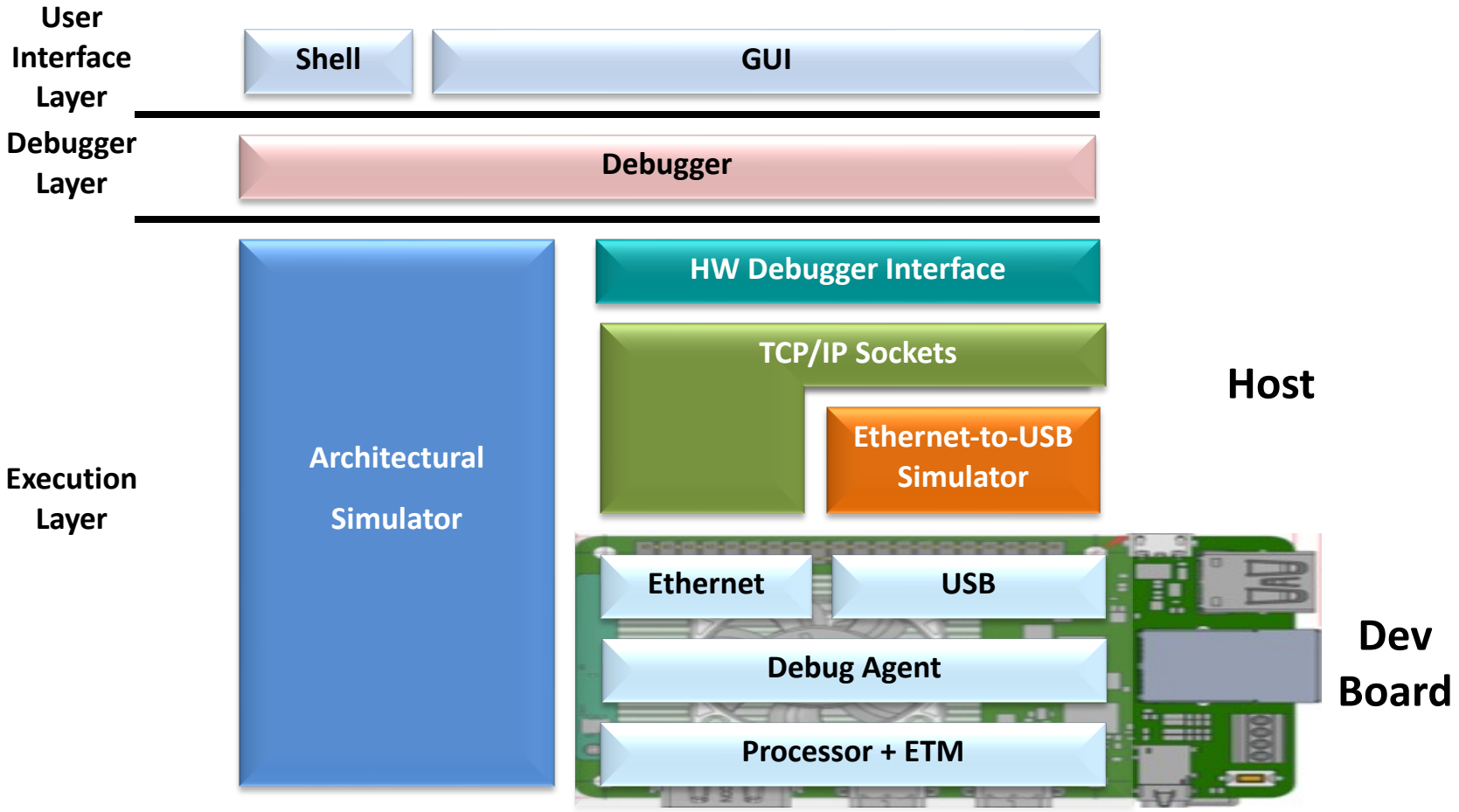
- A debugger is not an Integrated Development Environment (IDE)
  - Although the two can be integrated, they are separate entities.

- A debugger loads in a program (compiled executable, or interpreted source code) and allows the user to trace through the execution.

- Debuggers typically do disassembly, stack traces, expression watches, and more.

Boult, uccs.edu

Typical Integrated Debug Environment



Setting up a functional debug environment

- It is the first thing to do!!
  - Don't wait until you have a bug.

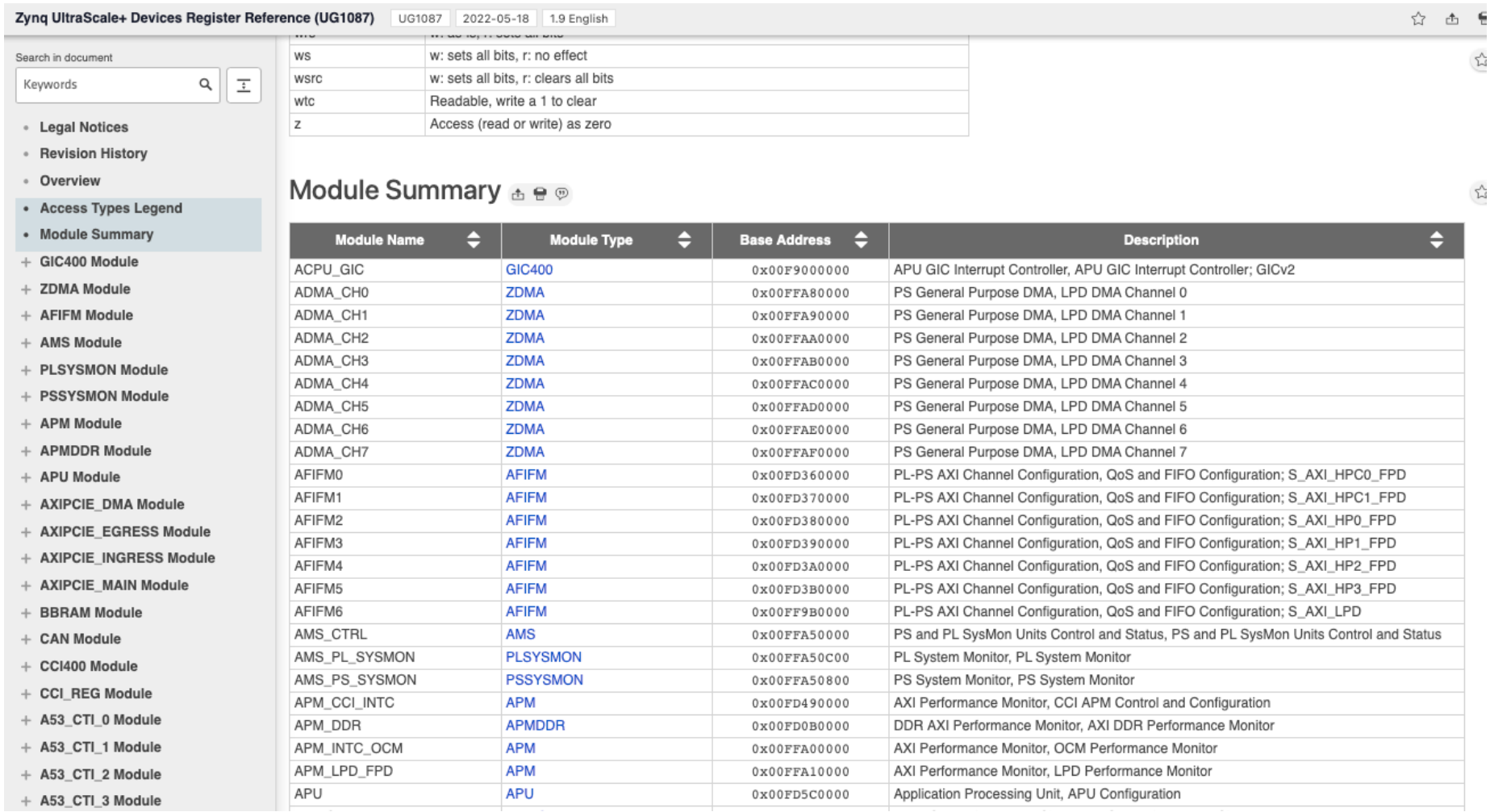
- Determine how you are going to verify that each component of the design is working correctly.
  - Start small --- get individual components working.
  - Use slower clock frequency if it looks like there is a setup time violation.
    - NOTE: Doesn't work for Hold-Time issues.
    - Might work for crosstalk issues
  - Stub off signals using registers i.e., do not hardwire to 1'b1 or 1'b0
    - Takes forever to recompile FPGA bit file
    - Easier to just write a register to change the pattern to your DUT
- Setup test pattern environment
  - Can use RAM for data storage (fastest & hardest)
  - Use shell scripts (slowest & easiest)
  - Linux application (good compromise)

Setting up a functional debug environment (cont)

- Use Xilinx LogicIP cores
  - AXI Performance Monitor
- Develop special AXI components
  - Gated Counters
  - Logic analyzer

- REMINDER: Debug hardware and software are intrusive
  - Software
    - Slows down other software
  - Hardware
    - Pushes on timing closure
    - Consumes FPGA resources

ZynqMP registers



Zynq UltraScale+ Devices Register Reference (UG1087) UG1087 2022-05-18 1.9 English

Search in document

Keywords

- Legal Notices
- Revision History
- Overview
- Access Types Legend
- Module Summary
- GIC400 Module
- ZDMA Module
- AFIFM Module
- AMS Module
- PLSYSMON Module
- PSSYSMON Module
- APM Module
- APMDDR Module
- APU Module
- AXIPCIE_DMA Module
- AXIPCIE_EGRESS Module
- AXIPCIE_INGRESS Module
- AXIPCIE_MAIN Module
- BBRAM Module
- CAN Module
- CCI400 Module
- CCI_REG Module
- A53_CTI_0 Module
- A53_CTI_1 Module
- A53_CTI_2 Module
- A53_CTI_3 Module

ws	w: sets all bits, r: no effect
wsrc	w: sets all bits, r: clears all bits
wtc	Readable, write a 1 to clear
z	Access (read or write) as zero

Module Summary

Module Name	Module Type	Base Address	Description
ACPU_GIC	GIC400	0x00F9000000	APU GIC Interrupt Controller, APU GIC Interrupt Controller; GICv2
ADMA_CH0	ZDMA	0x00FFA80000	PS General Purpose DMA, LPD DMA Channel 0
ADMA_CH1	ZDMA	0x00FFA90000	PS General Purpose DMA, LPD DMA Channel 1
ADMA_CH2	ZDMA	0x00FFAA0000	PS General Purpose DMA, LPD DMA Channel 2
ADMA_CH3	ZDMA	0x00FFAB0000	PS General Purpose DMA, LPD DMA Channel 3
ADMA_CH4	ZDMA	0x00FFAC0000	PS General Purpose DMA, LPD DMA Channel 4
ADMA_CH5	ZDMA	0x00FFAD0000	PS General Purpose DMA, LPD DMA Channel 5
ADMA_CH6	ZDMA	0x00FFAE0000	PS General Purpose DMA, LPD DMA Channel 6
ADMA_CH7	ZDMA	0x00FFAF0000	PS General Purpose DMA, LPD DMA Channel 7
AFIFM0	AFIFM	0x00FD360000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HPC0_FPD
AFIFM1	AFIFM	0x00FD370000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HPC1_FPD
AFIFM2	AFIFM	0x00FD380000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HP0_FPD
AFIFM3	AFIFM	0x00FD390000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HP1_FPD
AFIFM4	AFIFM	0x00FD3A0000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HP2_FPD
AFIFM5	AFIFM	0x00FD3B0000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_HP3_FPD
AFIFM6	AFIFM	0x00FF9B0000	PL-PS AXI Channel Configuration, QoS and FIFO Configuration; S_AXI_LPD
AMS_CTRL	AMS	0x00FFA50000	PS and PL SysMon Units Control and Status, PS and PL SysMon Units Control and Status
AMS_PL_SYSMON	PLSYSMON	0x00FFA50C00	PL System Monitor, PL System Monitor
AMS_PS_SYSMON	PSSYSMON	0x00FFA50800	PS System Monitor, PS System Monitor
APM_CCI_INTC	APM	0x00FD490000	AXI Performance Monitor, CCI APM Control and Configuration
APM_DDR	APMDDR	0x00FD0B0000	DDR AXI Performance Monitor, AXI DDR Performance Monitor
APM_INTC_OCM	APM	0x00FFA00000	AXI Performance Monitor, OCM Performance Monitor
APM_LPD_FPD	APM	0x00FFA10000	AXI Performance Monitor, LPD Performance Monitor
APU	APU	0x00FD5C0000	Application Processing Unit, APU Configuration

https://docs.xilinx.com/r/en-US/ug1087-zynq-ultrascale-registers/Module-Summary

PLL Control Register example

APLL_CTRL (CRF_APB) Register

APLL_CTRL (CRF_APB) Register Description

Register Name	APLL_CTRL
Relative Address	0x0000000020
Absolute Address	0x00FD1A0020 (CRF_APB)
Width	32
Type	rw
Reset Value	0x00012C09
Description	APLL Clock Unit Control

APLL_CTRL (CRF_APB) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
Reserved	31:27	rw	0x0	reserved.
POST_SRC	26:24	rw	0x0	Select the pass-thru clock source for PLL Bypass mode. 0xx: PS_REF_CLK 100: VIDEO_REF_CLK 101: ALT_REF_CLK 110: AUX_REF_CLK 111: GT_REF_CLK
Reserved	23	rw	0x0	reserved.
PRE_SRC	22:20	rw	0x0	Select the clock source for PLL input. 0xx: PS_REF_CLK 100: VIDEO_REF_CLK 101: ALT_REF_CLK 110: AUX_REF_CLK 111: GT_REF_CLK
Reserved	19:18	rw	0x0	reserved.
Reserved	17	rw	0x0	reserved.
DIV2	16	rw	0x1	Enable the divide by 2 function inside of the PLL. 0: no effect. 1: divide clock by 2. Note: this does not change the VCO frequency, just the output frequency.
Reserved	15	rw	0x0	reserved.
FBDIV	14:8	rw	0x2C	Feedback divisor integer portion for the PLL.
Reserved	7:4	rw	0x0	reserved.
BYPASS	3	rw	0x1	PLL Clock Bypass Mode. 0: normal PLL mode; the source clock is selected using [PRE_SRC]. 1: bypass the PLL; the source clock is selected using [POST_SRC].
Reserved	2:1	rw	0x0	reserved.
RESET	0	rw	0x1	PLL reset. 0: active. 1: reset. Note: Program the PLL into bypass mode before resetting the PLL.

MMAP Debugging commands

- All components are memory mapped. Therefore, it is easy to access registers and memory using the Linux mmap() system call.
- There are five simple commands which will help you debug your designs:

```
1) /usr/bin/pm: This command writes to a memory location
Put Memory - USAGE:  pm (Address) (write data) (optional repeat #)

2) /usr/bin/dm: This command displays a memory location(s)
Display Memory - USAGE:  dm (address) (repeat)

3) /usr/bin/fm: This command fills a range of memory
Fill Memory - USAGE: fm (address) (write data) (#addresses) (data incr)

4) /usr/bin/frm: This command fills a range of memory with random data
Fill Memory Random Data - USAGE:  frm (address) (# addresses)

5) /usr/bin/smb: This command This command sets a bit in a memory word
Set/Clear memory bit - USAGE:  smb (address) (bit number) (data)
```

- Four of these commands can "brick" your system in a femto-second.
  - Be very careful

frm – fill memory with random data

```
int fd = open("/dev/mem", O_RDWR|O_SYNC);
if(fd == -1)
{
printf("Unable to open /dev/mem. Need to be
root\n");
return -1;
}

lp_cnt = 1;         // Write at least 1 location
offset = 0;
target_addr = strtoul(argv[1], 0, 0);

  value = 0;
    if (argc == 3) {
        lp_cnt  = strtoul(argv[2], 0, 0);
    }
    if (lp_cnt > 0x400)  { // Max is 4096 bytes
        lp_cnt = 0x400;
        printf("Setting max repeat
                value to 0x400\n");
    }

regs = (unsigned int *)mmap(NULL,
                            MAP_SIZE,
                            PROT_READ|PROT_WRITE,
                            MAP_SHARED,
                            fd,
                            target_addr &
                            ~MAP_MASK);
```

```
srand(time(0));  // Seed the random number
generator using TOD

/* ---------------------------------------
 *   Main loop
 */

while (lp_cnt) {

  value = rand();   // Assign random data

  address = regs +
            (((target_addr + offset)
             & MAP_MASK)>>2);

  *address = value;     // perform write command
    printf("0x%.8x" , (target_addr +
                              offset));
// display register value
printf(" = 0x%.8x\n", *address);

    lp_cnt  -= 1; // decrement loop
    offset  += 4; // WORD aligned

} // End of while loop
```

Script based example

```
#!/bin/sh

# Enable IOMUX port for CSPI
/bin/pm 0x020e0144 0x00000001
/bin/pm 0x020e0148 0x00000001
/bin/pm 0x020e014c 0x00000001
/bin/pm 0x020e01cc 0x00000001
/bin/pm 0x020e07dc 0x00000002
/bin/pm 0x020e024c 0x00000000

# Set drive strength for SPI_SCLK
/bin/pm 0x020e0514 0x00002090

# Enable clock gating to ECSPI
/bin/pm 0x020c406c 0x00300c03

# Reset SPI
/bin/pm 0x02008008 0x01f01010

# Enable SPI
#/bin/pm 0x02008008 0x01f00af9
/bin/pm 0x02008008 0x01f002f9

/bin/pm 0x0200800c 0x00000100


# Eanble interrupts
/bin/pm 0x02008010 0x00000010


# Enable test mode
/bin/pm 0x02008020 0x80000000


# Enable GPIO_19 as a 48 MHz reference clock
/bin/pm 0x020e0220 0x00000003
#/bin/pm 0x020e05f0 0x0001b088
/bin/pm 0x020e05f0 0x000020e9
/bin/pm 0x020c4060 0x010e00c0


# Dump registers
#/bin/dm 0x02008000 0x10
#/bin/dm 0x020c4000
#/bin/dm 0x020c406c
```

Scripting – running a complete test

```
#!/bin/sh
#
# ADDRESSES:
# CDMA 0xb000_0000
# BRAM 0xb000_4000
# SHA  0xb000_6000
# OCM  0xfffc_0000
#
########################################
## Fill data in OCM to be transferred
/usr/bin/fm 0xfffc0000 0x0 100 0 > /dev/null

## Write 'abc' to OCM
/usr/bin/pm 0xfffc0000 0x6362610a > /dev/null
/usr/bin/pm 0xfffc0004 0x80000000 > /dev/null


# Setup tail of the 512-bit chunk
/usr/bin/pm 0xfffc003c 0x0020 > /dev/null


/usr/bin/fm 0xb0004000 0x0 100 0 > /dev/null
########################################
## Write the destination address to the CDMA unit
/usr/bin/pm 0xb0000020 0xb0004000 > /dev/null


## Write the Source address to the CDMA unit
/usr/bin/pm 0xb0000018 0xfffc0000 > /dev/null


## Start the transfer
/usr/bin/pm 0xb0000028 0x50 > /dev/null
```

```
############################################################
# Set the number of chunks in the SHA Uni
/usr/bin/pm 0xb0006014 0x1 > /dev/null

# Set the starting BRAM address
/usr/bin/pm 0xb0006018 0xb0004000 > /dev/null


# Reset SHA Unit
/usr/bin/pm 0xb0006000 0x0 > /dev/null
# Enable SHA conversion
/usr/bin/pm 0xb0006000 0x3 > /dev/null
# Disable SHA conversion
/usr/bin/pm 0xb0006000 0x2 > /dev/null


echo "---------------------------"
echo "Expected results
0xb0000020 = 0xedeaaff3
0xb0000024 = 0xf1774ad2
0xb0000028 = 0x88867377
0xb000002c = 0x0c6d6409
0xb0000030 = 0x7e391bc3
0xb0000034 = 0x62d7d6fb
0xb0000038 = 0x34982ddf
0xb000003c = 0x0efd18cb
"


echo "Actual results"
/usr/bin/dm 0xb0006020 8
```

Debug Registers

```
// Address decoding for reading registers
case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
  4'h0   : reg_data_out <= slv_reg0;                // Control register
  4'h1   : reg_data_out <= {29'b0,                  // status register
                            sha_complete,           // Need to be sticky bits
                            sha_clear_reg_n,        // Ditto
                            sha_digest_valid};      // Ditto
  4'h2   : reg_data_out <= slv_reg2;                // Address to the BRAM
  4'h3   : reg_data_out <= axi_bram_read_data;      // Read data from the BRAM
  4'h4   : reg_data_out <= axi_bram_write_data;     // Write data to the BRAM
  4'h5   : reg_data_out <= sha_num_chunks;          // Number of SHA 512-bit chunks
  4'h6   : reg_data_out <= sha_bram_addr_start;     // Starting address chunk in BRAM
  4'h7   : reg_data_out <= 32'hfeedbeef;            // CANARY
  4'h8   : reg_data_out <= sha_digest_reg[255:224];
  4'h9   : reg_data_out <= sha_digest_reg[223:192];
  4'hA   : reg_data_out <= sha_digest_reg[191:160];
  4'hB   : reg_data_out <= sha_digest_reg[159:128];
  4'hC   : reg_data_out <= sha_digest_reg[127: 96];
  4'hD   : reg_data_out <= sha_digest_reg[ 95: 64];
  4'hE   : reg_data_out <= sha_digest_reg[ 63: 32];
  4'hF   : reg_data_out <= sha_digest_reg[ 31:  0];

always @( posedge sha_clk or negedge sha_rst_n)
  begin : reg_reset
  integer i;
      if ( sha_rst_n == 1'b0 )                  // In RESET
      begin
         sha_digest_reg      <= 256'h67444444_89ABCDEF_19691969_69DECADE_DEADBEEF_01234567_89ABCDEF_01234567;
 end
```

Performance analysis (and debugging)

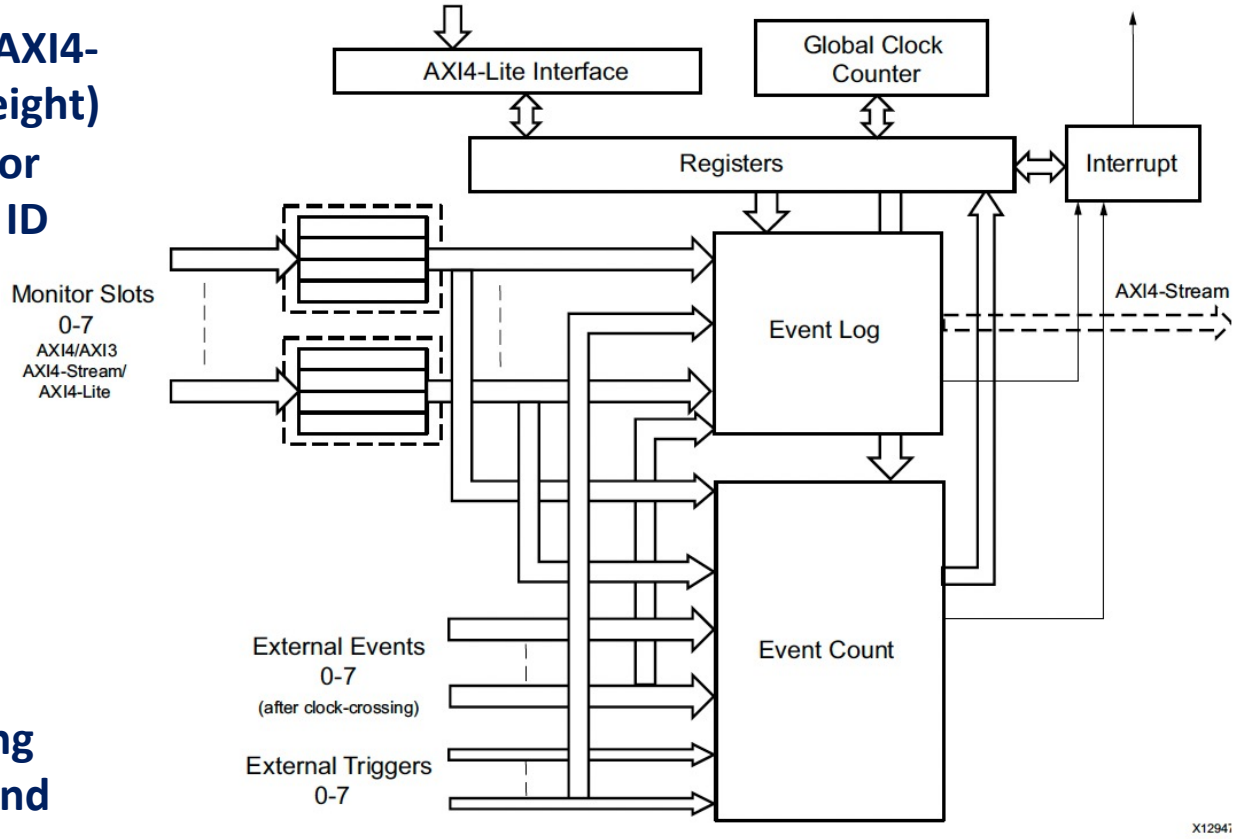
- Speeds and Feeds
  - Clock frequencies
    - Penalties for crossing multiple clock crossing boundaries
      - Synchronizers, etc.
  - Width
    - Weakest link problem i.e., packing and unpacking

- Typical Metrics
  - Bus Utilization – How idle is the pipe?
  - Latency – How long does it take 1 to get through?
  - How does it scale 1 – N?

- Reminder: Instrumentation is NOT free

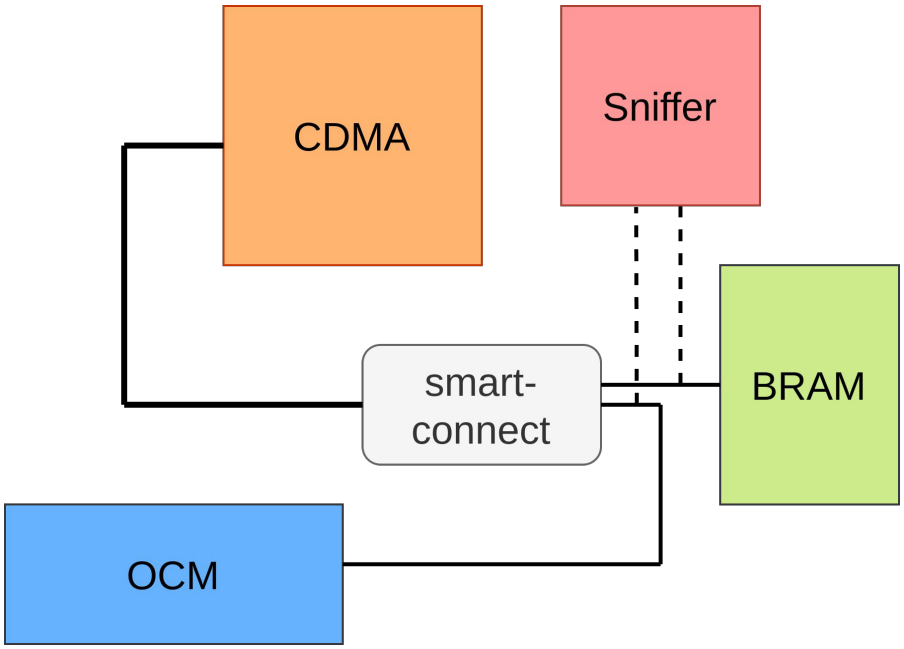
AXI Performance Monitor

- **Features**
  - Connects as a 32-bit slave on AXI4-Lite interface
  - Configurable number of (AXI4/AXI3/AXI4-Stream/AXI4-Lite) monitor slots (up to eight)
  - Flexible support for monitor slots with any data width, ID width and frequency
  - Free running Global Clock Counter
  - Supports AXI and external events logging
  - Supports AXI and external events counting
  - Supports external event triggering and cross probing between event counting and event logging



Custom AXI Bus Sniffer

- **Motivation**
  - **You are a startup and ChipScope is expensive**
  - **You have some gnarly FPGA related bugs that may be observable on the AXI bus.**



Courtesy of: Jatin Khare and Abhijjith Venkkateshraj -- Final Project EE382N-4 Spring 22

Custom AXI Bus Sniffer

- Debugging IP that shows you what happens inside the FPGA data flow.
- Once the `void transfer()` gives the control to the PL, the PS must not intervene until the transfer is over.
- Idea: make the PL capable of storing its own signal information.
- Probes the AXI signals used for data transfer.
- Runs the transaction and presents the post simulation results.

Capabilities of AXI Sniffer

- Supports data widths up to 1024 bits within the same Sniffer IP configuration.
- Easy scalable to probe any new signals of interest with minimal overhead.
- Provides design specific burst information.
- Shows the (Address Specific & Total) Transfer and Transaction counts.
- Can show the Read and Write data in the bus, corresponding to the address of interest.
- Supports unaligned address requests.
- Later, one can also visualize the data via customized AXI timing diagram using generated .json file from the PS code.

Quips

Quips from all over

- Programs must be written for people to read, and only incidentally for machines to execute.
- The big optimizations come from refining the high-level design, not the individual routines.
- Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
- Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
- Some of the best programs are done on paper. Putting it into the computer is just a minor detail.
- If you optimize everything, you will always be unhappy.
- Simple things should be simple. Complex things should be possible.

More quips

- If debugging is the process of removing software bugs, then programming must be the process of putting them in.
- The perfect kind of architectural decision is the one that never has to be made
- Programming isn't about what you know; it's about what you can figure out.
- First, solve the problem. Then, write the code.
- Programming is breaking of one big impossible task into several very small possible tasks
- Premature optimization is the root of all evil.
- Simplicity is a prerequisite for reliability.
- Redundant comments are just places to collect "lies and misinformation".
- If you must choose between clarity and brevity, "Always choose clarity in your code."
- Before software can be reusable; it first must be usable…
- Spent a month one day debugging an embedded system.

Questions