# Spring 2018

## EE 382N-4: Advanced Micro-Controller Systems

## Lab Assignment #1

## Due Feb 28th, 2018

---

**Overview:**

1) Using Vivado, build a circuit that instantiates a 1024x32 BRAM in the FPGA. The BRAM needs to be Dual-Ported. Each port should be hooked up to an AXI slave port. See block Diagram below.

2) Implement a pseudo-random number generator using a 32-bit LFSR in the FPGA that can used for the memory test. The value of the LFSR should change each time it is accessed.

3) Write a memory test using C that runs on Linux to test the BRAM.

---

**Memory Test Specification**:

1) Write a memory test that does a pseudo random address and data test of the BRAM memory using Port A.

   a) Randomly vary the data while randomly varying the address. Use the linear feedback shift register implemented in the FPGA to generate the address and the data patterns. Note that an LFSR cannot generate the "zero" value.

   b) Randomly vary the (write -> read-back/validate) timing so that the memory has the time to loose its state. In other words, the time between the write to an address and the corresponding read from the same address should also vary randomly.

2) Write a pseudo random address and data test of the BRAM memory that writes data to Port-A and reads the results from Port-B.

   a) Randomly vary the data while randomly varying the address. Use the linear feedback shift register implemented in the FPGA to generate the address and the data patterns. Note that an LFSR cannot generate the "zero" value.

   b) Randomly vary the (write -> read-back/validate) timing so that the memory has the time to loose its state. In other words, the time between the write to an address and the corresponding read from the same address should also vary randomly.

3) Write a pseudo random address and data test of the BRAM memory that writes data to Port-B and reads the results from Port-A.

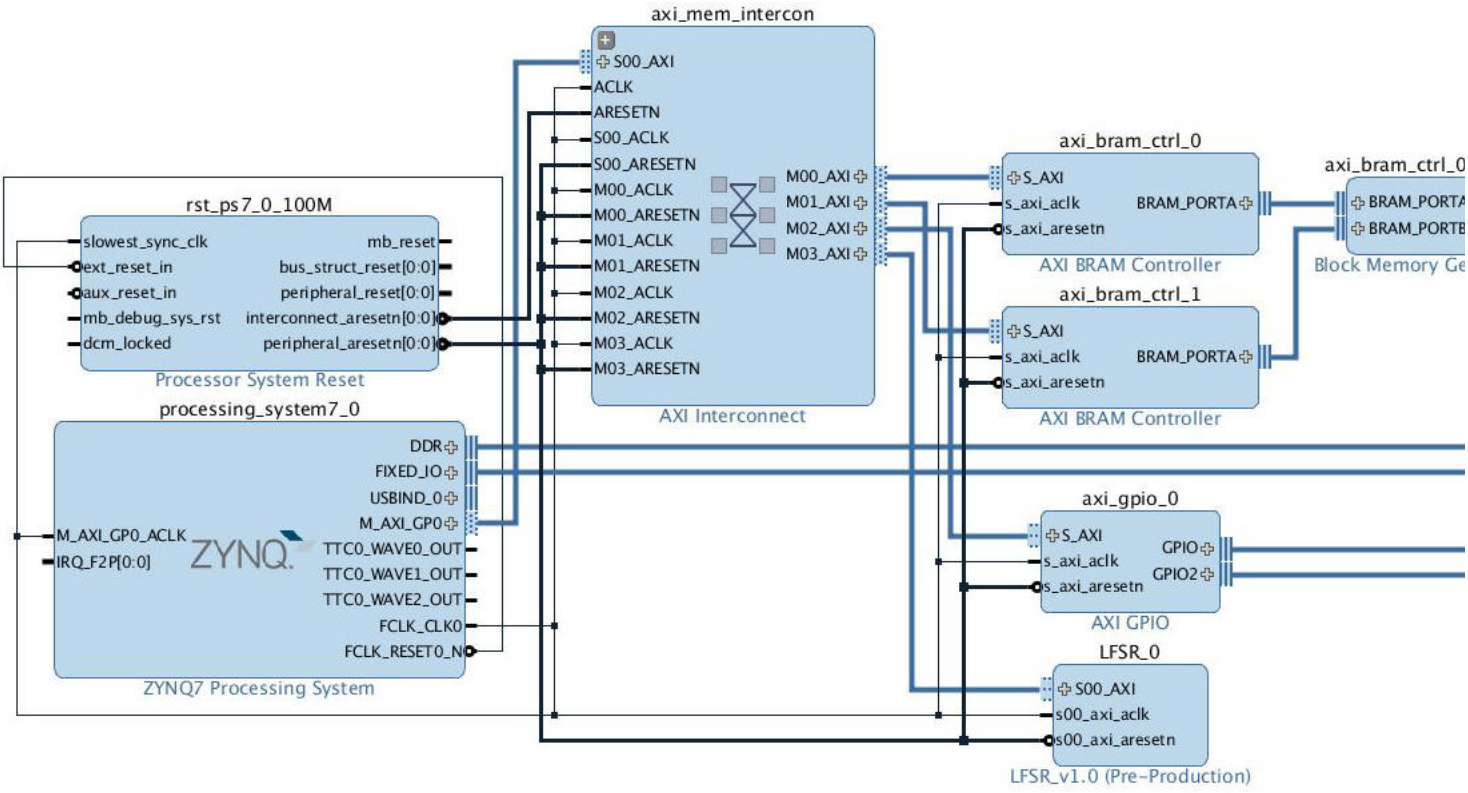
   a) Randomly vary the data while randomly varying the address. Use the linear feedback shift register implemented in the FPGA to generate the address and the data patterns. Note that an LFSR cannot generate the "zero" value.

   b) Randomly vary the (write -> read-back/validate) timing so that the memory has the time to loose its state. In other words, the time between the write to an address and the corresponding read from the same address should also vary randomly.

***NOTE: All 3 tests should generate readable diagnostics that explain what is broken and what needs to be fixed.***

---

**Block Diagram:**




**Environmental Details:**

1) Vivado is available on the Linux machines. Refer to this web page ECE-Linux-Machines for details on accessing the Linux machines remotely. NOTE: VNC is no longer supported.

2) All software will be compiled on the ZedBoard using the GNU tool suite. Do not use the Vivado SW development tools. These are for bare-metal implementations.

**Implementation Details**

1) The LFSR will need to be implemented in Verilog. It needs to provide a 32-bit PRN that can be used for both the address and data values in the memory test routine. NOTE: an LFSR does not generate a "0x0000_0000" value so you will need to generate it in software.

2) Instantiate a GPIO block that can be used for generating debug information. Use the LEDs to indicate test status.

3) Use the following address map for the peripherals:

Address Editor

| Cell | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| processing_system7_0 | | | | | |
| Data (32 address bits : 0x40000000 [ 1G ]) | | | | | |
| axi_bram_ctrl_0 | S_AXI | Mem0 | 0x4000_0000 | 8K | 0x4000_1FFF |
| axi_gpio_0 | S_AXI | Reg | 0x4120_0000 | 4K | 0x4120_0FFF |
| axi_bram_ctrl_1 | S_AXI | Mem0 | 0x4000_2000 | 8K | 0x4000_3FFF |
| LFSR_0 | S00_AXI | S00_AXI_reg | 0x43C0_0000 | 4K | 0x43C0_0FFF |

**Deliverables:**

1) Download your bit file to the ZedBoard and demonstrate your code running on the ZedBoard.

2) Upload your C-code and Vivado directories to Canvas.

3) Write a README summarizing the design and testing process

**Tutorials:**

An FPGA Tutorial using the ZedBoard
Zynq Design from Scratch
Zynq Development
Free Electrons
Zynq Training