Spring 2023

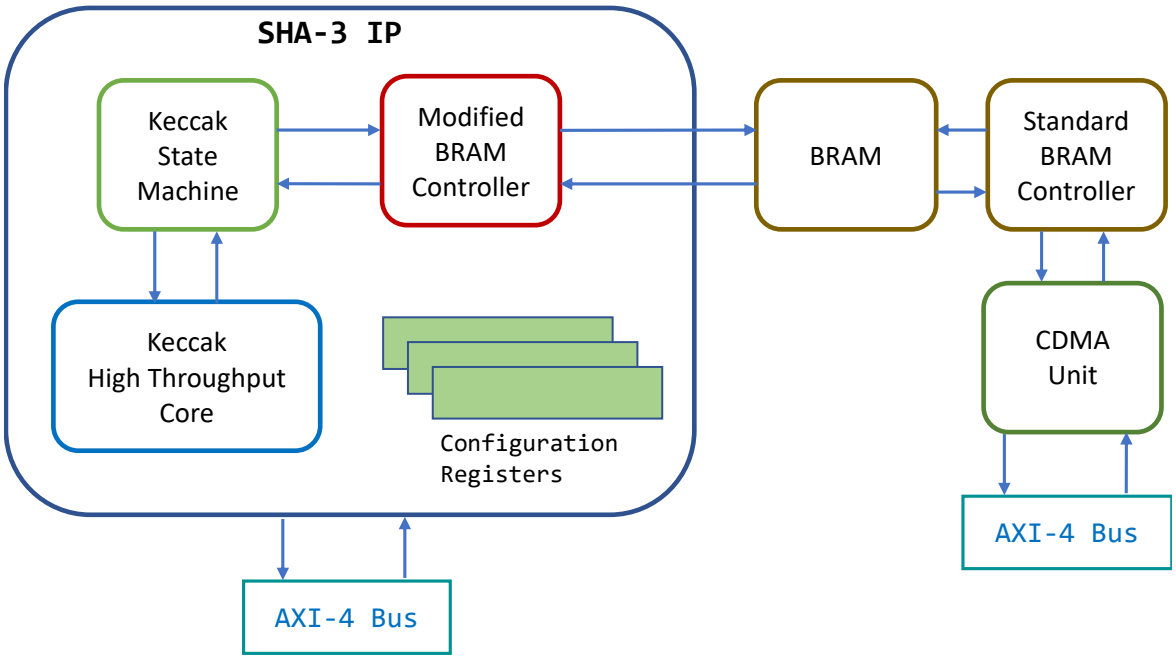
EE 382N-4: Advanced Micro-Controller Systems

Lab Assignment #3

DUE MARCH 24TH, 2023

Lab Goals:

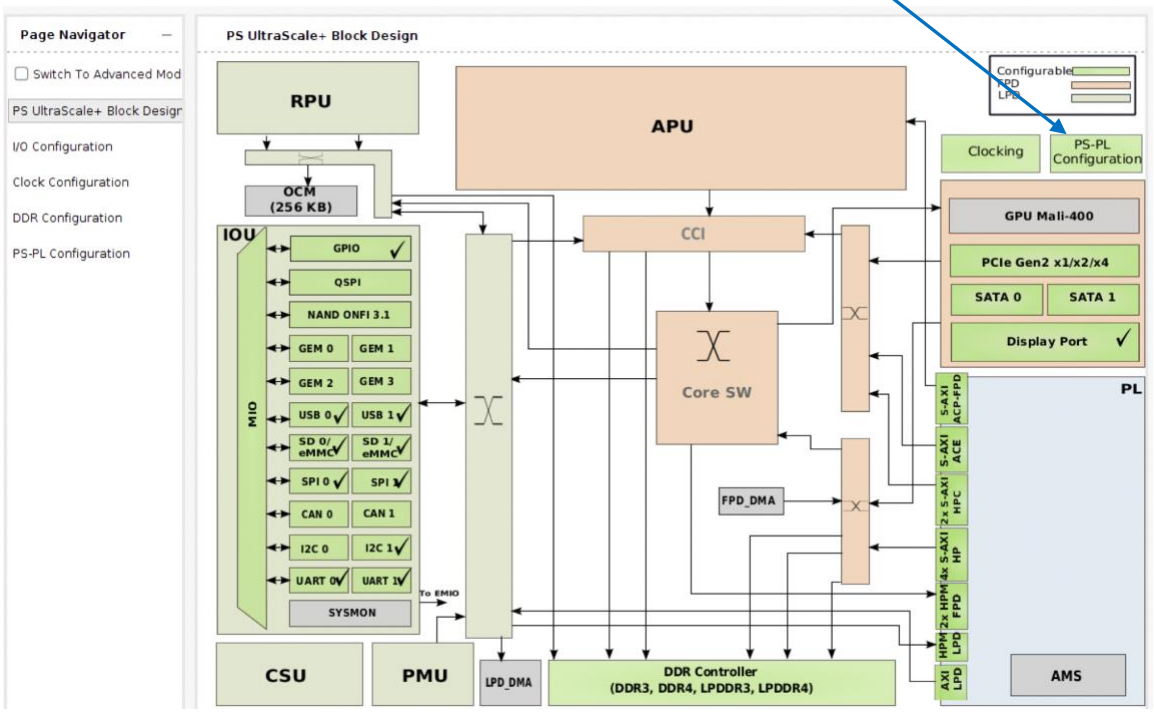
This lab extends Lab-2 to include a SHA-3 (Keccak-512) hardware accelerator and focuses on improving the throughput performance of the system (including both SW and HW). The SHA-3 unit is connected to a dual-ported BRAM as shown in the block diagram below. The BRAM provides the data for the SHA-3 unit. As in Lab 2, the BRAM is loaded using the CDMA unit. The Keccak-512 accelerator is implemented in Verilog in the Accelerator_Lab_3 block. The initial Vivado system design will be provided. It is the first pass implementation from which you will modify to improve the system performance. As mentioned in class there are several ways to improve throughput in the PL block, including wider data buses, improved state machine control parallelism of components, and pipelining. You will need to investigate a minimum of two improvement options and obtain significant performance gains to receive full credit for the lab.



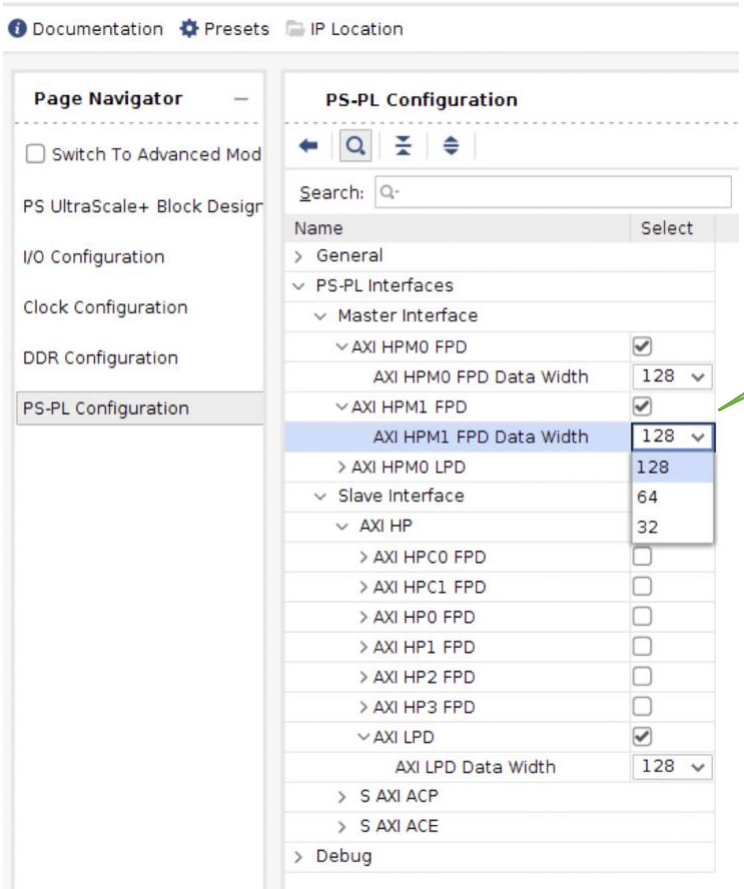
Initial prep:

1. Review the following design specification on the SHA-3 IP we are using for this lab: sha3.pdf
2. Review the AXI bus architecture of the ZynqMP as shown on Page 14 of the technical reference manual: zynq-ultrascale-trm.pdf

3. Review the programmable configuration features of the PS-PL:

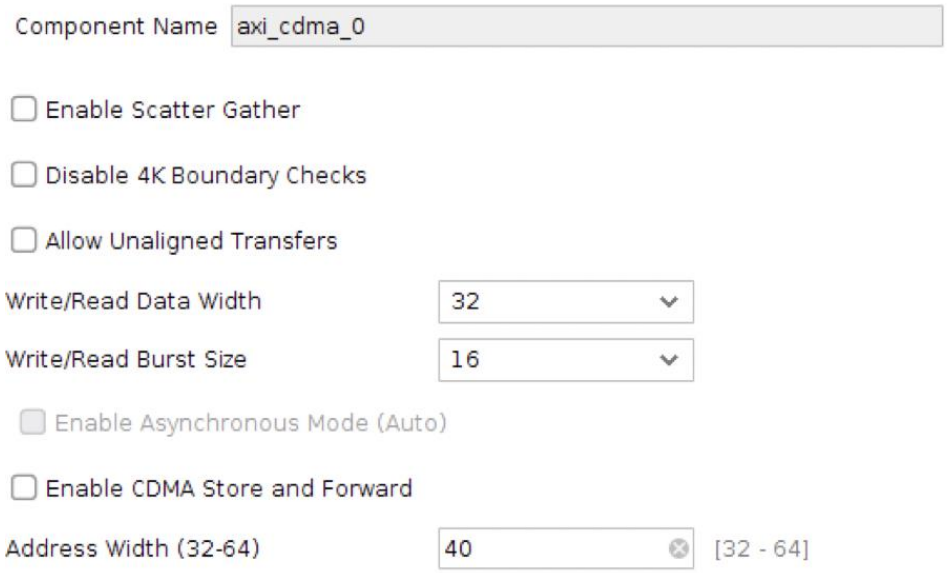


Zynq UltraScale+ MPSoC (3.2)

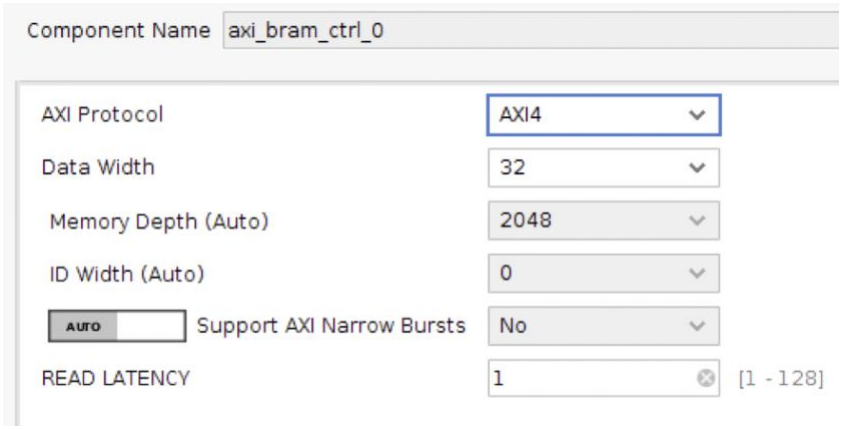


Bus widths can be changed for the Master and Slave AXI ports on the PL

4. Review the customization features of the CDMA unit, such as enabling CDMA Store and Forward.



5. Review the customization features of the BRAM Controller unit:



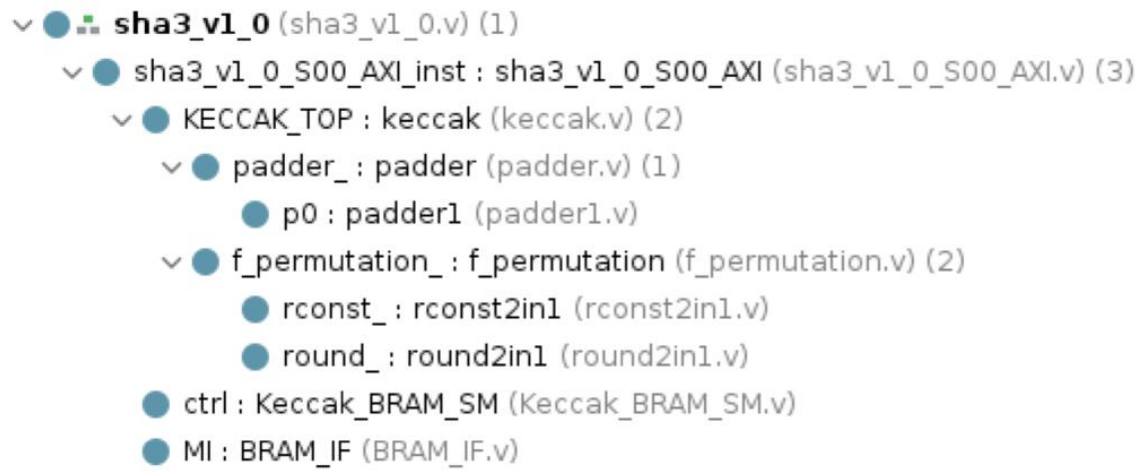
Some things to consider when modifying the BRAM block:

a. Increasing the bus width of the BRAM dual ported memory can be asymmetrical i.e., one port can be wider than the other. However, there are limitations as to the ratio of the difference between the two ports.
b. Increasing the bus width requires increasing the memory size in the address editor (see below).
c. When building large amounts of memory using BRAMs, Vivado synthesizes the new memory by concatenating individual BRAM blocks together. This will cause numerous speed paths as the individual BRAMs are spread across the FPGA fabric. It is highly likely you will run into timing issues when you set the width of the BRAM memory to 1024 bits. Note: 32 of the 36 available individual BRAMs will be used.

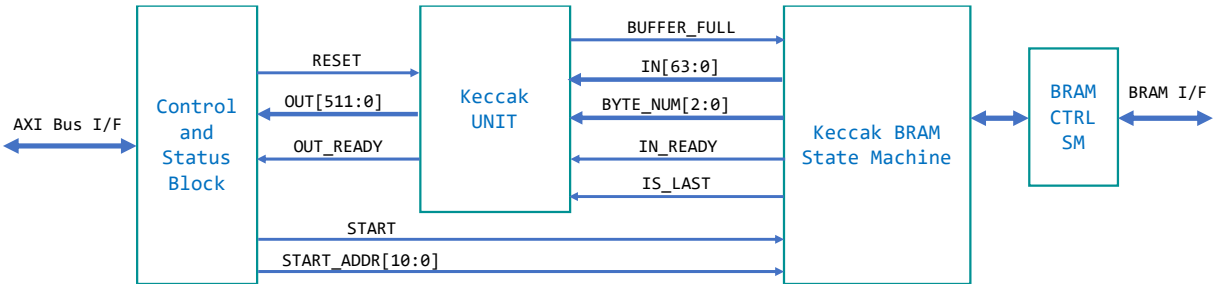
SHA-3 Design

The SHA-3 core is IP from Open-Cores. It is an implementation of Keccak-512 algorithm. There is a difference between the current day SHA-3 algorithm and the Keccak-512 algorithm primarily in the padding core. The following website provides both SHA-3 and Keccak results: https://emn178.github.io/online-tools/keccak_512.html Use this website to confirm the hash results when doing performance enhancements to the Baseline design provided with this lab.

The source tree for the SHA-3 IP in the "ip_repo" is shown below:



The detailed block diagram for the SHA-3 IP is shown below:



The keccak.v module and the sub-modules are the unmodified original files from Open-Cores and should not be heavily modified without extensive verification. There is Verilog simulation and verification environment provided with the IP located here: http://projects.ece.utexas.edu/courses/spring_23/ee382n4-17685/arch/labs/SP23_LAB_3/Keccak_2021_SIMS/keccak-master/

NOTE: Use this environment to verify any changes made to the Open-Cores IP.

The Keccak_BRAM_SM.v module controls the Keccak unit and the customized BRAM I/F module. These modules need lots of help and are a good place to look for performance gains.

Past project teams have focused on the following techniques for improving performance:

- Increase Bus Widths
- Parallel Reads
- Two Pipelined Reads
- Pipeline + Parallelism with FIFO

The register assignments in the Keccak_BRAM_SM.v module are:

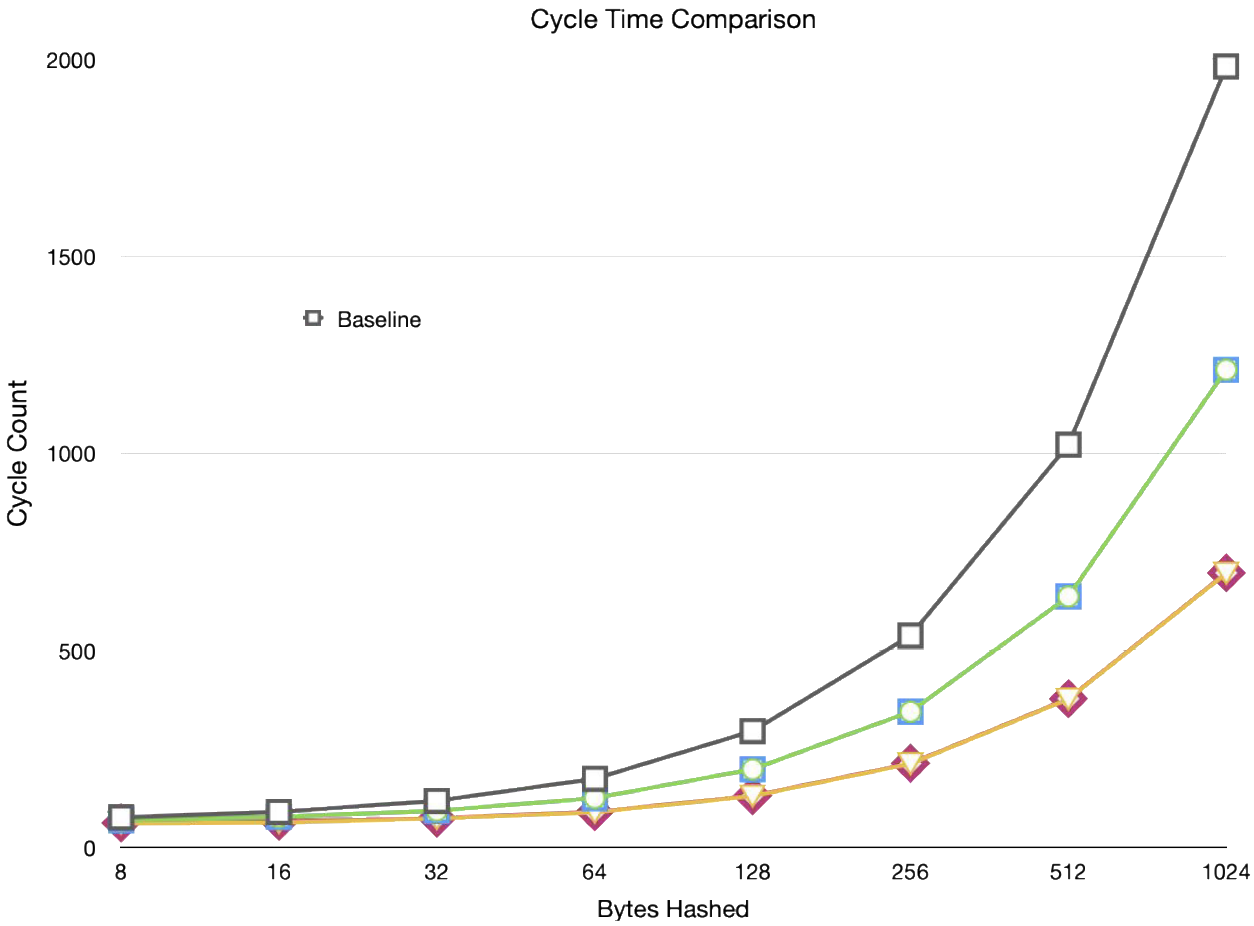
```
Reg 0 : Control register
            Bit[0]:  Start Keccak unit
            Bit[1]:  Reset Keccak unit
Reg 1 : Status Register:
            Bit[0]:  keccak_complete
            Bit[1]:  axi_start_read
            Bit[2]:  axi_start_write
            Bit[3]:  bram_complete
            Bit[8]:  keccak_start_read
            Bit[9]:  keccak_ready
            Bit[10]: keccak_last

Reg 2  : Address to the BRAM          // Peephole (see below)
Reg 3  : Read data from the BRAM      // Peephole (see below)
Reg 4  : Write data to the BRAM       // Peephole (see below)
Reg 5  : keccak_num_bytes             // Number of bytes to
                                      // send to the Keccak Unit
Reg 6  : keccak_bram_addr_start       // Starting BRAM address of data
                                      // to be hashed
Reg 7  : 32'hfeedbeef;                // Signature
Reg 8-E: Not Used
Reg F  : 32'hFEEDCAB2;                   // Another signature
Reg 10 : keccak_hash_reg[511:480];
Reg 11 : keccak_hash_reg[479:448];
Reg 12 : keccak_hash_reg[447:416];
Reg 13 : keccak_hash_reg[415:384];
Reg 14 : keccak_hash_reg[383:352];
Reg 15 : keccak_hash_reg[351:320];
Reg 16 : keccak_hash_reg[319:288];
Reg 17 : keccak_hash_reg[287:256];
Reg 18 : keccak_hash_reg[255:224];
Reg 19 : keccak_hash_reg[223:192];
Reg 1A : keccak_hash_reg[191:160];
Reg 1B : keccak_hash_reg[159:128];
Reg 1C : keccak_hash_reg[127:96];
Reg 1D : keccak_hash_reg[95:64];
Reg 1E : keccak_hash_reg[63:32];
Reg 1F : keccak_hash_reg[31:0];
```

The Keccak_BRAM_SM.v module provides the capability to write to the BRAM through a "peephole" mechanism. This is useful because the CDMA controls the other port to BRAM. This interface is used for debugging. The following shell script shows how to test the Keccak Unit using the peephole.

```
#!/bin/sh

# Fill OCM memory with all fffff's
/bin/fm 0xfffc0000 0xffffffff 1023  0 > /dev/null

# Write the Destination address to the CDMA unit
/bin/pm 0xB0000020 0xB0080000

# Write the Source address to the CDMA unit
/bin/pm 0xB0000018 0xfffc0000

# Start the DMA transfer
/bin/pm 0xB0000028 0x1023

# Reset Keccak Unit
/usr/bin/pm 0xA0080000 0x02 > /dev/null
/usr/bin/pm 0xA0080000 0x00 > /dev/null

# Generate hash for 16 bytes
/usr/bin/pm 0xA0080014 16 > /dev/null

# Start Keccak Unit"
/usr/bin/pm 0xA0080000 0x01 > /dev/null
/usr/bin/pm 0xA0080000 0x00 > /dev/null

echo "Expected results:
0xa0080040 = 0xc2d1e942
0xa0080044 = 0x1945bdac
0xa0080048 = 0x3aa06074
0xa008004c = 0x8d4b7f67
0xa0080050 = 0x7bcb22b5
0xa0080054 = 0xa06be2e3
0xa0080058 = 0xe1cc4a7c
0xa008005c = 0xaa8e8b9e
0xa0080060 = 0xddbb88a7
0xa0080064 = 0x43c530cb
0xa0080068 = 0x58367898
0xa008006c = 0x8f4d5893
0xa0080070 = 0xdc4fb0f5
0xa0080074 = 0xaefa63b2
0xa0080078 = 0xcd06dbf8
0xa008007c = 0xdd1843e5
"
echo "Actual results:"
# Dump memory
/usr/bin/dm 0xA0080040 16
```

Lab 3 procedure

As mentioned above, you will be modifying components of the provided design and determining the amount of performance gains compared to the Baseline. The Capture Timer from Lab-2 will provide the cycle count for each of the tests performed. The tests will be run on 8-1024 bytes hashed as shown below. You will be comparing two performance improvement techniques to the BASELINE implementation.



1. Download the tar file containing the SHA-3 Vivado project from: LAB_3.tar.gz This is a self-contained project, with an updated "ip_repo" which has the SHA-3 IP. The final directory structure will look like this:

```
BASELINE
CONSTRAINTS
ip_repo
Performance-1
Performance-2
```

   Do not modify the BASELINE files. Copy the Vivado project files from the BASELINE directory into the other two performance directories and make modifications there.

2. Download the new DTB from **here**
   We need a new DTB because the size of the BRAM memory grew from 8K to 128K
   a. Convert the DTB to a DTS and confirm that the new address map is correct.
   b. Change the compatible statement to match what you have in your kernel modules. Convert the DTS back to a DTB and rewrite it back to the BOOT sector.
   c. Reboot the Ultra96 board

3. Download the system.bit.Lab_3 file from **here** and insert it into the FPGA using the fpga_util command.

4. Confirm that the BASELIINE project is working correctly using the script above. Make changes to OCM data and rerun the script. Change the following fm command from

   ```
   /bin/fm 0xfffc0000 0xffffffff 1023  0 > /dev/null
   ```

   to

   ```
   /bin/fm 0xfffc0000 0xeeeedddd 1023  0 > /dev/null
   ```

   Confirm that the resulting 512-bit hash is correct. This will require that you go the following web page and **check the correct answer**

5. As mentioned above there are several techniques to improve performance:
   a) Increase Bus Widths
   b) Parallel Reads
   c) Two Pipelined Reads
   d) Pipeline + Parallelism with FIFO

   Research the web for other examples of performance improvement techniques.
   In your report explain why you picked the two techniques you used for this lab assignment.

   Graph the performance improvements using the template shown above. If there are performance improvements explain where they are coming from. If not explain why there was no improvement. You may need to generate additional instrumentation blocks to help determine what is going on.

6. Determine if should now modify your application software and kernel module to take advantage of the new hardware configuration.
   a. Graph the performance improvements with the new software changes.
   b. If there are performance improvements explain where they are coming from. If not explain why there was no improvement. You may need to generate additional instrumentation blocks to help determine what is going on.

Address Mapping (New):

The address map has changed from Lab_2 because the BRAM grew from 8K to 128K to accommodate increased bus widths (if needed for more performance).

Name	Interface	Slave Segment	Master Base Addr... ^1	Range	Master High Address
Network 0					
/axi_cdma_0					
/axi_cdma_0/Data (40 address bits : 1T)					
/zynq_ultra_ps_e_0/SAXIGP6	S_AXI_LPD	LPD_DDR_LOW	0x00_0000_0000	2G	0x00_7FFF_FFFF
/axi_bram_ctrl_CDMA/S_AXI	S_AXI	Mem0	0x00_B008_0000	128K	0x00_B009_FFFF
/zynq_ultra_ps_e_0/SAXIGP6	S_AXI_LPD	LPD_LPS_OCM	0x00_FF00_0000	16M	0x00_FFFF_FFFF
Network 1					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x0					
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_0/S	S_AXI	Reg	0x00_A000_0000	64K	0x00_A000_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_0/S	S_AXI	Reg	0x00_A000_0000	64K	0x00_A000_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_1/S	S_AXI	Reg	0x00_A001_0000	64K	0x00_A001_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_uart16550_1/S	S_AXI	Reg	0x00_A001_0000	64K	0x00_A001_FFFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A002_0000	4K	0x00_A002_0FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A002_0000	4K	0x00_A002_0FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_1/S_AXI	S_AXI	Reg	0x00_A002_1000	4K	0x00_A002_1FFF
/ULTRA96_IO/BD_CTL_GPIO/axi_gpio_1/S_AXI	S_AXI	Reg	0x00_A002_1000	4K	0x00_A002_1FFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_gpio_2/S_AXI	S_AXI	Reg	0x00_A002_2000	4K	0x00_A002_2FFF
/ULTRA96_IO/Low_Speed_MEZZ/axi_gpio_2/S_AXI	S_AXI	Reg	0x00_A002_2000	4K	0x00_A002_2FFF
/ULTRA96_IO/SYS_MGMT/axi_gpio_3/S_AXI	S_AXI	Reg	0x00_A002_5000	4K	0x00_A002_5FFF
/ULTRA96_IO/SYS_MGMT/axi_gpio_3/S_AXI	S_AXI	Reg	0x00_A002_5000	4K	0x00_A002_5FFF
/ULTRA96_IO/SYS_MGMT/system_management_wiz	S_AXI_LITE	Reg	0x00_A002_6000	8K	0x00_A002_7FFF
/ULTRA96_IO/SYS_MGMT/system_management_wiz	S_AXI_LITE	Reg	0x00_A002_6000	8K	0x00_A002_7FFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_0/S00_	S00_AXI	S00_AXI_reg	0x00_A003_0000	64K	0x00_A003_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_0/S00_	S00_AXI	S00_AXI_reg	0x00_A003_0000	64K	0x00_A003_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_1/S00_	S00_AXI	S00_AXI_reg	0x00_A004_0000	64K	0x00_A004_FFFF
/ULTRA96_IO/Low_Speed_MEZZ/PWM_w_Int_1/S00_	S00_AXI	S00_AXI_reg	0x00_A004_0000	64K	0x00_A004_FFFF
/Capture_Timer_0/S00_AXI	S00_AXI	S00_AXI_reg	0x00_A005_0000	4K	0x00_A005_0FFF
/Capture_Timer_0/S00_AXI	S00_AXI	S00_AXI_reg	0x00_A005_0000	4K	0x00_A005_0FFF
/sha3_0/S00_AXI	S00_AXI	S00_AXI_reg	0x00_A008_0000	128K	0x00_A009_FFFF
/sha3_0/S00_AXI	S00_AXI	S00_AXI_reg	0x00_A008_0000	128K	0x00_A009_FFFF
/axi_cdma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_B000_0000	4K	0x00_B000_0FFF
/axi_cdma_0/S_AXI_LITE	S_AXI_LITE	Reg	0x00_B000_0000	4K	0x00_B000_0FFF

Deliverables

1) Vivado files
2) Application and kernel module code
3) Graphs and other visual aids
4) Writeup answering the questions in the Lab Procedure section above.
5) Setup a time with the TA to demo and review your results.