# Spring 2024

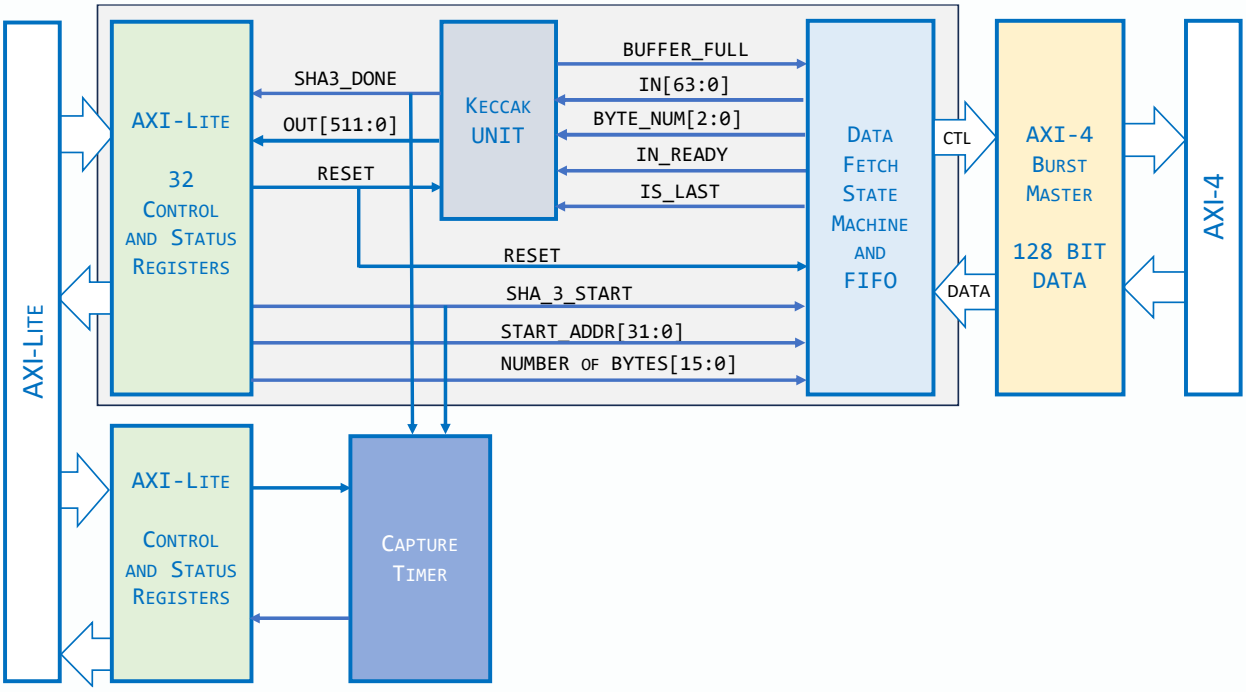
## EE 382N-4: Advanced Micro-Controller Systems

### Lab Assignment #3

**Due March 29th, 2024**

### Overview:

This lab combines portions of Lab 1 & Lab 2 to include a SHA-3 accelerator block that connects to the AXI-4 bus as shown in the block diagram below:




The SHA-3 Verilog (Keccak Unit) is instantiated in the AXI-LITE slave module (AXLM). There is a Data-Fetch state machine (DFSM) that is also instantiated in the AXLM that communicates to the AXI4 Burst-Master (AXLM) block that was used in Lab #1. The big difference between Lab #1 and Lab #3 is that all data transfers from the OCM are 128 bits wide. The crossbar switch was removed between the Burst-Master and the PS to force all transactions to be 128 bits. The Keccak unit consumes 64-bits at a time and has some time variability between operations. This will require some buffering (FIFO?) between the DFSM and the Keccak Unit. A "chunk" of data will be deposited in the OCM and fetched by the DFSM.

The number of bytes to be fetched will be in a control register in the AXLM The Keccak unit will generate a 512-bit result for the "chunk" of data. The hash is packed into sixteen 32-bit registers in the AXLM. The Keccak Unit will interrupt the processor once it has completed the Keccak-512 operation. This lab requires one kernel module to handle the interrupt.

The hashing result will be compared against a SW generated value using a Keccak conversion routine that is available on the WEB.

We will be using Keccak-512 Verilog code from OpenCores. The code has already been instantiated in the Vivado SHA-3 baseline model that you will download. You will also be downloading an older version of the lab where a CDMA was used to move to data from the OCM to BRAM. A custom BRAM I/F was used to move data from the BRAM to the Keccak Unit. The DFSM for this Lab should be patterned after the custom BRAM I/F state machine used in the previous lab.

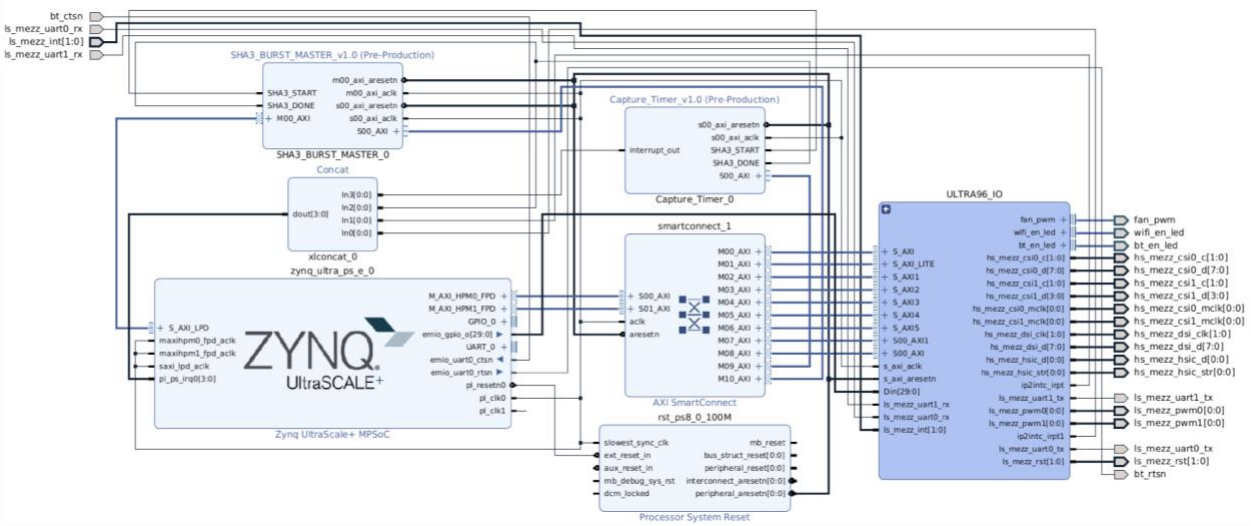
## Downloads:

1) Download the Spring 2023 LAB #3 Vivado schematic, IP_REPO and Constraints from HERE. Put this in a directory that is obvious that it is not the BASELINE for this lab. Make sure you **don't** share IP_REPOs or CONSTRAINTS with the BASELINE that you are going to download next.

2) Download the SHA-3 BASELINE Vivado schematic, IP_REPO and Constraints for this lab from HERE.

## Design Process:

1) You should see the following schematic when you open the BASELINE Vivado schematic for this LAB. NOTE: This is a shell schematic that you will need complete.




2) Modify the Capture-Timer state machine to accommodate the SHA3_START and SHA3_DONE signals. The SHA3_START will start the DFSM in the SHA3_BURST_MASTER and the timer at the same time. Once the 512-bit hash is generated the SHA3_DONE signal will stop the timer while causing an interrupt. As mentioned earlier you will need to repurpose the Timer-Capture kernel module from Lab #2 to handle the interrupt. The timer will measure the amount of time it takes to run the entire SHA-3 hashing function.

3) The SHA3_BURST_MASTER component needs to be updated. You have the flexibility to architect your own solution. There is shell to start from in the Verilog code that is provided with the schematic. It is highly encouraged to look at the Keccak control state machine from last Spring.

4) You will need to create your own DTB for this Lab. The instructions are located HERE

5) Write an application program that can support receiving a string or file name to perform the hashing function:

```
./test_keccak-512 -s "123 abc"
./test_keccak-512 -f file.txt
```

The application will put the hex values in the OCM, determine the number of bytes and start the accelerator.

## Deliverables:

1) Demonstrate a working Keccak accelerator in the FPGA using PM & DM commands. Build a script like the following:

```
#!/bin/sh

# Fill OCM memory with all fffff's
/bin/fm 0xfffc0000 0xffffffff 4  > /dev/null

# 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 = 0x00000000
0xa0080044 = 0x00000000
0xa0080048 = 0x3aa06074
0xa008004c = 0x00000000
0xa0080050 = 0x70000000
0xa0080054 = 0xa06be2e3
0xa0080058 = 0x00000000
0xa008005c = 0xaa8e8b9e
0xa0080060 = 0x00000000
0xa0080064 = 0x43c530cb
0xa0080068 = 0x50000000
0xa008006c = 0x8f4d5893
0xa0080070 = 0xdc4fb0f5
0xa0080074 = 0x00000000
0xa0080078 = 0xcd06dbf8
0xa008007c = 0x00000000
"
echo "Actual results:"

# Dump memory
/usr/bin/dm 0xA0080040 16
```

   Demonstrate how you determined the expected results answer for generating a hash of sixteen hexadecimal “F”s.

   *NOTE: The above script works for the 2023 project only. You will need to modify it accordingly*

2) Demonstrate the SW application that accepts inputs a string (sixteen “F”s) or a file name and runs executes the hashing function in the accelerator. Use various configurations of sterams and files to confirm correct operation of your implementation.

3) Measure the conversion time and display using the same format as used in Lab #2.

4) Write a design report summarizing the design and testing process.

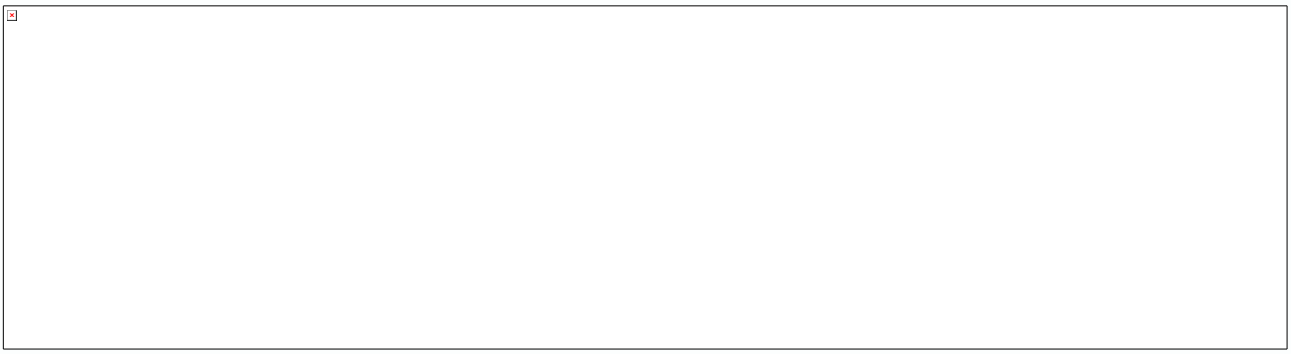
5) Upload your files to CANVAS.

## Verilog Simulation Environment:

The Verilog for the Keccak unit itself can be simulated on the LRC machines. This can be quite helpful if you want to confirm what you are seeing in Vivado. The Verilog simulation directory is located at:
https://projects.ece.utexas.edu/~mcdermot/arch/labs/SP24_LAB3/Keccak_2024_SIMS/keccak-master/

To run a simulation, execute the following commands on one of the Linux servers:

```
module load cadence/2022
cd simulation/verilog_sims/simulation
irun -f irun.args -input restore.tcl
```



## Keccak-512 Tutorials


[1] Guido Bertoni, Joan Daemen, Michael Peeters and Gilles Van Assche,
The Keccak sponge function family: Specifications summary
http://keccak.noekeon.org/specs_summary.html

[2] NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition,
NIST. Oct. 2012.
http://www.nist.gov/itl/csd/sha-100212.cfm

[3] SHA-3, Wikipedia, the free encyclopedia,
http://en.wikipedia.org/wiki/SHA3

[4] The Keccak reference, version 3.0,
http://keccak.noekeon.org/Keccak-reference-3.0.pdf

[5] Keccak implementation overview, version 3.2,
http://keccak.noekeon.org/Keccak-implementation-3.2.pdf

[6] Understanding Keccak512
https://he3.app/en/blogs/understanding-keccak512-hash-a-comprehensive-guide-for-developers/