Introduction/Overview of Lab 1

ECE 4156/6156 Advanced Hardware-Oriented Security and Trust

Spring 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Lab 1 is Assigned and is Due on Jan. 26th

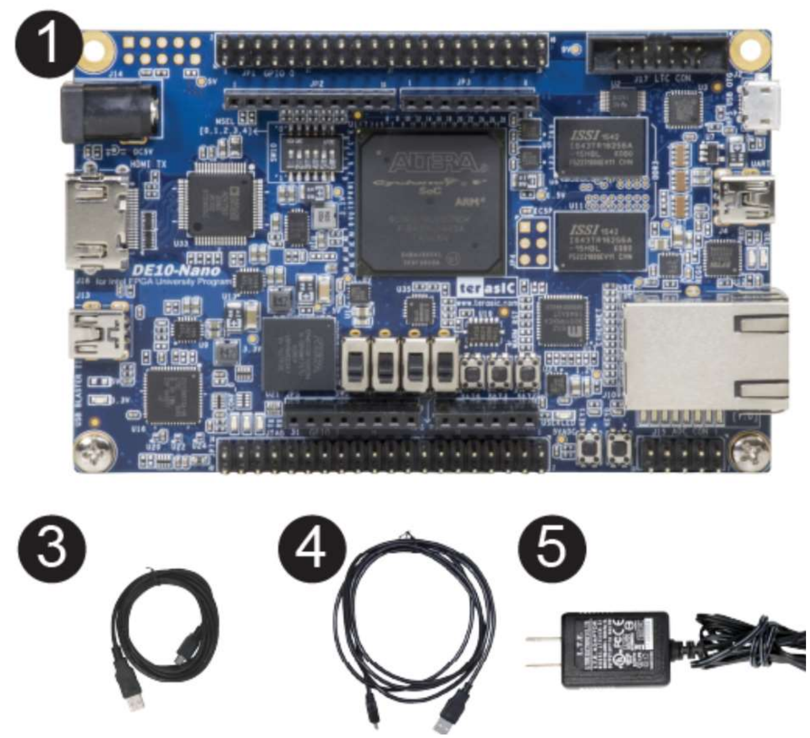
- Check out course web page
  - http://mooney.gatech.edu/Courses/ECE4156/
- Look under “Homeworks/Labs/Exams”
  - http://mooney.gatech.edu/Courses/ECE4156/hwlabexam/index.html
- Lab2 will be posted in two weeks, approximately

Prelab

Simulating, Synthesizing and Implementing a Secure Hash Algorithm using ModelSim and Quartus

DE-10 Board Contents

Note: the Quick Start Guide may be missing, but this is not needed as the document is available online.



1. DE10-Nano Board
2. DE10-Nano Quick Start Guide
3. Type A to Mini-B USB Cable x1
4. Type A to Micro-B USB Cable x1
5. Power DC Adapter (5V)

Simulating SHA256 Using ModelSim

- The Secure Hash Algorithm (SHA) is a cryptographic hash function used to map data of arbitrary size into a fixed size and is designed to be a one-way function
- In this lab, we will first simulate and functionally verify a VHDL implementation of SHA256
- We will learn how to create a VHDL testbench to perform the simulation by
  - Generating clock and reset signals
  - Issuing input stimulus and checking for expected output
- We will verify the correct operation of the hardware implementation of the SHA256 by comparing the results to a software implementation

Synthesizing and Implementing SHA256 Targeting an Intel FPGA

- We will synthesize the SHA256 top level module along with all the submodules needed to implement the algorithm
- We will also implement and map the design to an Intel FPGA
- SPOILER ALERT! Some of the steps might generate some errors
  - You will be asked to identify those errors and provide a hypothesis as to why you think these errors are happening
  - Do not worry YET! You will not be asked to solve these problems in this lab

Prelab Outcome

- Learn VHDL as you go
  - VHDL basics (entities, architectures, …)
  - VHDL testbenches
  - Note: VHDL is NOT a software language. If you are unfamiliar with VHDL, try to approach VHDL without bringing any assumptions from how software functions.
- Learn how to simulate a hardware design using Mentor Graphics ModelSim to check for correct module behavior
  - Monitoring specific signals in a design and exporting signal waveforms
  - Checking for correct results of specific test cases
- Learn how to synthesize and implement a hardware design targeting FPGAs using Quartus
  - Monitoring resource utilization results
  - Analysis of possible synthesis and implementation errors

Linux command : sha256sum

Sha256 --help

```
ychen414@ece-linlabsrv01.ece.gatech.edu>sha256sum --help
Usage: sha256sum [OPTION]... [FILE]...
Print or check SHA256 (256-bit) checksums.
With no FILE, or when FILE is -, read standard input.

  -b, --binary         read in binary mode
  -c, --check          read SHA256 sums from the FILEs and check them
      --tag            create a BSD-style checksum
  -t, --text           read in text mode (default)
  Note: There is no difference between binary and text mode option on GNU system.

The following four options are useful only when verifying checksums:
      --quiet          don't print OK for each successfully verified file
      --status         don't output anything, status code shows success
      --strict         exit non-zero for improperly formatted checksum lines
  -w, --warn           warn about improperly formatted checksum lines

      --help     display this help and exit
      --version  output version information and exit

The sums are computed as described in FIPS-180-2.  When checking, the input
should be a former output of this program.  The default mode is to print
a line with checksum, a character indicating input mode ('*' for binary,
space for text), and name for each FILE.

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
For complete documentation, run: info coreutils 'sha256sum invocation'
```

Linux command : sha256sum

1. Create a text file with your input (I call it "test_vector1.txt")
   - without spaces or next-line character
   - Content of text file is "c98c8e55"
2. Generate a binary file using your text file as input
   - "xxd –r –p test_vector1.txt > test_vector1.bin"
3. Run sha256sum
   - "sha256sum –b test_vector1.bin"

```
ychen414@ece-linlabsrv01.ece.gatech.edu>xxd -r -p test_vector1.txt  > test_vector1.bin
ychen414@ece-linlabsrv01.ece.gatech.edu>ls
test_vector1.bin  test_vector1.txt
ychen414@ece-linlabsrv01.ece.gatech.edu>sha256sum -b test
test_vector1.bin  test_vector1.txt
ychen414@ece-linlabsrv01.ece.gatech.edu>sha256sum -b test_vector1.bin
7abc22c0ae5af26ce93dbb94433a0e0b2e119d014f8e7f65bd56c61ccccd9504 *test_vector1.bin
```

Windows command : Get-FileHash

- Same .bin file as previous slide
- There is no “xxd” command in windows
- Probably easiest to just use the linux commands

```
PS C:\Users\K\Desktop> Get-FileHash test_vector.bin

Algorithm       Hash                                                                    Path
---------       ----                                                                    ----
SHA256          7ABC22C0AE5AF26CE93DBB94433A0E0B2E119D014F8E7F65BD56C61CCCCD9504        C:\Users\K\Desktop\test_vector.bin
```

Lab 1

SHA256 VHDL Code Modification

Fixing Errors in Synthesis and Implementation

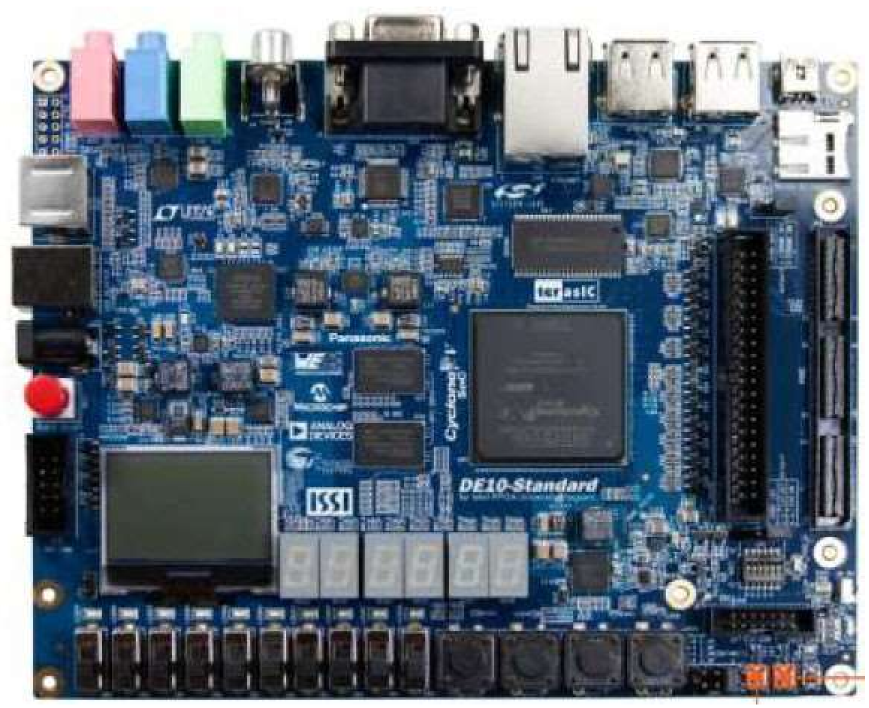
- In this lab, you will be given a VHDL code that contains some errors
- You will be tasked to identify possible design problems by
  - Issuing specific test cases to test for correct functionality
  - Synthesizing the design using specific timing constraints
  - Implementing the design using specific resource constraints
- You will then be tasked to resolve the problems that arose during the simulation, synthesis and implementation of the target design
- VHDL code modifications to fix
  - Possible timing violations
  - Unacceptable resource utilization
  - Functional errors

SHA-256 VHDL Design Hierarchy

- To better understand a design, you have to first understand how the modules and submodules interact together
- SHA 256 hierarchy:
- gv_sha256.vhd
  - sha256_control.vhd
  - sha256_padding.vhd
  - sha256_msg_sch.vhd
  - sha256_hash_core.vhd
  - sha256_regs.vhd
  - sha256_Kt_rom.vhd
  - sha256_Ki_rom.vhd

DE-10 FPGA Development Board

- FPGA has a limited number of I/O pins
- Check the number of I/O pins required by our current top level design
- The DE-10 FPGA has a total of 288 pins, only!
- Thus the current SHA-256 design cannot be implemented as is on the FPGA or else some of the module's inputs and outputs won't be mapped to pins

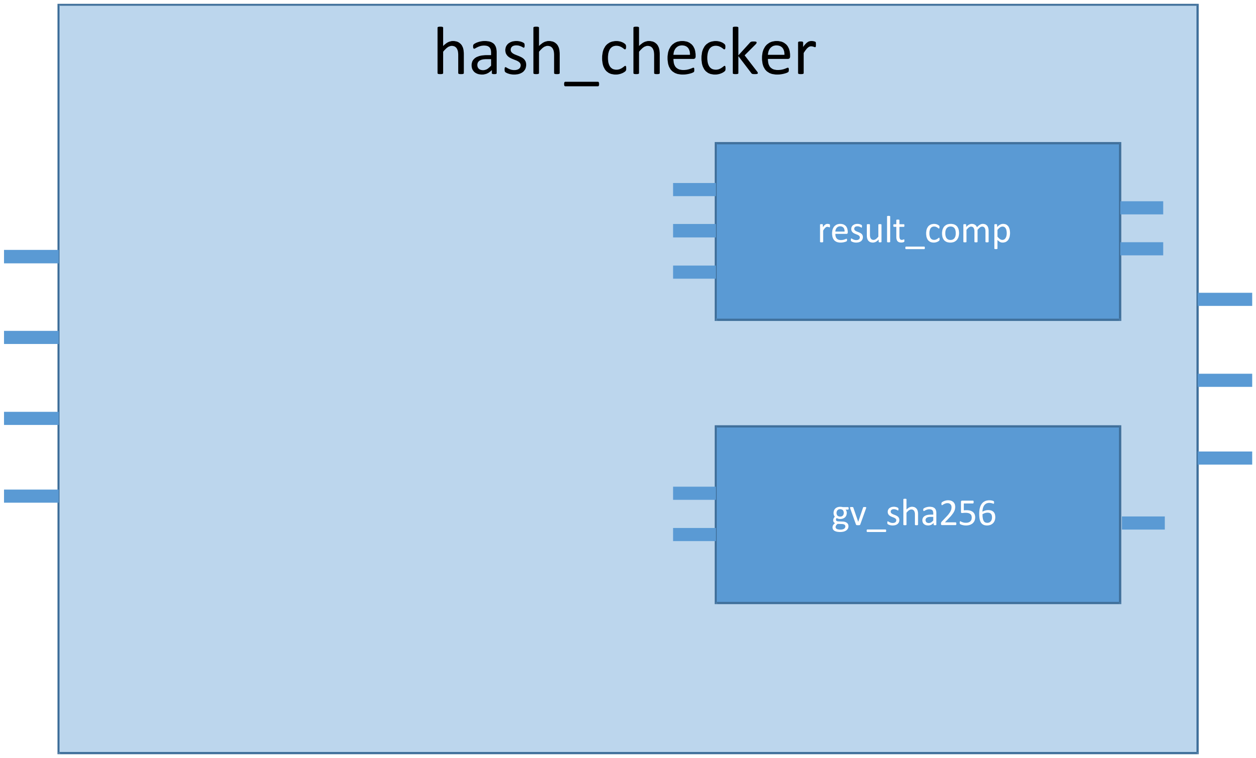


Solve I/O Utilization Problem

Lab 1

*Note that the number of I/O pins in the pictures are not accurate representation of the actual design

Lab 2



Control Assumes a Certain Behavior

- Passing incorrect signal values at unexpected time instances could cause errors
- Testbench cases in PreLab have some errors that are fixed for you
- In Lab 1, the testbench will have some errors in feeding the input test cases
- You will have to fix the timing now!
- HINTS
  - Look through the sha256_control.vhd file for expected behavior
  - It’s true that a picture is worth a thousand words, but it might be also true that a few words are worth a thousand lines of code!

Lab 2

Verifying AES In VHDL and C

Using the microSD Card Slot on the DE10 Board

- Part of lab 2 involves using a microSD card pre-programmed with an operating system and AES encryption program that interfaces with the AXI bus on the FPGA
- The jumpers on the board must be set to the positions in the leftmost image shown below
- The microSD card must be inserted into the microSD card slot on the DE10 board as shown in the center and right images below
- You can use a finger to push the SD card into or out of the slot
- **Recommended:** insert the SD card *after* setting the jumpers, and while the board is still powered off