Lab 4
MD5 Cracker

Prof. Kredo

Due: By 23:59 Friday, April 17

Introduction

In this lab you will accomplish several goals:

- Extend Lab 3 to reverse an MD5 digest
- Implement a basic state machine
- Verify module operation by writing testbenches
- Test your design on the DE1-SoC board

Lab 4 must be done individually.

Requirements

Your objective is to extend Lab 3 so that your system finds the 128-bit value that results in a specified MD5 digest entered by the user. You will accomplish this by iterating through all possible 128-bit values until you find the value—this is called a brute force attack on MD5. The user enters a target digest according to the operations outlined in Lab 2 when the `SW9` is low (similar to Lab 3) and then searching for the value after the user toggles `SW9` high and presses the *Action* button. Thus, your Lab 4 implementation will have two modes of operation: (1) identical to Lab 2 when `SW9` is low and (2) find a value that generates the provided MD5 digest when `SW9` is high.

The normal sequence of events is provided below. You will not be tested beyond this process.

1. Ensure `SW9` is in the low position
2. Reset the board with the *Reset* button
3. Enter or modify a 128-bit MD5 digest as described in Lab 2
4. Raise `SW9`
5. Press the *Action* button (which causes your system to search for the value that generates the MD5 digest)
6. Press the $Left$ and *Right* buttons to display the value
7. Lower `SW9` and continue at Step 3

Since your system must be able to repeatedly compute original values for given digests, you should not modify the 128-bit digest input by the user when you search for the value. When `SW9` is high, the $Left$ and *Right* buttons should scroll through the found value digest similar to the operation of Lab 2.

I/O

Use the following buttons for the system operations.

Function	Physical Button
Action	KEY3
Left	KEY2
Right	KEY1
Reset	KEY0

Testbench

For each module in this lab, you must write a testbench that verifies the functionality of the module. You do not need to test every possible input combination (for example, testing all 256 combinations of data from the switches), but your testbenches should thoroughly test your modules. When writing your testbenches, follow the guidelines below.

- Include an error signal that your testbench raises whenever an error occurs.
- Clearly comment your testbench code to document the exact behavior you are testing.
- Ensure your tests cover all paths through your module description.

Your top-level module should contain very little or no logic. For this lab, you do not need a testbench for your top-level module.

Timing

The DE1-SoC board contains a 50 MHz clock for use in your system. **Do not divide or modify the clock unless necessary to meet timing requirements.** If you must divide your clock, do so with a PLL. Your system should run with a single clock.

All submissions must pass the timing analyzer successfully. Only add false path constraints on those paths which truly have no timing requirements.

Evaluation

Submit your assignment by creating a project archive in Quartus II by selecting 'Archive Project...' from the Project Menu. Submit only your project archive (qar) file. Before archiving your project, ensure the 'File set' is 'Source Control' by clicking the 'Advanced...' button in the Archive Project window.

Your lab will be evaluated based on:

- 10 points: Accepts and displays MD5 digest
- 20 points: Iterates over message values
- 40 points: Computes correct image for given MD5 digest
- 10 points: System passes timing tests
- 20 points: Testbenches

Hints

- Test and debug in steps. Start with a subset of the lab requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Evaluation requirements are ordered in a way to ease this implementation process.
- Write your testbenches early and verify each module works before integrating it into the system and before programming your board.
- Altera has many free online video tutorials to help learn the Quartus II software suite.

Synopsis Constraints File

Put the lines below in a text file with the extension sdc, such as lab3.sdc, to inform Quartus about the clock in your system. Replace `YOUR_CLOCK_NAME` with the name of your clock signal in your top-level module.

```
create_clock -period 20 [get_ports YOUR_CLOCK_NAME]
derive_pll_clocks
derive_clock_uncertainty
```

If you use a PLL, then put the undivided clock name as `YOUR_CLOCK_NAME` and not the slower, divided output clock.

Extra Credit

Extra credit is available for systems that achieve certain performance levels, measured in average hash computations per time unit.

The table below illustrates the average hashes per second your system must compute to earn the indicated extra credit points.

Extra Points	**Average Hashes per Second**
5	1×10^6
10	2×10^6
15	30×10^6
20	75×10^6