**HDL Report**

**EECE 643**

**By, Rahul Parikh**

**Index**

[**1.** **System Requirements** 3](#_Toc419120946)

[**** **Summary of System Description and Specification** 3](#_Toc419120947)

[**** **Inputs and Outputs** 3](#_Toc419120948)

[**2.** **Block Diagram** 5](#_Toc419120949)

[**** **Description of each module in the system** 6](#_Toc419120950)

[**** **RTL Schematic View** 7](#_Toc419120951)

[**** **Inside view of md5\_output module** 8](#_Toc419120952)

[**3.** **Algorithmic State machine diagram** 9](#_Toc419120953)

[**4.** **State Table** 11](#_Toc419120954)

[**5.** **Conclusion** 12](#_Toc419120955)

# **System Requirements**

## **Summary of System Description and Specification**

* + Here in lab4 our main objective is to crack the MD5 using a brute force attack. System has input switches through which we can enter the MD5 digest that we wish to crack. We then have to find the corresponding 128-bit input combination that would render the MD5 digest that was input. So, our system needs to iterate through all possible 128-bit combinations that would generate the said MD5 digest.
  + The solution follows a two stage process wherein the first stage the user enters the target 128-bit MD5 digest by taking in an 8-bit value sixteen times through the 8 switches provided on the DE1-SoC board.
  + In the second stage, the system attempts to crack the MD5 digest.
  + Since the DE1-SoC board only has six seven segment displays, only 24-bits of the 128-bit output is displayed at a time. This 24-bit window can be shifted by one byte either in the left or the right direction using the push-to-ON buttons on the DE1-SoC board which have been configured to perform the left or the right shift of the window respectively.

## **Inputs and Outputs**

* + There are total of 13 inputs out of which 9 are switch inputs and the remaining 4 are push buttons.
  + Switch 0-7 are to give system a digest value and switch 9 is to select in which state system should function.
  + When switch 9 is high system is in MD5 cracker state, and when switch 9 is low it is in user input mode.
  + We can go through all 128 bit by push buttons. The push button 0 is reset button. It will reset all the system.
  + Push button 1 is to shift right in the 128 bit register where we can store our MD5 digest, and push button 2 is to shift left in the 128 bit register.
  + Push button 3 will work differently in each state. All other push button will work same way in each state. If our system is in MD5 cracker state and if we hit push button 3 it will start MD5 cracker operation until then despite the switch 9 is high MD5 cracker will not start calculating for new digest user have entered and it will display the previous results. But in user input state if we push this button it will capture the value currently on switch 7 through 0 will be stored into lowest byte of the 24 bit window.
  + We can shift through whole cracked original message signal once it is calculated after pushing button 3 in MD5 cracker state.
  + We have total 42 outputs in our system. We have in our system 6 seven segment display. As one seven segment comprise 7 bits to drive it we need 42 output to drive 6 seven segment.
  + We have 6 seven segment displays to display the values. So in every state we will only display 24 bits out of 128 bits.

# **Block Diagram**

main module

Inputs

clk & rst

One\_time

(Right shift)

One\_time

(Capture or Action)

One\_time

(Left shift)

7 bits outputs to seven segments

key\_input[2:0]

128 bit digest value

out\_window

(Output 30 bit user input)

out\_digest

(Output 30 bit digest value)

md5\_output

(produce md5 digest)

register

(Store 128bit user input)

sw\_in[7:0]

128 bit user input

sw\_in[8]

Control signal

Mux select which display module will work

display

(dis6)

display

(dis5)

display

(dis4)

display

(dis1)

display

(dis2)

display

(dis3)

seg5[6:0]

seg4[6:0]

seg3[6:0]

seg2[6:0]

seg1[6:0]

seg[6:0]1

## **Description of each module in the system**

* + One\_time module: This module is to debounce push button switches. This module synchronized with system clock. It will take original input signal as input and in output produce clean one time pulse for the given input signal. We can select rising or falling edge by giving parameter EDGE appropriate value. If parameter EDGE has value 1 then it is rising edge and if it is 0 then it is falling edge.
  + register module: This module is also synchronized with system clock. This module take switch9 as input. Value of this input select weather this module will work in user input state or MD5 cracker state. This module take four push buttons as inputs. Buttons are left\_ shift, right\_shift, capture and rst. This inputs are the output of one\_time module. If any shift button is pressed it will move 24 bit window by 1byte as per the shift buttons pressed. It will store 8bit value to least significant byte of 24 bit window when capture button is pressed. This module will work in this way when it is working in user input state. But when system is in md5 cracker state this module is disabled or we can say not working. This module will give 128 bit register in output.
  + md5\_output module: This module is also synchronized with system clock. This module will take 128 bit output of register module as input. This module will also take switch9 as input. This module internally has 128 bit counter which start when action button is pressed in md5 cracker state. In user input state this module is disabled. Counter will give internal module md5\_operation input value and at the output of md5\_operation we will have digest value for that counter value. Then output of md5\_operation module will be compared with 128 bits provided by the user. If value is matched then counter value will be given as output to out\_digest module.
  + out\_window module: This module is also synchronized with system clock. This module take 128 bit from register module and push buttons as input. This module will create 5bit value for each seven segment to display hexadecimal number. This module is also determines whether the data on register is valid or not. If data is not valid then it will display dash on seven segment. Also at the most significant byte it will not display anything on the seven segments other than most significant byte. This module also only work in user input state and in md5 cracker state it is disabled.
  + out\_digest module: This module is also synchronized with system clock. This module takes 128bit input from md5\_output module. This module will work only in md5 cracker state. It will create output of 30 bit output. This 30 bit has 6 different 5 bits values to give to 6 different seven segment to display numbers. In this module window will shift by one byte when any of the shift button is pressed.
  + Display module: This is a combinational module. Basically it is 5 to 7 decoder. It will take 5 bit in input from out\_window or out\_digest and produce corresponding 7 bit value to display hexadecimal number on seven segment display.

## **RTL Schematic View**

* + This figure contains how pll and main module is connected. Here in our design we have pll which provides us 75MHz frequency. We can get different frequencies from pll as per the design requirements.

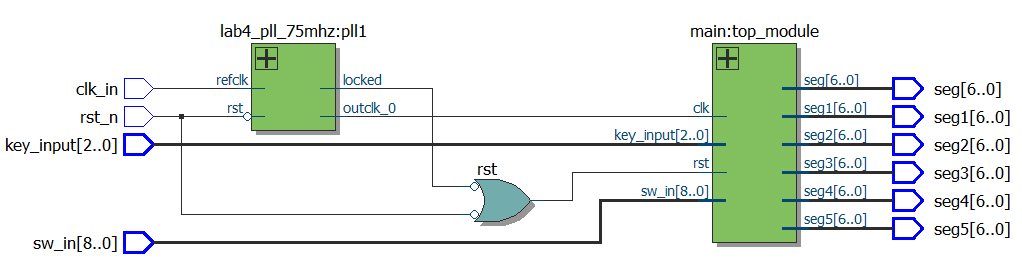
****

Figure 1: Top level module schematic

* This figure shows us how modules which described earlier connected inside the main module. This is the schematic view of block diagram.

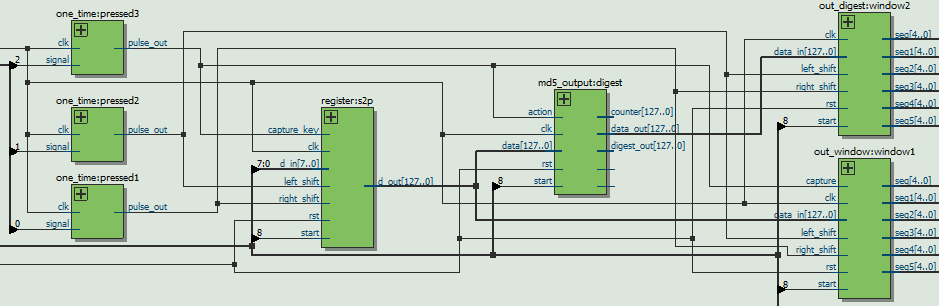


Figure 2: Inside view of main module.

## **Inside view of md5\_output module**

* + Inside md5\_output module md5\_operation modules are connected as per the diagram.
  + Here in md5\_output there are total 64 md5\_operation module is connected in this fashion.
  + Flip-flops are added to increase throughput and system can perform faster.

Q

D

clk

Q

D

round[i]

clk

message[i] [j+1]

current\_state[i] [j+1]

next\_state[i] [j+1]

md5\_operation

(round[i],

phase[j+1])

phase[j+1]

round[i]

next\_state[i] [j]

message[i] [j]

phase[j]

current\_state[i] [j]

md5\_operation

(round[i],phase[j])

# **Algorithmic State machine diagram**

Begin

Display cracked md5 message value.

Done==1

Computing md\_5

State

action==1

sw\_in[8]==1

0

1

Hardware

Reset

Hardware

Reset

User input

State

1

rst==1

1

rst==1

0

0

0

capture==1

0

1

1

Capture 8bit value

left\_shift==1

0

1

0

Move window left by 1 byte

0

left\_shift==1

1

1

Move window right by 1 byte

* **Description of algorithmic state machine**
  + When power button is on and at moment system start system will work in state which is selected by user. If sw\_in[8] is high system will go into md5 cracker state and if it is low then system will start in user input state.
  + In user input state user can enter md5 digest to the system in this state. In this state user can capture 8 bit values and store them into 128 bit register. We can move 24 bit window and store all 128 bit value.
  + In md5 cracker state we can crack the digest which we have entered in the user input state. First in this state we will wait for user to press action button. As action button pressed the internal process of cracking digest will start and system will remain processing state. Once process is done and done bit is high the original message will be displayed on seven segment. In this display state we can move 24 bit window and go through all 128 bit value. Once we done our work in this state then we can go to user input state to change digest value. We can go through this process over and over again as it is infinite loop.

# **State Table**

* In out\_window module we have one state machine. To describe functionality of the state machine I have provided state transition table below.
* Here MSB is rst and LSB is capture.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | {rst,capture} | | | |
| States | 00 | 01 | 10 | 11 |
| Data valid state | Data valid state  Output:data\_valid[I]=1 | Data valid state  Output:data\_valid[I]=1 | Data not valid state  Output:data\_valid[I]=0 | Data not valid state  Output:data\_valid[I]=0 |
| Data not valid state | Data not valid state  Output:data\_valid[I]=0 | Data valid state  Output:data\_valid[I]=1 | Data not valid state  Output:data\_valid[I]=0 | Data not valid state  Output:data\_valid[I]=0 |

# **Conclusion**

In this lab I learnt how to design a Verilog HDL module when dealing with complex requirements or programs. In this lab I have used many key design approaches that can affect the performance of the program by the virtue of the resources used and the various timing constraints that are associated with the said resources that are utilized. This gave me a good understanding of how to manage the resources by analyzing the hardware implication of the written code. For effective debugging, I exploited the benefits offered by writing test benches that provided insight to the real-time timing analysis of the written modules. Furthermore, this approach of debugging makes it easier to evaluate designs regardless of their size. After completion of this project, my overall understanding of concepts surrounding digital design improved to a great extent in addition to learning how to design effectively in Verilog HDL.