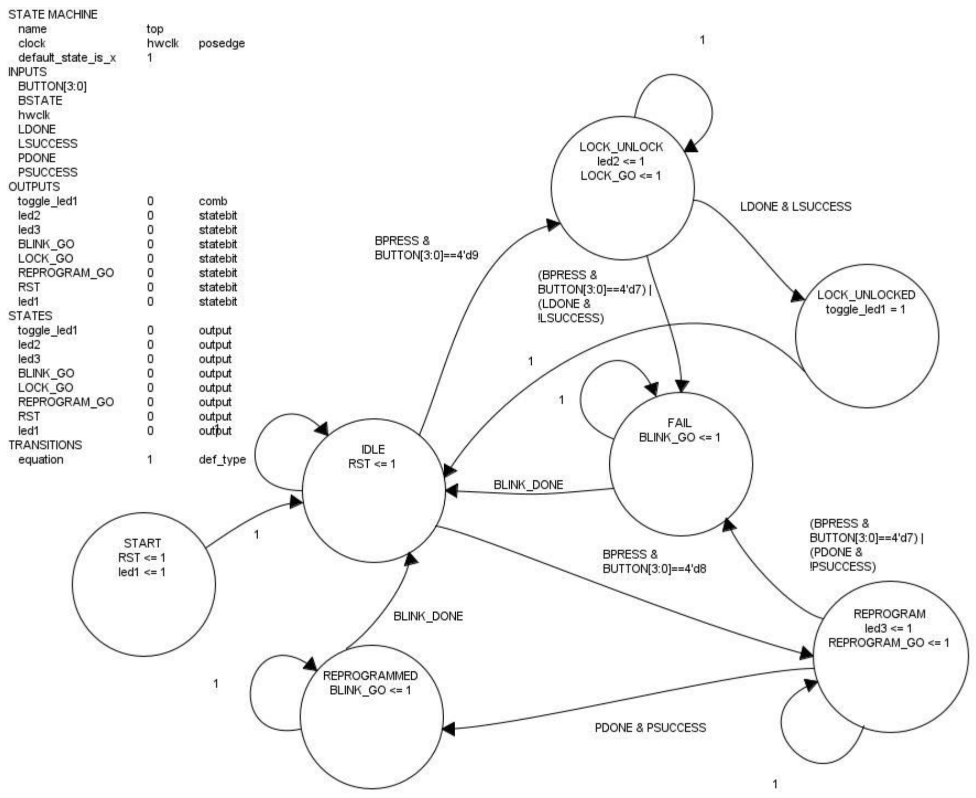
Dylan Lee
205300889

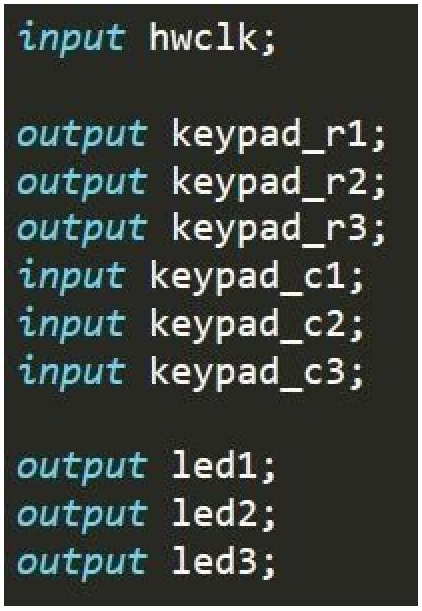
FPGA Project Report

For my implementation of the FPGA project, I included five primary FSMs, three of which include datapaths. In addition to these, I had three smaller helper modules and the provided Button_code modules.

At the highest level of the hierarchy, the top module directs the flow of the entire system. It consists of seven states: START, IDLE, LOCK_UNLOCK, LOCK_UNLOCKED, REPROGRAM, REPROGRAMMED, and REPROGRAMMED. The machine begins in the START state, where registers and values are initialized, as is led1, which represents the lock/unlock state of the machine. From here, the machine immediately transitions to the IDLE state, where it waits for input. If it receives a nine, it will go to the LOCK_UNLOCK state, where the user inputs a UC. A control signal is sent to an instance of the enter_code module and the machine remains in the state until a status signal is received back. LED2 is also lit. If the user inputs the correct UC, the LOCK_UNLOCKED state is reached, where the led is toggled and the machine returns to IDLE. If the user fails, the FAIL state is reached, where the blink module is told to blink 0.5 seconds off, 1 second on three times. After this has been completed, it returns to the IDLE state. From the IDLE state, if an eight is received, the REPROGRAM state is reached, where LED3 is lit, and the REPROGRAM module will run. It will check for the PC, then accept a new UC and check the second UC against the first. Success will lead to REPROGRAMMED, where the BLINK module is made to blink on for 0.2 and off for 0.2 seconds, five times. Failure will lead to the FAIL state. The inputs and outputs of the module primarily deal with the hardware itself, such as the keypad rows and columns, the hardware clock, and the LEDs.The top module also contains the enterDigits module, which was provided, and outputs the current BUTTON, and where it is pressed, BSTATE. This latter signal is put through a synchronizer to obtain BPRESS, which goes high after the falling edge of BSTATE, which corresponds to a button being released. Although the top module doesn't have a RST, it asserts RST on the other modules when it is in the START and IDLE states.

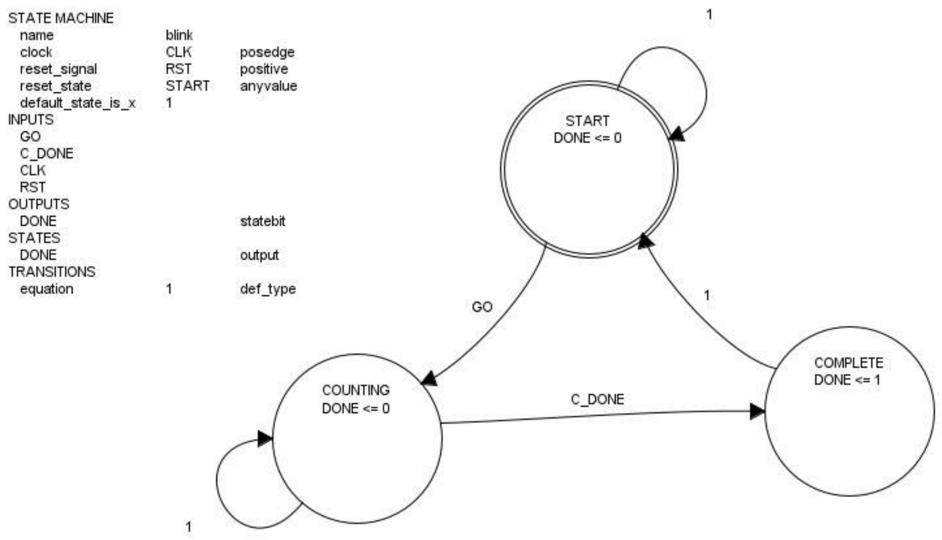


STATE MACHINE		
name	top	
clock	hwclk	posedge
default_state_is_x	1	
INPUTS		
BUTTON[3:0]		
BSTATE		
hwclk		
LDONE		
LSUCCESS		
PDONE		
PSUCCESS		
OUTPUTS		
toggle_led1	0	comb
led2	0	statebit
led3	0	statebit
BLINK_GO	0	statebit
LOCK_GO	0	statebit
REPROGRAM_GO	0	statebit
RST	0	statebit
led1	0	statebit
STATES		
toggle_led1	0	output
led2	0	output
led3	0	output
BLINK_GO	0	output
LOCK_GO	0	output
REPROGRAM_GO	0	output
RST	0	output
led1	0	output
TRANSITIONS		
equation	1	def_type

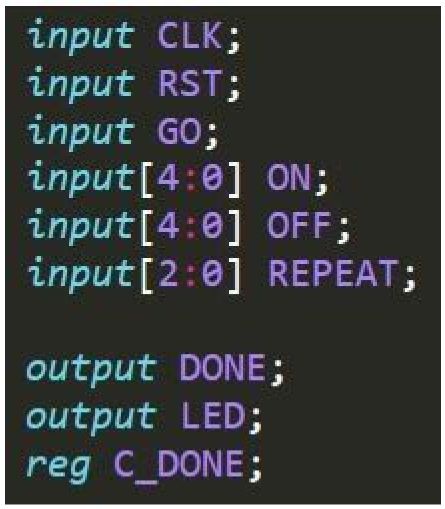


```
input hwclk;

output keypad_r1;
output keypad_r2;
output keypad_r3;
input keypad_c1;
input keypad_c2;
input keypad_c3;

output led1;
output led2;
output led3;
```

The blink module is relatively simple. It consists of three states: START, COUNTING, and COMPLETE. It remains in the START state until given the GO signal from the top module, at which point it transitions to the COUNTING state, where it activates the module's datapath. Once the datapath has run, the COMPLETE state is reached and DONE is asserted, allowing the top module to continue running. The datapath is basically just a counter. It is incremented by one every clock edge, and the status of the LED changes based on its value. The datapath inputs ON, and OFF specify the number of deciseconds that the LED should remain on and off, respectively. The REPEAT input specifies how many times the LED should blink before turning off. These signals are dependent on the top module's state, which also controls which LED is assigned the output LED. I chose to go with the interpretation that led2 blinks if led2 is on when the FAIL state is incurred, and led3 blinks if led3 is on. GO, and C_DONE are status signals.

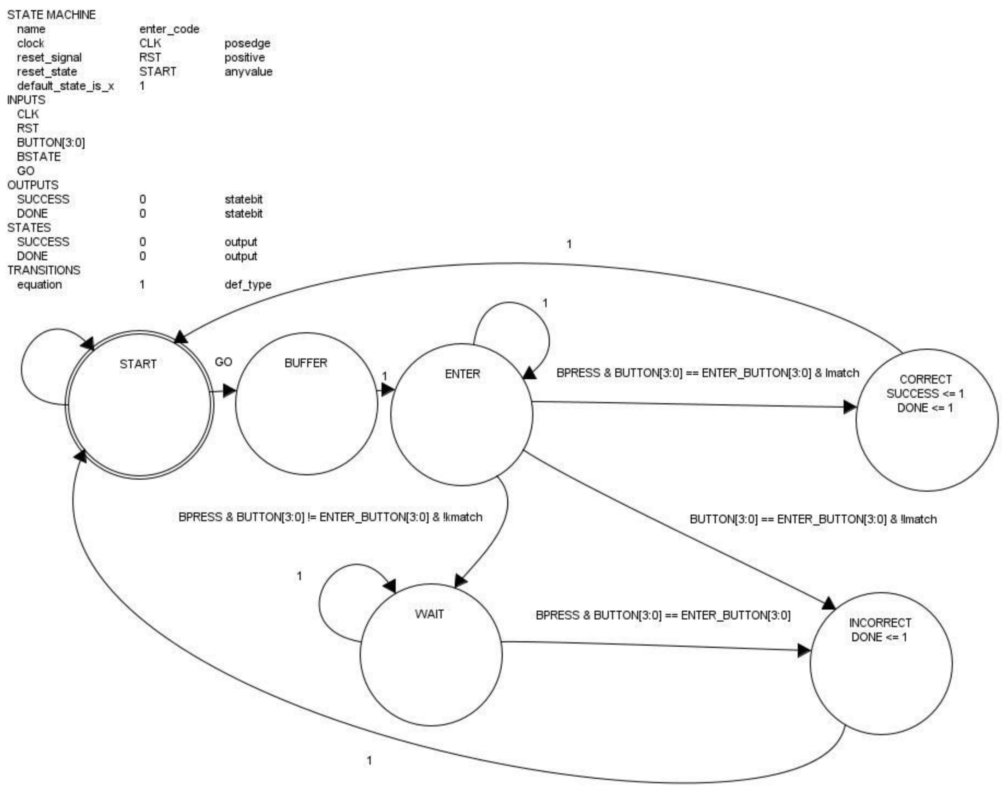


```
STATE MACHINE
   name                blink
   clock               CLK       posedge
   reset_signal        RST       positive
   reset_state         START     anyvalue
   default_state_is_x  1
INPUTS
   GO
   C_DONE
   CLK
   RST
OUTPUTS
   DONE                          statebit
STATES
   DONE                          output
TRANSITIONS
   equation            1         def_type
```



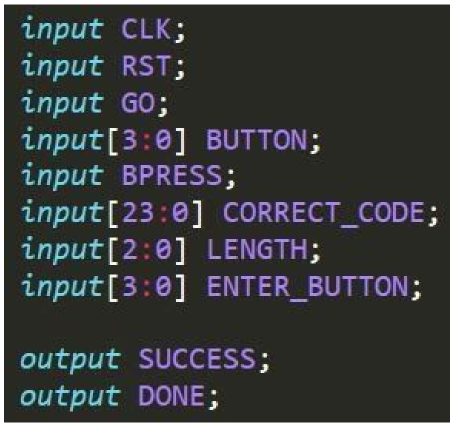
```
input CLK;
input RST;
input GO;
input[4:0] ON;
input[4:0] OFF;
input[2:0] REPEAT;

output DONE;
output LED;
reg C_DONE;
```

The enter_code module is used twice, in the top and reprogram modules. It follows the general structure for a digital lock laid out in discussion and the textbook. It has five states: START, ENTER, WAIT, INCORRECT, and CORRECT. It communicates with its parent module with GO, SUCCESS and DONE. When GO is asserted, the machine transitions to the ENTER state. This is where the datapath portion of this module comes in. If BPRESS is high, BUTTON is compared against the four most significant bits of CORRECT_CODE, which is left-shifted by four at each button press. An index is also kept, and two signals keep track of whether the keys and lengths of the input and code are equal. These two signals are called kmatch and match, respectively. Because this module is used for the UC unlock and the UC reprogramming, an input named ENTER_BUTTON is required, which is 4'd9 in the prior case and 4'd8 in the latter. If at a button press, the button doesn't match the current code, it goes to a wait state where it waits for the ENTER_BUTTON to be pressed, where it then moves to INCORRECT. If the lengths don't match, the machine also goes here. But, if the input matches all of CORRECT_CODE, it goes to the CORRECT state, where DONE and SUCCESS are true. The datapath is where two of the smaller modules play a role. The counter module keeps a three bit counter. Whenever a valid BUTTON is pressed when the machine is in the ENTER state, the index is incremented. This index signal is used in the codeReg to get the corresponding integer from CORRECT_CODE, starting from zero at the MSB of CORRECT_CODE. This is output on current_button.

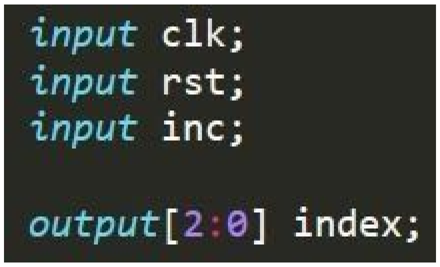


```
STATE MACHINE
  name                enter_code
  clock               CLK          posedge
  reset_signal        RST          positive
  reset_state         START        anyvalue
  default_state_is_x  1
INPUTS
  CLK
  RST
  BUTTON[3:0]
  BSTATE
  GO
OUTPUTS
  SUCCESS             0            statebit
  DONE                0            statebit
STATES
  SUCCESS             0            output
  DONE                0            output
TRANSITIONS
  equation            1            def_type
```

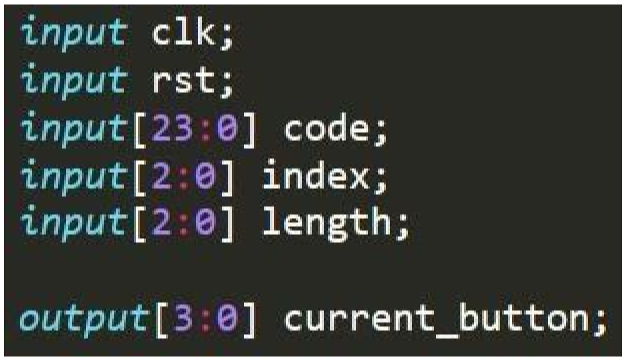
START
GO
BUFFER
1
ENTER
1
BPRESS & BUTTON[3:0] == ENTER_BUTTON[3:0] & Imatch
CORRECT
SUCCESS <= 1
DONE <= 1
1
BPRESS & BUTTON[3:0] != ENTER_BUTTON[3:0] & !kmatch
BUTTON[3:0] == ENTER_BUTTON[3:0] & !lmatch
1
WAIT
BPRESS & BUTTON[3:0] == ENTER_BUTTON[3:0]
INCORRECT
DONE <= 1
1



```
input CLK;
input RST;
input GO;
input[3:0] BUTTON;
input BPRESS;
input[23:0] CORRECT_CODE;
input[2:0] LENGTH;
input[3:0] ENTER_BUTTON;

output SUCCESS;
output DONE;
```

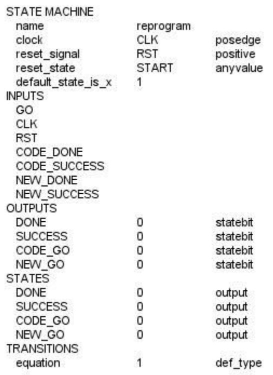


```
input clk;
input rst;
input inc;

output[2:0] index;
```

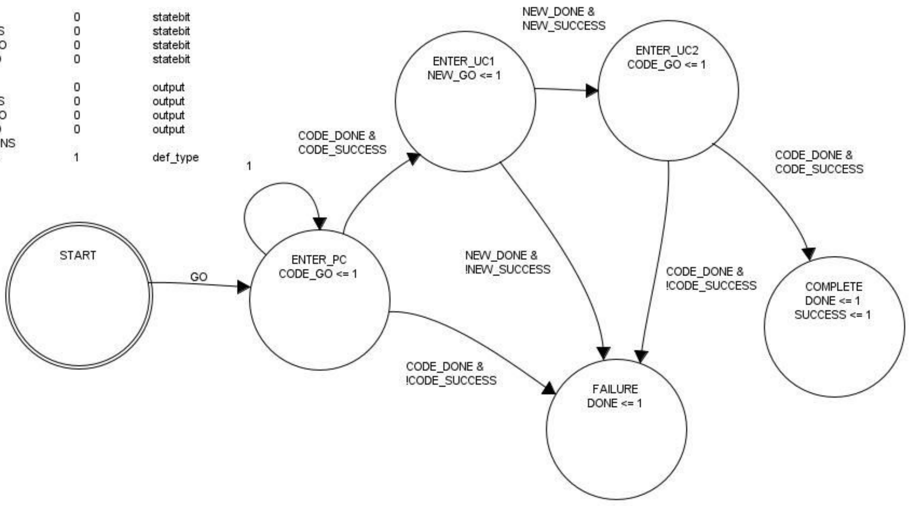


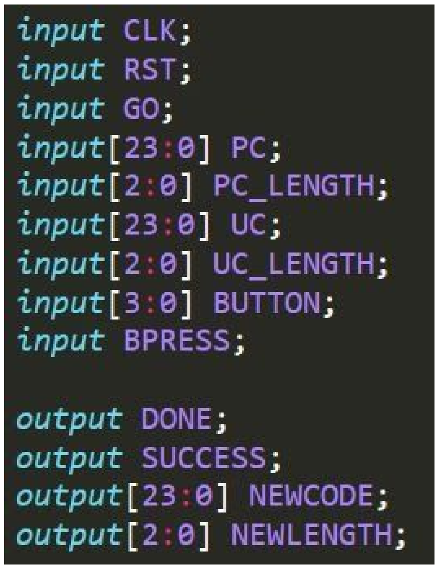
```
input clk;
input rst;
input[23:0] code;
input[2:0] index;
input[2:0] length;

output[3:0] current_button;
```

The reprogram module contains one enter_code module and one enter_new_code module. It has six states: START, ENTER_PC, ENTER_UC1, ENTER_UC2, COMPLETE, and FAILURE. The same module is used for the ENTER_PC and ENTER_UC2 states. It is reset and loaded with new inputs for ENTER_UC2. GO, DONE, and SUCCESS are the control signals between top and reprogram, and NEWCODE and NEWLENGTH contain the value that UC is updated to if the COMPLETE state is reached.



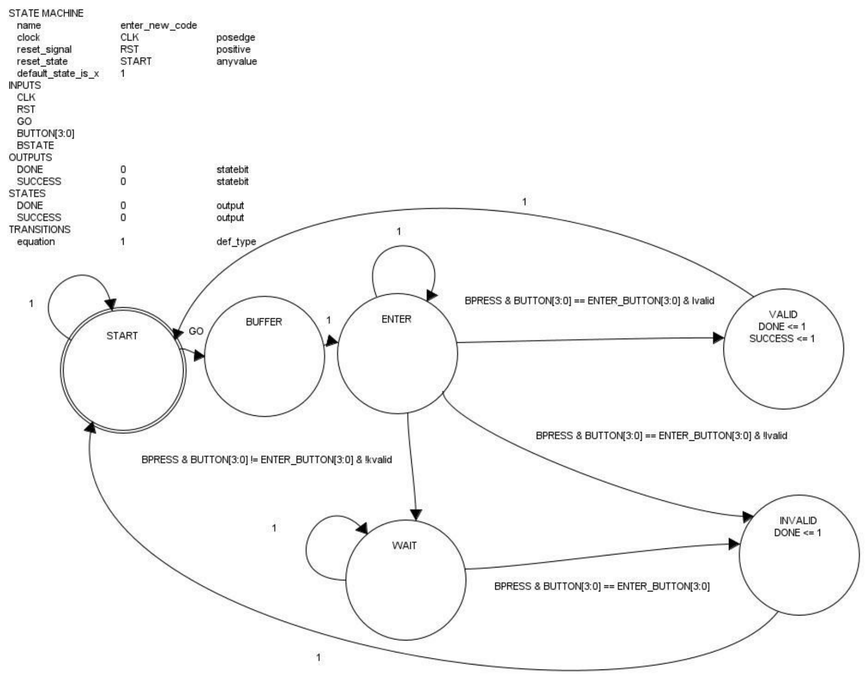
```
STATE MACHINE
  name               reprogram
  clock              CLK          posedge
  reset_signal       RST          positive
  reset_state        START        anyvalue
  default_state_is_x 1
INPUTS
  GO
  CLK
  RST
  CODE_DONE
  CODE_SUCCESS
  NEW_DONE
  NEW_SUCCESS
OUTPUTS
  DONE               0            statebit
  SUCCESS            0            statebit
  CODE_GO            0            statebit
  NEW_GO             0            statebit
STATES
  DONE               0            output
  SUCCESS            0            output
  CODE_GO            0            output
  NEW_GO             0            output
TRANSITIONS
  equation           1            def_type
```





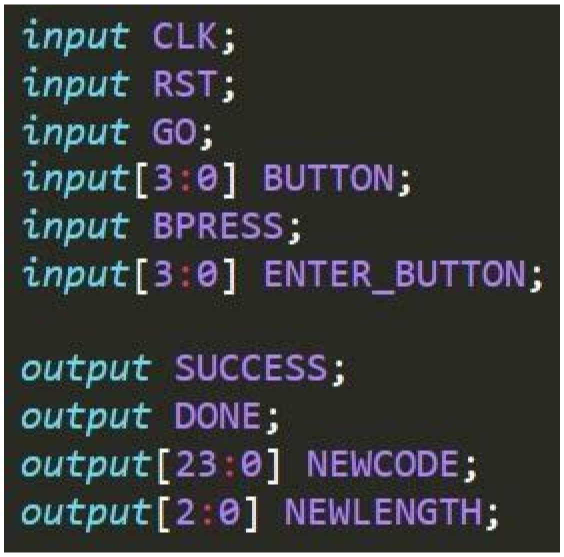
```
input CLK;
input RST;
input GO;
input[23:0] PC;
input[2:0] PC_LENGTH;
input[23:0] UC;
input[2:0] UC_LENGTH;
input[3:0] BUTTON;
input BPRESS;

output DONE;
output SUCCESS;
output[23:0] NEWCODE;
output[2:0] NEWLENGTH;
```

The enter_new_code module is for inputting a new UC to the system. It’s state diagram is almost identical to that of enter_code, except that kmatch and lmatch are replaced by kvalid and lvalid, where are used to determine whether the inputted code is a valid UC. The datapath implementation is therefore different, and it uses a module called newCodeReg. This takes in the current BUTTON press as input, as well as shift, a signal that tells the module whether a valid button press is taking place, and whether the current state is ENTER. If so, newCodeReg left shifts its stored code and increments its index. This code is wired to the output of the enter_new_code module, which is in turn wired to the output of the reprogram module. GO, DONE, and SUCCESS are control signals.

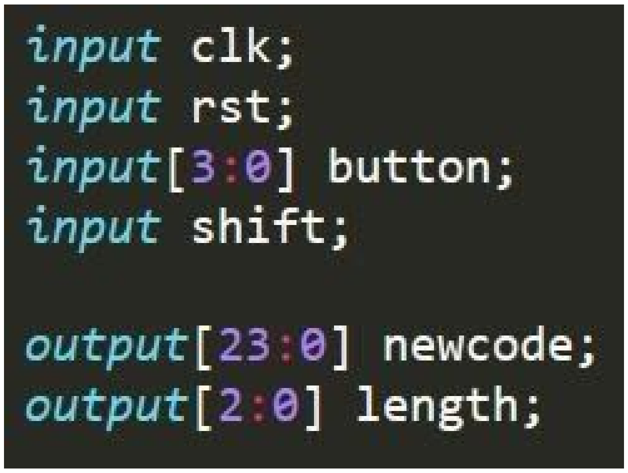


```
STATE MACHINE
  name                enter_new_code
  clock               CLK              posedge
  reset_signal        RST              positive
  reset_state         START            anyvalue
  default_state_is_x  1
INPUTS
  CLK
  RST
  GO
  BUTTON[3:0]
  BSTATE
OUTPUTS
  DONE                0                statebit
  SUCCESS             0                statebit
STATES
  DONE                0                output
  SUCCESS             0                output
TRANSITIONS
  equation            1                def_type
```

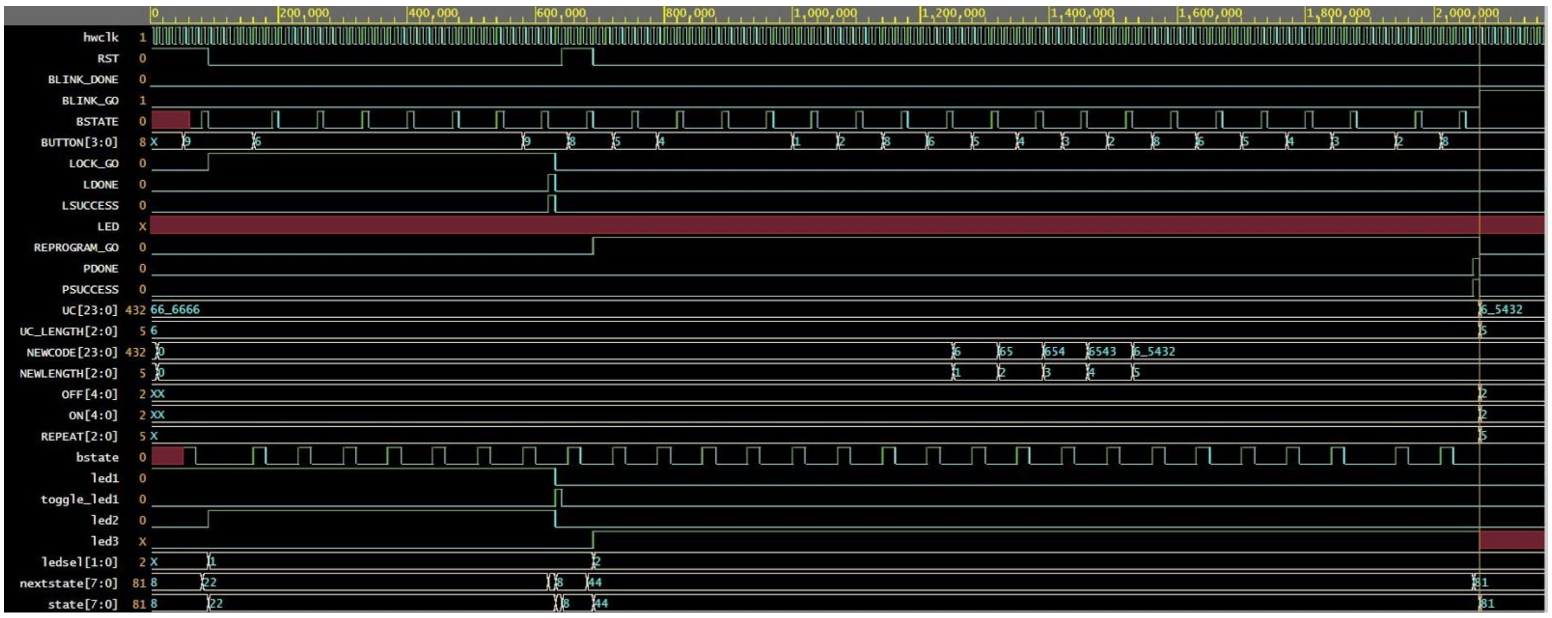
START
BUFFER
ENTER
VALID
DONE <= 1
SUCCESS <= 1
INVALID
DONE <= 1
WAIT
GO
1
BPRESS & BUTTON[3:0] == ENTER_BUTTON[3:0] & lvalid
BPRESS & BUTTON[3:0] == ENTER_BUTTON[3:0] & !lvalid
BPRESS & BUTTON[3:0] != ENTER_BUTTON[3:0] & !kvalid
BPRESS & BUTTON[3:0] == ENTER_BUTTON[3:0]



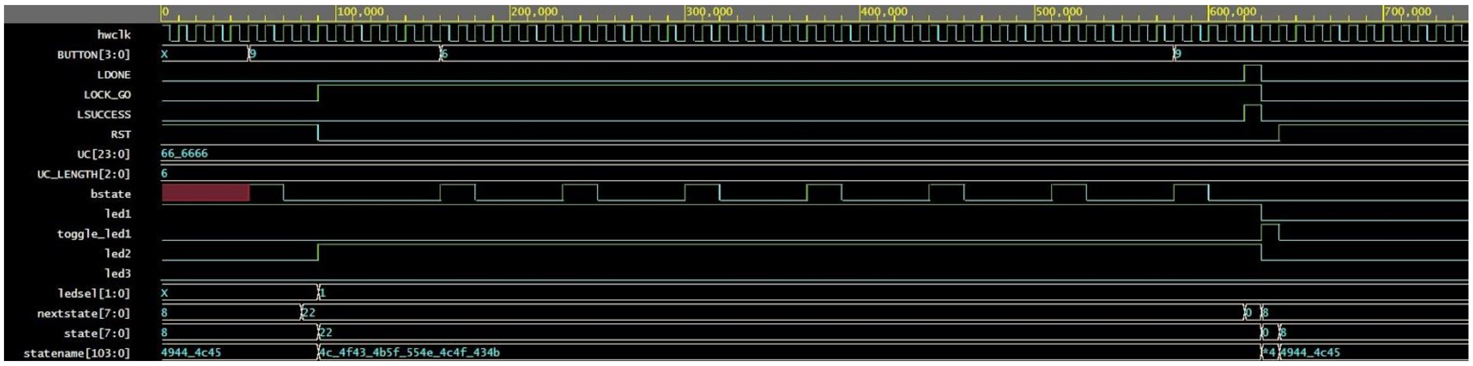
```
input CLK;
input RST;
input GO;
input[3:0] BUTTON;
input BPRESS;
input[3:0] ENTER_BUTTON;

output SUCCESS;
output DONE;
output[23:0] NEWCODE;
output[2:0] NEWLENGTH;
```



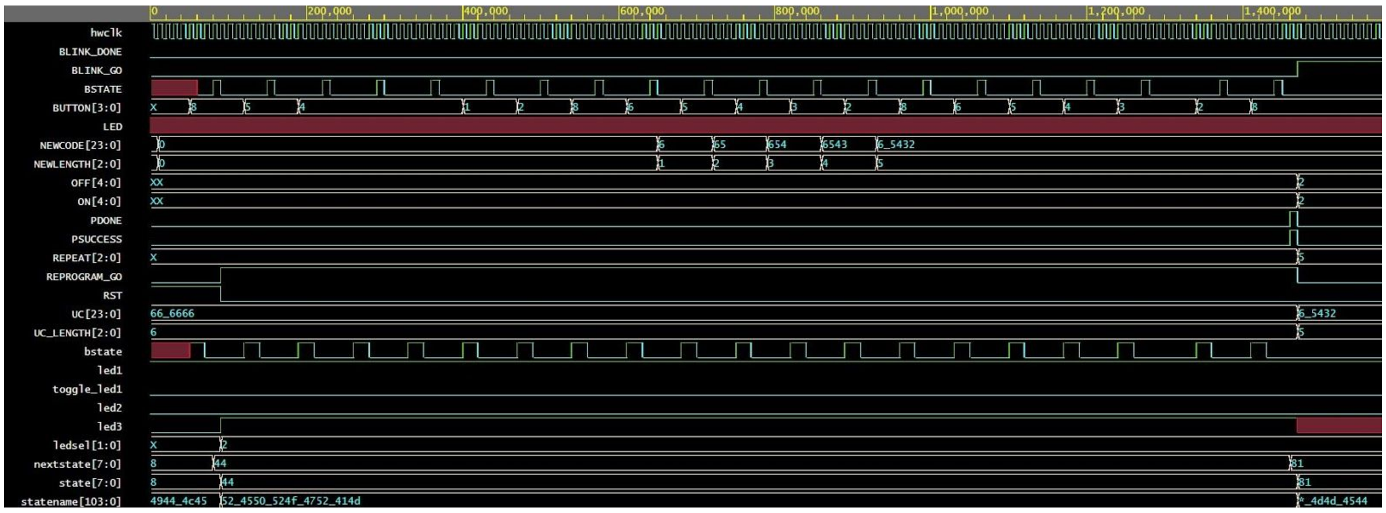
```
input clk;
input rst;
input[3:0] button;
input shift;

output[23:0] newcode;
output[2:0] length;
```



Pictured above is an example timing diagram for the top module, with some of its inputs, outputs, and control signals. As the blink module has not been used yet, its output is red. First, the lock is unlocked, which toggles led1. After that, the code is reprogrammed, which can be seen by the change in UC[23:0] at the yellow line to the right of the diagram.



This is a close-up of the enter_code unlock portion of the larger time diagram. Notice the simultaneous raising of LDONE and LSUCCESS to indicate the correct UC was entered.



This is a close-up of the reprogramming portion of the larger time diagram. Notice the simultaneous raising of PDONE and PSUCCESS, as well as the changing of UC and UC_LENGTH and ON and OFF, to indicate success.

The PC can be changed in the top module. It is a parameter, as is its length, PC_LENGTH. PC is a concatenation of six four bit integers, which can be changed. PC_LENGTH is a three bit integer and must be specified for the lock to work correctly. For PCs less than six integers, right-justify the integers, so the most significant bits are don't-cares, or random values.

As I did not work with a partner, I did all the work on this project.