21.11.2024

Finite State Machine

Contents

Task Description

A company building a finite state machine with a digital lock. It uses switches for getting password. Each password switch should have led to indicate passwords bit (if led open that switch is 1 if not switch is 0). Passwords are ascii codes of letters, and each letter is one stage password. For the ascii code each letter has a number up to 127. So that each stage has 7 bits to indicate binary number for that ascii. For change stage, system work at each stage correct switches should be open and button should be short pushed. There are 3 stages for finite state machines. If a wrong password is submitted system count that each of that until 5 then system is lockout and not give response correct password or wrong passwords. Moreover, this system indicates stage, or lockout via seven segment display. For each stage it shows stage number and for lockout seven segment display “L”. If the system is

locked out the only solution is long pushed to button, then system goes to stage 0. System is reset wrong number counter at each stage. Debouncer decides short or long pushed to button. Debouncer get clock rising edge and wait until it is enough to say state is change it reflect state is changed.

Acceptance Test

The company representative will the system out at least 1 correct and 1 wrong answer at each stage and go on with cycle 3 stages after 3 stages system should be at stage 0. Then system trying 5 or more wrong passwords should be submitted to see lockout stage. Then trying to correct the password to see if the stage changing or not . The system should not change its stage and long push to button should reset lockout system.

Requirements

The requirements of the system are listed below, including interface (IIRQ), functional(FRQ#) and timing(TRQ#) requirements.

Interface Requirements:

IRQ1: Input: binary switches that maintain state once toggled by user.
IRQ2: Output: Led should be intensity of 1 led for one switch to indicate switch open or not.
IRQ3: Output: Seven-segment display shows each stage.

Functional Requirements:

FRQ1: Reading switches which are High or Low then LEDs should indicate status.
FRQ2: There are four states “State=0, State=1, State=2, State=lockout”. The system should track current state and can be change state when requirements satisfied.
FRQ3: At each state there will be unique password.
FRQ4: If the correct password switches open then the user shortly-push central button to change state.
FRQ5: After a full cycle like (state= 0->1->2) state should be again 0 to go on to initial state.
FRQ6: System should keep wrong count at each state. If the wrong password submitted, then system count them. If the system goes another state until counter goes up to 5, then counter should reset itself.
FRQ7: If a user submit more than 5 wrong password then the system change states to lock.
FRQ8: At lock state system should not be change state although correct password submitted.
FRQ9: System can be changed if long-push happened to central button and state should be 0 as initial.

FRQ10: Seven segment display stages. For 0 to 2 it shows state number and for lock state it shows L in the display.
FRQ11: Debouncer should describe how much time is enough for short and long push.

Timing Requirements:

TRQ1: Short push time should be bigger than 0.655 ms debouncing period.
TRQ2: Long push time should be bigger than 1.34 sec debouncing period.
TRQ3: Central button should remain high for at least 0.655ms. If sorter than this System will not accept this push and there will be glitch.

Implementation

We describe this circuit in VHDL and use Vivado to automatically synthesize and implement it for use on the Basys3 board.

Input

Switches
Using switches on Basys3 to indicate which switch opens, that helps to user can easily enter password. Opening 7 switches (0 to 6) to indicate ascii binary code of password (IRQ1). SW0 is the least significant bit and SW6 is the most significant bit, so that basys3 can understand password.(FRQ1)

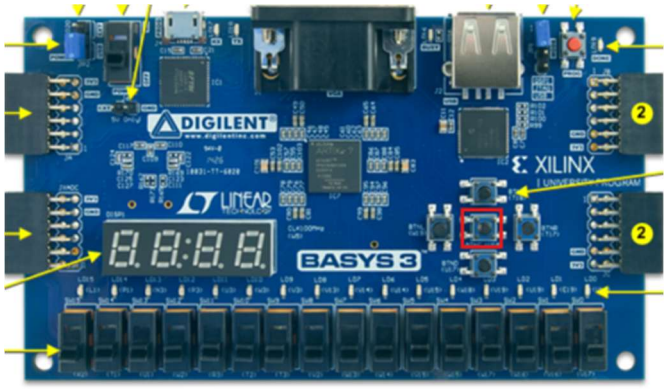
```
## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {sw[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports {sw[6]}]
```

Clock
Basys3 has default 100Mhz clock for the execution that means the period is the 10 nanoseconds.

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

Central Button

On basys3 there are more than 1 button however, we just use central button. This button is used for getting submission of password. If the click time is less than 0.655ms then it is not getting any action, if it is between 0.655ms and 1.34 sec it is short pushed that is for submission, and if it is clicked longer than 1.34 sec it is long pushed that is for reset the system lockout.




```
##Buttons
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {btnC[0]}]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]
```

Output

LEDS

Activating switch LEDs to indicate which switches toggle on and off. For doing that we connect each switch to a led (For instance: SW0=>Led0, SW6=>LED6). So that user can see which bit s/he is implementing. (IRQ2)

```
## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports {led[5]}]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {led[6]}]
```

Common Anode

Each anode has seven segment displays. We activate all anode pins to use just one seven segment display (an <= "1110").

```
#set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports dp]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {an[0]}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {an[1]}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {an[2]}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {an[3]}]
```

Seven Segment Display

For using seven segment display activating seg and giving specific value for each state, to seven segment helps to show system current state. (IRQ3)

```
##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]
```

State	Seven Segment Value
0	1000000
1	1111001
2	0100100
Lock	1000111
Other	1111111

Debouncer

Debouncer is a method of understanding how many times a button clicks and get accurate result of clicked button. Because of reverse inductance when we click button it is 0 then it rapidly changes it is value for a while. To get clear results on digital system we wait until a few clocks to ignore those rapid fluctuations.

In this project we need two debouncer. First debouncer decide button is steady HIGH for a time between 0.655ms and 1.34sec. The debouncer method takes buttonC and return btn_d and in sequential logic we keep btn_d_d which helps to compare if the button state is changed or not. For long-push debouncer it gets btnC as input and return rstbtn_d as return. Again, in sequential logic we keep rstbtn_d_d for comparison.

We keep comparison in a Boolean btn_d_re for short click and rstbtn_d_re for reset button and it helps to operate our combinational and sequential algorithms.
Note: FRQ4, FRQ11, TRQ1, TRQ2, TRQ3 are satisfied here.

Combinational Logic

Combinational logic doesn't include clk so that we can think that analog and continuous. In combinational logic algorithms we are deciding state changes. To do that we have state_t0 which is current state, according to state_t0 and conditional if else system checks requirements satisfied or not. The Requirements are, if system lockout='1' which indicates password is wrongly submitted over 5 times then state_t1 next state will be lock. If the button clicked shortly and password is correct it goes next state. If the button is clicked and the password is wrong then system should be in the same state. Other than this conditions also result with next state will be the same as current one. These requirements when state_t0 at 0,1,2 however when current state is lock, there is just one requirement rstbtn_d_re should be "1" then next state will be zero. Moreover, each stage has a unique password for stage:0 =>password:0000000, stage:1 =>password:0000001, stage:2 =>password:0000010.

Note: FRQ2, FRQ3, FRQ5, FRQ8, FRQ9 are satisfied here.

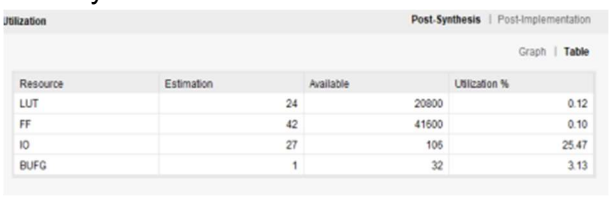
Sequential Logic

Sequential logic algorithms work with clock, so it run every 10 nanosecond. At each clock system get button data for comparison. Then it changes state_t0 to state_t1 this can be result as changing state or can be same state. A variable (sevsegval) defined for quickly implement if seven segment display should change. For State=0 -> sevsegval=0, State=1 -> sevsegval=1, State=2 -> sevsegval=2, State=lock -> sevsegval=4. Then at architectural behavioral process has condition for seven segments display correct state.

To count wrong submission there is a counter name "wrong_pwd_count". That count wrong password submission by a conditional code. When state is 0,1 or 2 then If btn_d_re is "1" and password of current stage is correct then wrong_pd_count will be 0, if password is wrong then wrong_pwd_count is increasing one. When state is lock then rst_btn_d_re should be "1" for wrong_pwd_count =0.
After that there is conditional statement for check wrong_pwd_count is 6 then it changes lockout value to "1" else lockout is "0".
Note: FRQ6, FRQ10 are satisfied here.

Utilization

Post Synthesis utilization table:

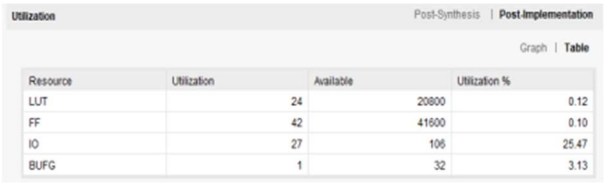


Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	24	20800	0.12
FF	42	41600	0.10
IO	27	106	25.47
BUFG	1	32	3.13

Post Implementation utilization table:



Utilization | Post-Synthesis | Post-Implementation

Graph | Table

Resource	Utilization	Available	Utilization %
LUT	24	20800	0.12
FF	42	41600	0.10
IO	27	106	25.47
BUFG	1	32	3.13

Verification

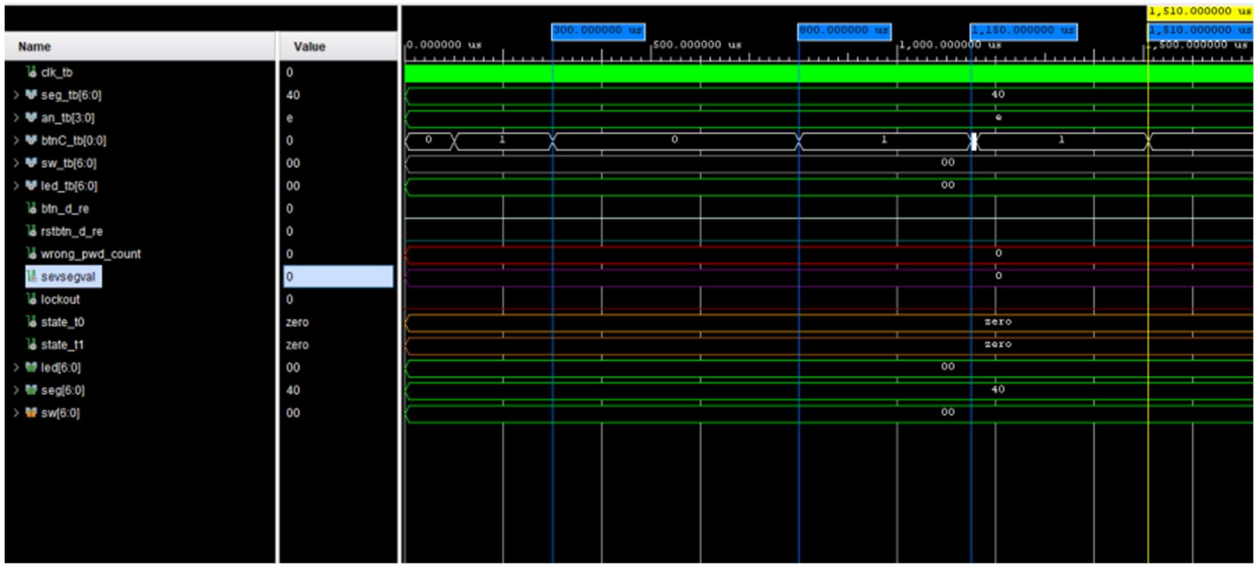
At verification system should work to satisfy functional and timing requirements. For doing that a prepared testbench is ready, but using real debouncer period time will be more costly we use 1.31 ms for long button press.

In box text are codes and red writings are commands.

```
--t=0
        sw_tb<="0000000"; --correct code
        btnC_tb(0) <= '0';
        wait for 100 us;
        --t=100
-- short pulse, shouldn't work
        btnC_tb(0) <= '1';
        wait for 200 us;
        btnC_tb(0) <= '0';
        wait for 500 us;
        --t=800 desired output not change in stage but buttonc
Should be 1  wrong count=0
```

```
   -- short pulse, glitch, then another short pulse
        -- together they should have worked, but due to the
glitch in between they shouldn't
        btnC_tb(0) <= '1';
        wait for 350 us;
        btnC_tb(0) <= '0';
        wait for 10 us;
        btnC_tb(0) <= '1';
        wait for 350 us;
        btnC_tb(0) <= '0';
        wait for 700 us;
        --t=2210us desired output is button is 1 for 800-1150
then 810-1160us
```


```
  -- short pulse, glitch, then another short pulse
        -- together they should have worked, but due to the
glitch in between they shouldn't
        btnC_tb(0) <= '1';
        wait for 350 us;
        btnC_tb(0) <= '0';
        wait for 10 us;
        btnC_tb(0) <= '1';
        wait for 350 us;
        btnC_tb(0) <= '0';
        wait for 700 us;
        --t=2210us desired output is button is 1 for 800-1150
then 810-1160us
```

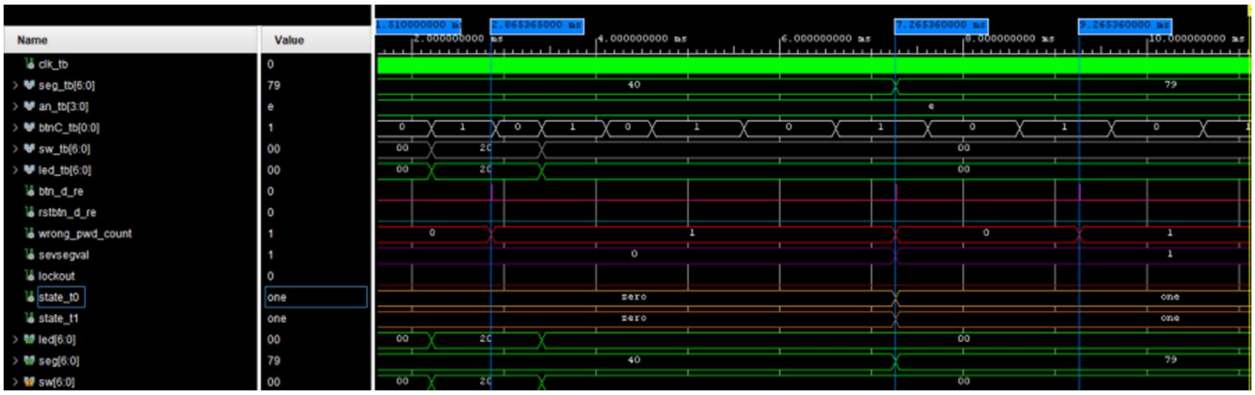
According to testbench we are first enter correct password for state 1. After waiting 100 ns we press the button so that btnC become 1 for 200 ns then it again be 0. It is a short pulse which is

debouncer will not except because we are expecting to press time should be bigger than 655 ns. So, this pulse is not working and not accepted as submission. At t=800 ns there are no change in values except btnC it shortly become 1 and then return 0. After 800ns system get btnC click for 350 ns then it become 0 for 10 ns then it returns 1 for 350ns. Although complete time is okey for btn_d_re can be 1 unless there is no glitch. There is exactly 1 clock cycle between btnC is 0 to 1 however system understand as it is a basic fluctuation. So that it is not submitted password and except btnC nothing changes. Now we are on t=2210ns

```
sw_tb<="0100000";
    -- Wrong password so system count up wrong counter
    btnC_tb(0) <= '1';
    wait for 700 us;--t=2910 btnc should be 1
    btnC_tb(0) <= '0';
    wait for 500 us;--t=3410 btnc should be one and stage
should be 1
     sw_tb<="0000000";
    -- long pulse, should work, state should go from "zero"
to "one"
    btnC_tb(0) <= '1';
    wait for 700 us;--t=4110 btnc should be 1
    btnC_tb(0) <= '0';
    wait for 500 us;--t=4610 btnc should be one and stage
should be 1
```

```
-- now state transition test
    btnC_tb(0) <= '1'; -- this will not work !! because the
debouncing period works both ways, we should have waited
approx. 150 ms more for this.
    wait for 1 ms;
    --t=4610 us
    btnC_tb(0) <= '0';
    wait for 1 ms;--t=6810

        btnC_tb(0) <= '1'; -- now this will work
    wait for 1 ms;
    btnC_tb(0) <= '0';
    wait for 1 ms; --Desired output wrong count should be
1
    --t=8810
```



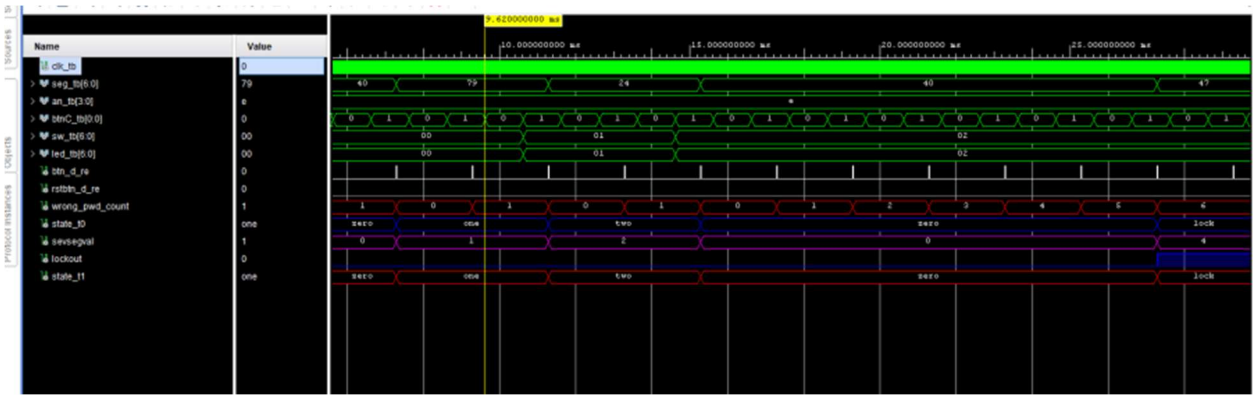
Wrong password submitted so wrong count increase between 2.865ms 7.26ms. The reason is wrong count reset at 7.26 is system change its state 0 to 1. At 4110ns correct password for state 1 is submitted, then system wait clock cycle and button to change state_t1 will be 1 then state_t0 become 1. Again, with change of state_t0 seven segment display variable seg changed. Then at 9.26ms system get wrong password input. Again wrong counter become 1. Also change in the switches reflected by change at led so we can understand where switches are changed.

```
sw_tb<="0000001"; -- state from "one " to "two"
        btnC_tb(0) <= '1';
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
--Desired output stage should be 2 and wrong count should
be 0
        --t=10810
        btnC_tb(0) <= '1';
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms; --Desired output wrong count should be
1
        --t=12810
```

```
sw_tb<="0000010";
        btnC_tb(0) <= '1'; -- state from "two" to "zero"
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=14810
        -- For lockout mode:
        btnC_tb(0) <= '1';--wrong count 1
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=16810
```

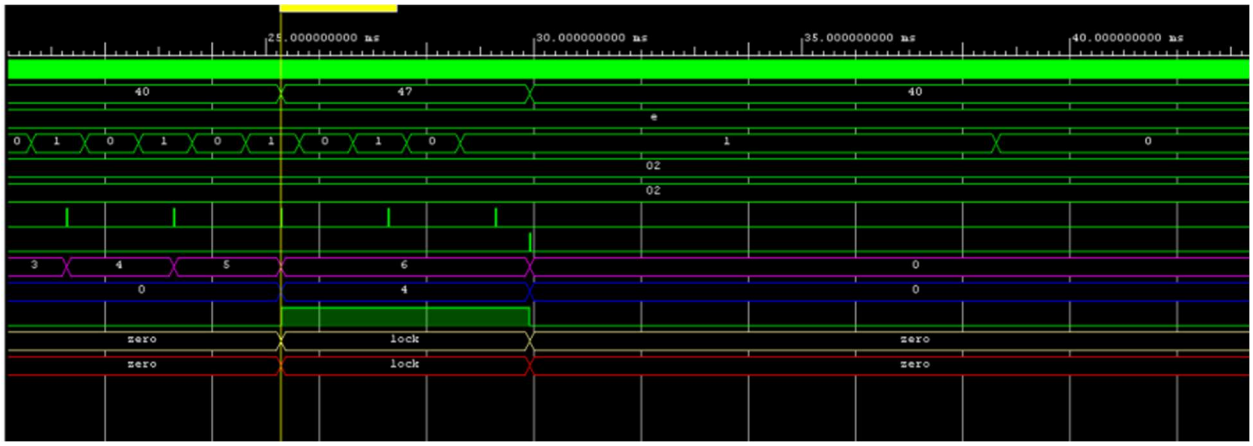
```
  btnC_tb(0) <= '1';--wrong count 2
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=18810
        btnC_tb(0) <= '1';--wrong count 3
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=20810
      btnC_tb(0) <= '1';--wrong count 4
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=22810
```

```
btnC_tb(0) <= '1';--wrong count 5
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=24610
btnC_tb(0) <= '1';--wrong count 6
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=26610
        --system lockout
btnC_tb(0) <= '1';--wrong count not change it is already
lockout
        wait for 1 ms;
        btnC_tb(0) <= '0';
        wait for 1 ms;
        --t=28610
```



At t=9000ns system stage is 1 and at 9.22ms wrong password is submitted so that system wrong count is increased to 1. At 11.20ms correct password of state 2 is submitted so state change after 11.20ms to state 2. At t=13.26ms again one more wrong answer submitted and wrong count increase. Then correct password of state 2 is submitted and system goes to state 0 and reset wrong count. That shows that state cycle and reset of wrong password counter works well. After state is being 0, 5 wrong password submitted. At t=27ms system goes into lock state and wrong count is 5. After one more wrong password system not giving any response and counter become 6. To reset lockout state there should be long press on the button so system should reset states.

```
btnC_tb(0) <= '1'; -- state from "lockout" to "zero"
        wait for 10 ms;
        btnC_tb(0) <= '0';
        wait;
```



We can see that long press to button reset lockout station. At test bench we can predict at what time what will be happen so when we get the simulation data and compare them they seems they validate each other.

References

1- https://digilent.com/reference/programmable-logic/basys-3/start