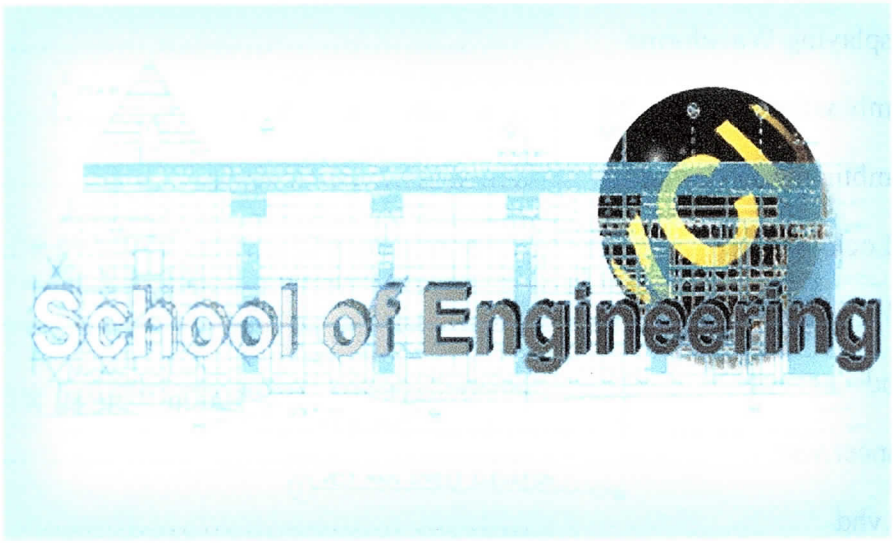
# EGRE 427 Lab/Homework No. 2

| | |
|---|---|
| Title | FPGA-based Hardware Design Using VHDL |
| Name | Chad Chapman |
| Date Performed | 23-Dec-10 |
| Date Submitted | 25-Dec-10 |




I pledge this work to be my own _____

# Table of Contents

**Project Description**

# ELECTRICAL ENGINEERING

School of Engineering


## EGRE 427 Advanced Digital Design

## Lab/Homework No. 2

## FPGA-based Hardware Design Using VHDL

This homework must be pledged as your own (individual) work – write it out and sign it.

1) Write a VHDL description that connects the momentary push buttons (BTN0-BTN3) to the LEDs (LD0-LD3) on the Digilent Nexys board. Use one instance of the switch debouncer and one instance of the synchronizer (source code available on the course Blackboard page) on the input signal from each of the buttons. The debouncer should be clocked from the normal 50 MHz FPGA clock. However, the synchronizer should be clocked from a slower clock that you will use to clock your state machine in problem 2. In this case, that clock should have a frequency of 10 Hz. Use a clock divider component (source code also available on the course web page) to divide the 50 MHz FPGA clock down to 10 Hz for the synchronizers. Demonstrate the proper functioning of this part of the assignment to the instructor or the TA before moving onto the next part of the assignment.

2) Write a VHDL description of an electronic combination lock. Upon powerup or reset, the combination lock will be in the locked state. The state of the lock will be indicated by the LEDs on the Nexys board. If the lock is locked, then the LD0 LED will be on and the LD1 LED will be off. If the lock is unlocked, then the LD0 LED will be off and the LD1 LED will be on. The combination shall be a sequence of 4 digits, each of which can take on the value of 0-4. The combination will be input by pressing the buttons on the Nexys board. The combination for this lock is the sequence 3-1-0-2. If the buttons are pressed in this order, the lock will open for 10 seconds and then relock. If the buttons are pressed in the wrong order, then the combination lock will wait until 4 of the buttons have been pressed (i.e., the user has tried a complete combination) and then it will blink the LD0 and LD1 LEDs in a opposing on-off sequence, once per second for 5 seconds. During this 5 second period, any pressing of the buttons by the user will be ignored. You MUST use the 3-process state machine description style shown in class for this assignment in order to receive any credit. Demonstrate the proper functioning of this part of the assignment to the instructor or the TA before turning in the complete assignment.

For this assignment, you must turn in a cover sheet with your name, date, and a brief writeup which includes a state transition diagram of your combination lock state machine and description of how it works. You must also turn in copies of your VHDL files and your UCF file that you used to map the design into the FPGA on the Nexys board.

# Combination Lock FSM

Upon inspection of the state transition diagram and the corresponding code we clearly show that this machine is a Moore machine. The FSM gets it's input from a signal named 'btn_pressed' which is of type STD_LOGIC_VECTOR(3 DOWNTO 0). The input is directly received from the signal 'sync_out' which resides inside of the architecture for Electronic_Combination_Lock. The input at each state is explicitly specified for each case and maps to an output within the process Output_Function.

When we are inside of the finite state machine we go from one state to another until we reach the states labeled UNLOCKED and ERROR. If we are in either of these two states we prevent the FSM from going to the next desired state by using a signal called 'enable'. The conditions are described by the following figures:

```
-- We reach this state if and only if the correct combination of
-- buttons is pressed.
WHEN UNLOCKED =>
   IF enable = '0' THEN
      NEXT_STATE <= UNLOCKED;
   ELSE
      NEXT_STATE <= LOCKED;
   END IF;
```

Figure 1 – Logic for UNLOCKED.

```
-- We visit the 'ERROR' state if any of the buttons pressed is
-- incorrect.
WHEN ERROR =>
   IF enable = '0' THEN
      NEXT_STATE <= ERROR;
   ELSE
      NEXT_STATE <= LOCKED;
   END IF;
```

Figure 2 – Logic for ERROR.

As shown in Figure 1, when enable is set to '0' we want to stay in the state called UNLOCKED and the same idea apply as well for Figure 2. While we are at these two states the finite state machine that produces the required output to the LED's is performing some kind of operation. Although not explicitly stated in the VHDL file, when enable is set to '1' we want the FSM to go to the next desired state. We have labeled the transitions from UNLOCKED to LOCKED and from ERROR to LOCKED to show that this is the behavior we expect. More details concerning the signal called enable will be discussed in the section for LED_CONROL_FSM.

The output for this machine will be string "01" when the FSM is still accepting inputs from the buttons on the FPGA. When the output string is "10" this information is provided to the FSM that controls the output for the LED's when the correct sequence of buttons is pressed by the user. If the output string is "11" we have reached the ERROR state and the FSM that controls the LED's will provide the required output.

Before we can present the state transition table first we must provide the definition of a Moore Machine. A Moore machine can be defined as a 6-tuple $M(Q, \Sigma, \Lambda, \delta, G, q_0)$, where

1. $Q$ is a finite set of states,
2. $\Sigma$ is a finite set called the input alphabet,
3. $\Lambda$ is a finite set called the output alphabet,
4. $\delta: Q \times \Sigma \longrightarrow Q$ is the transition function,
5. $G: Q \longrightarrow \Lambda$ is the output function, and
6. A unique start state $q_0 \in Q$.

The description of the CombinationLockFSM is as follows:

1. $Q = \{LOCKED, Right1\_P, Right2\_P, Right3\_P, Right4\_P,\}$
   $Right1\_R, Right2\_R, Right3\_R,$
   $Wrong1\_P, Wrong2\_P, Wrong3\_P, Wrong4\_P,$
   $Wrong1\_R, Wrong2\_R, Wrong3\_R,$
   $UNLOCKED, ERROR\}$,

2. $\Sigma = \{0000,0001,0010,0100,1000, enable = 0, enable = 1\}$,

3. $\Lambda = \{led\_select = 00, led\_select = 01, led\_select = 10, led\_select = 11\}$,

4. The transition function represented as a transition table,

| State | Input | Next State |
|---|---|---|
| LOCKED | btn_pressed = 0000 | LOCKED |
| LOCKED | btn_pressed = 0001 | Wrong1_P |
| LOCKED | btn_pressed = 0010 | Wrong1_P |
| LOCKED | btn_pressed = 0100 | Wrong1_P |
| LOCKED | btn_pressed = 1000 | Right1_P |
| Right1_P | btn_pressed = 1000 | Right1_P |
| Right1_P | btn_pressed = 0000 | Right1_R |
| Right1_R | btn_pressed = 0000 | Right1_R |
| Right1_R | btn_pressed = 0010 | Right2_P |
| Right1_R | btn_pressed = 0001 | Wrong2_P |
| Right1_R | btn_pressed = 0100 | Wrong2_P |
| Right1_R | btn_pressed = 1000 | Wrong2_P |
| Right2_P | btn_pressed = 0010 | Right2_P |

| Right2_P | btn_pressed = 0000 | Right2_R |
|---|---|---|
| Right2_R | btn_pressed = 0000 | Right2_R |
| Right2_R | btn_pressed = 0001 | Right3_P |
| Right2_R | btn_pressed = 0010 | Wrong3_P |
| Right2_R | btn_pressed = 0100 | Wrong3_P |
| Right2_R | btn_pressed = 1000 | Wrong3_P |
| Right3_P | btn_pressed = 0001 | Right3_P |
| Right3_P | btn_pressed = 0000 | Right3_R |
| Right3_R | btn_pressed = 0000 | Right3_R |
| Right3_R | btn_pressed = 0100 | Right4_P |
| Right3_R | btn_pressed = 0001 | Wrong4_P |
| Right3_R | btn_pressed = 0010 | Wrong4_P |
| Right3_R | btn_pressed = 1000 | Wrong4_P |
| Right4_P | btn_pressed = 0100 | Right4_P |
| Right4_P | btn_pressed = 0000 | UNLOCKED |
| UNLOCKED | enable = 0 | UNLOCKED |
| UNLOCKED | enable =1 | LOCKED |
| Wrong1_P | btn_pressed = 0000 | Wrong1_R |
| Wrong1_P | btn_pressed = 0001 | Wrong1_P |
| Wrong1_P | btn_pressed = 0010 | Wrong1_P |
| Wrong1_P | btn_pressed = 0100 | Wrong1_P |
| Wrong1_P | btn_pressed = 1000 | Wrong1_P |
| Wrong1_R | btn_pressed = 0000 | Wrong1_R |
| Wrong1_R | btn_pressed = 0001 | Wrong2_P |
| Wrong1_R | btn_pressed = 0010 | Wrong2_P |
| Wrong1_R | btn_pressed = 0100 | Wrong2_P |
| Wrong1_R | btn_pressed = 1000 | Wrong2_P |
| Wrong2_P | btn_pressed = 0000 | Wrong2_R |
| Wrong2_P | btn_pressed = 0001 | Wrong2_P |
| Wrong2_P | btn_pressed = 0010 | Wrong2_P |
| Wrong2_P | btn_pressed = 0100 | Wrong2_P |
| Wrong2_P | btn_pressed = 1000 | Wrong2_P |
| Wrong2_R | btn_pressed = 0000 | Wrong2_R |
| Wrong2_R | btn_pressed = 0001 | Wrong3_P |
| Wrong2_R | btn_pressed =0010 | Wrong3_P |
| Wrong2_R | btn_pressed =0100 | Wrong3_P |
| Wrong2_R | btn_pressed =1000 | Wrong3_P |
| Wrong3_P | btn_pressed = 0000 | Wrong3_R |
| Wrong3_P | btn_pressed = 0001 | Wrong3_P |
| Wrong3_P | btn_pressed = 0010 | Wrong3_P |
| Wrong3_P | btn_pressed = 0100 | Wrong3_P |
| Wrong3_P | btn_pressed = 1000 | Wrong3_P |
| Wrong3_R | btn_pressed = 0000 | Wrong3_R |
| Wrong3_R | btn_pressed = 0001 | Wrong4_P |

| | | |
|---|---|---|
| Wrong3_R | btn_pressed = 0010 | Wrong4_P |
| Wrong3_R | btn_pressed = 0100 | Wrong4_P |
| Wrong3_R | btn_pressed = 1000 | Wrong4_P |
| Wrong4_P | btn_pressed = 0000 | ERROR |
| Wrong4_P | btn_pressed = 0001 | Wrong4_P |
| Wrong4_P | btn_pressed = 0010 | Wrong4_P |
| Wrong4_P | btn_pressed = 0100 | Wrong4_P |
| Wrong4_P | btn_pressed = 1000 | Wrong4_P |
| ERROR | enable = 0 | ERROR |
| ERROR | enable =1 | LOCKED |

5. The output function represented as a transition table,

| State | Output |
|---|---|
| LOCKED | led_select = 01 |
| Right1_P | led_select = 01 |
| Right2_P | led_select = 01 |
| Right3_P | led_select = 01 |
| Right4_P | led_select = 01 |
| Right1_R | led_select = 01 |
| Right2_R | led_select = 01 |
| Right3_R | led_select = 01 |
| Wrong1_P | led_select = 01 |
| Wrong2_P | led_select = 01 |
| Wrong3_P | led_select = 01 |
| Wrong4_P | led_select = 01 |
| Wrong1_R | led_select = 01 |
| Wrong2_R | led_select = 01 |
| Wrong3_R | led_select = 01 |
| UNLOCKED | led_select = 10 |
| ERROR | led_select = 11 |

6. $q_0 = LOCKED.$

# LED_Control_FSM

The LED_Control_FSM is a bit more complex than the CombinationLockFSM. Now we are using a counter and status type to help model the behavior of the FSM. Figures 3 and 4 shows this how it is written in the .VHD file.

```
TYPE status_type IS (LOCKED, UNLOCKED, INCORRECT);
SIGNAL present_status, next_status : status_type;
```

Figure 3 – TYPE status_type.

The TYPE called 'status_type keeps track of the state of the CombinationLockFSM and it's current_status is determined by the signal led_select. This allows us to choose the next path to take in the LED_Control_FSM without having to use any nested 'if then' statements and it helps us determine the value enable_out is going to assume with respect to status. By choosing the next value enable_out is going to take, status_type plays a role in the behavior of the CombinationLockFSM when it is at the states LOCKED, UNLOCKED, and ERROR.

```
SUBTYPE counter IS INTEGER RANGE 15 DOWNTO 0;
SIGNAL present_count, next_count : counter;
```

Figure 4 – SUBTYPE counter.

The SUBTYPE counter is used to keep track of the time since the status changed from LOCKED to UNLOCKED and from LOCKED to INCORRECT. This allows us to set or clear the enable_out bit when 'present_status' is not set to LOCKED since we no longer care what values led_select holds. The logic for 'status_type' UNLOCKED and INCORECT compares the current value held by counter to determine when 'present_status' needs to be assigned a new value. The logic used is shown in the figures below.

```
WHEN UNLOCKED =>
   IF present_count < 10 THEN
      next_status <= UNLOCKED;
      enable_out <= '0';
   ELSIF present_count = 10 THEN
      next_status <= LOCKED;
      enable_out <= '1';
   ELSE
      next_status <= UNLOCKED;
      enable_out <= '0';
   END IF;
```

Figure 5 – Logic for UNLOCKED

```
WHEN INCORRECT =>
   IF present_count < 5 THEN
      next_status <= INCORRECT;
      enable_out <= '0';
   ELSIF present_count = 5 THEN
      next_status <= LOCKED;
      enable_out <= '1';
   ELSE
      next_status <= INCORRECT;
      enable_out <= '0';
   END IF;
```

Figure 6 – Logic for INCORRECT

In Figure 5, we test to see when 'present_count' is less than 10. We repeat this comparison every second until $present\_count = 10$. Once this happens we change the 'enable_out' bit to' 1' so that the CombinationLockFSM may continue to accept input signals originating from the buttons. If $present\_count < 10$ the 'enable_out' bit remains at '0' and the LED_Control_FSM continues to generate output for the LED's. Similar combinational logic is used for when

$present\_status = INCORRECT$. Instead in this case we increment count until $present\_count = 5$. All of this information from the TYPE status_type is used in state $S_0$ to determine which path to take in the 'Transition_Function.'

The status of this FSM can be described as its own FSM and can be described according to the definition as $M(Q, \Sigma, \Lambda, \delta, G, q_0)$ where:

1. $Q = \{LOCKED, UNLOCKED, INCORRECT\}$
2. $\Sigma = \{led\_select = "01", led\_select = "10", led\_select = "11",$
   $present\_count = 1, present\_count = 2, present\_count = 3,$
   $present\_count = 4, present\_count = 5, present\_count = 6,$
   $present\_count = 7, present\_count = 8, present\_count = 9,$
   $present\_count = 10\}$,
3. $\Lambda = \{enable\_out = 0, enable\_out = 1\}$,
4. The transition function represented as a transition table,

| States | Input | Next State |
|---|---|---|
| LOCKED | $led\_select = "01"$ | LOCKED |
| LOCKED | $led\_select = "10"$ | UNLOCKED |
| LOCKED | $led\_select = "11"$ | INCORRECT |
| UNLOCKED | $present\_count < 10$ | UNLOCKED |
| UNLOCKED | $present\_count = 10$ | LOCKED |
| INCORRECT | $present\_count < 5$ | INCORRECT |
| INCORRECT | $present\_count = 5$ | LOCKED |

5. The output function represented as a transition table,

| State | Output |
|---|---|
| LOCKED | $enable\_out = 1$ |
| UNLOCKED | $enable\_out = 0$ |
| INCORRECT | $enable\_out = 0$ |

6. $q_0 = LOCKED$.

The LED Control FSM can be described as a Mealy machine. A Moore machine can be defined as a 6-tuple $M(Q, \Sigma, \Lambda, \delta, G, q_0)$, where

1. $Q$ is a finite set of states,
2. $\Sigma$ is a finite set called the input alphabet,
3. $\Lambda$ is a finite set called the output alphabet,

4. $\delta: Q \times \Sigma \longrightarrow Q$ is the transition function,
5. $G: Q \times \Sigma \longrightarrow \Lambda$ is the output function, and
6. A unique start state $q_0 \in Q$.

The description of this FSM is as follows:

1. $Q = \{S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, DONE\}$,
2. $\Sigma = \{LOCKED, UNLOCKED, INCORRECT\}$,
3. $\Lambda = \{led\_out = 00, led\_out = 01 = led\_out = 10, led\_out = 11\}$,
4. The transition function represented as a transition table,

| State | Input | Next State |
|---|---|---|
| S0 | LOCKED | S0 |
| S0 | UNLOCKED | S6 |
| S0 | INCORRECT | S1 |
| S1 | INCORRECT | S2 |
| S2 | INCORRECT | S3 |
| S3 | INCORRECT | S4 |
| S4 | INCORRECT | S5 |
| S5 | INCORRECT | S16 |
| S6 | CORRECT | S7 |
| S7 | CORRECT | S8 |
| S8 | CORRECT | S9 |
| S9 | CORRECT | S10 |
| S10 | CORRECT | S11 |
| S11 | CORRECT | S12 |
| S12 | CORRECT | S13 |
| S13 | CORRECT | S14 |
| S14 | CORRECT | S15 |
| S15 | CORRECT | S16 |
| S16 | CORRECT | DONE |
| DONE | CORRECT | S0 |
| DONE | INCORRECT | S0 |

5. The output function, $G: Q \times \Sigma \longrightarrow \Lambda$ represented as a transition table,

| State | Input | Output |
|---|---|---|
| S0 | LOCKED | $led\_out = 01$ |
| S0 | UNLOCKED | $led\_out = 00$ |
| S0 | INCORRECT | $led\_out = 00$ |

| | | |
|---|---|---|
| S1 | INCORRECT | $led\_out = 01$ |
| S2 | INCORRECT | $led\_out = 10$ |
| S3 | INCORRECT | $led\_out = 01$ |
| S4 | INCORRECT | $led\_out = 10$ |
| S5 | INCORRECT | $led\_out = 01$ |
| S6 | UNLOCKED | $led\_out = 10$ |
| S7 | UNLOCKED | $led\_out = 10$ |
| S8 | UNLOCKED | $led\_out = 10$ |
| S9 | UNLOCKED | $led\_out = 10$ |
| S10 | UNLOCKED | $led\_out = 10$ |
| S11 | UNLOCKED | $led\_out = 10$ |
| S12 | UNLOCKED | $led\_out = 10$ |
| S13 | UNLOCKED | $led\_out = 10$ |
| S14 | UNLOCKED | $led\_out = 10$ |
| S15 | UNLOCKED | $led\_out = 01$ |
| S16 | LOCKED | $led\_out = 00$ |
| DONE | LOCKED | $led\_out = 01$ |

6. $q_0 = S0$.

VHDL and UCF files

The VHDL files that are written or modified and the UCF file are included in this section. The UCF file used in our design is shown below in Figure 7.

```
#PINLOCK_BEGIN
#Pinout for NEXYS2 Board
# System clock connected to GCK0
NET "clk"     LOC ="B8";
# leds
NET "leds<3>" LOC = "K14";
NET "leds<2>" LOC = "K15";
NET "leds<1>" LOC = "J15";
NET "leds<0>" LOC = "J14";
# btns
NET "btns<3>" LOC = "H13";
NET "btns<2>" LOC = "E18";
NET "btns<1>" LOC = "D18";
NET "btns<0>" LOC = "B18";
```

Figure 7 – CombinationLockPins.ucf

# Simulation Waveforms

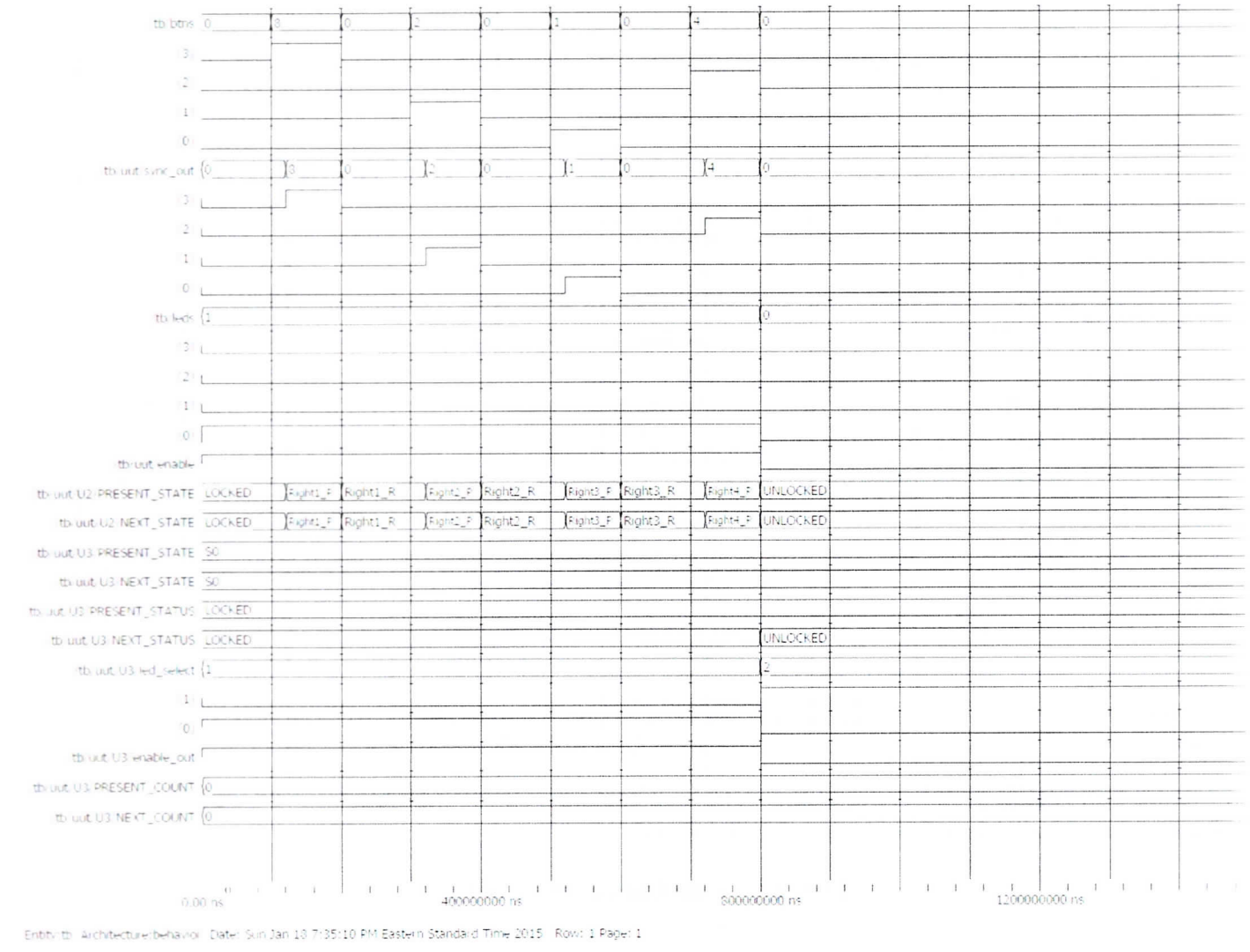



Figure 8 – Correct Input 1 to 1,400,000,000 ns

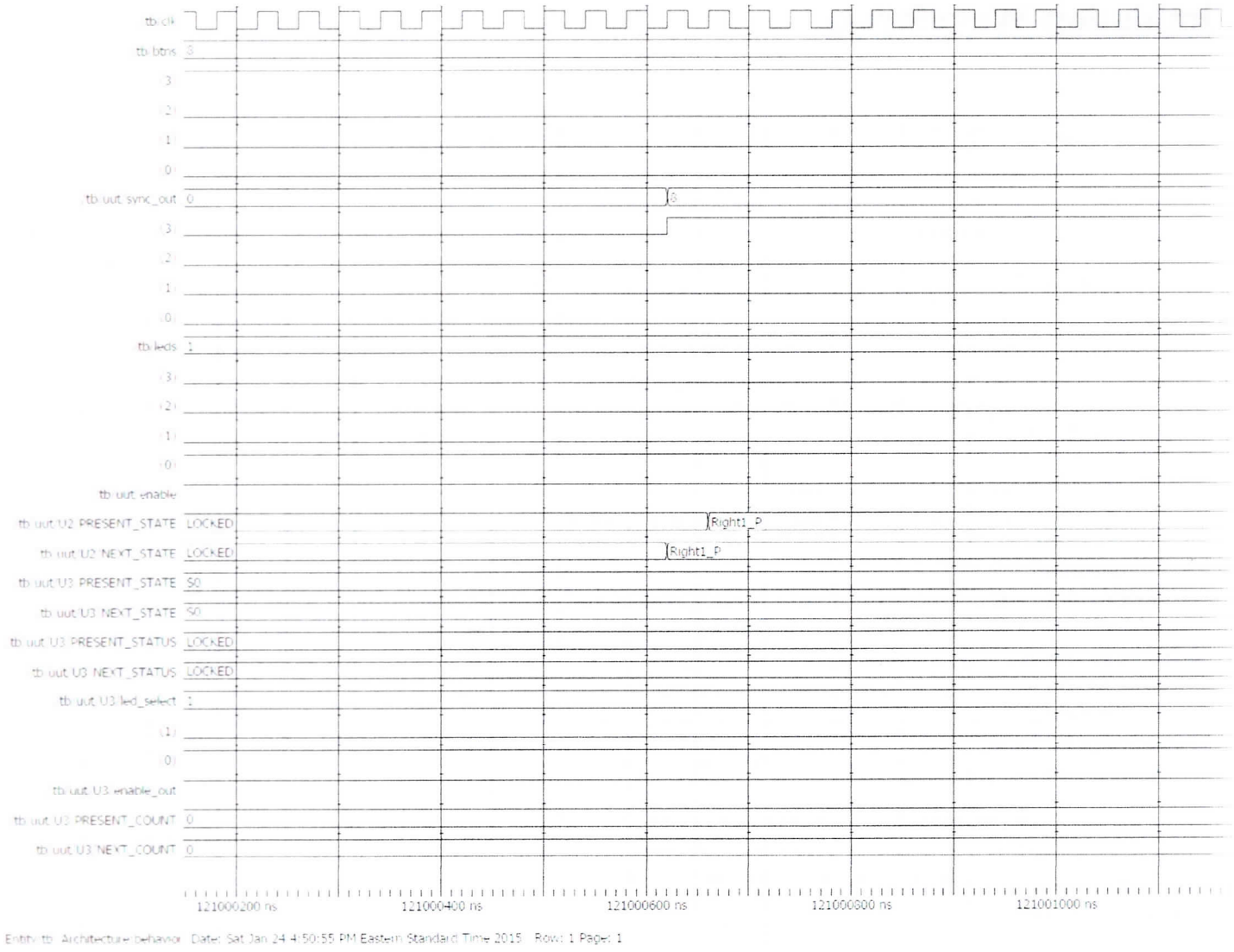



Figure 9 – First Correct Input Detected

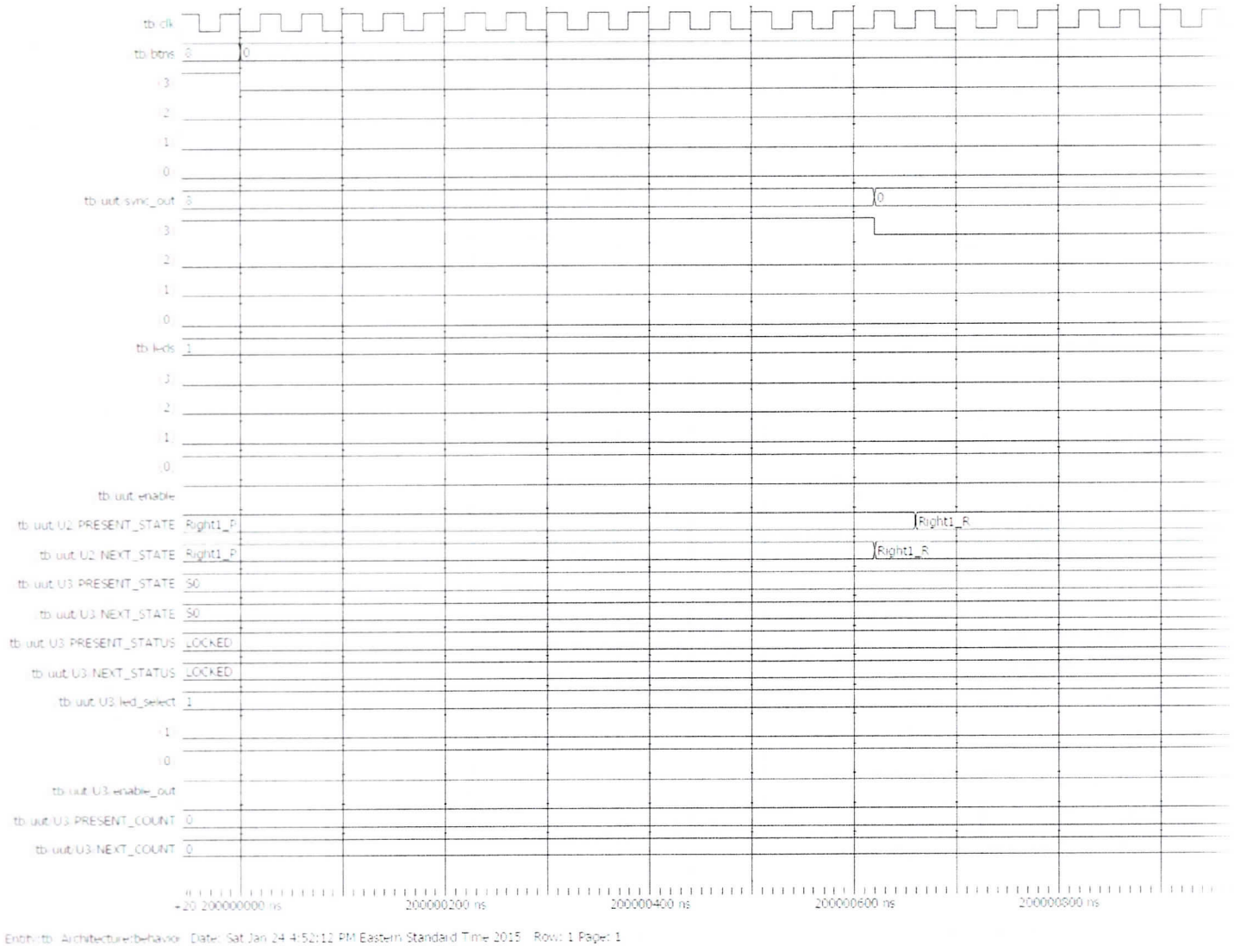



Figure 10 – First Correct Input Released

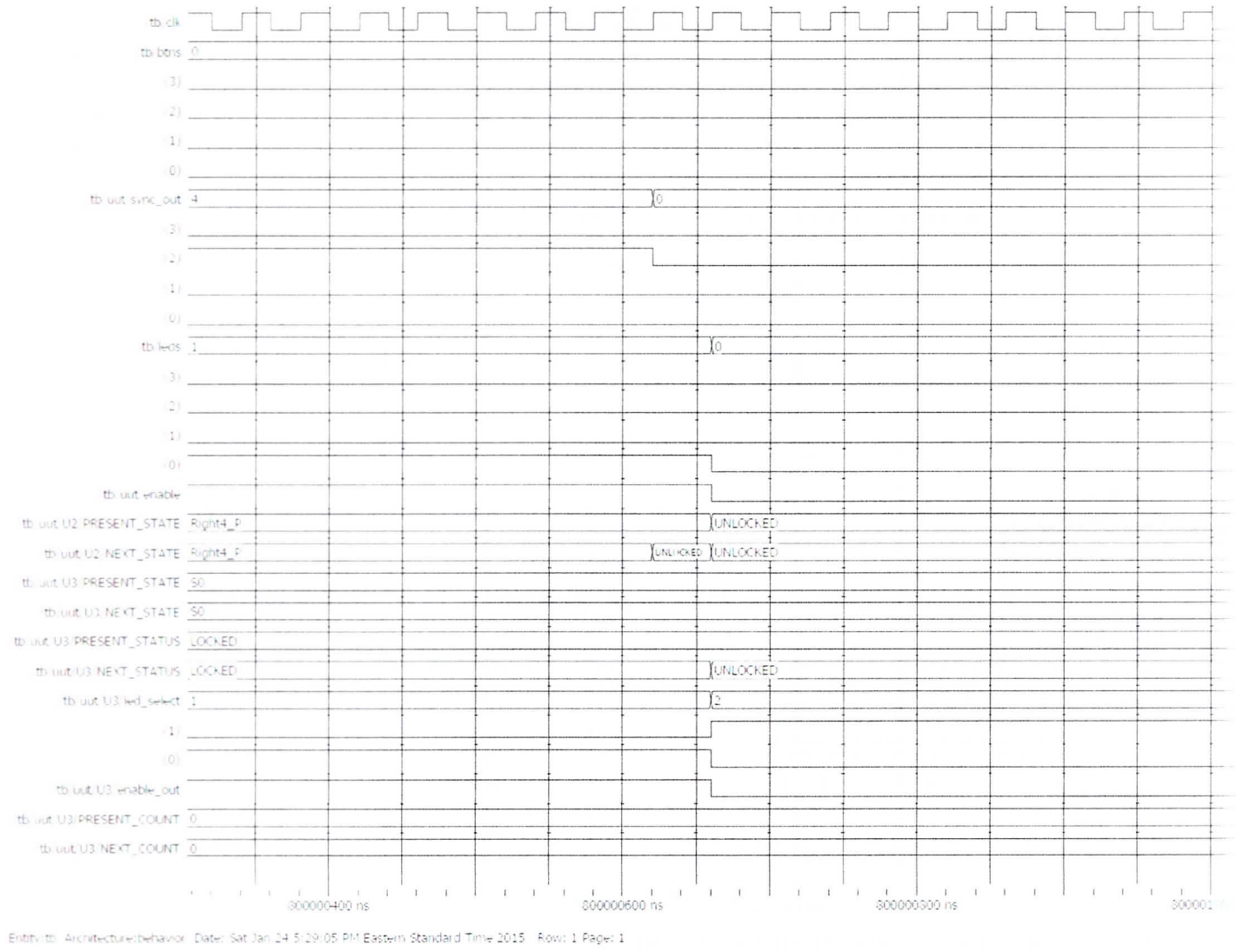



Figure 11 – Combination Lock is Unlocked

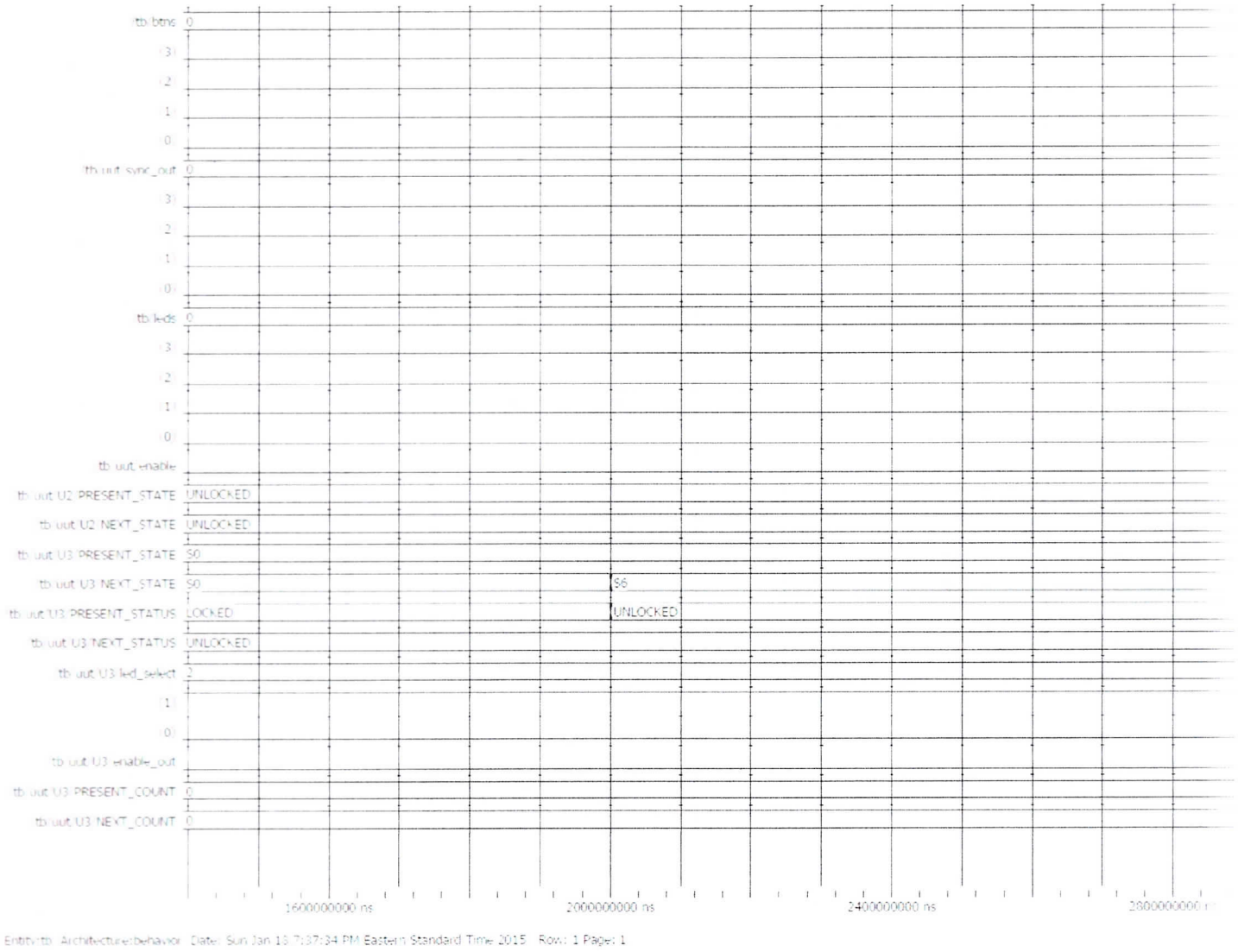



Figure 12 – Correct Input 1,400,000,000 to 2,800,000,000 ns

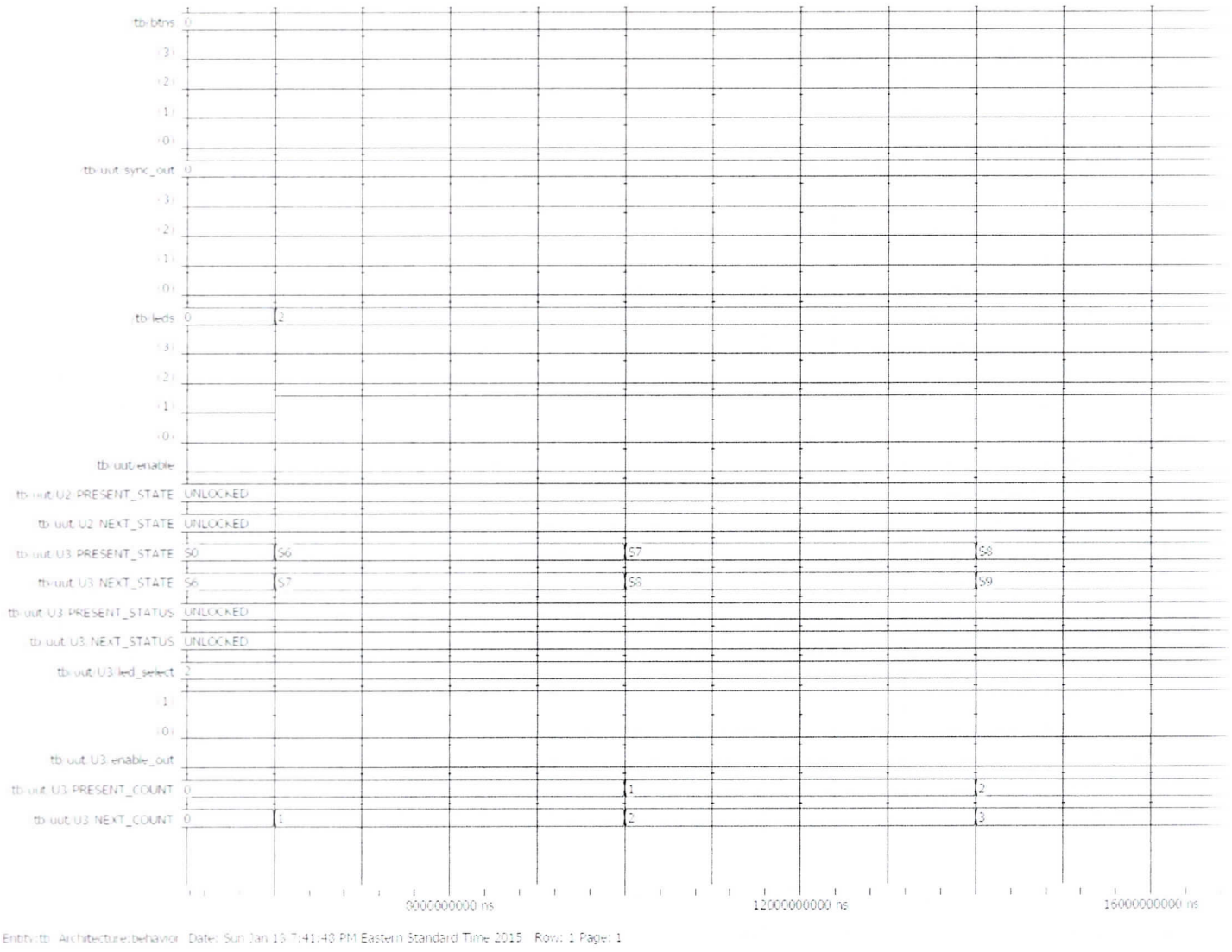



Figure 13 – Correct Input 5,000,000,000 to 16,000,000,000 ns

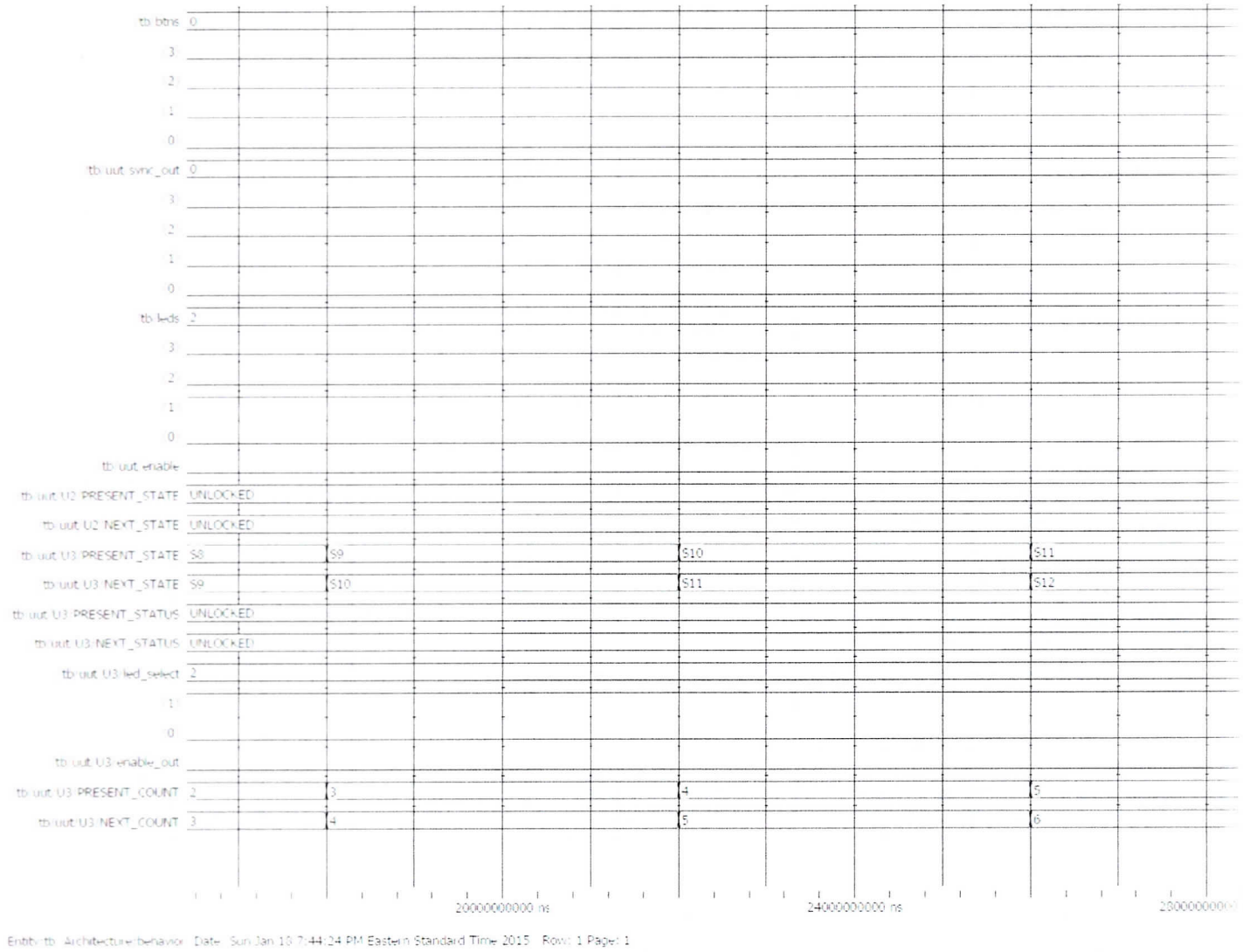



Figure 14 – Correct Input 16,000,000,000 to 28,000,000,000 ns

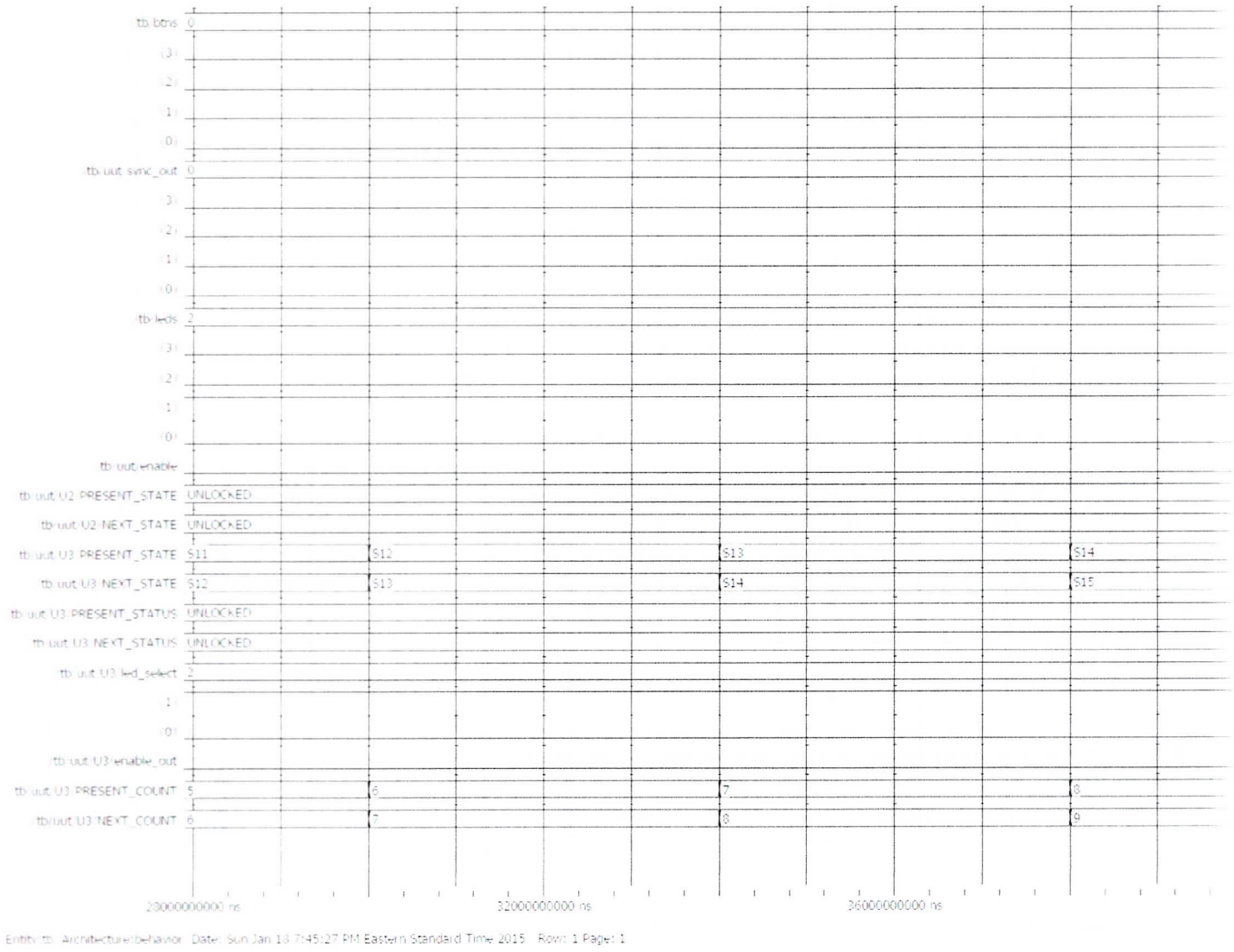



Figure 15 – Correct Input 28,000,000,000 to 39,000,000,000 ns

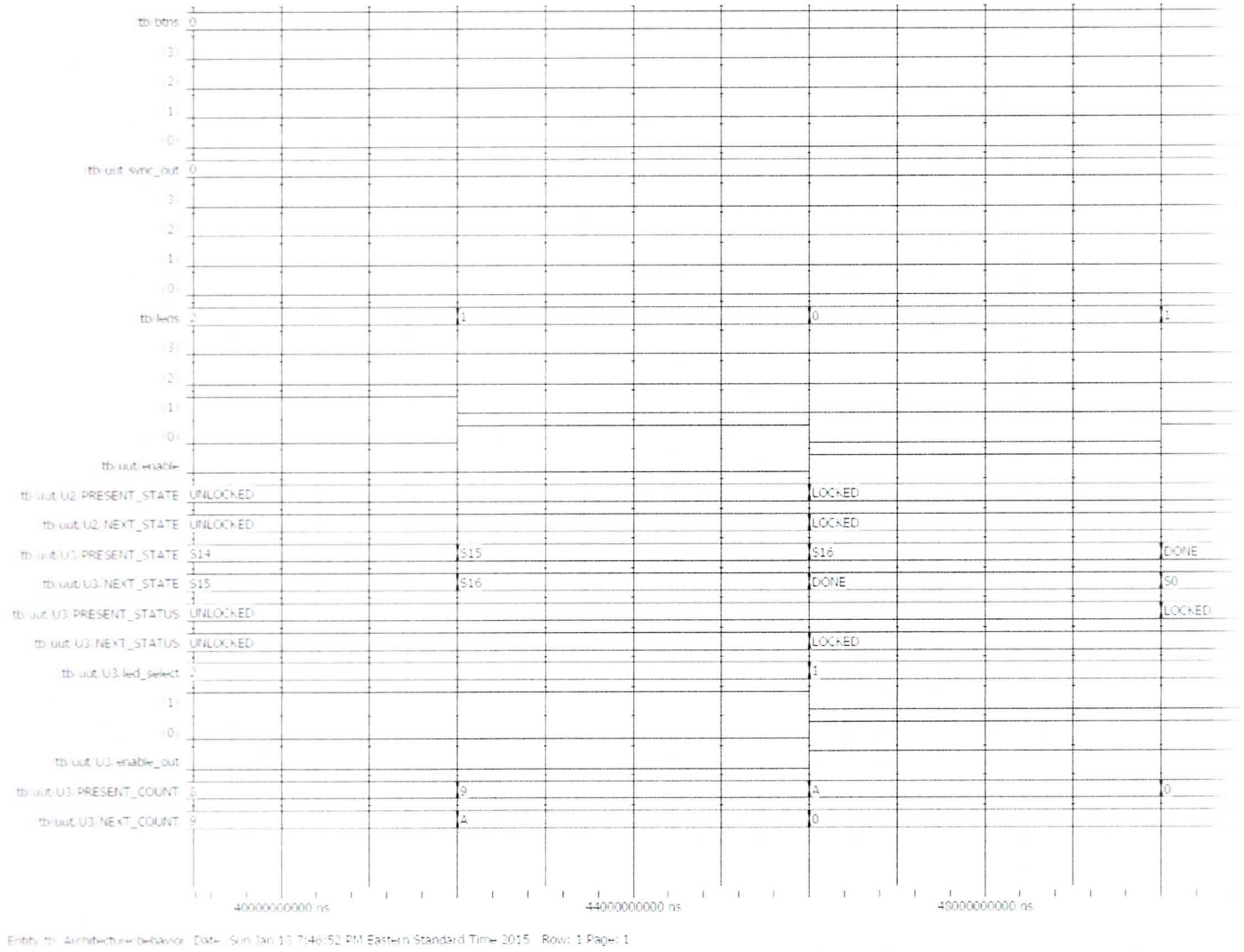



Figure 16 – Correct Input 39,000,000,000 to 50,000,000,000 ns

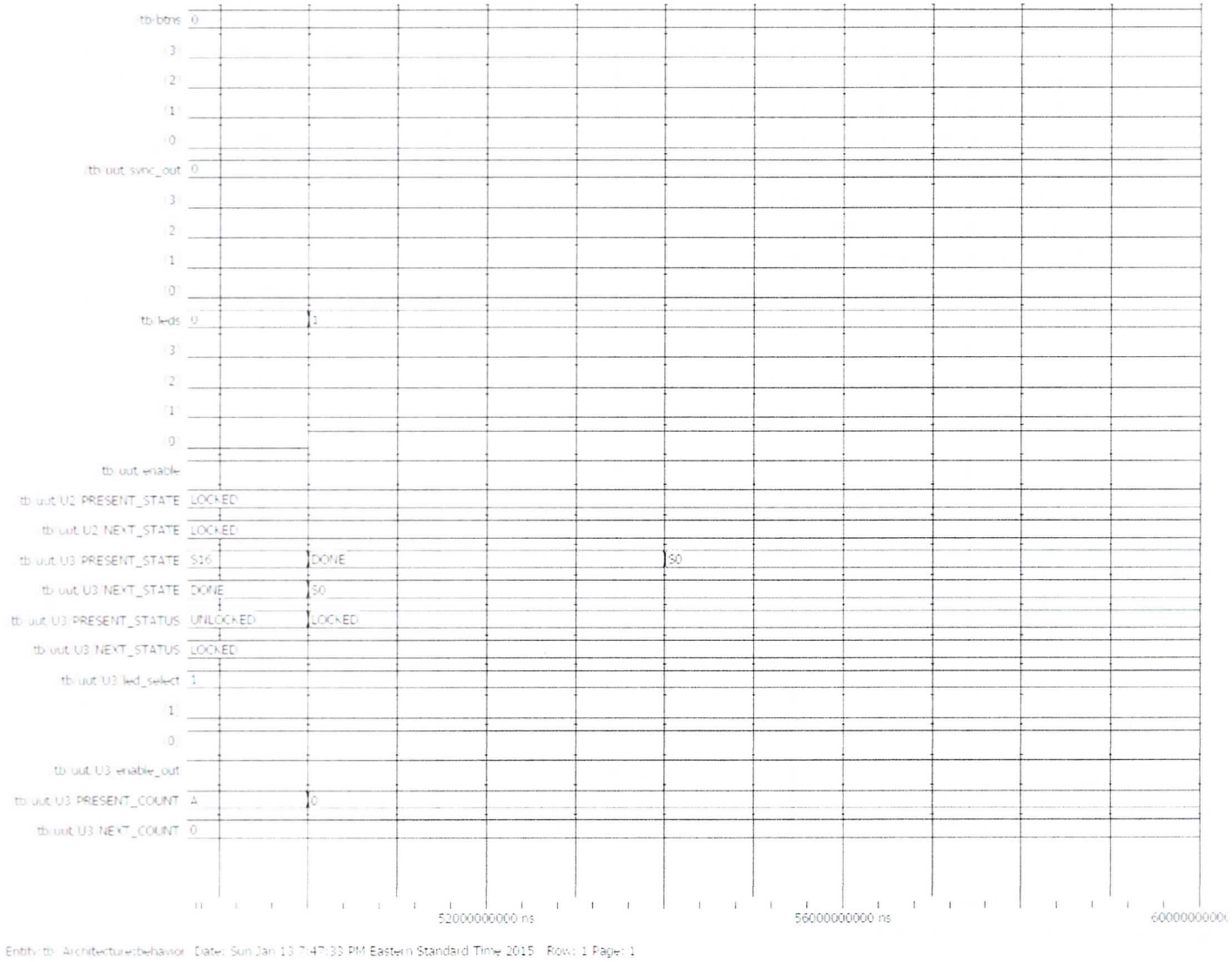



Figure 17 – Correct Input 50,000,000,000 to 60,000,000,000 ns

The figures 18 – 21 illustrate cases when the wrong input combination of <0, 3, 2, 1> is entered into the electronic combination lock and the ignoring of other inputs when the FSM is in an ERROR state.

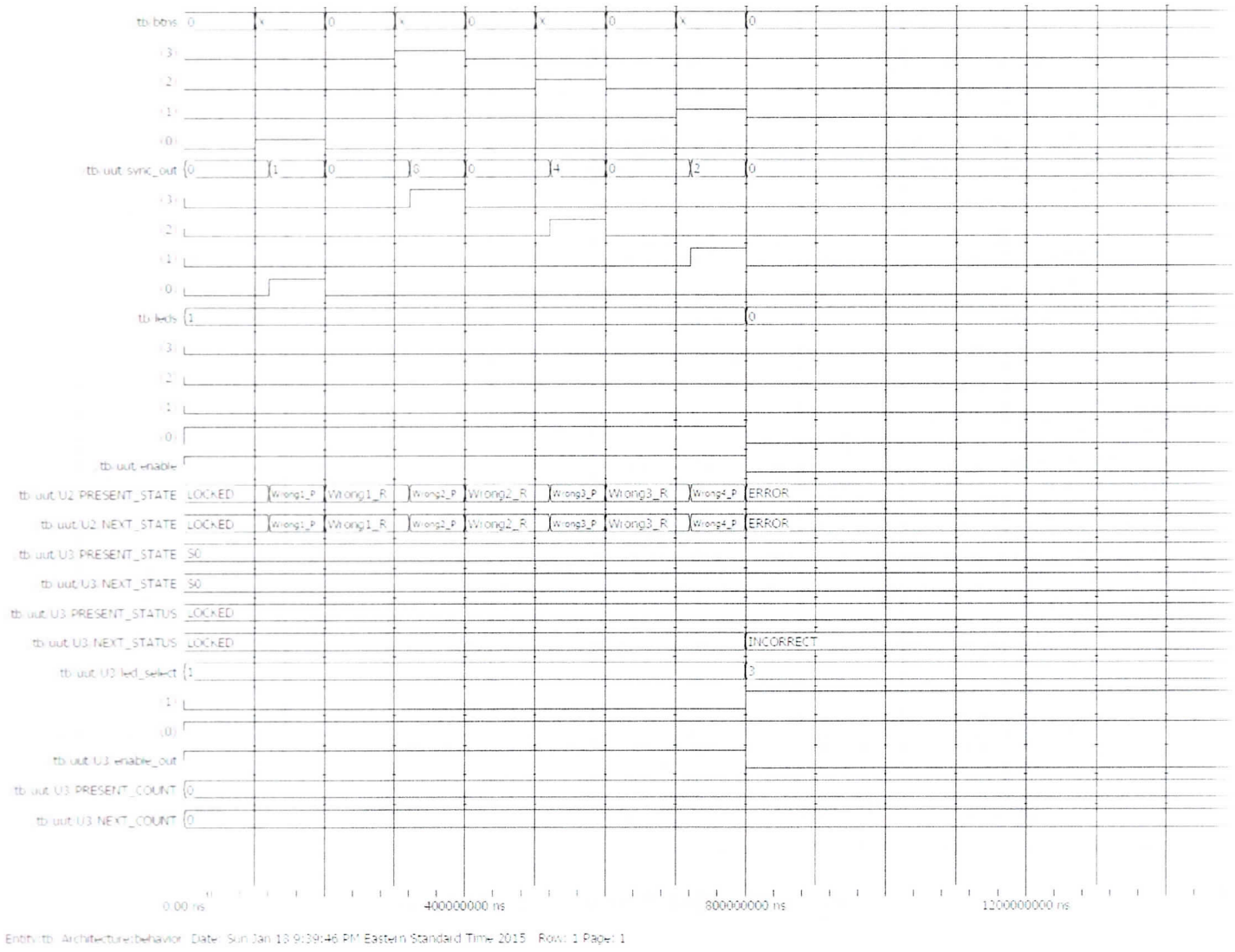



Figure 18 – Incorrect Input 1 to 1,400,000,000 ns

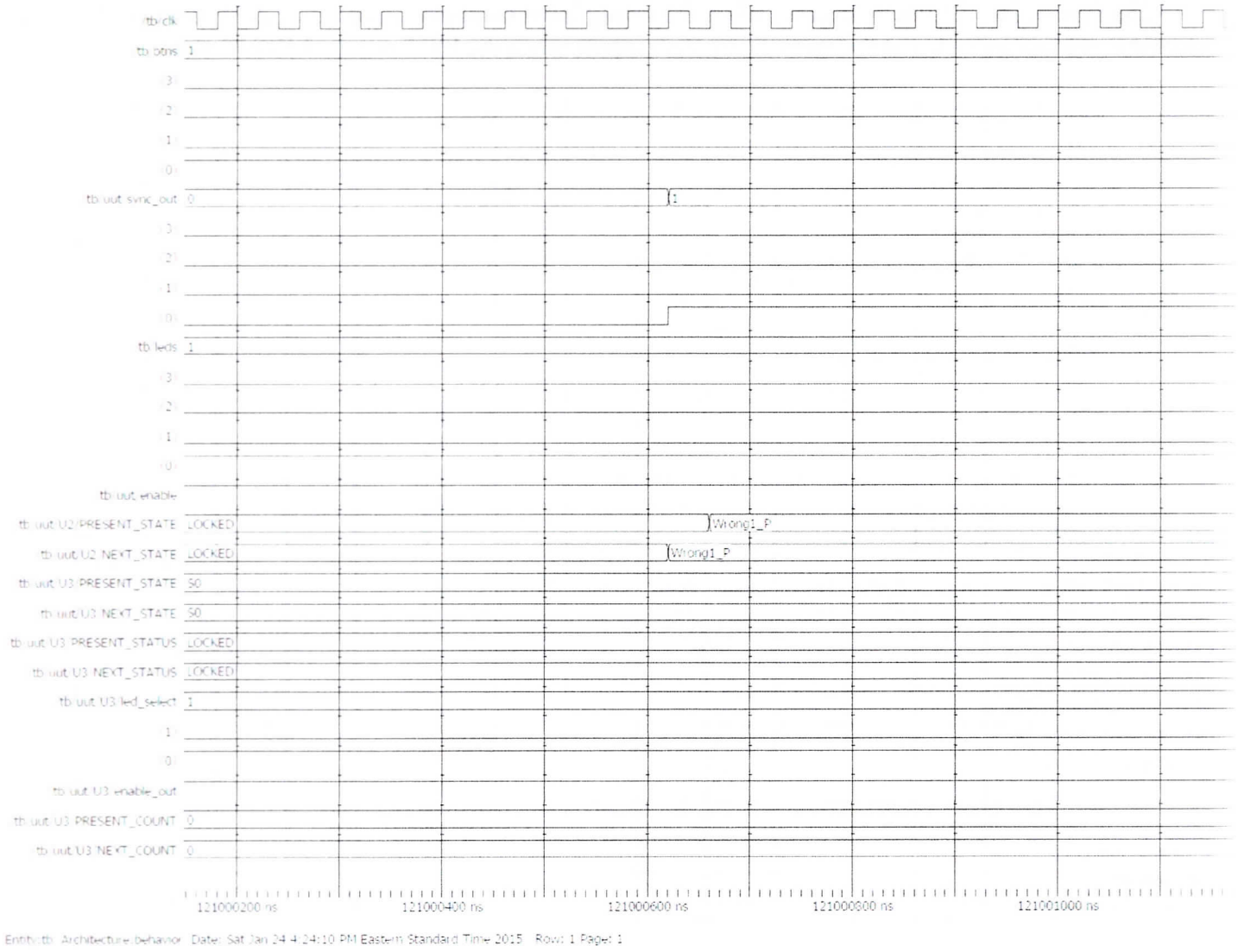



Figure 19 – First Wrong Input is Detected

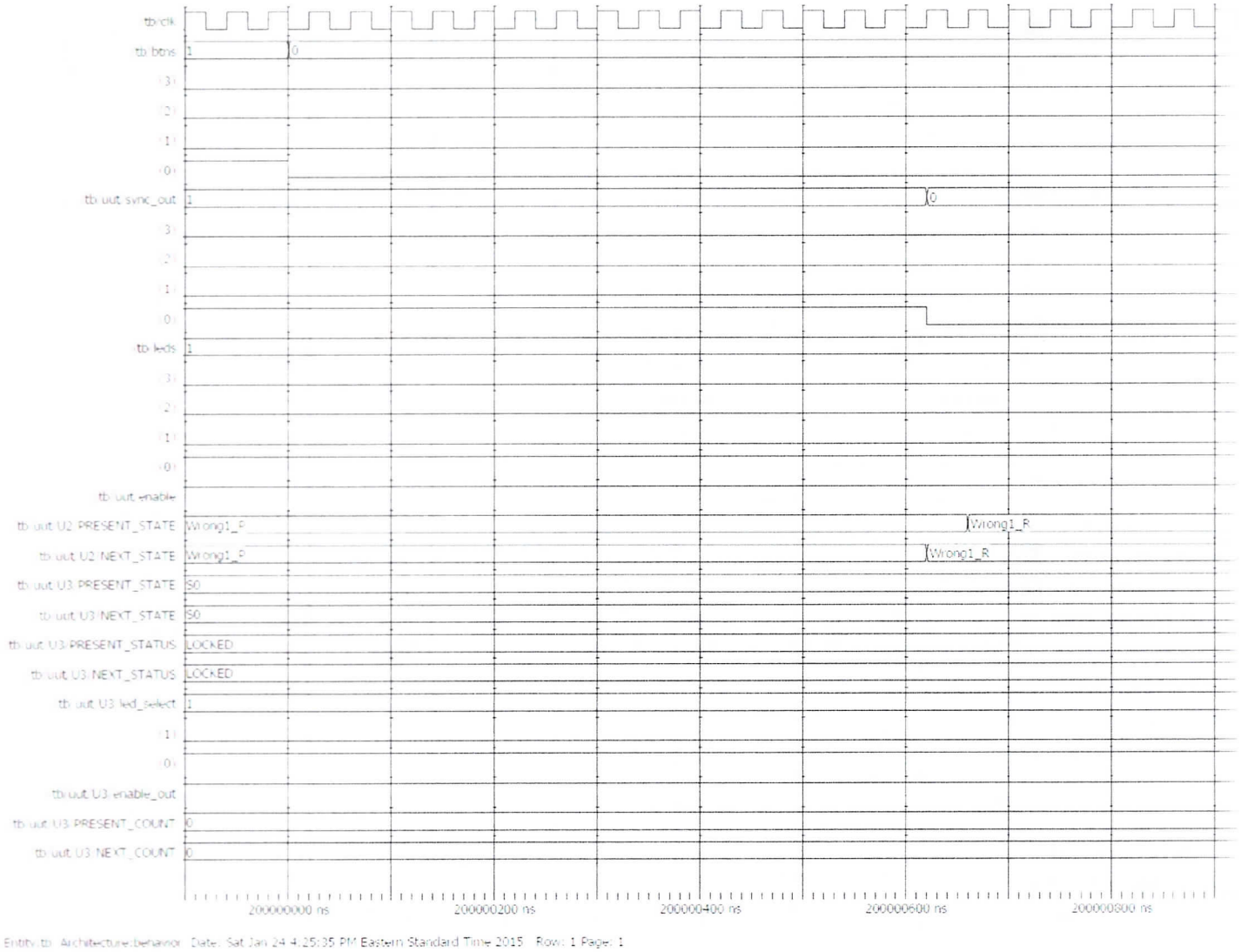



Figure 20 – First Wrong Input Button is Released

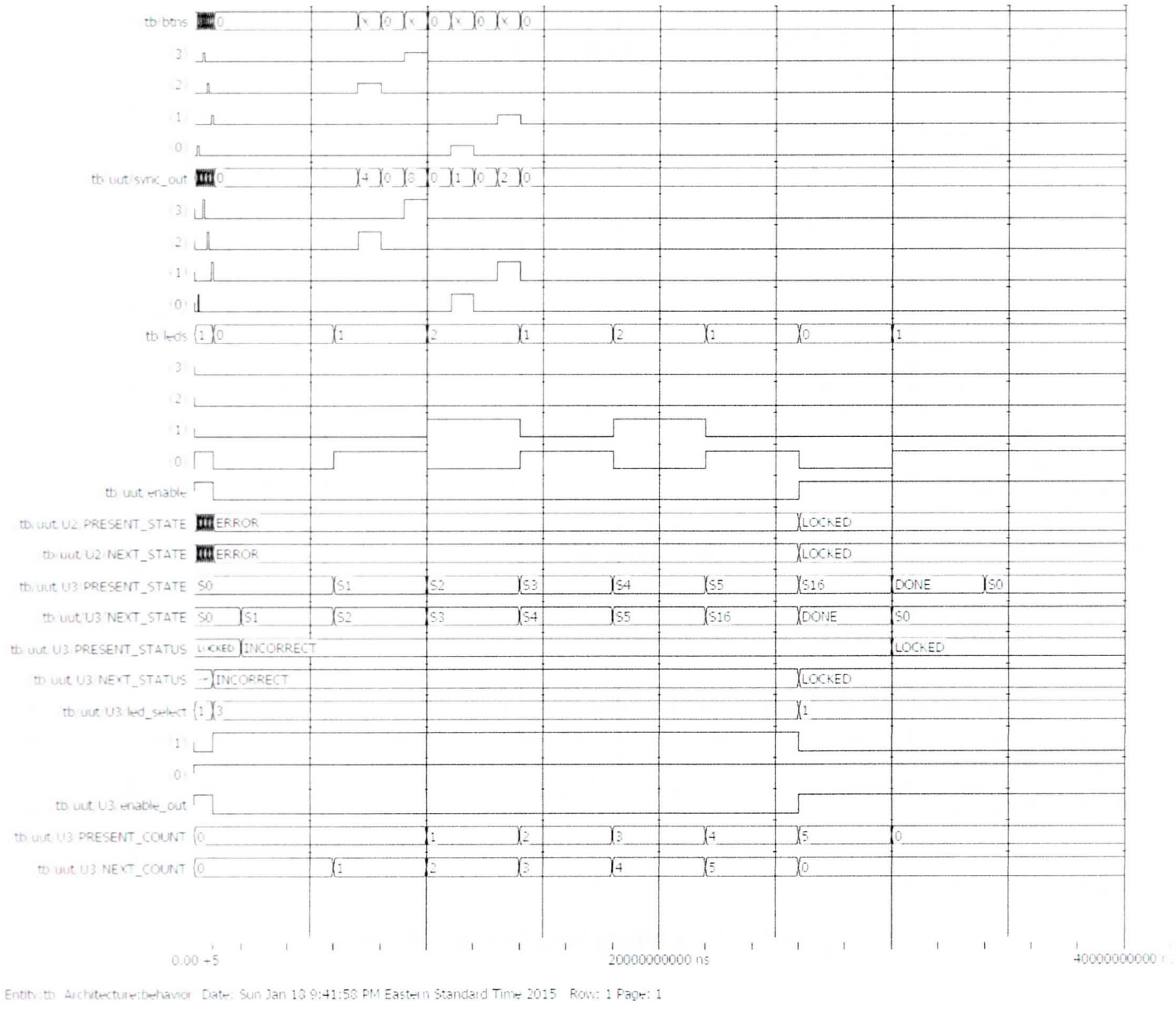



Figure 21 – Incorrect Input and Demonstrating Ignored Inputs.

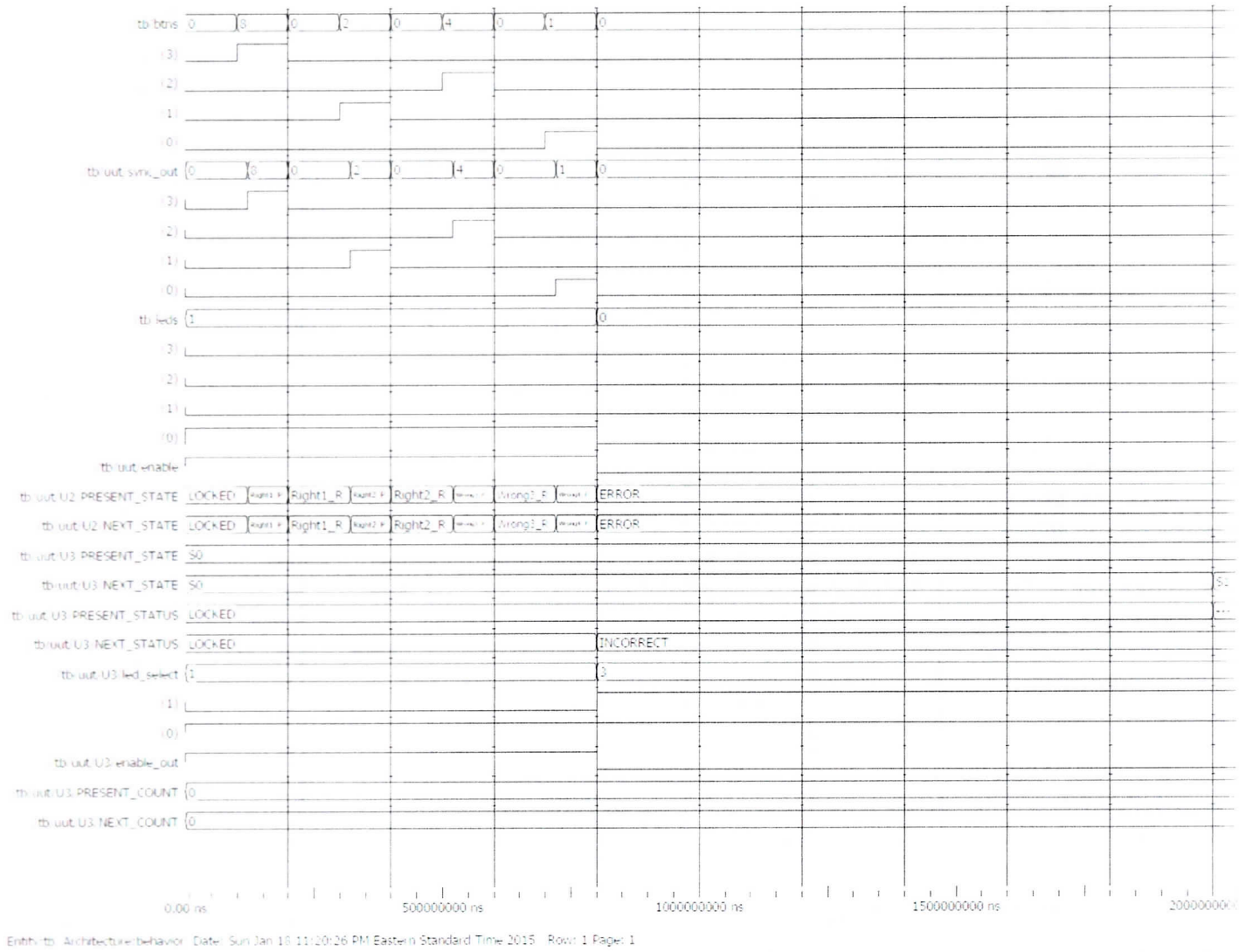



Figure 22 – Partial Correct Input 0 to 200,000,000 ns

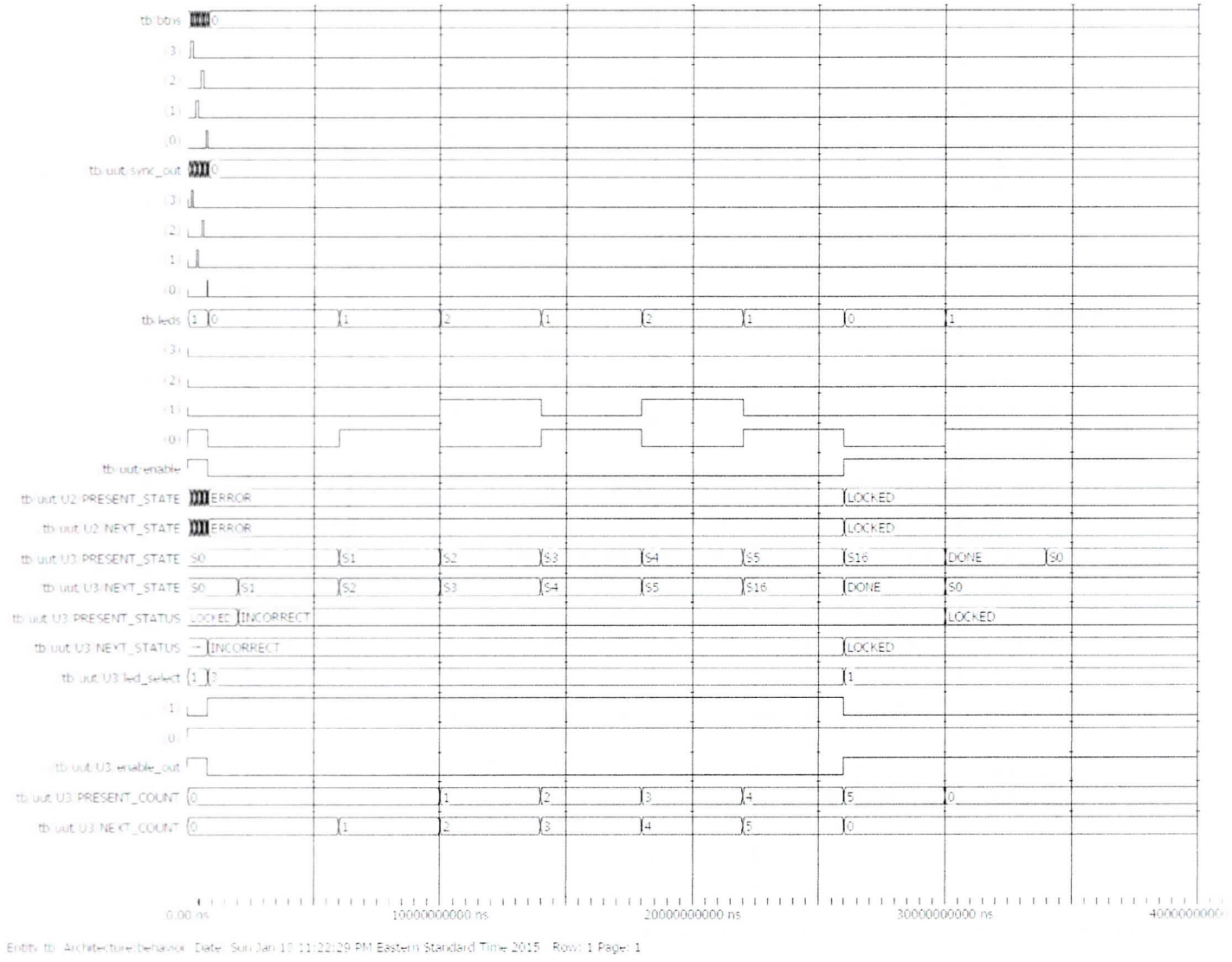



Figure 23 – Partial Correct Input 0 to 40 seconds

The .vhd files attached after this page are the following in the order shown.

1. Script for displaying waveforms.
2. Electronic_Combination_Lock_tb.vhd
3. Electronic_Combination_Lock.vhd
4. CombinationLockFSM.vhd
5. LED_Control_FSM.vhd
6. LEDTimer.vhd
7. SwitchDebouncer.vhd
8. Synchronizer.vhd
9. ClockDivider.vhd

The only thing we need to discuss about LEDTimer.vhd is that the constant used in the generic map is different from the one in clock_divider.vhd in order to create a time pulse signal to be used by the LED_Control_FSM in order to time the outputs correctly.

```
vsim tb
view -undock wave
add wave sim:/tb/clk
add wave sim:/tb/btns
add wave sim:/tb/uut/sync_out
add wave sim:/tb/leds
add wave sim:/tb/uut/enable
add wave sim:/tb/uut/U2/PRESENT_STATE
add wave sim:/tb/uut/U2/NEXT_STATE
add wave sim:/tb/uut/U3/PRESENT_STATE
add wave sim:/tb/uut/U3/NEXT_STATE
add wave sim:/tb/uut/U3/PRESENT_STATUS
add wave sim:/tb/uut/U3/NEXT_STATUS
add wave sim:/tb/uut/U3/led_select
add wave sim:/tb/uut/U3/enable_out
add wave sim:/tb/uut/U3/PRESENT_COUNT
add wave sim:/tb/uut/U3/NEXT_COUNT
radix -hexadecimal
run 40000000000 ns
```