Lab 8: Keyboard (Calculator and Caps Lock Control)

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Contents

Lab 8: Keyboard (Calculator and Caps Lock Control)

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Lab 8-1: Numbers And Alphabets

Design Specification

Source Code

global

Global variables

Keyboard Decoder

Inout PS2_DATA, PS2_CLK

Input rst, clk

Output [511:0] key_down, [8:0] last_change, key_valid

OnePulse

Output signal_single_pulse

Input signal, clock

Symbol To 7-Segment

Output [7:0] seg7_code

Input [8:0] kb_code

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3] d_sel, [7:0] d_out

Input: clk, rst, [7:0] d0, [7:0] d1, [7:0] d2, [7:0] d3

Exp 1

Output [3:0] d_sel, [7:0] d_out, [8:0] last_change, key_valid

Inout PS2_DATA, PS2_CLK

Input rst, clk

Design Implementation

Global Variables The global variables are used across the whole project.

Keyboard Decoder It′s the provided keyboard decoder by TAs.

OnePulse It′s the provided one-pulse module by TAs.

Symbol To 7-Segment Convert the keyboard codes to the patterns of 7-segment display, including number 1 to 9, alphabets A, S, M, and enter key. The alphabet A represents addition and will be shown as 8`b1001110_1` in the 7-segment display. Meanwhile, the alphabet S and M represents subtraction and multiplication and will be shown as 8`b1111110_1` and 8`b1001000_1`.

7-Segment Frequency Divider To generate the 1000 Hz clock, I use variables `counter_in` and `counter_out` to count from 0 to 50000. The counter_in will store the value for the next time step and pass the value to the `counter_out` when the clock raises. The reason why we need 50000 counting is each counting is triggered only when the clock raises, so the circuit will count 1 more for every twice clock pulses.

7-Segment Display Since we can only control one digit of the 7-segment display each time, I design a module that takes the 4-digit patterns as input and shows the 1 digit on the display when the clock raises. Whenever the clock raises, the module will switch the control d_sel to different digit and shows the corresponding digit. Take an example, when the first clock raise occur, the module will set `d_sel` = 4`b1110` and `d_out` = `d0`. As for second clock pulse, the module will output `d_sel` = 4`b1101` and `d_out` = `d1` and so on.

Exp 1 Combine all the modules listed above. The keyboard decoder decodes the keyboard signal and output the code of the key that is pressed. Then, `symbol_to_7segment` converts the keyboard code to the 7-segment pattern, which will be shown in the 7-segment display.

I/O Pin Assignment

I/O	PS2_CLK	PS2_DATA	rst	clk	key_valid
LOC	C17	B17	V17	W5	L1

I/O	d_sel[0]	d_sel[1]	d_sel[2]	d_sel[3]
LOC	U2	U4	V4	W4

I/O	last_change[0]	last_change[1]	last_change[2]	last_change[3]
LOC	U16	E19	U19	V19

I/O	last_change[4]	last_change[5]	last_change[6]	last_change[7]	last_change[8]
LOC	W18	U15	U14	V14	V13

I/O	d_out[0]	d_out[1]	d_out[2]	d_out[3]	d_out[4]	d_out[5]	d_out[6]	d_out[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

Block Diagram

Lab 8-1: Numbers And Alphabets
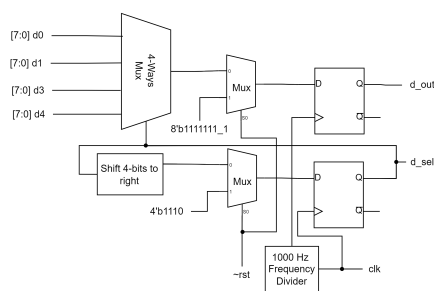
Figure 1: Lab 8-1 Logic Diagram

RTL Simulation
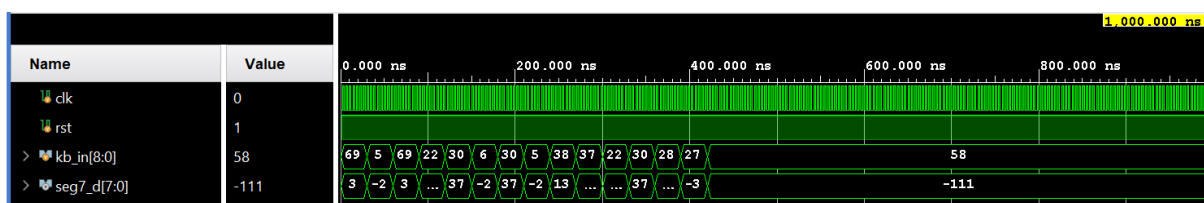
Figure 2: Lab 8-1 RTL Simulation

Lab 8-2: Single-Digit Adder

Design Specification

Source Code

global

Global variables

Keyboard Decoder

Inout PS2_DATA, PS2_CLK

Input rst, clk

Output [511:0] key_down, [8:0] last_change, key_valid

OnePulse

Output signal_single_pulse

Input signal, clock

Frequency Divider

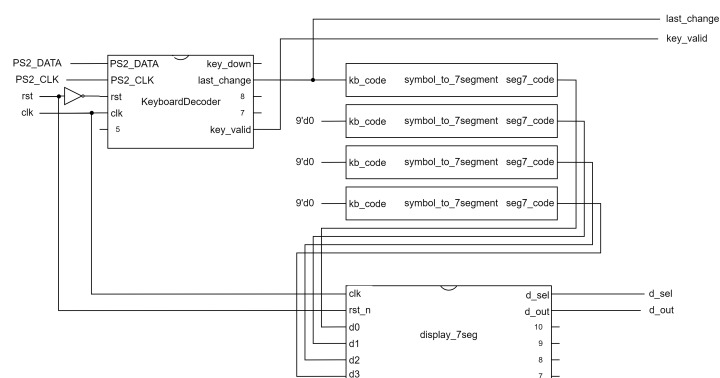Output: [29:0] clk_out

Input: clk, rst_n

Controller

output [7:0] a, [7:0] b, [7:0] sum, [7:0] kb_num

Input [8:0] kb_in, key_valid, clk, rst

Binary To 7-Segment Convertor

Input: [7:0] i

Output: [3:0] P, [7:0] D

Symbol To Binary

Output [7:0] binary

Input [8:0] kb_code

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3] d_sel, [7:0] d_out

Input: clk, rst, [7:0] d0, [7:0] d1, [7:0] d2, [7:0] d3

Exp 2

Output [3:0] d_sel, [7:0] d_out, [8:0] last_change, key_valid

Inout PS2_DATA, PS2_CLK

Input rst, clk

Design Implementation

Global Variables   The global variables are used across the whole project.

Keyboard Decoder   Same as lab 8-1

OnePulse   Same as lab 8-1

7-Segment Frequency Divider   Same as lab 8-1

7-Segment Display   Same as lab 8-1

Frequency Divider   The frequency divider module output a vector which represents the clock divided by 2^1, 2^2, 2^3..., 2^{30}. It can be simply implemented by adding 1 to a variable `[30:0]` `clk_out` at each clock pulse.

Controller   In lab 8-2, I use F1 and F2 to indicate the augend `a` and the addend `b`. Whenever the key F1 is pressed, the user can only modify the value of the augend `a` in the following until the key F2 is pressed. It can be implemented by a finite state machine and it always detect the latest pressed key `kb_in` and `key_valid` signal. When the module captures the latest key is F1 and the `key_valid` signal is 1, it will set the internal variable `digit` to 0. When the module captures the F2 in the same way, it will set the internal variable `digit` to 1.

In the mean time, the module also captures numbers 0 ~ 9 and key "enter". When the module captures numbers 0 ~ 9, it will convert the keyboard code `kb_in` into binary number `kb_num` and set augend `a` or addend `b` according to the variable `digit`. Whenever the enter key is pressed or the `rst` signal is low, the module will reset the augend `a` and addend `b` to 0.

Finally, the module will compute the sum of augend `a` and addend `b` and store them in the variable `sum`.

Binary To 7-Segment Convertor   Convert 4-bit binary number to 7-segment display with switch-case syntax.

Symbol To Binary   The module simply convert the keyboard codes of 0 ~ 9 in the left side into binary numbers.

7-Segment Frequency Divider   Same as lab 8-1

7-Segment Display   Same as lab 8-1

Exp 2   The module combines all modules listed above. Firstly, the module gets the latest pressed key `last_change` and signal `key_valid` from `KeyboardDecoder`. Then pass the signal into `controller` and output the augend a, addend b, and sum `sum`. Finally, the augend a, addend b, and the sum `sum` will be converted into 7-segment pattern by the module `segment7`, which will be shown in the 7-segment display by the module `display_7seg`

I/O Pin Assignment

I/O	PS2_CLK	PS2_DATA	rst	clk	key_valid
LOC	C17	B17	V17	W5	L1

I/O	d_sel[0]	d_sel[1]	d_sel[2]	d_sel[3]
LOC	U2	U4	V4	W4

I/O	last_change[0]	last_change[1]	last_change[2]	last_change[3]
LOC	U16	E19	U19	V19

I/O	last_change[4]	last_change[5]	last_change[6]	last_change[7]	last_change[8]
LOC	W18	U15	U14	V14	V13

I/O	d_out[0]	d_out[1]	d_out[2]	d_out[3]	d_out[4]	d_out[5]	d_out[6]	d_out[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

Block Diagram

Lab 8-2: Single-Digit Adder

Exp 2 (exp_2.v)
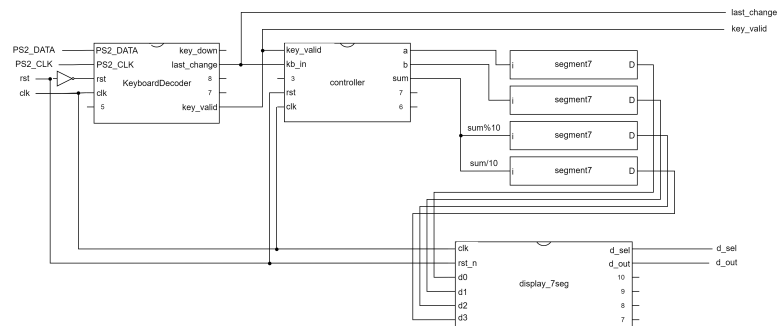
Controller (controller.v)

Figure 3: Lab 8-2 Logic Diagram

RTL Simulation
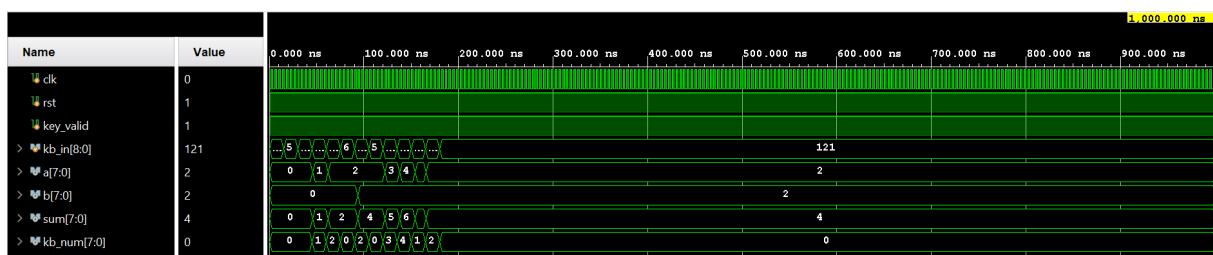
Figure 4: Lab 8-2 RTL Simulation

Lab 8-3: 2-Digit Calculator

Design Specification

Source Code
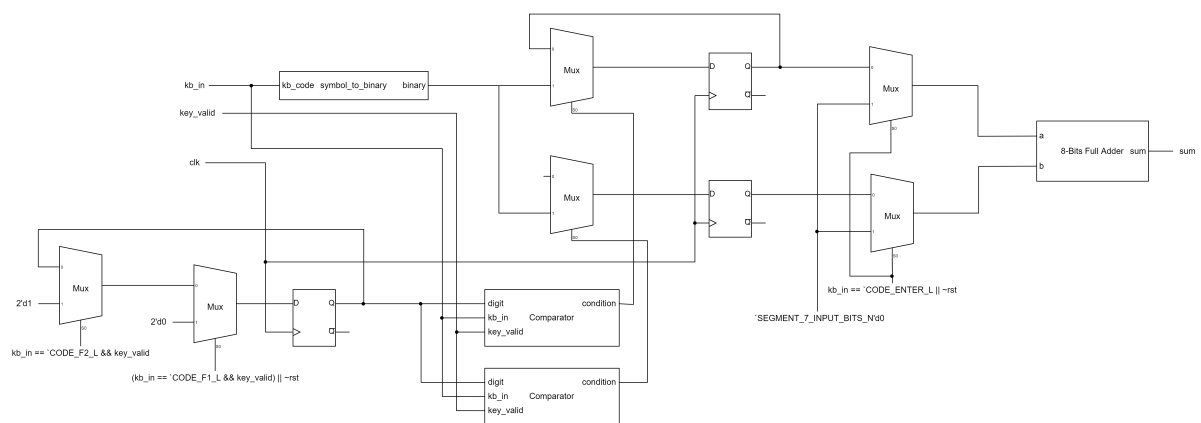
global

Global variables

Keyboard Decoder

Inout PS2_DATA, PS2_CLK

Input rst, clk

Output [511:0] key_down, [8:0] last_change, key_valid

OnePulse

Output signal_single_pulse

Input signal, clock

Count

Input clk, key_valid, [511:0]key_down, [8:0]last_change, rst_n

Output [3:0] in0, [3:0] in1, [3:0] in2, [3:0] in3, [13:0] value, [2:0] state

Value

Input [13:0] in

Output [3:0] out0, [3:0] out1, [3:0] out2, [3:0] out3

Number Checker

Output is_number

Input [1:0] lr, [8:0] kb_in

Binary To 7-Segment Convertor

Input: [7:0] i

Output: [3:0] P, [7:0] D

Symbol To Binary

Output [7:0] binary

Input [8:0] kb_code

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3] d_sel, [7:0] d_out

Input: clk, rst, [7:0] d0, [7:0] d1, [7:0] d2, [7:0] d3

Decimal Display

Output [3:0] d_sel, [7:0] d_out

Input [3:0] d0, [3:0] d1, [3:0] d2, [3:0] d3, clk, rst

Exp 3

Output [3:0] d_sel, [7:0] d_out, [8:0] last_change, key_valid

Inout PS2_DATA, PS2_CLK

Input rst, clk

Design Implementation

Global Variables   The global variables are used across the whole project.

Keyboard Decoder   Same as lab 8-1

OnePulse   Same as lab 8-1

Count   This module is a state machine that determines the digit will be set. In state 0 and 1, the calculator will set the tens digit and the unit digit of the first number respectively. In state 2, the calculator will decide the operator, which is addition, subtraction, or multiplication. In state 3 and 4, the calculator will set the tens digit and the unit digit of the second number. In state 5, the module will compute the result according to the input numbers and operator and goes back to state 0 when the enter key is pressed.

Value   Convert the input value `in` to 4-digit decimal number in `out0`, `out1`, `out2`, and `out3`.

Number Checker   Check whether the input keyboard code `kb_in` is a valid number or not. The input argument `lr` can determine which side of number key is valid, left, right, or both side.

7-Segment Frequency Divider   Same as lab 8-1

7-Segment Display   Same as lab 8-1

Binary To 7-Segment Convertor   Same as lab 8-2

Symbol To Binary   Same as lab 8-2

7-Segment Frequency Divider   Same as lab 8-1

7-Segment Display   Same as lab 8-1

Decimal Display   The module shows the binary number `d0`, `d1`, `d2`, and `d3` on the 7-segment display. It convert the binary number `d0`, `d1`, `d2`, and `d3` to 7-segment pattern and then, put them into the module `display_7seg` to show the number on the 7-segment display.

Exp 3   This is the top module of lab 8-3. It combines all modules listed above. The `KeyboardDecoder` outputs the latest key that pressed and then, the module `controller` takes it as input and output the 3 numbers `a`, `b`, and `sum`. Finally, the module will choose to show the variables `a` and `b` or `sum` and the module `dec_disp` takes the variables `a`, `b`, and `sum` as input and show them on the 7-segment display.

I/O Pin Assignment

I/O	PS2_CLK	PS2_DATA	rst_n	clk
LOC	C17	B17	V17	W5

I/O	d[0]	d[1]	d[2]	d[3]
LOC	U2	U4	V4	W4

I/O	D_ssd[0]	D_ssd[1]	D_ssd[2]	D_ssd[3]	D_ssd[4]	D_ssd[5]	D_ssd[6]	D_ssd[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

Block Diagram

Figure 5: Lab 8-3 Logic Diagram

RTL Simulation

Figure 6: Lab 8-3 RTL Simulation

Lab 8-4: Caps Control

Design Specification

Source Code

global

Global variables

Keyboard Decoder

Inout PS2_DATA, PS2_CLK

Input rst, clk

Output [511:0] key_down, [8:0] last_change, key_valid

OnePulse

Output signal_single_pulse

Input signal, clock

Finite State Machine

Input clk, key_valid, [511:0] key_down, [8:0] last_change

Output led_state
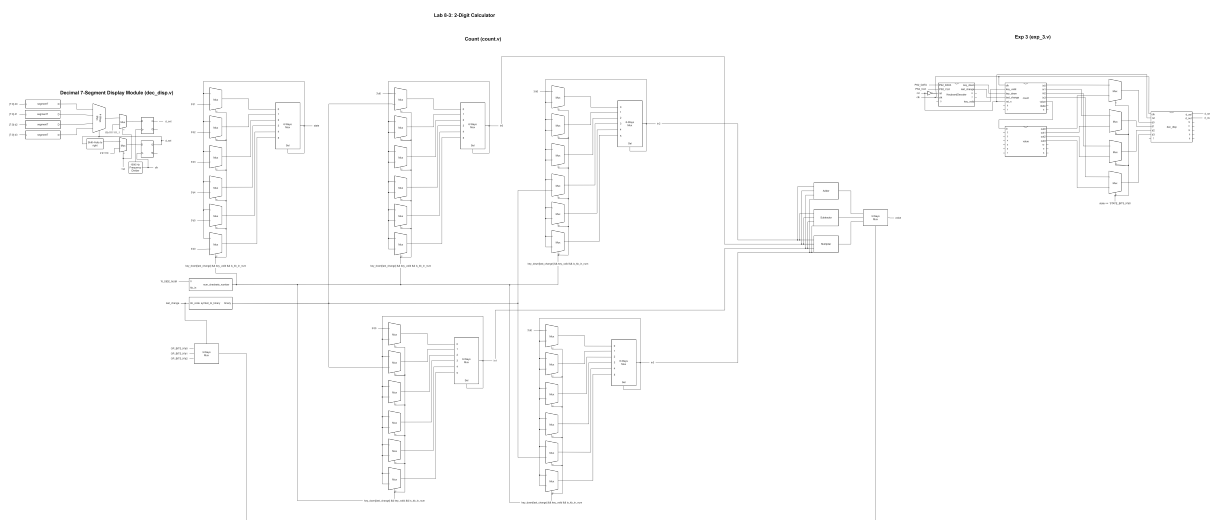
Key Control

Input clk, [511:0]key_down, key_valid, [8:0] last_change

Output [4:0] char

Shift Control

Input clk, key_valid, [8:0] last_change

Output shift_state

Out Control

Input [4:0] char, rst, state, clk, shift_state

Output [6:0] led

Exp 4

Input clk, rst, PS2_DATA, PS2_CLK

Output led_state, [6:0] led, is_pressed

Global Variables   The global variables are used across the whole project.

Keyboard Decoder   Same as lab 8-1

OnePulse   Same as lab 8-1

Finite State Machine   The module `fsm` detect the status of the cap key. Whenever the cap key is clicked, the state will change. I use `key_down[last_change] == 1'b1 && key_valid && tmp == `CODE_CAP_L` to detect the click of cap key and generate one-pulse signal.

Key Control   The module converts the keyboard code `last_change` into ASCII code **char** of the alphabets.

Shift Control   The module captures the press of shift key. When the shift key is pressed, the state goes to 1, otherwise, the state will be 0.

Out Control   The module determines to output the lower or the upper case of the character **char** according to 2 arguments `state ^ shift_state`. The module convert the argument **char** into ASCII code in lower or upper cases by switch statement.

Exp 4 It′s the top module of lab 8-4 and it combines all modules mentioned above. First, it pass the output of the module `KeyboardDecoder`, including `key_down`, `last_change`, and `key_valid` to module `fsm` and `shift`. The module `fsm` and `shift` output 2 signal `state` and `shift_state`. In the mean time, the module `key` convert the keyboard code into binary numbers. Finally, the module `out` takes `state`, `shit_state`, and **char** as input and output the ASCII code of the alphabets.

I/O Pin Assignment

I/O	PS2_CLK	PS2_DATA	rst	clk
LOC	C17	B17	V17	W5

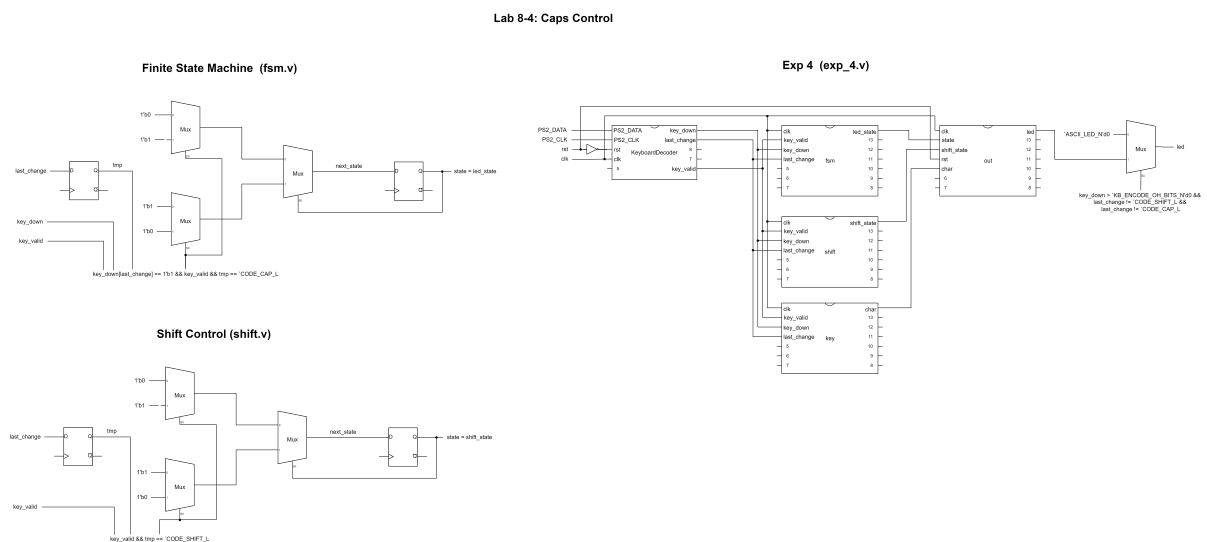| I/O	led[0]	led[1]	led[2]	led[3]	led[4]	led[5]	led[6]	is_pressed	led_state
LOC	U16	E19	U19	V19	W18	U15	U14	P1	L1

Block Diagram

Figure 7: Lab 8-4 Logic Diagram
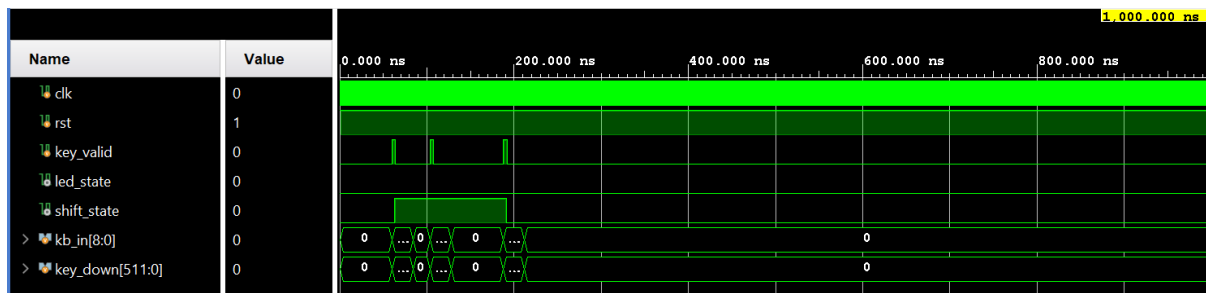
RTL Simulation

Figure 8: Lab 8-4 RTL Simulation

Discussion

In Lab 8-3, I spend lots of time in detecting the click of a key. I' ve tried implement a one-pulse module `keyboard` for the `key_down` signal and it works. However, it sometimes occurred race condition error when the module `keyboard` works with other module. It' s a weird exception, so I finally give up using `keyboard` module to make one-pulse signal. I also found that `key_down[ last_change] && key_valid && last_change == target_key` is a great alternative way to detect the `target_key`. As a result, I use this way to detect the click of the keyboard and handle state transition.

Conclusion

The lab 8 is much harder than lab 7 and it combines the keyboard controlling. It' s a interesting that controlling the keyboard with FPGA. I also learn how to capture the keyboard signal and make it become one-pulse signal.

Reference

- None