

EL6463 Advanced Hardware Design

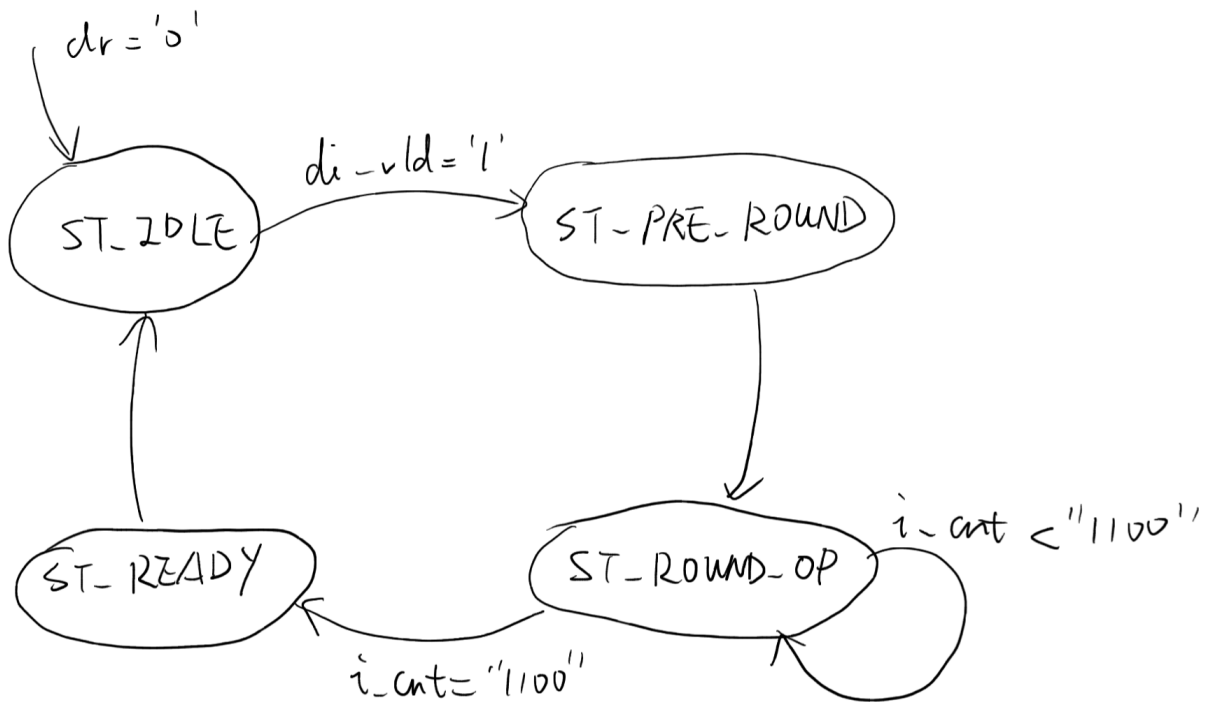
Lab #5

Name: Chen Shen

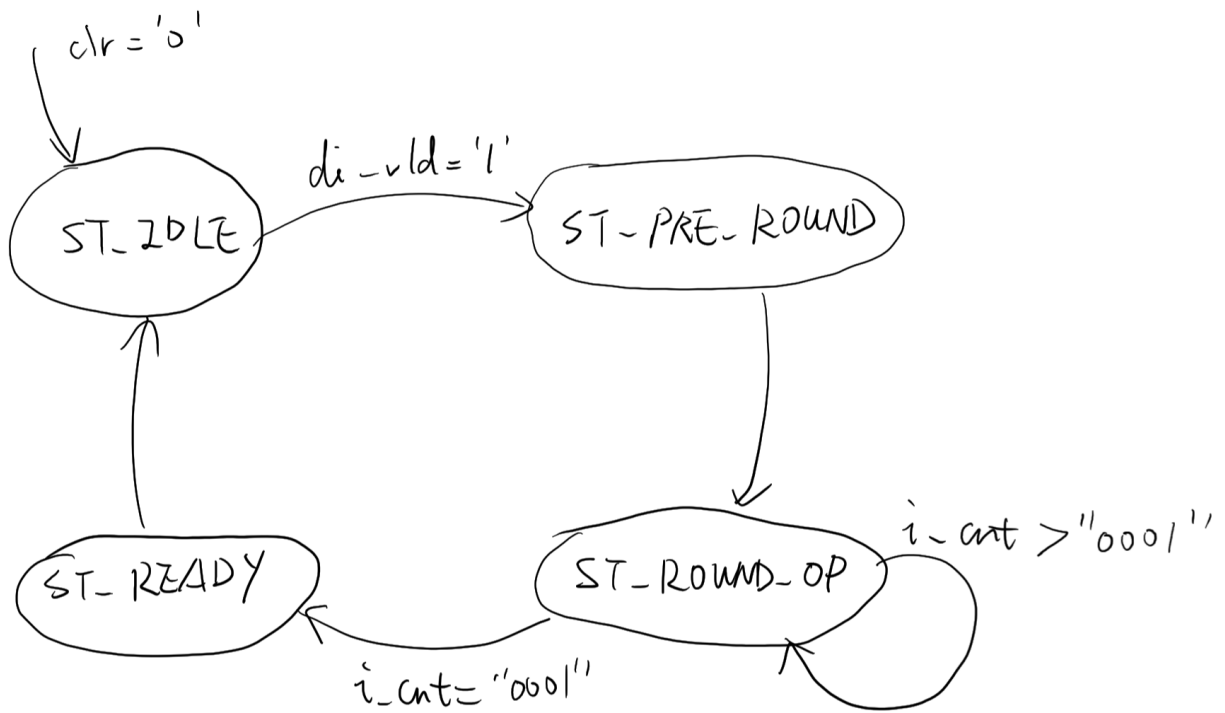
netID: cs5236

FSM Diagram

RC5 Encryption



RC5 Decryption



Test Cases

In my test bench, I used 3 din for encryption and decryption.

din = x"0000000000000000"

din = x"0123456789abcdef"

din = x"ffffffffffffffffffff"

Hand Calculation

Encryption: din = x"0000000000000000"

Before the main loop, we have A = x"00000000" and B = x"00000000".

$$\begin{aligned}
 \text{din} &= \text{x"}0000\ 0000\ 0000\ 0000\text{"} \\
 \underline{a} &= \underline{\text{x"}0000\ 0000\text{"}} \quad \underline{b} = \underline{\text{x"}0000\ 0000\text{"}}
 \end{aligned}$$

In the first loop, after the first line, we have A = x"46f8e8c5". After the second line, we have B = x"2529792d".

$i = 1$

$$ab_xor = a \oplus b = x"00000000"$$

$$a_rot = ab_xor \lll b = x"00000000" \lll 0 = x"00000000"$$

$$a = a_rot + S[2] = x"00000000" + x"46F8E8C5" \\ = x"46F8E8C5"$$

$$ba_xor = b \oplus a = x"46F8E8C5" \oplus x"00000000" = x"46F8E8C5"$$

$$b_rot = ba_xor \lll a = x"46F8E8C5" \lll x"46F8E8C5" \\ = x"46F8E8C5" \lll "00101" \\ = x"46F8E8C5" \lll 5$$

$$x"46F8E8C5" = \begin{array}{cccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}$$

5 MSBs move to LSBs ↑

$$b_rot = "11011110001101000100010101000" = x"DF1D18A8"$$

$$b = b_rot + S[3] = x"DF1D18A8" + x"460C6085" = x"2529792D"$$

$$\underline{a = x"46F8E8C5"} \quad \underline{b = x"2529792D"}$$

In the second loop, after the first line, we have $A = x"a3354804"$. After the second line, we have $B = x"8a0e959b"$.

$$i = 2$$

$$ab_xor = a \text{ xor } b = x''46F8E8C5'' \text{ xor } x''2529792D''$$

$$a = "0100\ 0110\ 1111\ 1000\ 1110\ 1000\ 1101\ 0101"$$

$b = "0010\ 0101\ 0010\ 1001\ 0111\ 1001\ 0010\ 1101"$

$$ab_xor = "0110\ 0011\ 1101\ 00011001\ 0001\ 1111\ 1000" = x"63D191E8"$$

$$a_{-wt} = ab_{-xor} \lll b = x''63D191E8'' \lll x''2529792D''$$

$$= x''630191E8'' \lll ''01101''$$

$$= x'' 63D191E8'' \lll 13$$

$x''63D191E8'' = 11011000111010001001000111101000''$
 13 MSBs move to LBSs

$a_rot = "00110010001111010000110001111010" = x"323D0C7A"$ more to 2553

a = a-not + S[4] = x"323D0C7A" + x"70F83B8A"

$= x' A3354804''$

$$b \oplus a = x''2529792D'' \oplus x''A3354804''$$

$b = "0010\ 0101\ 0010\ 1011\ 0111\ 1001\ 0010\ 1101"$

$$a = "1010\ 0011\ 0011\ 0101\ 0100\ 1000\ 0000\ 0100"$$

$ba_xor = "10000110000111000011000100101001" = x'861C3129"$

$$b_not = b_xor \lll a = x''861c3129'' \lll x''A2354804''$$

$$= x''861C3129'' \lll ''00100''$$

$$= x^{86103129} \lll 4$$

$x = 861C3129_{16} = 0000\ 1000\ 0110\ 0000\ 1100\ 0011\ 0000\ 1001\ 0010\ 1000\ 1001_{2}$

b-not = "0110 0001 1100 0011 0001 0010 1001 1000" = x"61C31298"

$$b = b_{\text{rot}} + S[5] = x''61c312f8'' + x''284B8303'' = x''8A0E959B''$$

a = x" A3354804 " b = x" 8A0E959B "

In the third loop, after the first line, we have `A = x"4a87f340"`. After the second line, we have `B = x"b6ab53fd"`.

$i = 3$

$$ab_xor = a \oplus b = x" A3354804 " \oplus x" 8A0E959B "$$

$$a = " 1010 \ 0011 \ 0011 \ 0101 \ 0100 \ 1000 \ 0000 \ 0100 "$$

$$b = " 1000 \ 1010 \ 0000 \ 1110 \ 1001 \ 0101 \ 1001 \ 1011 "$$

$$ab_xor = " 0010 \ 1001 \ 0011 \ 1011 \ 1101 \ 1101 \ 1001 \ 1111 " = x" 293BDD9F "$$

$$a_rot = ab_xor \ll b = x" 293BDD9F " \ll x" 8A0E959B "$$

$$= x" 293BDD9F " \ll " 1101 "$$

$$= x" 293BDD9F " \ll 27$$

$$x" 293BDD9F " = " \underbrace{0010 \ 1001 \ 0011 \ 1011 \ 1101 \ 1101 \ 1001 \ 1111}_{27 \text{ MSBs}} \uparrow$$

more to LBS

$$a_rot = " 1111 \ 1001 \ 0100 \ 1001 \ 1101 \ 1110 \ 1110 \ 1100 " = x" F949DEEC "$$

$$a = a_rot + S[6] = x" F949DEEC " + x" 513E1454 "$$

$$= x" 4A87F340 "$$

$$ba_xor = b \oplus a = x" 8A0E959B " \oplus x" 4A87F340 "$$

$$b = " 1000 \ 1010 \ 0000 \ 1110 \ 1001 \ 0101 \ 1001 \ 1011 "$$

$$a = " 0100 \ 1010 \ 1000 \ 0111 \ 1110 \ 0011 \ 0100 \ 0000 "$$

$$ba_xor = " 1100 \ 0000 \ 1000 \ 1001 \ 0110 \ 0110 \ 1101 \ 1011 " = x" C08966DB "$$

$$b_rot = ba_xor \ll a = x" C08966DB " \ll x" 4A87F340 "$$

$$= x" C08966DB " \ll " 00000 "$$

$$= x" C08966DB " \ll 0$$

$$= x" C08966DB "$$

$$b = b_rot + S[7] = x" C08966DB " + x" F621ED22 " = x" B6AB53FD "$$

$$a = x" 4A87F340 "$$

$$b = x" B6AB53FD "$$

Decryption: $din = x" 0000000000000000 "$

Before the main loop, we have $A = x" 00000000 "$ and $B = x" 00000000 "$.

$$din = x" 0000000000000000 "$$

$$a = x" 00000000 "$$

$$b = x" 00000000 "$$

In the first loop, after the first line, we have $A = x" 93c8774f "$. After the second line, we have $B = x" 9afb9c80 "$.

i=12

$$b_minus = b - s[25] = x"00000000" - x"65046380" = x"9AFB9C80"$$

$$b_rot = b_minus \ggg a = x"9AFB9C80" \ggg x"00000000" \\ = x"9AFB9C80"$$

$$b = b_rot \text{ xor } a = x"9AFB9C80" \text{ xor } x"00000000" = x"9AFB9C80"$$

$$a_minus = a - s[24] = x"00000000" - x"F6CC1431" \\ = x"0933EBCF"$$

$$a_rot = a_minus \ggg b = x"0933EBCF" \ggg x"9AFB9C80" \\ = x"0933EBCF" \ggg "00000" \\ = x"0933EBCF"$$

$$a = a_rot \text{ xor } b = x"0933EBCF" \text{ xor } x"9AFB9C80"$$

$$a_rot = "0000 \ 1001 \ 0011 \ 0011 \ 1110 \ 1011 \ 1100 \ 1111"$$

$$b = "1001 \ 1010 \ 1111 \ 1011 \ 1001 \ 1100 \ 1000 \ 0000"$$

$$a = "1001 \ 0011 \ 1100 \ 1000 \ 0111 \ 0111 \ 0100 \ 1111" \\ = x"93C8774F"$$

$$\underline{a = x"93C8774F"} \quad \underline{b = x"9AFB9C80"}$$

In the second loop, after the first line, we have $A = x"e99c86aa"$. After the second line, we have $B = x"6130d88b"$.

$$i = 11$$

$$b_{\text{minus}} = b - s[22] = x"9AFB9C80" - x"43192304" = x"57E2797C"$$

$$b_{\text{rot}} = b_{\text{minus}} \gg a = x"57E2797C" \gg x"93C8774F"$$

$$= x"57E2797C" \gg \gg "0111"$$

$$= x"57E2797C" \gg \gg 15$$

$$x"57E2797C" = \begin{array}{cccccccc} 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}$$

move to MSBs

$$b_{\text{rot}} = "111100101111000101011111100100" = x"F2F8AFE4"$$

$$b = b_{\text{rot}} \text{ xor } a = x"F2F8AFE4" \text{ xor } x"93C8774F"$$

$$b_{\text{rot}} = "111100101111000101011111100100"$$

$$a = "100100111001000011101110100111"$$

$$b = "01100001001100001101100010001011" = x"6130D88B"$$

$$a_{\text{minus}} = a - s[22] = x"93C8774F" - x"30D76B0A" = x"62F10C45"$$

$$a_{\text{rot}} = a_{\text{minus}} \gg b = x"62F10C45" \gg x"6130D88B"$$

$$= x"62F10C45" \gg \gg "0101"$$

$$= x"62F10C45" \gg \gg 11$$

$$x"62F10C45" = \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

move to MSBs

$$a_{\text{rot}} = "10101000101011001011100010001" = x"88AC5E21"$$

$$a = a_{\text{rot}} \text{ xor } b = x"88AC5E21" \text{ xor } x"6130D88B"$$

$$a_{\text{rot}} = "10101000101011001011100010001"$$

$$b = "01100001001100001101100010001011"$$

$$a = "111010011001100010000110101010" = x"E99C86AA"$$

$$\underline{a = x"E99C86AA"} \quad \underline{b = x"6130D88B"}$$

In the third loop, after the first line, we have A = x"4987f97f". After the second line, we have B = x"20b04007".

$$i = 10$$

$$b_minus = b - S[21] = x''6135D88B'' - x''AE162167'' = x''B31AB724''$$

$$b_rot = b_minus \ggg a = x''B31AB724'' \ggg x''E99C86AA''$$

$$= x''B31AB724'' \ggg ''01010''$$

$$= x''B31AB724'' \ggg 10$$

$$x''B31AB724'' = ''1011\ 0011\ 0001\ 1010\ 1011\ 0111\ 0010\ 0100''$$

↑
move to MSBs

7 LSRs

$$b_rot = ''1100\ 1001\ 0010\ 1100\ 1100\ 0110\ 1010\ 1101'' = x''C92CC6AD''$$

$$b = b_rot \text{ xor } a = x''C92CC6AD'' \text{ xor } x''E99C86AA''$$

$$b_rot = ''1100\ 1001\ 0010\ 1100\ 1100\ 0110\ 1010\ 1101''$$

$$a = ''1110\ 1001\ 1001\ 1100\ 1000\ 0110\ 1010\ 1010''$$

$$b = ''0010\ 0000\ 1011\ 0000\ 0100\ 0000\ 0000\ 0111'' = x''20B04007''$$

$$a_minus = a - S[20] = x''E99C86AA'' - x''4DBFCA76'' = x''9BDCBC34''$$

$$a_rot = a_minus \ggg b = x''9BDCBC34'' \ggg x''20B04007''$$

$$= x''9BDCBC34'' \ggg ''00111''$$

$$= x''9BDCBC34'' \ggg 7$$

$$x''9BDCBC34'' = ''1001\ 1011\ 1101\ 1100\ 1011\ 1100\ 0011\ 0100''$$

↑
move to MSBs

7 LSRs

$$a_rot = ''0110\ 1001\ 0011\ 0111\ 1011\ 1010\ 0111\ 1000'' = x''6937B978''$$

$$a = a_rot \text{ xor } b = x''6937B978'' \text{ xor } x''20B04007''$$

$$a_rot = ''0110\ 1001\ 0011\ 0111\ 1011\ 1010\ 0111\ 1000''$$

$$b = ''0010\ 0000\ 1011\ 0000\ 0100\ 0000\ 0000\ 0111''$$

$$a = ''0100\ 1001\ 1000\ 0111\ 1111\ 1010\ 0111\ 1111'' = x''4987F97F''$$

$$\underline{a = x''4987F97F''}$$

$$\underline{b = x''20B04007''}$$

Resource Utilization

RC5 Encryption

	Synthesis stage	Place and Route stage
LUT and FF pairs usage	446 LUTs and 167 FFs	125
IOB usage	32	32
RAM/DSP blocks used (if any)	0	0

RC5 Decryption

	Synthesis stage	Place and Route stage
LUT and FF pairs usage	385 LUTs and 167 FFs	127
IOB usage	32	32
RAM/DSP blocks used (if any)	0	0

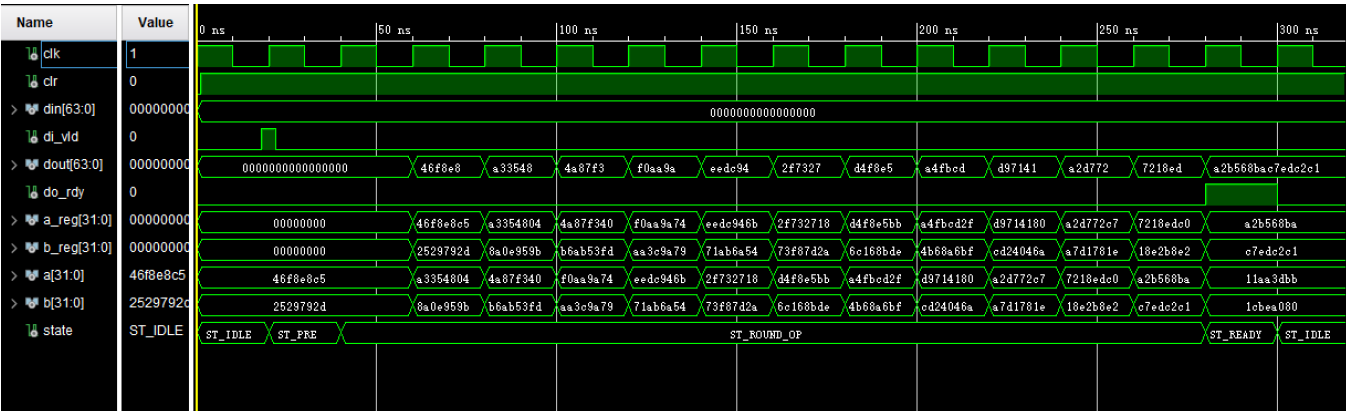
Simulation

*Notice that all the simulations use the same clock peroid - 20 ns

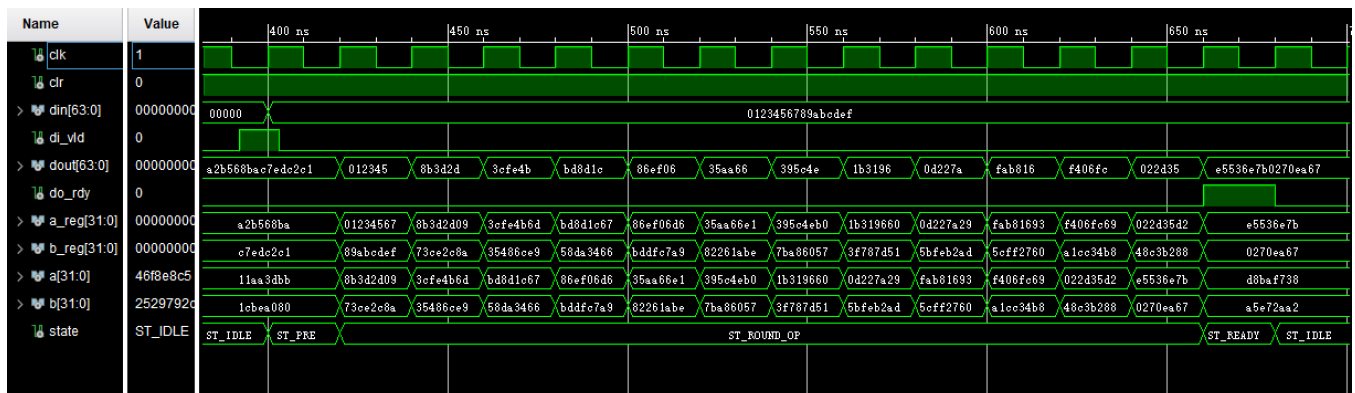
Functional Simulation

RC5 Encryption

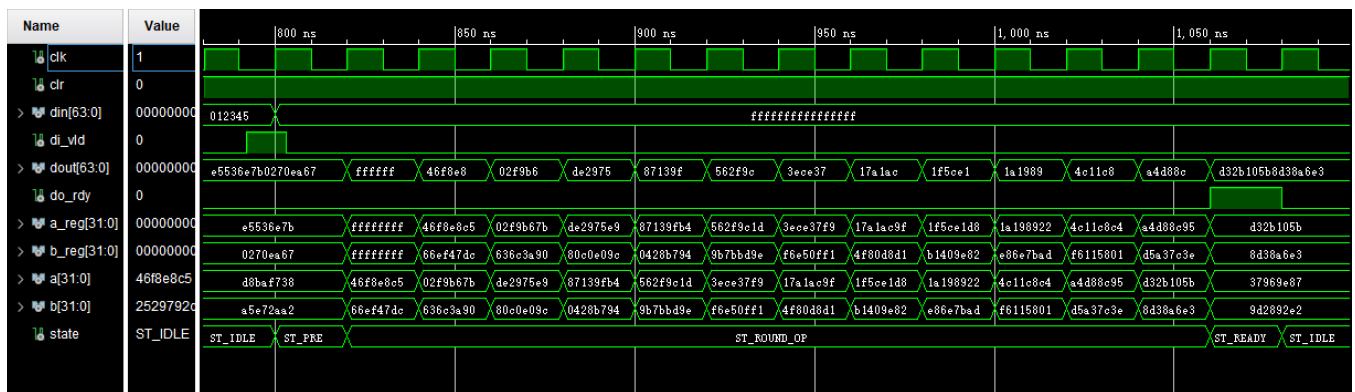
din = x"0000000000000000"



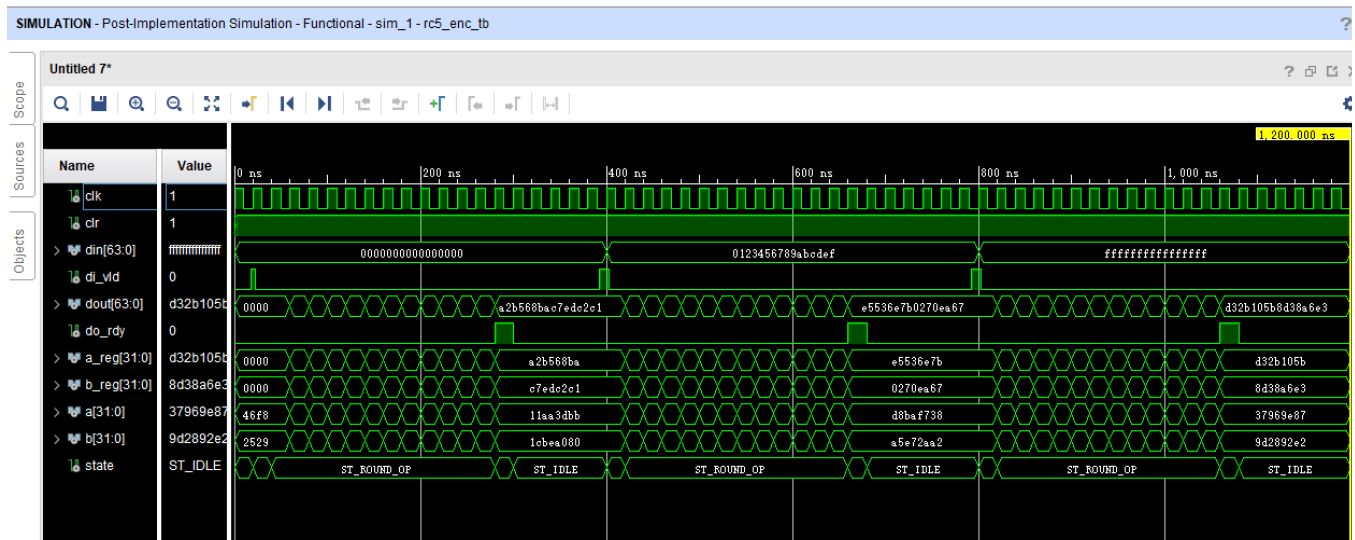
din = x"0123456789abcdef"



din = x"ffffffffffffffff"

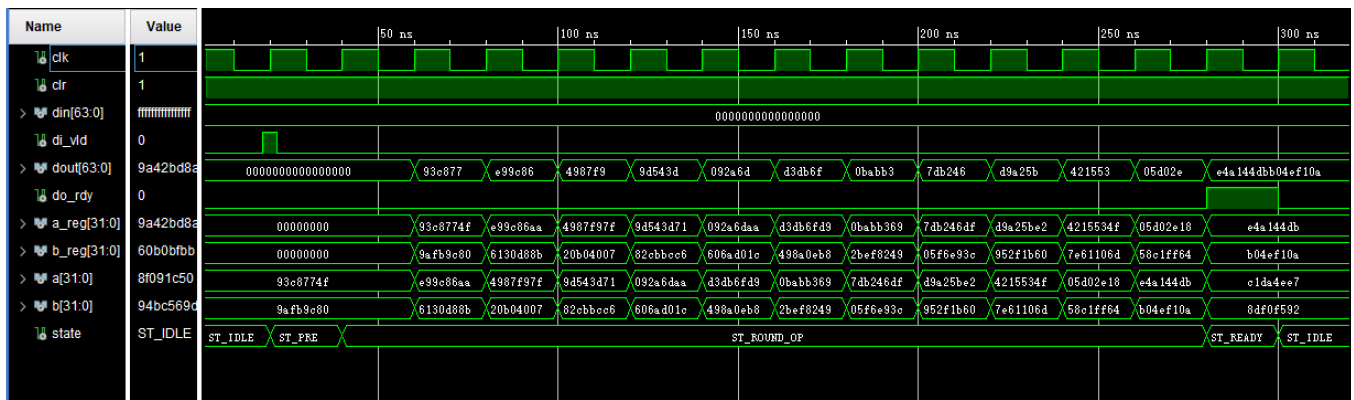


The Whole Screenshot

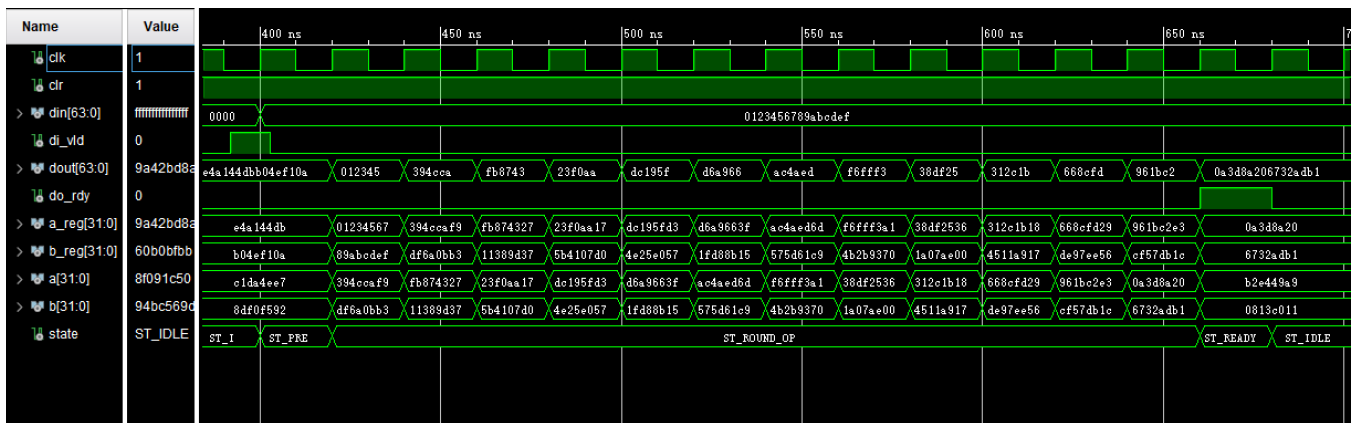


RC5 Decryption

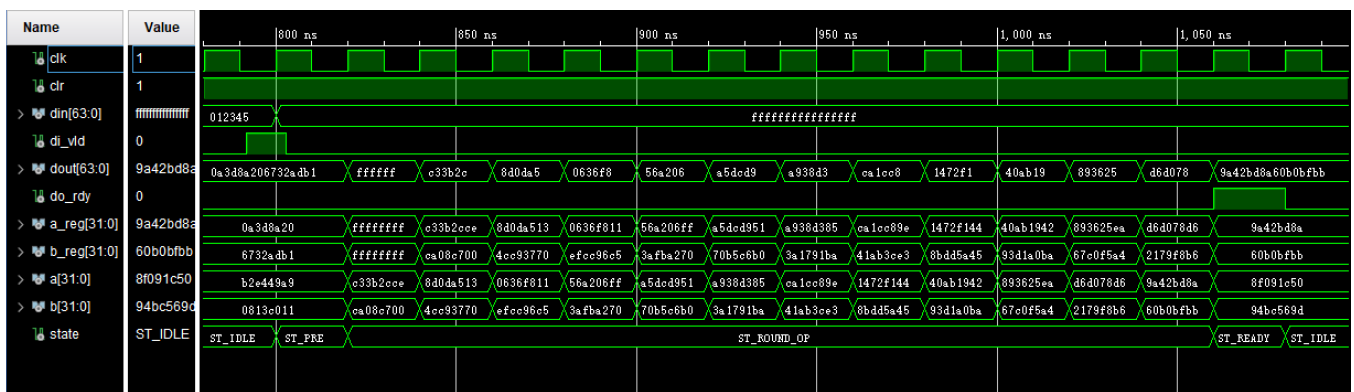
din = x"0000000000000000"



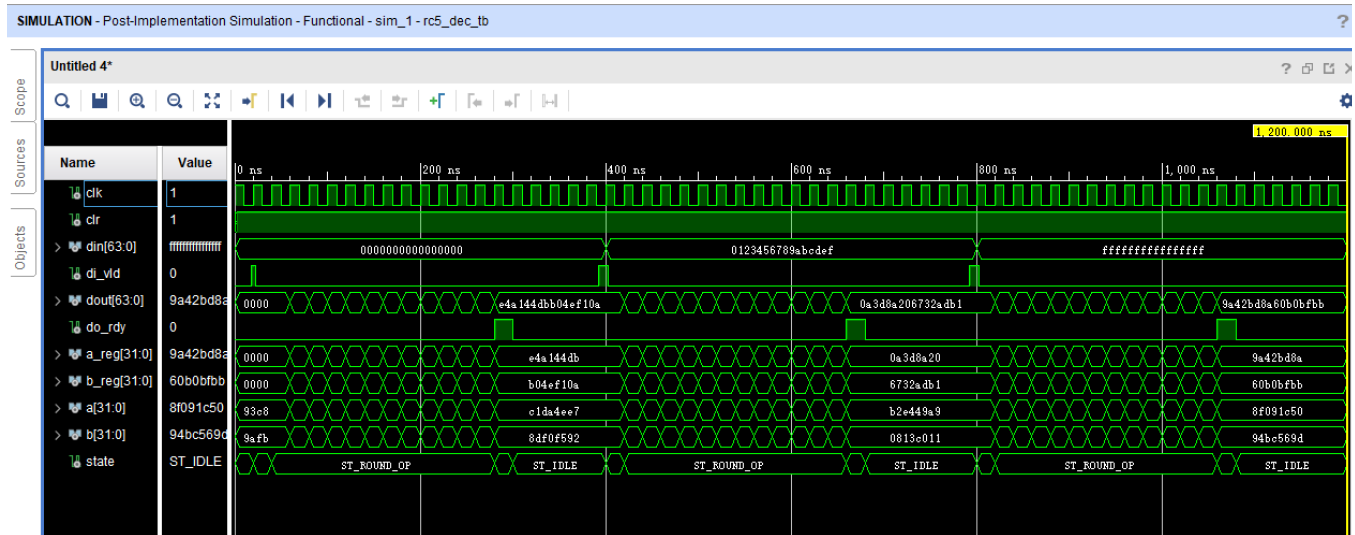
din = x"0123456789abcdef"



din = x"ffffffffffffffff"



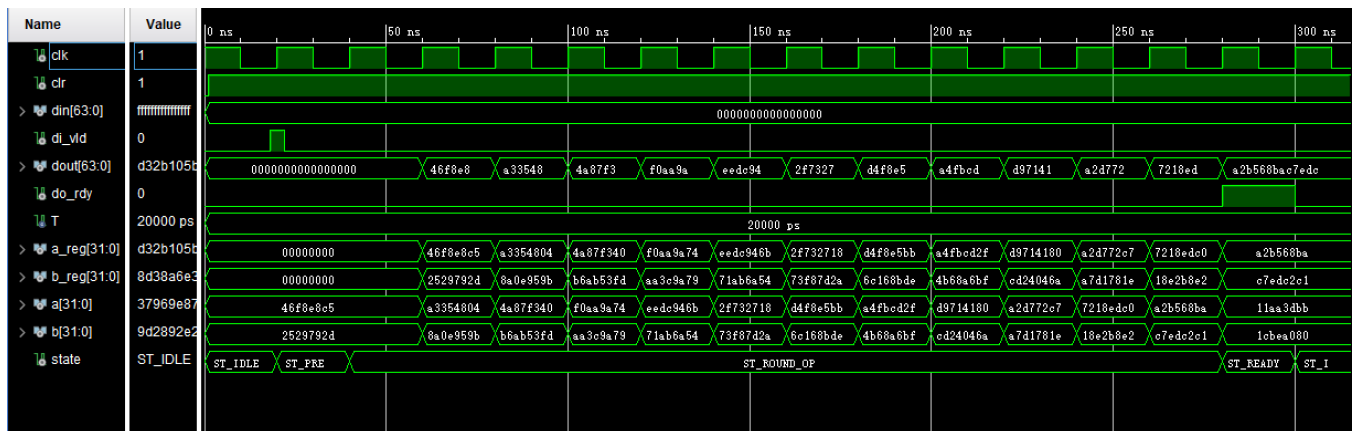
The Whole Screenshot



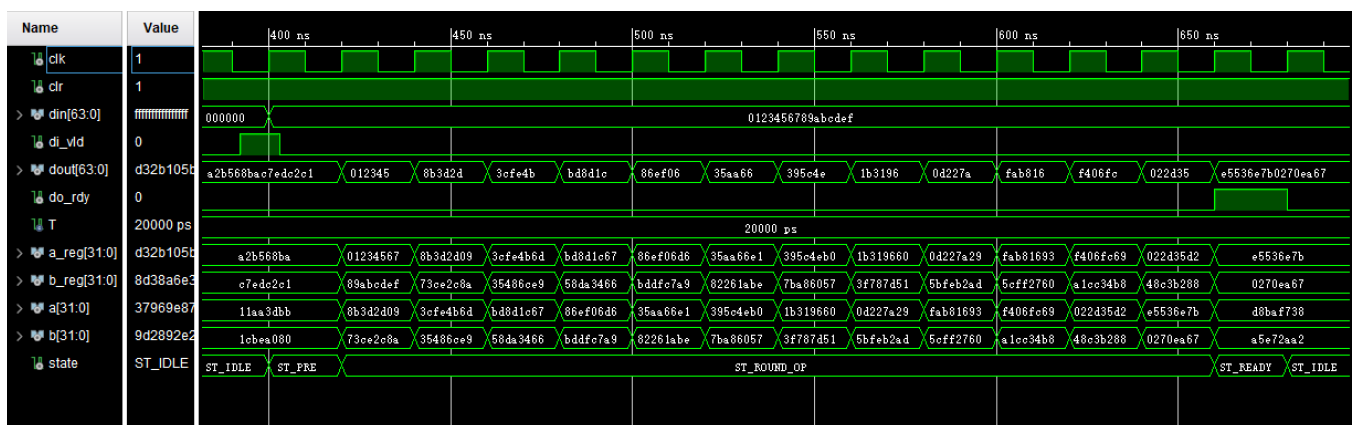
Timing Simulation

RC5 Encryption

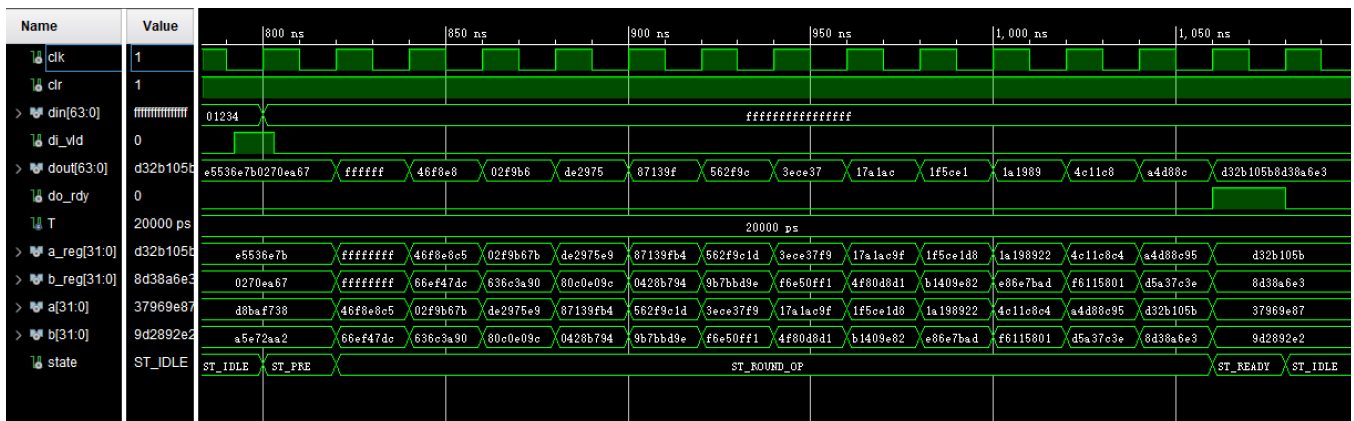
din = x"0000000000000000"



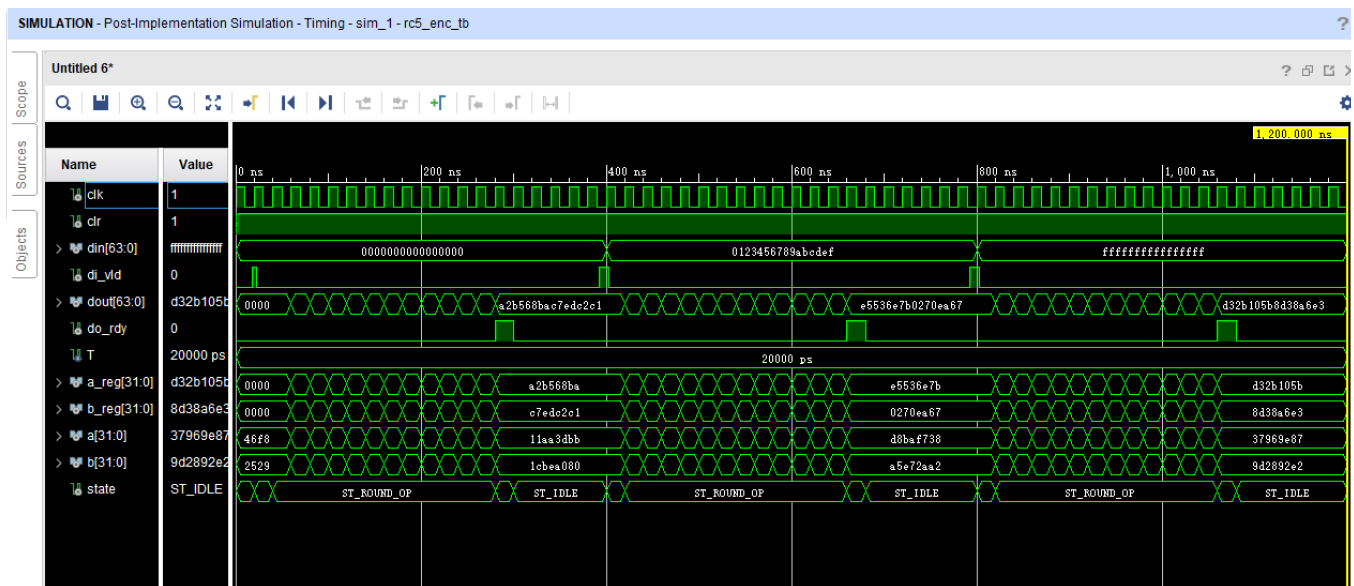
din = x"0123456789abcdef"



din = x"ffffffffffffffff"

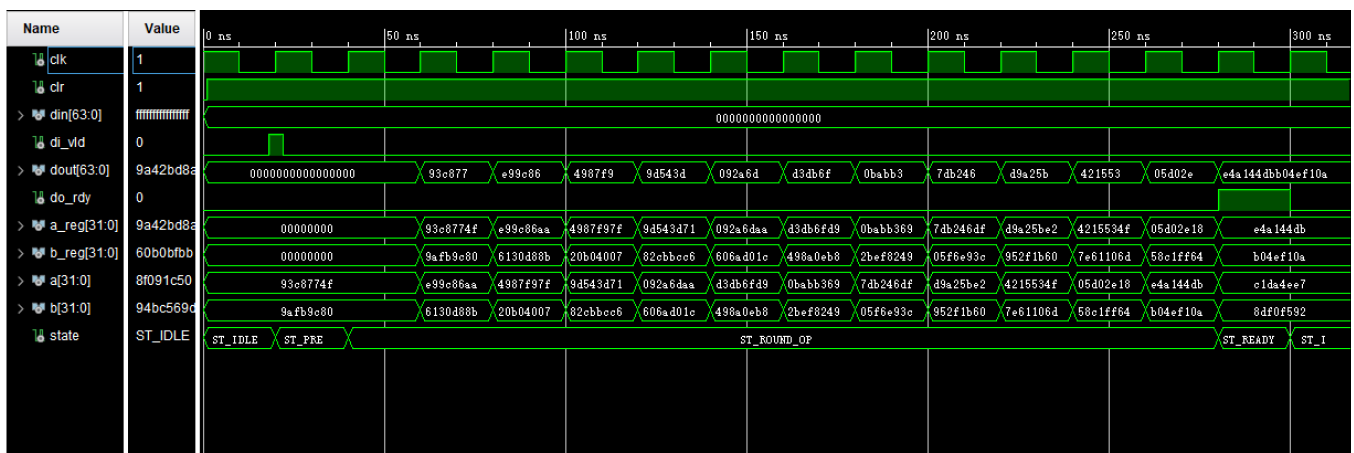


The Whole Screenshot



RC5 Decryption

din = x"0000000000000000"



din = x"0123456789abcdef"

asked for a clock period WNS shorter and it would still be fine. Thus we can answer the minimum period and maximal frequency through this way. (I found this explanation on <http://bilauer.co.il/blog/2017/01/vivado-minimal-period-timing/>)

RC5 Encryption

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.195 ns	Worst Hold Slack (WHS):	0.105 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Falling Endpoints:	0	Number of Falling Endpoints:	0	Number of Falling Endpoints:	0
Total Number of Endpoints	287	Total Number of Endpoints	287	Total Number of Endpoints	156

Minimum period: 12.805 ns

Maximum clock frequency: 78.094 MHz

Latency: 12 clock cycles

RC5 Decryption

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.720 ns	Worst Hold Slack (WHS):	0.177 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Falling Endpoints:	0	Number of Falling Endpoints:	0	Number of Falling Endpoints:	0
Total Number of Endpoints	287	Total Number of Endpoints	287	Total Number of Endpoints	156

Minimum period: 11.280 ns

Maximum clock frequency: 88.652 MHz

Latency: 12 clock cycles

Port Map

Clock Signal

First, I generated a **clock signal** with period 20 ns, and mapped the clock signal c1k (logic bit) in my design to it.

Because the hardware needs time to deal with the data in each clock cycle, if we use a too small period, for example, 10 ns, the time will be not enough to cover all the computations in each clock cycle. So we need set the clock cycle properly.

Buttons

In my design, 3 buttons (reset, up, down and center) are used.

The reset signal c1r (logic bit) is mapped to **CPU reset button**, which serves as an asynchronous reset signal. When clicking on reset button, the signals including input (din), output (dout), registers (a_reg and b_reg) and some intermediate signals (i_cnt, do_rdy, .etc) will be initialized with a certain value (generally 0). Notice that the reset signal is low level effective. That is why it must be mapped to the CPU Reset button.

The signals up_btn (logic bit) and down_btn (logic bit) are mapped to **up button** and **down button**, which are used to modify the value of input vector. When clicking on up button, the displaying 8-bit hexadecimal number will increase by 1. Similarly, when clicking on down button, the displaying 8-bit hexadecimal number will decrease by 1. These two button only work when the input vector is being displayed. In addition, in order to avoid continued increasing/decreasing, I used a buffer signal for each button. By checking the value of button and the corresponding buffer at every clock rising edge, I can decide the rising edge of button signal is within which clock cycle. Thus, the function will be triggered only once at a time. Also, thanks to the high frequency clock signal, the delay cannot be detected by us human beings.

Center button is mapped to the signal di_vld, which is used to tell the system that input value is ready and computation can take place.

Switches

In my design, 10 switches (I/O switch, A/B switch, and 8 switches to decide the modifying bits) are used.

The signal io_sw (logic bit) is mapped to the first right-handed switch (**I/O switch**). This switch is used to switch the display (on 7 segments) between input vector and output vector.

The signal ab_sw (logic bit) is mapped to the second right-handed switch (**A/B switch**). This switch is used to switch the display (on 7 segments) between vector A (32 most significant bits) and vector B (32 least significant bits).

The signal mod_hex (8-bit logic vector) is mapped to 8 switches (**modifying switch**). Each bit of the vector corresponds to a certain segment. When modifying the hexadecimal value with the two buttons mentioned above, only the segments of which the corresponding switch is set to 1 will change.

LEDs

In my design, only one LED is used.

The signal `do_rdy` is mapped to the first left-handed LED. When output is ready, the LED will be on. Or it remains off.

7 Segment Display

In my design, all 7 segment display are used.

The current displaying value are determined by the two switch mentioned above (**I/O switch** and **A/B switch**).

In order to perform a proper function of displaying, I generate a slow clock, comparing with the clock `clk` (20 ns period). This clock signal for 7 segment display has a period of $20 * 2^{16}$ ns and it is used to switch among all the 8 digits. At the rising edge of display clock signal (`disp_clk(15)`), the anode select (signal `seg_sel`) and corresponding value to be displayed (signal `seg_val`) will change. As a result, we can get a suitable refresh rate. With this rate, the 8 digits can be different and no overlapping occurs.

Brief Summary

By modifying the `.xdc` file, we can set the period of clock signal. In this way, each divide by two will use a flip-flop to implement. However, the number of flip-flop which can be used as frequency divider is limited. So the clock period can only be changed within a certain range. If we want a really slow clock, for example, a period of 1 sec. It can not be realized directly by flip-flop hardware. Thus, we need to declare a vector as counter to count the number of rising edges of clock signal.

The most significant difference between buttons and switches is that the switches have two stable states (0 and 1) while the buttons only have one (0). That's why I used buttons as triggers for each function and used switches to represent different states or modes.

There are 8 digits of 7 segment display but only 16 LEDs. As we all know, each digit of 7 segment display can show one bit of hexadecimal number, which takes 4 binary bits. 32 binary bits can be displayed at a time with 8 hexadecimal bits. So I chose to use 7 segment display to show the long vector (64 logic bits). Besides, the LEDs are used as indicators.

For more details, please go over my VHDL codes.

Demo Video

https://youtu.be/Anmj_HO9hy4