EL6463 Advanced Hardware Design

Lab #4

Name: Chen Shen

netID: cs5236

Test Cases

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

din = x"00000000000000000"

din = x"0123456789abcdef"

din = x"fffffffffffffff"

Hand Calculation

Encryption: din = x"0000000000000000"

Before the main loop, we have A = x''000000000'' and B = x''000000000''.

$$din = x'' 0000 0000 0000 0000''$$
 $a = x'' \overline{00000000''} b = x'' 0000 0000''$

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

```
1=2
  ab_{-}xor = a \times or b = x''46F8E8C5'' \times or x''2529792D''
       a = 10000 \text{ ollo 1111 1000 1110 1000 1101 0101}
b = 10000 \text{ olol oclo 10010111 1001 0010 1101}
  ab-xor = "0110 0011 /101 00011001 0001 1111 1000"= x"63 D191 E8"
  a_wt = ab_xor <<<br/>b = x"63D191E8" <<< x"2529792D"
                         = x"630191E8" <<< "01101"
  = x" 63 D191 E8" <<< 13

X" 63 D191 E8" <<< 13

13 MSBs | more to LSBs
  a-rot = "001100100011 11010000 1100 0111 1010" = X"323DOCTA"
  a = a-not + S[4] = x"323DOC7A" + x"70783B8A"
                       = v"A3354804"
 ba_{-}xor = b \times or a = x'' 2529792D'' \times or x'' A3354804''
       a= 11 1010 0011 0011 0101 0100 1000 0000 0100 11
  ba-xor = "10000110 0001 1100 0011 0001 0010 1001"= x' &61(3)29"
  b-rot = ba-xor <<< a = x"861C3129" <<< x"A6354804"
                          = x" 861(3129" <<< "00100"
                           = x"861(3129" <<< 4
X" 861C3129" = "1000,0110000/110000/1000/00/00/
  b-rot = " 01/0 000/ 1100 00/1 000/ 00/0 100/ 1000 " = x"61631298"
  b = b_rot + 5[5] = x"6[c31298"+ x"284B8303" = x"8A0E9393"
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 $ab_{-xor} = a \times corb = x''A3354604'' \times cor x''8A0E95913''$ a_wt = ab_xor <<< b = x"293BDD9F" <<< x"8A0E959B" = x">13BD9F" <<< "11011" a-rot = "11110010100100100110111011101100" = x"F949DEEC" a = a - wt + S[6] = x"F949VEE(" + x"513Z!454"= x" 4A&77540" $ba_{-}xor = b \times or a = x'' 8AOE959B'' \times or x'' 4A87F340''$ p = " | 1000 | 1010 | 100 | 1010 | 100 | 1011 " a = 11 0(00 (0)0 | 000 | 0|1 | 1110 0010 0000 11 b-rot = ba-xor <<< a = x"C08766DB" <<< x"40877540" = x" cos9660B" <<< "00000" = x" co89660B " <<< 0 = X" COST 66DB " b = b_rot + 5[7] = x"Coff 66DB" + x"F621ED22" = x"B6AB53FD" a = x'' 44877540'' b = x'' 86/43537D''

Decryption: din = x"0000000000000000"

Before the main loop, we have A = x''000000000'' and B = x''000000000''.

$$din = x''000000000000000''$$
 $a = x''000000000'' = b = x''00000000''$

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

1=12 b_minu = b - 5 [25] = x"00000000" - x"65046380" = x"9AFB9c80" b_wt=b, minus >>> a = x "9AFB9C80" >>> x "00000000" - X'GAIRG(80" b = b_wt xx q=x" 9AFB9C80" Xor x"00000000" = x"9AFB9c80" a_minus= a- s[24] = x"00000000 - x''[6001431" = x"0933 TBCF" a_rot = a-minus >>> b = x"0983 EBCF" >>> x"9AFB9C80" - X'0933 EBCF" >>> " 60000" = x" 0933EBCF" a= a-rut xor b = x"0933 EBCF" xor x"9A789C80" a-vot = "0000 (00) 0011 1110 1011 1100 1111" P = 1,1001 1010 1111 1011 1001 1100 1000 02001. $\alpha = 1001 0011 1100 1000 0111 0111 0100 1111''$ - x" 9368774F"

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

1=11 b_minus = b- S[26] = x"9AFB9C80"- x"43192304" = x"67E2797C" b-rst = b-minus >>> a = x"57E2797C" >>> x"93C8774F" = x"57227976" ->> " 0 1111" = X"57217976" >>> 15 x"5772717C" = "0101 0111 1110 0010 0/11 100 0111 1100" L_ret = 1111 0010 1111 1000 1010 111 1110 0100" = x" FZT-8AFE4" 6=6- bot xor a = x" T2 T8 A FE4" xor x"95 C877 4F" h yst = 1111 00/0 1111 1000 (010 1111 1100 0/00" a = "1001 0011 1100 1000 0111 0111 0100 1111" a-minus = a - s(22) = x"93687747" - x"30076800" =x"62718645" a-not = a-minus >>> 6 = x"62F10 C45" >>> x"6160D888" = x" 62710C45" >>> 110101" - x" 62F10C45" >>> 11 X"627-10C45"= 116/10 00/0 /11000/0000 1/1000/00 0/0/" a_not = 1' 10001000 [0[0]1100 0[0] 1110 00[0 000]" = x'' 88AC5221"

a= a-rot xor b = x"88ACSZU" xor x"613.0888" a-rot = 1/10001000 101011100 0/01 1110 00/0 0001"

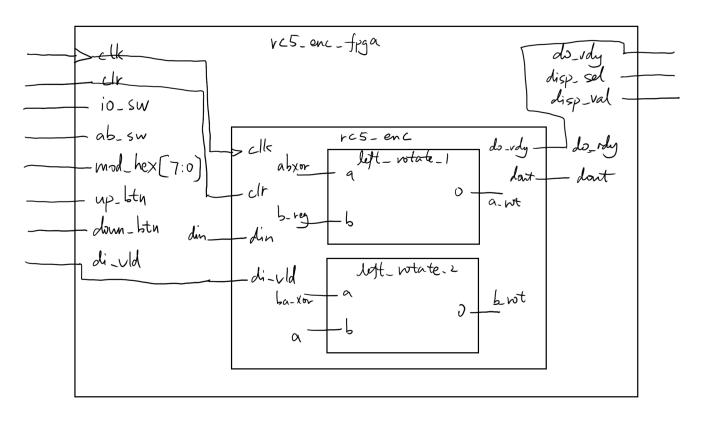
a=x"E99C86AA" b=x"6130D88B"

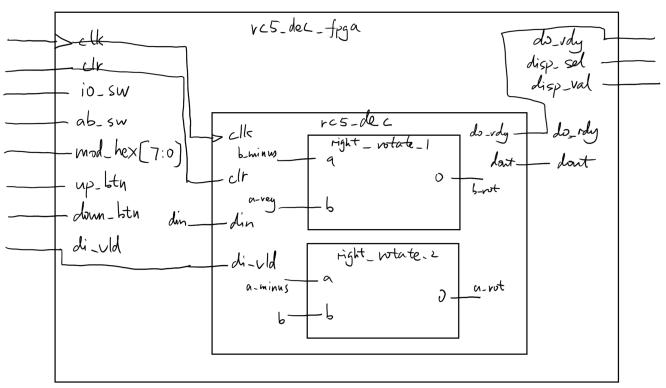
In the third loop, after the first line, we have A = x"4987f97f". After the second line, we have B = x''x"20b04007".

```
1=10
  b. minus = b- S[21] = x"6130D888" - x"AE162167" = x" B31 A B724"
  b-rot = b-minus >>7 a = x"1331AB724" >>> x" E99C86AA"
                       =x"B31AB724" >>> "101010"
                        = x'1831A18724" >>> 10
     X'B31AB719" = "1011 00/1 0011 1010 1011 01/1 0010 0100 1''

move to MSBs
   b= b-nut xor a = x" c92 c c640" xor x" E99(864A"
      p- rot = " 1100 100 100 100 0110 1010 1101"
      V= 11 [110 1001 1001 1000 0000 0110 1010 1010 "
   b = " 0010 0000 | 011 0000 0100 0000 0111 = x"2080 4007"
   a. minus = a- S[20] = x" Eggc 86AA" - x"4DBF CA76" = x"9BDC BC34"
   a-vot = a-minus >7> b = 8" 9BDC BC34" >>> x" 20606007"
                          =x"9BDCBC34" >> "00111"
                          =x"9BDCBC34" >>> 7
         X''9BDCBCS4' = [100 | 101 | 100 | 101 | 100 | 011 0100;" 7 LSA.
   arot= "0110 (00) 0011 0111 /011 100 0111 1000"=x"69378978"
    a= a-not xor b= x"693713978" xor x"20804007"
      arof = 1,0110 (00 0011 0111 /011 12) 0/11 (020),
      p = 1,0010 0000 1011 0000 0100 0000 0100 0111,
    α= " 0100 1001 1000 0111 1111 (001 0111 1111"=x"4987F97F"
    a=x"4987 F97F" 6 = x"20B04007"
```

Block Diagram





Resource Utilization

	Synthesis stage	Place and Route stage
LUT and FF pairs usage	510 LUTs and 230 FFs	186
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	390 LUTs and 230 FFs	131
IOB usage	32	32
RAM/DSP blocks used (if any)	0	0

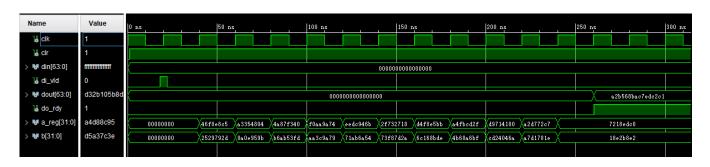
Simulation

Functional Simulation

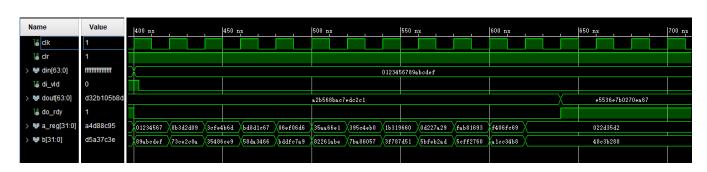
*In post-implementation simulation, the a_reg, b_reg, a and b are compiled to a lot of wires. So only some of these signals can be added to wave window.

RC5 Encryption

din = x"0000000000000000000



din = x"0123456789abcdef"



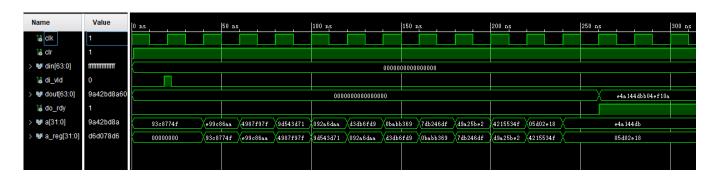
^{*}Notice that all the simulations use the same clock peroid - 20 ns

din = x"ffffffffffffff"

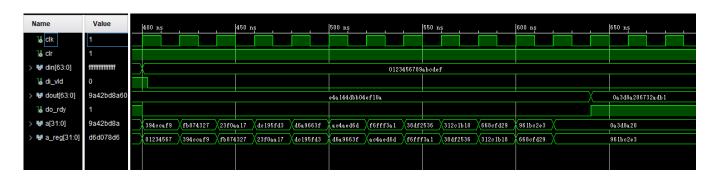
Name	Value	8	00 ns			850 ns				9	00 ns					950 ns				1, 000 1	ns .		1,	050 ns
l ä clk	1																							
l⊌ cir	1																							
> 😽 din[63:0]	fillillillilli	Ж										fff	ffffff	ffffff	f									
l⊌ di_vld	0																							
> 😻 dout[63:0]	d32b105b8d										e5536e7	ъ0270	ea67									√ d3	2Ъ1051	b8d
l⊌ do_rdy	1																							
> V a_reg[31:0]	a4d88c95	Х	ffffffff	46f8e8c5	02f9t	67Ъ	de2975	e9 X	87139fb	4	562f9c1	d X	3ece37:	f9	17a 1a	c9f	1f5ce1	18	1a 198922	40110	8c4	a4d88c	15	
> 😻 b[31:0]	d5a37c3e	Ж	ffffffff	66ef47dc	636c3	8a90	80c0e0:	9c X	0428Ъ79	4	9b7bbd9	• \	f6e50f:	f1	4f80d	8d1	Ъ1409е	32	e86e7bad	f6115	801	d5a37c	ie	
			<u> </u>	<u> </u>					The state of the s										The state of the s			The state of the s		

RC5 Decryption

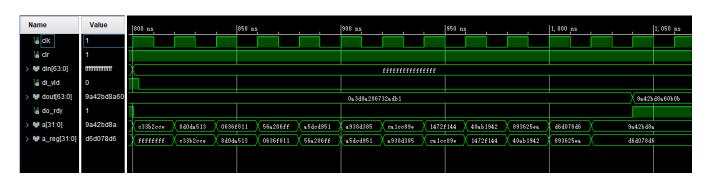
din = x"00000000000000000"



din = x"0123456789abcdef"



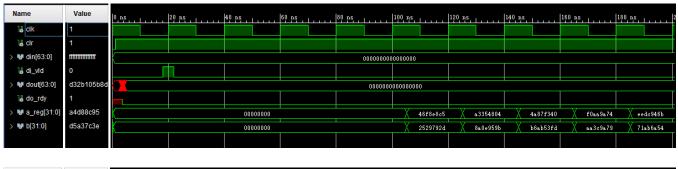
din = x"ffffffffffffff"



Timing Simulation

RC5 Encryption

din = x"00000000000000000"



Name	Value		2	220 ns	1	240 ns		260 ns	·	280 ns		300 ns. 1	 320 ns.	 340 ns		360 ns.	38	30 ns
ไå clk	1																	
1å clr	1																	
> 😽 din[63:0]	ffffffffffff										0000000000	0000000						
lå di_vld	0																	
> 🐶 dout[63:0]	d32b105b8d						000000	000000	0000						a2b56	1 8bac7edc2c1		
¹å do_rdy	1																	
> W a_reg[31:0]	a4d88c95		732718	Χ	d4f8e5bb	Χ	a4fbcd2f	Χ	d9714180	Χ	a2d772c7	Χ		72	18edc0			
> ₩ b[31:0]	d5a37c3e	73:	f87d2a	Χ	6c168bde	Χ	4b68a6bf	Х	cd24046a	Χ	a7d1781e	Х		18	e2b8e2			

din = x"0123456789abcdef"



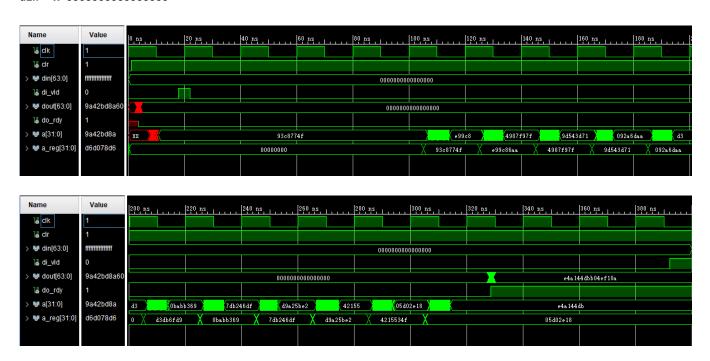
Name	Value	600 n	is. I	62	0 ns	640 ns	 660 ns	 680 ns	1	700 ns		720 ns		740 ns	 760 ns. 1	1	780 ns. 1	
ไ å clk	1																	
lå cir	1																	
> W din[63:0]	ffffffffffffff							01	23456789	abcdef								
lå di_vld	0																	
> W dout[63:0]	d32b105b8d		a2b5	68bac	7edc2c1						f1148	dc74dad4	4 e					
do_rdy	1																	
> W a_reg[31:0]	a4d88c95	18	265e0d	c3	Χ					a8:	f49383							
> W b[31:0]	d5a37c3e	86 X	bldfdf	44	χ					0e	73893c							

din = x"ffffffffffffff"

Name	Value	800	ns	820 ns		840 ns		860 ns	880 ns	900 ns		920 ns	1	940 ns	1	960 ns		980 ns
ไ ⊌ clk	1																	
ไ₀ clr	1																	
> W din[63:0]	ffffffffffff								fffffffff	fffffff								
lå di_vld	0																	
> W dout[63:0]	d32b105b8d								f11485dc74d	a d44 e								
do_rdy	1																	
> ₩ a_reg[31:0]	a4d88c95	a8f	ffffffff	ΪХ	46f8e8c5	Χ	02f9b67b	de2975e9	87139fb4	X 5	62f9c1d	X	lece37f9	Χ	17a1ac9f	Х	1f5ce1d8	X 1a 1989:
> ₩ b[31:0]	d5a37c3e	0c7	ffffffff	ΪX	66ef47dc	Χ	636c3a90	X 80c0e09c	0428Ъ794		b7bbd9e	X	6e50ff1	Х	4f80d8d1	Х	Ъ1409е82	Xe86e7ba
Name	Value	l1 00	n ne	11 020	D.F.	li nan -	25	11 060 pc	11 080 pe	li 100 pe		li 120 m		li 140 m	\r.	li 160	7.5	li 180 pe
	Value	1,00) ns	1, 020	ns	1,040	ns	1,060 ns	1,080 ns	1, 100 ns		1, 120 no		1, 140 n	15	1, 160	R5	1, 180 ns
Name		1,00) ns	1, 020	ns 	1,040	ns	1,060 ns	1,080 ns	1, 100 ns		1, 120 p.		1, 140 n	is	1, 160	ns .	1, 180 ns
la clk la clr	1	1,00	D ns	1, 020	ns	1,040	ns	1,060 ns	1,080 ns			1, 120 n		1, 140 n	is	1, 160	ns .	1, 180 ns
la clk la clr	1	1,00) ns	1, 020	ns	1,040	ns I	1,060 ps			<u> </u>	1, 120 n		1, 140 p	15	1, 160	ns	1, 180 ns
18 clk 18 clr 18 din[63:0] 18 di_vld	1	1,00	ns,			1, 040	ns L	1,060 ns				1, 120 n.		1, 140 n	15	1, 160	ns ,	1, 180 ns
18 clk 18 clr 18 din[63:0] 18 di_vld	1	1,00				1, 040	ns N	1,060 ps						1, 140 n	15	1, 160	NS .	1, 180 ns
16 ctr ■ din[63:0] 16 di_vld ■ dout[63:0]	1	1,00				1,040	ms X	1,060 ps		fffff				1, 140, n	15	1, 160	ns	1, 180 ns
lå clk lå clr lå di_(63:0] lå di_vld lå dout(63:0] lå do_rdy lå d_reg[31:0]	1 1 1 1 0 d32b105b8d 1		f11485d			1, 040	ns X	1,060 ps		e44d	d32b10			1, 140 r	15	1, 160	ns ,	1, 180 ns
lå clk lå clr voldin[63:0] lå di_vid voldin[63:0] lå do_rdy	1 milliminiiii 0 d32b105b8d 1 a4d88c95	1a19	f11485d			1, 040	ns X	1,060 ps		e44d	d32b10 38c95			1, 140 n	15	1, 160	ns	I, 180 ns

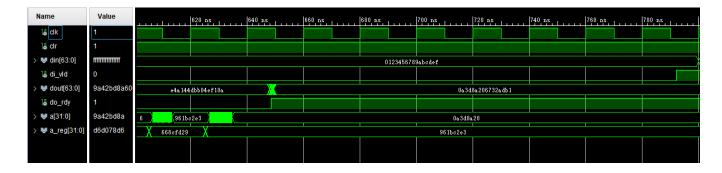
RC5 Decryption

din = x"0000000000000000"

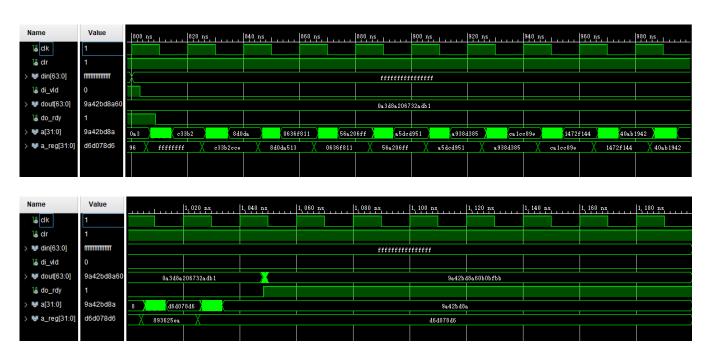


din = x"0123456789abcdef"





din = x"ffffffffffffff"



Speed of the Design

*Remember that in Vivado, only WNS (Worst Negative Slack) is provided in the report, which represents the max delay path (required time - arrival time). In my design, I used a clock signal having a period of 20 ns. So the requirement was 20 ns, and the slack is WNS (note that it is positive), which means that we could have 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):	6.516 ns	Worst Hold Slack (WHS):	0.210 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

Setup		Hold		Pulse Width	
Number of Falling Endpoints:	0	Number of Falling Endpoints:	0	Number of Falling Endpoints:	0
Total Number of Endpoints	412	Total Number of Endpoints	412	Total Number of Endpoints	219

Minimum period: 13.484 ns

Maximum clock frequency: 74.162 MHz

Latency: 12 clock cycles

RC5 Decryption

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	9.005 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	413	Total Number of Endpoints	413	Total Number of Endpoints	219

Minimum period: 10.995 ns

Maximum clock frequency: 90.950 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 clr (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 indecators.

For more details, please go over my VHDL codes.

Demo Video

https://youtu.be/oqQJlu5xBXE