Table of Contents

1） Introdution

a. This lab mainly foucus on building 4-bit and 8-bit mutiple function processor both using discrete logical circuit and FPGA board. The circuit is capable of complete the function of AND OR XOR and the revert of them. The discrete circuit is able to perform at 4-bit level.

2） Operation of the logic processor

a. Adjust the Din switch to the value we want register A to load then turn on load A. Turn off load A, then adjust the Din switch to the value we want register B to load, turn on load B, then turn off load B.

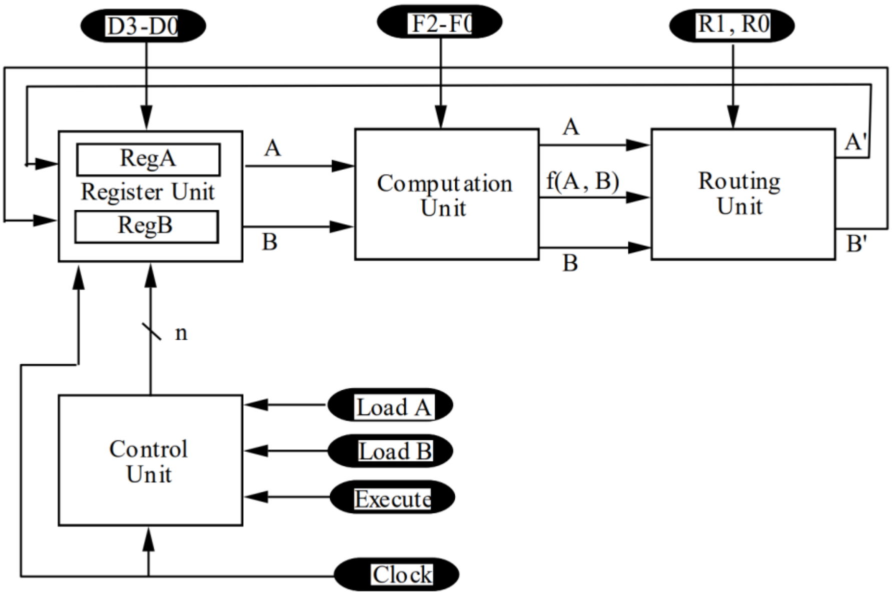
b. Adjust F and R to choose what funtion and how the output put into register A and register B, and turn on execute to run, then turn off execute whenever you like.

3） Written description, block diagram and state machine diagram of logic processor.

a. (1) register unit contains two 4-bit shift registers, they can take both load A, load B and S signal to either parallel loading or right shift one bit of high or low signal based on S signal. (2) computation unit takes two 1-bit signal to perform AND OR XOR NAND NOR XNOR 0 1 functions, the selection among all the functions is

decided by the 3-bit signal F.（3） routing unit takes 3 bit of signal coming from A, B and the computation unit, then based on the 2-bit signal R, the routing unit will send 2 bits back, and the selection and order are decided by R.（4）control unit takes signal E（execute）and S（shift） signal will be high for 4 clock cycle for its inner design.

b.

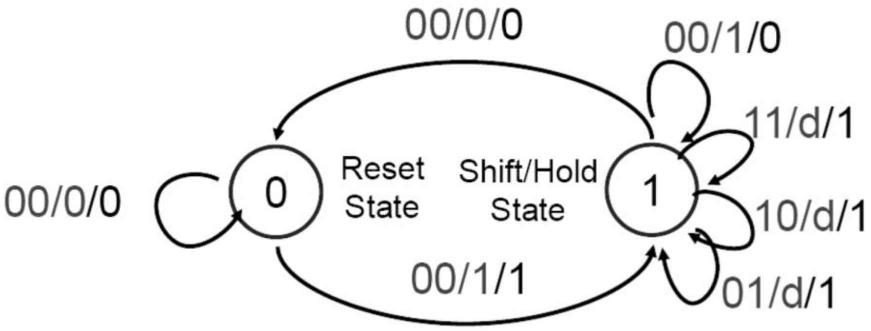


(the block diagram in lab manual basically represent my implementation)

c. State Machine Diagram

(i) Mealy Machine

(ii) (xx/x/x):（C1C0/Q/S）



4) Design steps taken and detailed circuit schematic diagram.
a. Written procedure of the design steps taken.

i. If you used kmaps or truth tables during design, include them here. Kmaps are usually helpful for creating the next state logic in the control unit. Additionally, you should show any transformations required to ‘map’ the logic expressions to the discrete logic chips in your kit (e.g., you do not have AND or OR chips – so how did you modify the logic to use NAND/NOR expressions?)

S-kmap(S*=c1+c2+EQ’)

EQ\C1C0	00	01	11	10
00	0	d	d	d
01	0	1	1	1
11	0	1	1	1
10	1	d	d	d

Q-kmap(Q*=c1+c0+E)

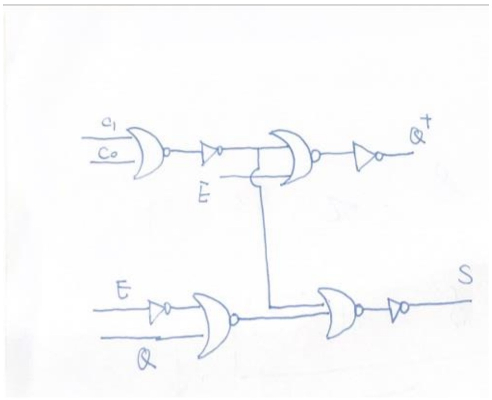
EQ\C1C0	00	01	11	10
00	0	d	d	d
01	0	1	1	1
11	1	1	1	1
10	1	d	d	d

We don’t need a inverter to transfer Q because in the flip-flop, we have the inverse output of Q, also since both Q and S need (c1+c0), which could bound together as one output (c1+c0)=((c1+c0)’)’=invert(nor(c1,c0))

While ((c1+c0)+E) could also be transfered into inverter and nor gate

For S signal we can directly use the (c1+c0) signal and the final step is to get EQ’, (EQ’)’=(E’+Q) and or gate can be easily appied by inverter and nor.

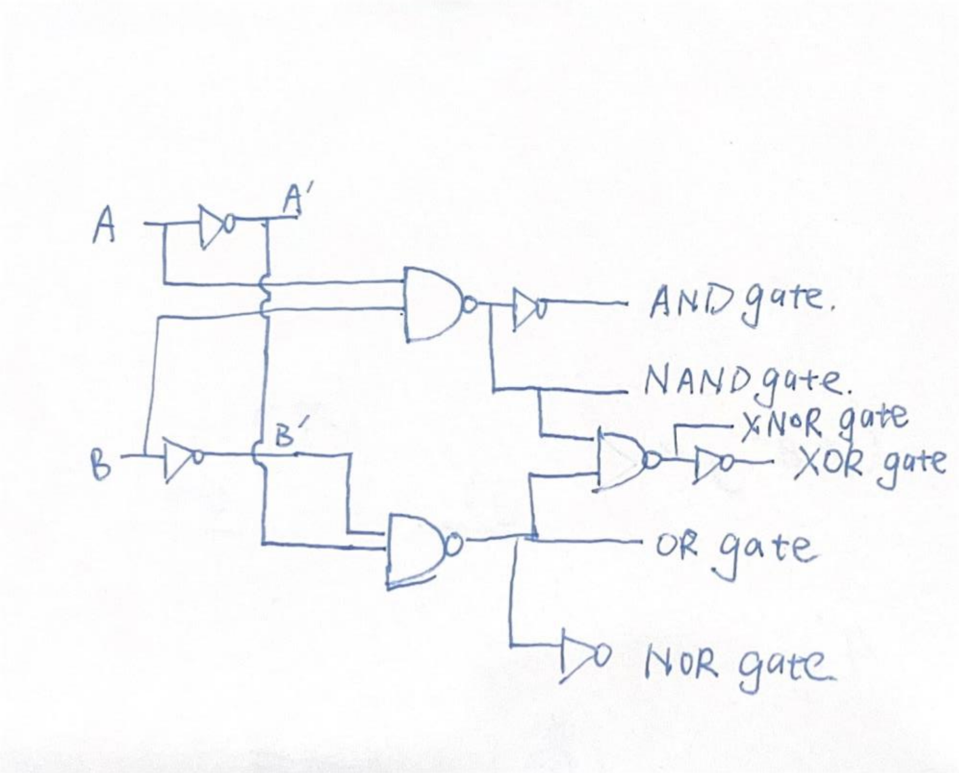
As result the design of Q and S is as following



Also for the calputation part, we also need to use nand and nor to finish the implementation:

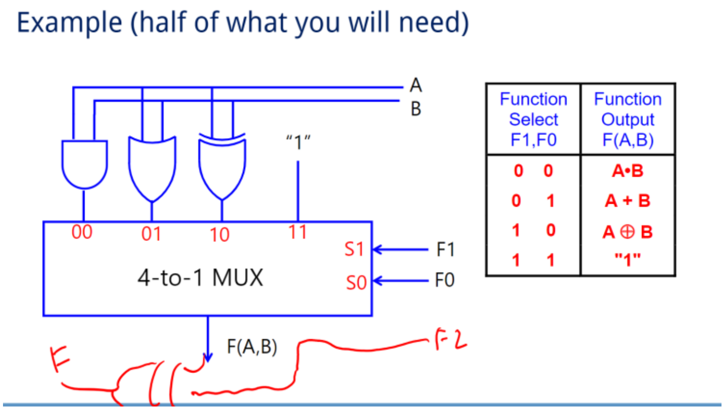
Since A’ and B’ might be used in the computation, we first use two inverters to generate

A’ and B’. (A+B) can be achieved by (A’B’)’ which is nand(A’,B’), at the same time we can get A NOR B, similarly, if we use NOT(A NAND B), we can get A NAND B , A and B at the same time. For A XOR B
A XOR B = AB’+BA’ and we add redundant term AB’+BA’+AA’+BB’=(A+B)(A’+B’)
We already find A+B and A’+B’ at the same time we have XNOR gate



ii. Written description of the design considerations taken (did you consider multiple implementations of the same circuit and the tradeoffs of each?)

We can also use the XOR gate so that we can only calculate half of the computation:


Example (half of what you will need)

Function Select F1,F0		Function Output F(A,B)
0	0	A•B
0	1	A + B
1	0	A ⊕ B
1	1	"1"

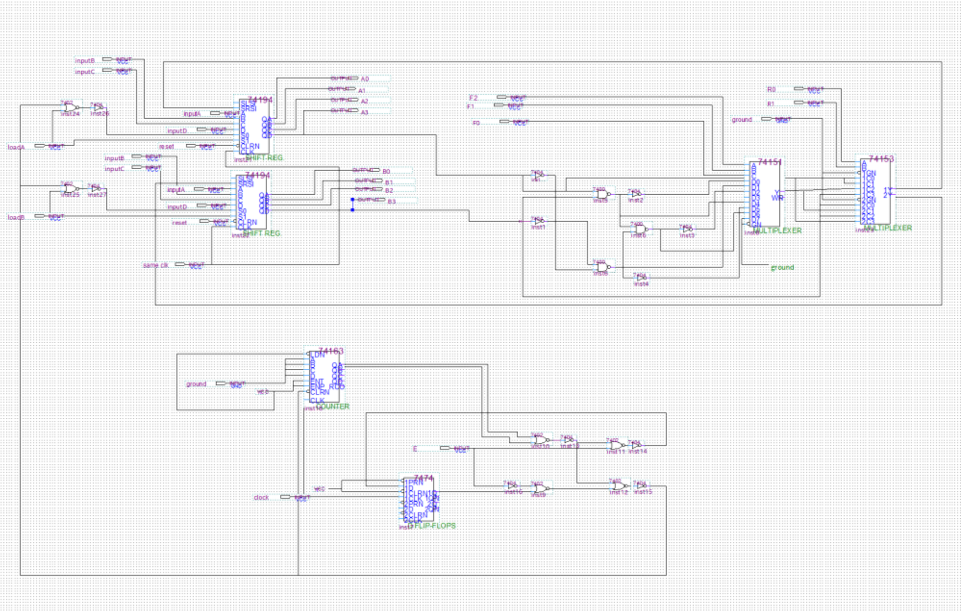
However this method can't fully take advantage of what we have already calculated. Also, to develop a XOR gate by using NAND and NOR will also occupy a lot of area on the breadboard.

b. Detailed Circuit Schematic

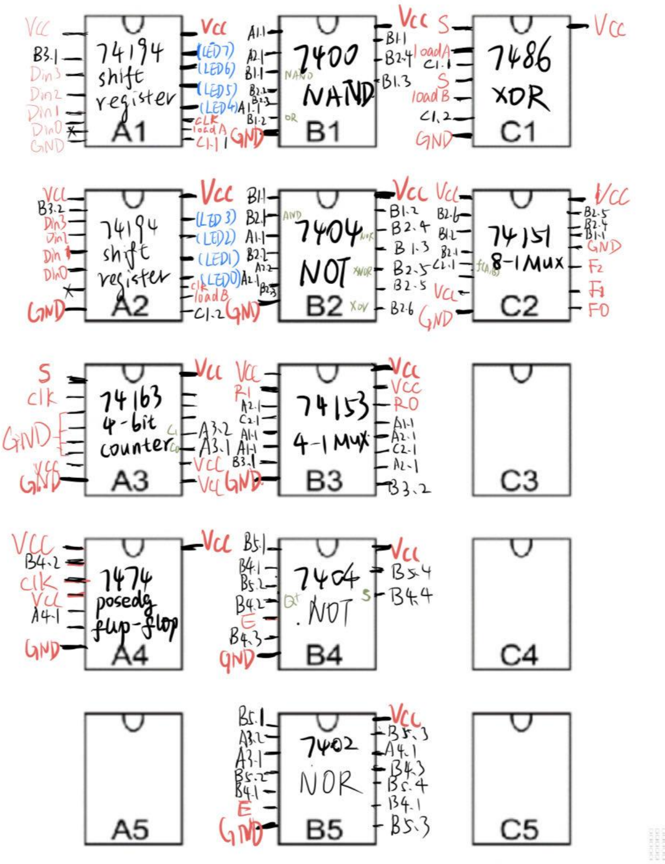
i. Draw a gate level schematic of your circuit. It is OK to use small standard blocks like MUXes, flipflops, and shift registers, but custom blocks like your control unit must have a gate level schematic.

ii. If the schematic becomes too large, components such as the control unit can be represented as black boxes on the top-level schematic, and a detailed schematic of that component can be included below.

iii. You must use a CAD tool (e.g., Fritzing, KiCAD, EAGLE, Quartus) for this portion. Note that Quartus has the standard 7400 parts in the schematic editor. Fritzing parts may be downloaded from the course website.



(5) Breadboard view / Layout sheet



74194
shift register
A1
VCC
B3.1
Din3
Din2
Din1
Din0
GND
Vcc
(LED7)
(LED6)
(LED5)
(LED4)
CLK
load A
C1.1
7400
NAND
B1
A1.1
A2.1
B1.1
B2.2
B2.3
A1.1
B1.2
GND
NAND
OR
Vcc
B1.1
B2.4
B1.3
7486
XOR
C1
S
load A
C1.1
S
load B
C1.2
GND
Vcc
74194
shift register
A2
VCC
B3.2
Din3
Din2
Din1
Din0
GND
Vcc
(LED3)
(LED2)
(LED1)
(LED0)
clk
load B
C1.2
7404
NOT
B2
B1.1
B2.1
A1.1
B2.2
A2.2
A2.1
B2.3
GND
AND
NOR
XNOR
XOR
Vcc
B1.2
B2.4
B1.3
B2.5
B2.5
B2.6
74151
8-1 Mux
C2
Vcc
B2.6
B1.2
B2.1
C1.1
VCC
GND
f(A,B)
Vcc
B2.5
B2.4
B1.1
GND
F2
F1
F0
74163
4-bit
counter
A3
S
clk
GND
VCC
GND
Vcc
A3.2
A3.1
VCC
Vcc
C1
C0
74153
4-1 Mux
B3
VCC
R1
A2.1
C2.1
A1.1
A1.1
B3.1
GND
Vcc
VCC
R0
A1.1
A2.1
C2.1
A2.1
B3.2
C3
7474
posedg
flip-flop
A4
VCC
B4.2
clk
VCC
A4.1
GND
Vcc
7404
NOT
B4
B5.1
B4.1
B5.2
B4.2
E
B4.3
GND
Q+
S
Vcc
B5.4
B4.4
C4
A5
7402
NOR
B5
B5.1
A3.2
A3.1
B5.2
B4.1
E
GND
Vcc
B5.3
A4.1
B4.3
B5.4
B4.1
B5.3
C5

(note : S is B4.4)

6) 8-bit logic processor on FPGA

a. Summary of all .SV modules and the changes you made to extend the processor to 8-bits. Specifically, you need to describe even modules which were provided, but you did not create.

module register_unit: is an 4or8-bit register module

It stores data (4 or 8 bits) based on control signals.

Clk is the clock, Reset resets to 0 on the clock's rising edge.

Load loads data D, Shift_En shifts data, and Shift_In is the data to be shifted.

Data_Out is the register's content, and Shift_Out is LSB.

module compute:

This module is a 1-bit ALU, performing different logic operations based on the control input F and storing the result in F_A_B

module router:

It takes three inputs and uses a control signal R to decide where to send the data, It has two lanes for data, A_In and B_In. It also gets data from another source called F_A_B.

R is a signal tells the router where to send the data.

module control:

It uses a clock signal and control inputs like Reset, LoadA, LoadB, and Execute whihc has 8 possible states (B, C, D, E, F, G, H, I,) in a state machine.

On each clock's rising edge, it updates the current state based on the reset condition or the next state.

State transitions depend on the Execute signal.

In state A, it responds to LoadA and LoadB, disabling Shift_En.

In state J, it disables Ld_A, Ld_B, and Shift_En.

In other states, it disables Ld_A and Ld_B while enabling Shift_En.

change I made: Bit Width of State Enum from [2:0] to [3:0], State Transition Conditions changed from case(A) and case(F) to case(A) and case(J)

module HexDriver: controls four 7-segment displays to show hexadecimal digits

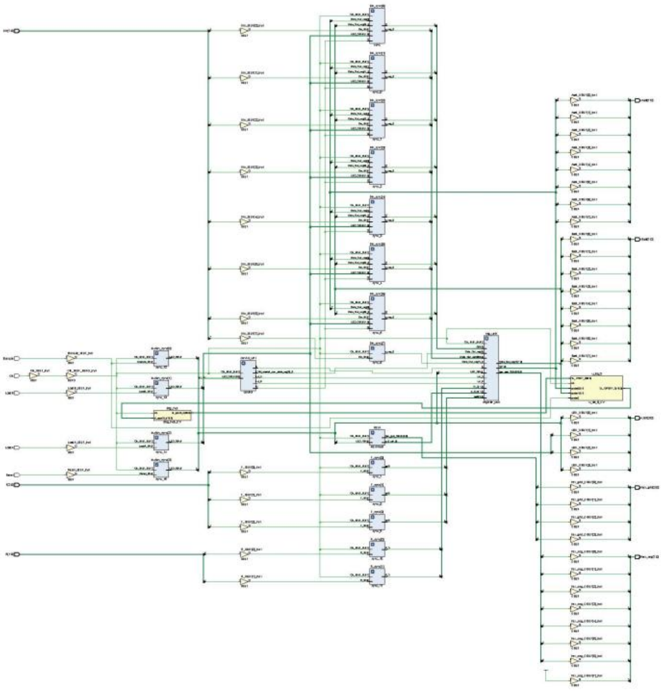
module processor: is a top-level module for a 4-bit logic processor. It includes inputs for clock, control signals, data, and selection, as well as outputs for debugging and display control. It integrates all the sub-modules.

change I made: Bit Width of Din from [3:0] to [7:0], output logic from [3:0] to [7:0] Aval Bval.

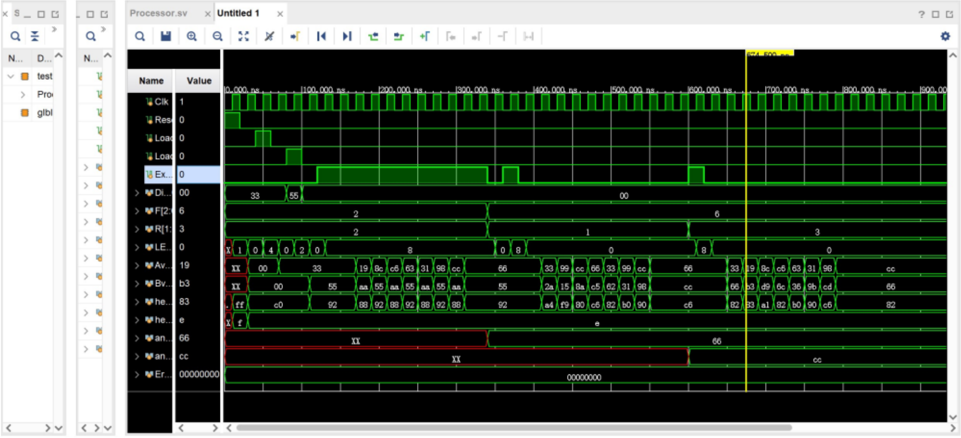
logic A, B, Din_S from [3:0] to [7:0]

For Hex driver:input change from .in({4'h0, 4'h0, B[3:0], A[3:0]}) to .in({B[7:4],B[3:0],A[7:4],A[3:0]}), Change form sync Din_sync[3:0] (Clk, Din, Din_S) to Din_sync[7:0] (Clk, Din, Din_S)

b. RTL block diagram - please only include the top-level design if using the RTL viewer.



c. Include a simulation of the processor that has notes (annotations) that give information such as what operation is being performed, where the result was stored, etc.



For Din, the value is the input, with the chage of LoadA and LoadB, we can load Aval and Bval, after the computation, the value is stored in Aval which is 66(8'h33 ^ 8'h55)(after 8 shifts)also for Bval, it shift 8 times and remain still.

The second computation is processed with A=66 B=55. The value of B is chaged into ~(8h’66 ^ 8'h55)=cc after the 8 shifts also for A value, it shift 8 times and remain still.

The third computation is processed with Aval and Bval are expected to swap, after 8 shift, the value is exchangeed.

d. Include procedure used to generate Vivado Debug Core trace, as well as the result of such trace executing an example operation. E.g. “Step 1, Set the trigger on signal <signal here>…”

Step1:open synthesis design and click “set up debug”

Step2: nets to debug: add nets to debug (ex Aval OBUF) and choose the probe type

Step3: choose the sample of data depth

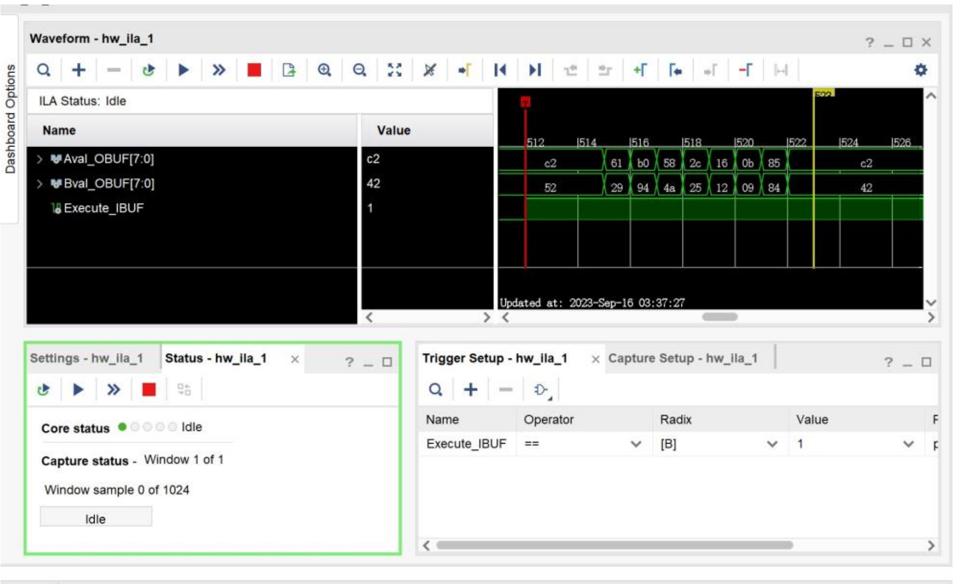
Step4: rerun the implementation design and click generate bit stream

Step5:auto connect and open target

Step6:Set up trigger condition

Step7:run the debug core and move the switch on board to trigger the debug core

e. Include the output of the debug core trace performing an operation on the 8-bit logic processor (this doesn’t need to be the same operation as the one your TA asked to demonstrate, but it needs to be non-trivial).



Waveform - hw_ila_1

ILA Status: Idle

Name	Value
Aval_OBUF[7:0]	c2
Bval_OBUF[7:0]	42
Execute_IBUF	1

Updated at: 2023-Sep-16 03:37:27

Settings - hw_ila_1 | Status - hw_ila_1

Core status ● Idle

Capture status - Window 1 of 1

Window sample 0 of 1024

Idle

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

Name	Operator	Radix	Value
Execute_IBUF	==	[B]	1

R0=1,R1=0,F=>B,

The operation is A and B, the initial value is C2 and 52, A and B is 42

Then B is changed into 42

the value of A and B change with each clock, ex. A=C2=11000010 then A after one cycle is 01100001 (move the least most significant bit to the left) which is 61

（7） Description of all bugs encountered, and corrective measures taken.

We encounter one bug stated as follows

(i) control unit acted Unsteadily when turning on execute， in normal case, we would expect such wave form generated after execute is high（C1C0/Q/S）. (00/1/1） =>（01/1/1） =>（10/1/1） =>（11/1/1） =>（00/1/0） stay at this state until execute off =>（00/0/0）

However, the actual signals are sometimes normal and sometimes, it will behave like（00/1/1） =>（00/1/0） It directly drop into the halt state without 4 counts. After some observation, I realize the reason, S in logic expreesion is S = C1 + C0 + Q'E Meaning it is first driven by E signal and after Q is high, it is driven by counter signal. In my case, as our counter is 4-bit, we need manually put a signal into CLR port to stop counting. In this case, we choose S signal because as S turn to low, we are done counting, however, there is a problem state as follows. If counter signal generate slower than S signal we will see in the second clock cycle, while Q is high, that will cause Q'E turn to low, and C1 and C0 has not turned to 01, these means S signal will also turn to low, and counter is driven by S! This is the exact reason of the ubcertainity, if counter signal generate faster, it will not appear, otherwise, it will direct halt. We consult TA and she gave us a solution of changing 74161 into 74163, and that worked.

(8) conclusion

a. This lab is done by complete two portions of tasks, we find that the first portion of

task is time comsuming and difficult compared to the second portion. In this lab, we understand the basics of building a processor, and the construction of different unit. We practice the correct use of swtich rather than floating the signal.

b. Post-lab question 1_
We can use an XOR gate to realize that. In this lab we connect F2 and f(A,B)(f(A,B) include AND OR XOR 1），by using this method, we can be saved from the trouble of using 8-1 MUX.

Post-lab question 2_
It can reduce the number of unit tests, instead of testing 8 functions, we now only need to test 4 + 1 funtions, it reduce the implementation time and increase the testability.

Post-lab question 3_
First, we derive the logic expreesions of S and Q, we implement these logic gate, then we implemnet flip-flop and counter to complete functionality. Lastly, we connect LED to the output signals to debug.
Mealy machine advantage: need fewer state to operate.
Disadvantage: it will act to input signal, maybe harder to operate

Moore machine: opposite of above.

Post-lab question 4_
Vsim generate simulation signal and debug core display the real signal on hardware.
Vsim is prefered when there are mutiple signals need to be tested.
Debug core is prefered when simulation is not giving the same response in reality.
c.(nothing)