Term Project 1 Report

ID: 2017120486 Name:서천함

1. Project summary

The project is about a classic missionary-cannibal problem. Given the order of crossing the river in the problem description, what we are supposed to do is to transform the present and next state missionaries and cannibals using combinational logic. We can design 3 inputs to represent the missionary's current state, cannibal's current state and direction respectively. Besides, the 2 outputs can be used to represent the missionary's next state, cannibal's next state. Based on truth table and 5-variable K-map we can easily optimize the logical equations for the output. Then we can design the module using the logical equations and finally get the simulation result of the module.

2. Module description(include description of implement) (6%)

Step1: Suppose the following 5 input variables and 4 output variables.
a: represents the most significant bit of missionary_curr
b: represents least significant bit of missionary_curr
c: represents the most significant bit of cannibal_curr
d: represents least significant bit of cannibal_curr
e: direction(left 0, right 1)
w: represents the most significant bit of missionary_next
x: represents least significant bit of missionary_next
y: represents the most significant bit of cannibal_next
z: represents least significant bit of cannibal_next

Step2: Based on the problem description, we can complete the truth table as follows.

a	b	c	d	e	w	x	y	z
0	0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	1
0	0	0	1	0	0	0	1	0
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1
0	0	1	0	1	0	0	0	0
0	0	1	1	0	1	1	1	1
0	0	1	1	1	0	0	0	1
0	1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1	1
0	1	0	1	0	1	0	1	0
0	1	0	1	1	1	1	1	1
0	1	1	0	0	1	1	1	1
0	1	1	0	1	1	1	1	1
0	1	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	1	1
1	0	0	1	0	1	1	1	1
1	0	0	1	1	1	1	1	1
1	0	1	0	0	1	1	1	1
1	0	1	0	1	0	0	1	0
1	0	1	1	0	1	1	1	1
1	0	1	1	1	1	1	1	1
1	1	0	0	0	1	1	0	1
1	1	0	0	1	1	1	1	1
1	1	0	1	0	1	1	1	0
1	1	0	1	1	0	1	0	1
1	1	1	0	0	1	1	1	1
1	1	1	0	1	1	1	0	0
1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	0	1

Step 3: According to the truth table, we can use 5-variable Karnaugh map to optimize the logical equations for w, x, y, z repectively.

w =acd+ae'+a'b+ bc+ b'c'e+ cde'+ c'd'
x = ad+ae'+bc+ cde'+c'd'+c'e
y = ab'+a'b+b'c'+ce'+c'd'e+de'
z = ab'd+a'bc+cd+c'e+d'e'

Step 4: Based on the logical equations, we can implement combinational logic of the problem with gate-level.

18 different inner net shoule be defined and we name them from w1 to w18.
The following table describes each wire and its input variables.

w1: acd	w10: ab'
w2: ae'	w11:b'c'
w3: a'b	w12: ce'
w4: bc	w13:c'd'e
w5: b'c'e	w14:de'
w6: cde'	w15:ab'd

w7: c'd'	w16:a'bc
w8: ad	w17:cd
w9: c'e	w18: d'e'

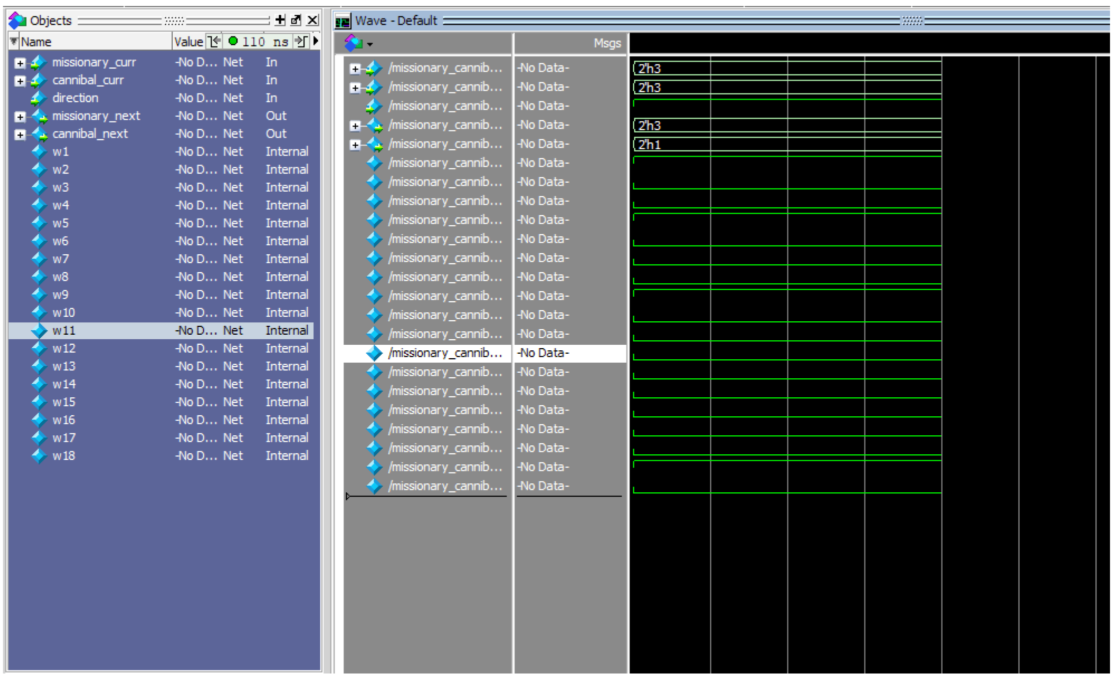
Gate inistantiation example:
1/ and (w1, missionary_curr[1], cannibal_curr[1], cannibal_curr[0]);
This means 3 inputs(a,c,d) and AND gate.
2/ or x(missionary_next[0],w8, w2, w4, w6, w7, w9);
This means 6 inputs and OR gate.

3. HDL code(Include program comment) (6%)

```
module missionary_cannibal(missionary_curr, cannibal_curr, direction,
missionary_next, cannibal_next);

    //Input declaration
    input[1:0]missionary_curr;
    input[1:0]cannibal_curr;
    input direction;
    output[1:0]missionary_next;
    output[1:0]cannibal_next;

    //Inner net definition
    wire w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15, w16,
w17, w18;

    //Basic logic gate instantiation

    // w
    and (w1, missionary_curr[1], cannibal_curr[1], cannibal_curr[0]);//3-input and
gate(acd)
    and (w2, missionary_curr[1], ~direction); //2-input and gate(ae')
    and (w3, ~missionary_curr[1], missionary_curr[0]);//2-input and gate(a'b)
    and (w4, missionary_curr[0],cannibal_curr[1]);//2-input and gate(bc)
    and (w5, ~missionary_curr[0], ~cannibal_curr[1], direction);//3-input and
gate(b'c'e)
    and (w6, cannibal_curr[1], cannibal_curr[0], ~direction);//3-input and gate(cde')
    and (w7, ~cannibal_curr[1], ~cannibal_curr[0]);//2-input and gate(c'd')
    or w(missionary_next[1],w1, w2, w3, w4, w5, w6, w7);

    // x
    and (w8, missionary_curr[1], cannibal_curr[0]);//2-input and gate(ad)
    and (w9, ~cannibal_curr[1], direction);//2-input and gate(c'e)
    or x(missionary_next[0],w8, w2, w4, w6, w7, w9);
```

```
//y
and (w10, missionary_curr[1], ~missionary_curr[0]);//2-input and gate(ab')
and (w11, ~missionary_curr[0], ~cannibal_curr[1]);//2-input and gate(b'c')
and (w12, cannibal_curr[1], ~direction);//2-input and gate(ce')
and (w13, ~cannibal_curr[1], ~cannibal_curr[0], direction);//3-input and gate(c'd'e)
and (w14, cannibal_curr[0], ~direction);//2-input and gate(de')
or y(cannibal_next[1],w10, w3, w11, w12, w13, w14);


//z
and (w15, missionary_curr[1], ~missionary_curr[0], cannibal_curr[0]);//3-input and gate(ab'd)
and (w16, ~missionary_curr[1], missionary_curr[0],
cannibal_curr[1]);//3-input and gate(a'bc)
and (w17, cannibal_curr[1], cannibal_curr[0]);//2-input and gate(cd)
and (w18, ~cannibal_curr[0], ~direction);//2-input and gate(d'e')
or z(cannibal_next[0], w15, w16, w17, w9, w18);

endmodule
```

4. Simulation screenshot

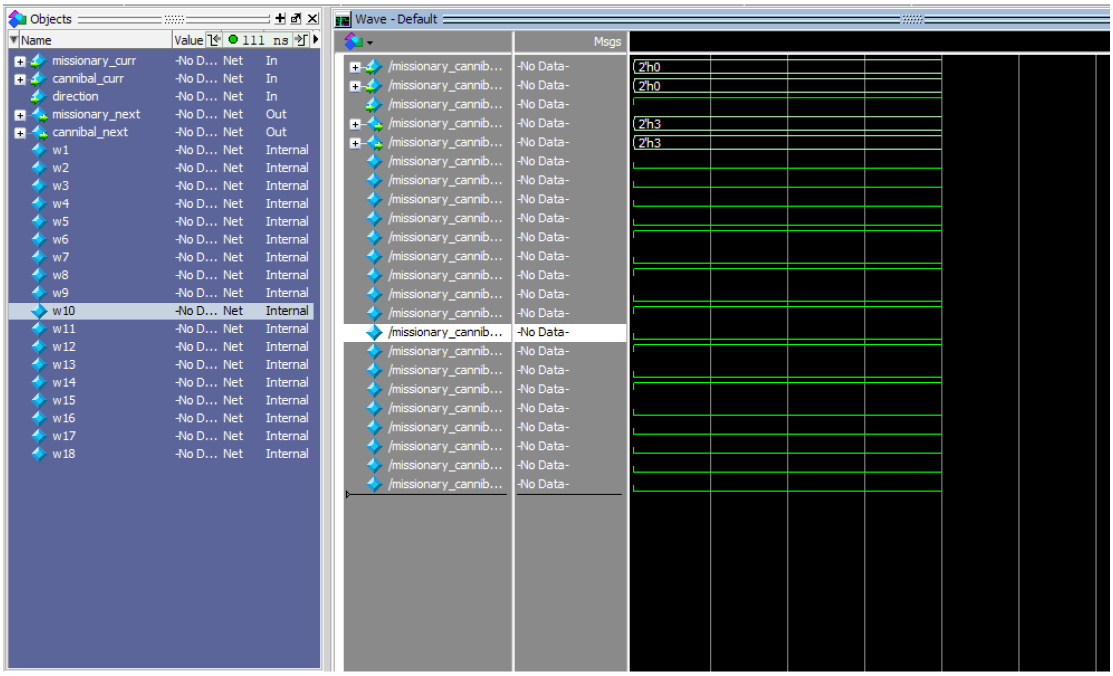


(Simulation script used:
quit -sim
vsim -gui work.missionary_cannibal
add wave *

force -deposit missionary_curr 2'h3 0
force -deposit cannibal_curr 2'h3 0
force -deposit direction 1 0

run 4ns)

Explaination:
Input: 1 1 1 1 1
Output: 1 1 0 1
-> Correct.



(Simulation script used:
quit -sim
vsim -gui work.missionary_cannibal

add wave *

force -deposit missionary_curr 0 0
force -deposit cannibal_curr 0 0
force -deposit direction 1 0

run 4ns)

Explaination:
Input: 0 0 0 0 1
Output: 1 1 1 1
-> Correct.