EECS 478 Project 2 Report

Yuxuan Zhang, uniqname: zyuxuan

Task1

(1) AND gate

First, check whether the input nodes and output node are pointing to null pointer and clean the truth table and fan in of the output node.

Then, add input1 and input2 as fan in, and set "2" as the number of variables.

Finally, add new entry"11" to the truth table, because only when input1 and input2 are both 1, the output will be 1.

(2) 3-input XOR gate

First, check whether the input nodes and output node are pointing to null pointer and clean the truth table and fan in of the output node.

Then, add input1, input2 and input3 as fan in, and set "3" as the number of variables.

Finally, add new entries to the truth table. Because the truth table of 3-input XOR gate is as follow:

input1	input2	input3	output
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

(3) 4-input MUX

First, check whether the input nodes and output node are pointing to null pointer and clean the truth table and fan in of the output node.

Then, add input1, input2, input3, input4, select1 and select2 as fan in, and set "6" as the number of variables.

Finally, add new entries to the truth table. Because the truth table of 4-input MUX is as follow:

select2	select1	input4	input3	input2	input1	output
0	0	-	-	-	1	1
0	1	-	-	1	-	1
1	0	-	1	-	-	1
1	1	1	-	-	-	1

Task2

(1) Create add module

Firstly, create input nodes, including input1, input2 and cin.

Create output nodes, including output and cout.

Create internal nodes C[i], where $0 \le i <$numBits.

The Boolean algebra we use to generate 1 bit full-adder is as follow:

$$s[i] = a[i] \oplus b[i] \oplus c[i-1], \text{ where } 0 < i < \text{numBits}$$

$$s[0] = a[0] \oplus b[0] \oplus \text{cin}.$$

$$c[i] = a[i] \cdot b[i] + a[i] \cdot c[i-1] + b[i] \cdot c[i-1], \text{ where } 0 < i < \text{numBits} - 1;$$

$$c[0] = a[0] \cdot b[0] + a[0] \cdot \text{cin} + b[0] \cdot \text{cin};$$

$$\text{cout} = a[\text{numBits}-1] \cdot b[\text{numBits}-1] + a[\text{numBits}-1] \cdot c[\text{numBits}-2] + b[\text{numBits}-1] \cdot c[\text{numBits}-2].$$

For adder, we need to generate internal nodes d[i], e[i], f[i] to store the temporary value.

We need 3 AND gate and 2 OR gate and a 3-input XOR gate for each 1-bit adder.

$$d[i] = a[i] \cdot b[i], e[i] = a[i] \cdot c[i-1], f[i] = b[i] \cdot c[i-1], g[i] = d[i] + e[i],$$

$$c[i] = f[i] + g[i], output[i] = s[i] = a[i] \oplus b[i] \oplus c[i-1]$$

Then I use a for loop to generate output nodes with numBits width. The terminal cases like i=0 or i=numBits-1 are considered as:

$$d[0] = a[0] \cdot b[0], e[0] = a[0] \cdot \text{cin}, f[0] = b[0] \cdot \text{cin}, g[0] = d[0] + e[0],$$

$$c[0] = f[0] + g[0], output[0] = s[0] = a[0] \oplus b[0] \oplus \text{cin}$$

$$d[\text{numBits}-1] = a[\text{numBits}-1] \cdot b[\text{numBits}-1], e[\text{numBits}-1] = a[\text{numBits}-1] \cdot c[\text{numBits}-1],$$

$$f[\text{numBits}-1] = b[\text{numBits}-1] \cdot c[\text{numBits}-2], g[\text{numBits}-1] = d[\text{numBits}-1] + e[\text{numBits}-1],$$

$$\text{cout} = f[\text{numBits}-1] + g[numBits - 1],$$

$$output[numBits - 1] = s[\text{numBits}-1] = a[\text{numBits}-1] \oplus b[\text{numBits}-1] \oplus c[\text{numBits}-1]$$

(2) Create subtract module

Firstly, create input nodes, including input1 and input2, and then create output nodes. We also need to create "1" node and "0" node as carry in and the inputs of XOR gate.

To calculate input1 minus input2, we need to turn input2 into 2's complemented format, namely

$$(x_k x_{k-1} x_{k-2} \cdots x_0)_{implemented} = x_k' x_{k-1}' x_{k-2}' \cdots x_0' + 1$$

I use 3-input XOR gate with the 3 inputs "0","1" and input2[i], because $0 \oplus 1 \oplus input2[i] = \overline{input2[i]}$. Use a for loop to inverse the value of each node of input2, and then use the add module to calculate $input2_{numBits-1}' input2_{numBits-2}' \cdots input2_0' + 1$.

Finally I use another adder to calculate input1 add the 2's complemented format of input2, and the result is input1 minus input2.

(3) Verification

i) ADDER: For 2-bit adder and 4-bit adder, I use the provided simulator to check the correctness of the result. Firstly generate the add1.blif, add2.blif and add4.blif, and for each BLIF file, I write a corresponding input file.

Using the follow command:

```
./simulator add1.blif<input1
```

to see whether the output is correct. And then I use the provided adder16.blif to compare with the result of my adder16_zyuxuan.blif by typing

```
./abc adder16.blif adder16_zyuxuan.blif.
```

ii) SUBTRACTOR:

I swap my sub16.blif with Zhejing Sang and Lifan Gong. And I use the following command:

```
./abc sub16.blif sub16_sang.blif
./abc sub 16.blif sub16_lifan.blif
```

to check the correctness of my BLIF files.

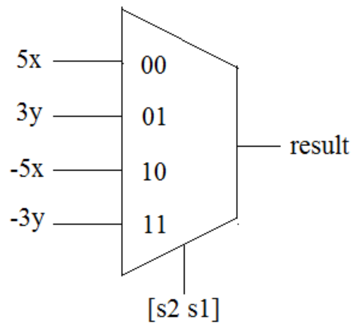
I also check 2-bit subtractor and 4-bit substractor by using simulator.

Task 3

First call the createSHIFTModule funciton to shift input1 2bits left to calculate 4×input1. Then call the createADDModule function to compute 4xinput1+input1, and we got 5×input1. In datapaths.cpp, we name the nodes of 5xinput1 "x_5". Use the same method to calculate 3xinput2. In datapaths.cpp, we name the nodes of 3×input2 "y_3".

To compare "x_5" with "y_3", the sign of "x_5", the sign of "y_3" and the sign of "x_5"-"y_3" all need to be considered. We firstly consider the sign of "x_5" and the sign of "y_3". Because the sign of a 2's complement binary number is the highest bit, so we can only check the highest bit of these two numbers. If the sign of these two numbers are different, the number with the highest bit being 1 is the smaller one, because a negative number is always smaller than a positive number. If the sign of these two numbers are the same, we must use a subtractor to decide which one is smaller.Using 5×input1 and 3×input2 as the inputs parameters of the createSUBModule function, we can check the sign of the result of the subtractor. If the sign of the result is negative, we can conclude that 5x is smaller than 3y. If the sign of the result is positive, we can conclude that 3y is smaller than 5x.

The possible output of abs(min(5x, 3y)) will be 5x, 3y, -5x, -3y. So I use a 4-input MUX to select which one should be the output. The diagram of this MUX can be seen in the following diagram.



The select signal s2,s1 depend on the highest bits of 5x-3y, 5x, 3y, which is shown in the following table. From the table, we can draw the conclusion that

$$s1 = 3y[18] + 5x[18]; \quad s2 = \overline{(5x\_3y)[18]} \cdot \overline{5x[18]} + \overline{5x[18]} \cdot 3y[18] + 3y[18] \cdot \overline{(5x\_3y)[18]}$$

5x-3y[18]	5x[18]	3y[18]	output	s2	s1
0	0	0	3y	0	1
0	0	1	-3y	1	1
0	1	0	-5x	1	0
0	1	1	-3y	1	1
1	0	0	5x	0	0
1	0	1	-3y	1	1
1	1	0	-5x	1	0
1	1	1	-5x	1	0