SoC설계

Lab#6
Ripple Carry Adder

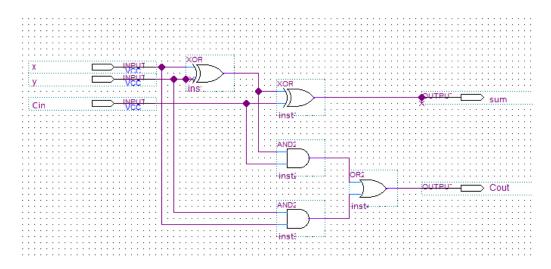
컴퓨터공학과 201402439 천원준

1. Purpose of the lab

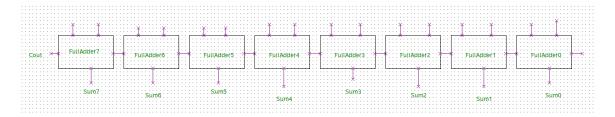
이번 과제의 목표는 4-to-1 MUX와 2-to-4 Decoder을 Structural, Dataflow, Behavioral Style로 구현하고 정상 작동하는지 확인해보는 것이다.

2. Design procedure

(1) Full-Adder



(2) 8-bit RCA



3. Simulation

Full Adder

```
//Full Adder
module FA(x, y, cin, cout, sum);
//input & output
input x, y, cin;
output cout, sum;

//assign {cout, sum} = a + b + c; //using concate, cout : higher bit, sum : lower bit
assign {cout, sum} = x + y + cin;
endmodule |
```

x, y, cin 세 가지 1비트짜리 입력을 받아, 전부 합한 2비트값을 cout(상위 1비트), sum(하위 1비트)로 나눠서 출력해줍니다.

8-bit Ripple Carry Adder

```
//8-bit Ripple Carry Adder
//Top module
module RCA(x, y, cin, cout, sum);

//input
input [7:0] x, y; //8-bit input
input cin; //1-bit

//output
output [7:0] sum; //8-bit
output cout; //1-bit

wire [6:0] carry; //carry among the FAS

FA FA0(.x(x[0]), .y(y[0]), .cin(cin), .cout(carry[0]), .sum(sum[0]));
FA FA1(.x(x[1]), .y(y[1]), .cin(carry[0]), .cout(carry[1]), .sum(sum[1]));
FA FA2(.x(x[2]), .y(y[2]), .cin(carry[1]), .cout(carry[2]), .sum(sum[2]));
FA FA3(.x(x[3]), .y(y[3]), .cin(carry[2]), .cout(carry[3]), .sum(sum[3]));
FA FA4(.x(x[4]), .y(y[4]), .cin(carry[3]), .cout(carry[4]), .sum(sum[4]));
FA FA5(.x(x[5]), .y(y[5]), .cin(carry[4]), .cout(carry[6]), .sum(sum[5]));
FA FA7(.x(x[7]), .y(y[7]), .cin(carry[6]), .cout(cout), .sum(sum[7]));
endmodule |
```

위에서 만든 Full Adder 모듈 8개를 이어 붙여, 8-bit Ripple Carry Adder를 만들었습니다. 첫번째 FA인 FA0의 입력값은 입력 x, y의 최하위 비트 1비트와 cin입니다. 출력 값은 각 비트의 합인 sum과 carry[0]입니다. 하위 FA로부터 나온 carry값을 그 바로다음 FA의 cin으로 넣어주는 방식으로 FA를 연결하여, 최상위 FA에서는 출력으로 나오는 carry를 cout에 넣어줍니다.

RCA testbench

```
`timescale 1ns/100ps
module RCA_tb;
//input(reg)
reg [7:0] x, y;
reg cin;
//output(wire)
wire cout;
wire [7:0] sum;
integer i, j;
integer num_correct, num_wrong;
reg [8:0] check;
RCA RCA0(.x(x), .y(y), .cin(cin), .cout(cout), .sum(sum));
initial begin
    num_correct = 0; num_wrong = 0;
for(i=0; i<16; i=i+1)begin</pre>
       \hat{x}=i; for (j=0; j<16; j=j+1) begin
           y=j;
cin = 1'b0;
           check = x + y + cin;
            if({cout, sum} == check)
               num_correct = num_correct+1;
               num_wrong = num_wrong+1;
        end
    end
    $display("num_correct = %d, num_wrong = %d", num_correct, num_wrong); //print
endmodule |
```

Testbench에서는 for문을 이용하여 입력값을 x, y 각각 0~15까지 바꿔주면서 합이 제대로 나오는지 확인하였습니다. 입력값을 단순 합한 값을 저장하는 check 변수와 RCA를 통해 나온 합을 비교하여, RCA가 올바르게 연산하였으면 매 연산마다 num_correct의 값을 1씩 증가시킵니다. 연산이 올바르지 않다면 num_wrong의 값을 1씩 증가시킵니다.

(2)	Msgs							
+/RCA_tb/x	00001111	00000000						
-/ /RCA_tb/y	00001111	00000000	00000001	00000010	00000011	00000100	00000101	00000110
/RCA_tb/cin	0							
<pre>/RCA_tb/cout</pre>	St0							
-/ /RCA_tb/sum	00011110	00000000	00000001	00000010	00000011	00000100	00000101	00000110
-/-/ /RCA_tb/i	16	0						
-/-/ /RCA_tb/j	16	0	1	2	3	4	5	6
-/- /RCA_tb/num_correct	256	0	1	2	[3	4	5	6
	0	0						
RCA_tb/check	000011110	000000000	000000001	000000010	000000011	000000100	000000101	000000110

X가 0일 때, y값에 따라서 결과값이 정확히 출력되는 것을 확인할 수 있습니다.

≨ 1 •	Msgs							
 /RCA_tb/x	00001111	00000101						
II - /RCA_tb/y	00001111	00000000	00000001	00000010	00000011	0000010	00000101	00000110
<pre>/RCA_tb/cin</pre>	0							
<pre>/RCA_tb/cout</pre>	St0							
	00011110	00000101	00000110	00000111	00001000	0000100	1 00001010	00001011
 → /RCA_tb/i	16	5						
	16	0	1	2	3	4	5	6
	256	80	81	82	83	84	85	86
+	0	0						
- → /RCA_tb/check	000011110	000000101	00000011	00000011	1 00000100	0 0000010	00000101	000001011

X가 5일 때, y값에 따라 합연산이 제대로 수행되는 것을 확인할 수 있습니다.

≨ 1 •	Msgs								
II - / /RCA_tb/x	00001111	00001011							
∓ - / /RCA_tb/y	00001111	00000000	00000001	000000	.0	00000011	00000100	00000101	00000110
<pre>/RCA_tb/cin</pre>	0				\bot				
<pre>/RCA_tb/cout</pre>	St0				\bot				
± –♦ /RCA_tb/sum	00011110	00001011	00001100	000011	1	00001110	00001111	00010000	00010001
	16	11							
	16	0	1	2	1	3	4	5	6
# /RCA_tb/num_correct		176	177	178		179	180	181	182
	0	0							
III - ♦ /RCA_tb/check	000011110	000001011	00000110	0 000001	.01	000001110	00000111	000010000	000010001

X가 11일 때, 합연산이 정확히 수행되고 있음을 확인할 수 있습니다.

\$ 1 →	Msgs										
∓ - ∜ /RCA_tb/x	00001111	00001111									
-/ /RCA_tb/y	00001111	00000111	00001000	00001001	00001010	00001011	00001100	00001101	00001110	00001111	
<pre>/RCA_tb/cin</pre>	0										
<pre>/RCA_tb/cout</pre>	St0										
-/ /RCA_tb/sum	00011110	00010110	00010111	(00011000	(0001100	1 00011010	00011011	00011100	00011101	00011110	
II - ∜ /RCA_tb/i	16	15									16
 /RCA_tb/j	16	7	.8	9	10	11	12	13	14	15	16
+- /RCA_tb/num_correct	256	247	248	249	250	251	252	253	254	255	256
	0	0									
RCA_tb/check	000011110	000010110	00001011	00001100	00001100	00001101	0 00001101	00001110	0 (00001110	000011110	

X가 15일 때, 합연산이 정확히 수행되었고, num_correct 변수를 통해 총 연산 횟수(256회) 중 256회 모두 정상적으로 합연산을 수행하였음을 확인 할 수 있습니다.

4. Evaluation

위의 시뮬레이션에서 매 연산마다 입력값을 단순 합연산하여 합한 check 값과 RCA를 통해 합한 {cout, sum} 값이 같음을 확인할 수 있습니다. 이를 통해 RCA를 통한 합연산이 제대로 수행되고 있음을 확인할 수 있습니다.

5. Discussion

이번 실습의 key는 Full Adder 8개를 이어 붙여, 하위 FA의 carry를 상위 FA의 입력으로 전달해주는 것입니다.

이번 과제를 통해 프로세서 상에서 덧셈 연산이 어떤 식으로 이루어지는지 깨닫게 되었습니다.

이 design의 단점은 모든 FA가 동시에 값을 내놓는 것이 아니고, 하위 FA의 연산이 끝나고 하위 FA의 carry를 전달받아야만 상위 FA의 연산을 수행할 수 있다는 것입니다. 그러므로 모든 FA의 연산이 수행되기 전의 매우 짧은 시간 동안 올바르지 않은 결과값을 출력으로 내놓을 수도 있습니다. 이를 보완하기 위해서는 모든 FA가 clock 신호에 맞추어서 결과를 내놓을 수 있도록 설계하는 방법이 있습니다.