6주차 결과보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.

해당 실험의 목적은 반감산기, 전감산기와 반가산기, 전가산기, 코드 컨버터에 대해 이해하고, 구현해봅니다. 또한, 카르노맵을 통해 SOP나 POS를 구할 수 있는 능력을 배양하고, 구한 함수식을 바탕으로 직접 구현해봅니다. Simulation에서 그치지 않고, 최종적으로 설계를 FPGA 보드에 올려, 동작을 검증하고, 입력 신호에 따른 출력 결과를 분석하고자 합니다.

2.

우선 첫 번째로 설명할 것은 half-adder입니다.

Input A	Input B	Output s	Output c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
`timescale 1ns / 1ps

module halfadder(
  input a, b,
  output s, c
);
assign s = a^b;
assign c = a&b;

endmodule
```

```
`timescale 1ns / 1ps

module halfadder_tb;
reg a, b;
wire s, c;

halfadder test(
  .a(a),
  .b(b),
  .s(s),
  .c(c)
);

initial begin
  a = 1'b0;
  b = 1'b0;
  #160
  $finish;
end

always@(a or b)begin
  a <= #80 ~a;
  b <= #40 ~b;
end

endmodule
```

Half Adder의 식은 Output s(sum) = A^B, Output c(carry) = A&B로 나타낼 수 있습니다. 두 개의 input 중 한 개가 1일 때 sum이 발생하고, 둘 다 1이여야 carry가 발생합니다.

두 번째로 서술할 것은 Full Adder입니다.

Input Cin	Input B	Input A	Output s	Output cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
`timescale 1ns / 1ps

module fulladder(
  input a, b, cin,
  output s, cout
);

assign s = a^b^cin;
assign cout = (a&b)|((a^b)&cin);

endmodule
```

```
`timescale 1ns / 1ps

module fulladder_tb;
reg a, b, cin;
wire s, cout;

fulladder test(
  .a(a),
  .b(b),
  .cin(cin),

  .s(s),
  .cout(cout)
);

initial begin
  cin = 1'b0;
  b = 1'b0;
  a = 1'b0;
  #160
  $finish;
end

always@(cin or b or a)begin
  cin <= #80 ~cin;
  b <= #40 ~b;
  a <= #20 ~a;
end

endmodule
```

Full Adder에서 Output s(sum) = a^b^cin, Output cout(carry) = cin&(a^b)+a&b 로 나타낼 수 있습니다. 세 개의 input 중 홀수 개가 1일 때 sum이 발생하고, 1이 2개 이상이여야 carry가 발생합니다.

3.

우선 첫 번째로 설명할 것은 half-subtractor입니다.

Input A	Input B	Output b	Output d
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

```
`timescale 1ns / 1ps

module halfsubtractor(
  input A, B,
  output b, D
);

assign D = A^B;
assign b = B&(~A);

endmodule
```

```
`timescale 1ns / 1ps

module halfadder_tb;
reg a, b;
wire s, c;

halfadder test(
  .a(a),
  .b(b),
  .s(s),
  .c(c)
);

initial begin
  a = 1'b0;
  b = 1'b0;
  #160
  $finish;
end

always@(a or b)begin
  a <= #80 ~a;
  b <= #40 ~b;
end

endmodule
```

Half Subtractor의 식은 Output b(borrow) = (~A)&B, Output D(Difference) = A^B로 나타낼 수 있습니다. 두 개의 input 중 홀수 개가 1일 때(똑같지 않을 때) Difference가 발생하고, A < B이면, 즉 A가 0이고 B가 1일 때 borrow가 발생합니다.

다음으로 설명할 것은 full-subtractor입니다.

Input A	Input B	Input Borrow	Output b	Output d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

```
`timescale 1ns / 1ps

module fullsubtractor(
  input A, B, Borrow,
  output b, D
);

assign b = ((~A)&B)|((~(A^B))&Borrow);
assign D = A^B^Borrow;

endmodule
```

```
`timescale 1ns / 1ps

module fullsubtractor_tb;
reg A, B, Borrow;
wire b, D;

fullsubtractor test(
  .A(A),
  .B(B),
  .Borrow(Borrow),

  .b(b),
  .D(D)
);

initial begin
  A = 1'b0;
  B = 1'b0;
  Borrow = 1'b0;
  #160
  $finish;
end

always@(A or B or Borrow)begin
  A <= #80 ~A;
  B <= #40 ~B;
  Borrow <= #20 ~Borrow;
end

endmodule
```

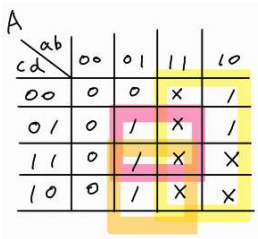
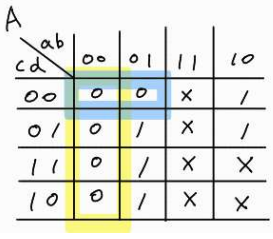
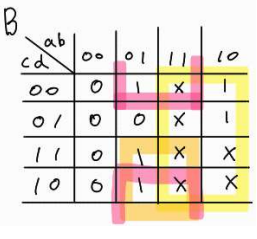
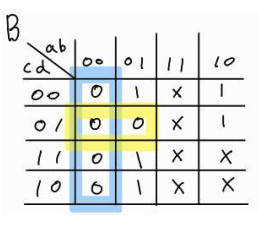
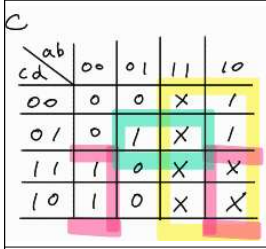
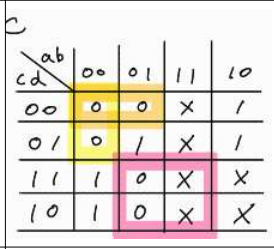
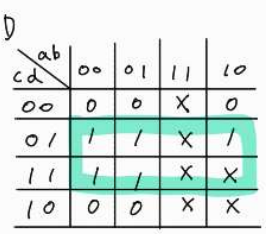
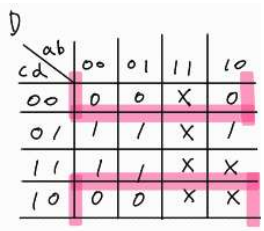
다음으로 설명하게 될 것은 Full Subtractor입니다.
Full Subtractor의 식은
Output b = (~A)&B+(~(A^B))&Borrow
Output d = A^B^Borrow로 나타낼 수 있습니다.
Output b는 A값은 0이고, input B와 Borrow의 값이 둘 다 0이 아니면 발생하거나, 모든 값이 1일 때 발생합니다. half subtractor와 full subtractor 모두 d(difference)는 input 값들에 xor을 취한 값이기 때문에 그 중 1이 홀수 개이면 발생합니다.

4.

In a	In b	In c	In d	Out A	Out B	Out C	Out D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	d	d	d	d
1	0	1	1	d	d	d	d
1	1	0	0	d	d	d	d
1	1	0	1	d	d	d	d
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d


```
`timescale 1ns / 1ps

module codeconverter(
  input a, b, c, d,
  output A, B, C, D
);

assign A = a|(b&d)|(b&c);
assign B = a|(b&(~d))|(b&c);
assign C = a|(b&(~c)&d)|((~b)&c);
assign D = d;

endmodule
```

```
`timescale 1ns / 1ps

module codeconverter_tb;
reg a, b, c, d;
wire A, B, C, D;

codeconverter test(
  .a(a),
  .b(b),
  .c(c),
  .d(d),

  .A(A),
  .B(B),
  .C(C),
  .D(D)
);

initial begin
  a = 1'b0;
  b = 1'b0;
  c = 1'b0;
  d = 1'b0;
  #160
  $finish;
end

always@(a or b or c or d)begin
  a <= #80 ~a;
  b <= #40 ~b;
  c <= #20 ~c;
  d <= #10 ~d;
end

endmodule
```

A in sop form	A in pos form	B in sop form	B in pos form
			
A = a+bd+bc	A = (a+c+d)(a+b)	B = a+bd'+bc	B = (a+b)(a+c+d')
C in sop form	C in pos form	D in sop form	D in pos form
			
C = a+bc'd+b'c	C=(b'+c')(a+b+c)(a+c+d)	D=d	D=d

Out A는 sop로 a+bd+bc, pos로 (a+b)(a+c+d)로 나타낼 수 있으며, Out B는 sop로 a+bc+bd', pos로 (a+b)(a+c+d')로 나타낼 수 있습니다. Out C는 sop로 a+b'c+bc'd, pos로 (a+c+d)(a+b+c)(b'+c')로 나타낼 수 있습니다. Out D는 sop와 pos 둘다 d로 나타낼 수 있습니다. 출력값에 대한 sop를 구한 다음 이를 이용해 간단하게 식으로 나타내 구현할 수 있었습니다. 8421(BCD)-2421 Code converter에서는 0000부터 1001까지의 입력값만 활용하기 때문에 1010부터 1111까지의 값들은 don't care로 두어 더욱 식을 간소화시킬 수 있었습니다.

5.

Full adder과 Half adder에서 Output s와 Full subtractor과 Half Subtractor에서 Output d는 모두 xor 게이트를 이용해 같은 결과가 나왔습니다. Full adder에서 c는 올라온 값, A, B 중 두 개 이상이 1일 경우 올림이 생긴다는 것으로 이해할 수 있습니다. Full Subtractor에서 b는 A와 B가 같은 값을 가지고 있는 상태에서 Borrow가 1일 때와 A는 0이고 B는 1일 때 내림 값이 있다는 것을 확인할 수 있습니다.

6.

오버플로우 : 가산기에서 n비트와 n비트를 더할 때 n+1비트의 결과가 나올 수 있는데, 이를 오버플로우라고 합니다. 제일 높은 MSB에서 carry가 발생했다는 의미와 같습니다. 이는 XOR 게이트를 통해 발견할 수 있는데, 오버플로우는 입력으로 주어지는 두 수가 같은 부호일 때에만 발생하므로, 같은 부호비트를 덧셈했을 시 부호가 바뀌면 오버플로우가 발생했다고 할 수 있습니다.