6주차 결과보고서

전공: 생명과학과 학년: 4학년 학번: 20182186 이름: 김승원

**1. 실험목표**

우선 adder와 subtractor의 개념을 이해한 다음, 마지막으로 code converter의 개념을 이해한다. Verilog를 사용해서 Full adder와 Half adder, Full subtractor와 Half adder를 구현해보고, 그리고 Code converter까지 구현해본다. 이후 FPGA를 통해 Verilog를 사용해서 구현한 회로가 동작을 하는지 확인한다.

**2. Half adder와 Full adder의 결과 및 과정 설명(진리표 포함)**

**2-1) Half adder**

**<design source code>**

`timescale 1ns / 1ps

module HA(

input a,b,

output s,c

);

assign s = a^b,

c = a&b;

endmodule

**<simulation source code>**

`timescale 1ns / 1ps

module tb\_HA;

reg A, B;

wire S, C;

HA HAGATE(.a(A), .b(B), .s(S), .c(C));

initial begin

A = 1'b0;

B = 1'b0;

end

always begin

A = #10 ~A;

B = #40 ~B;

end

initial begin

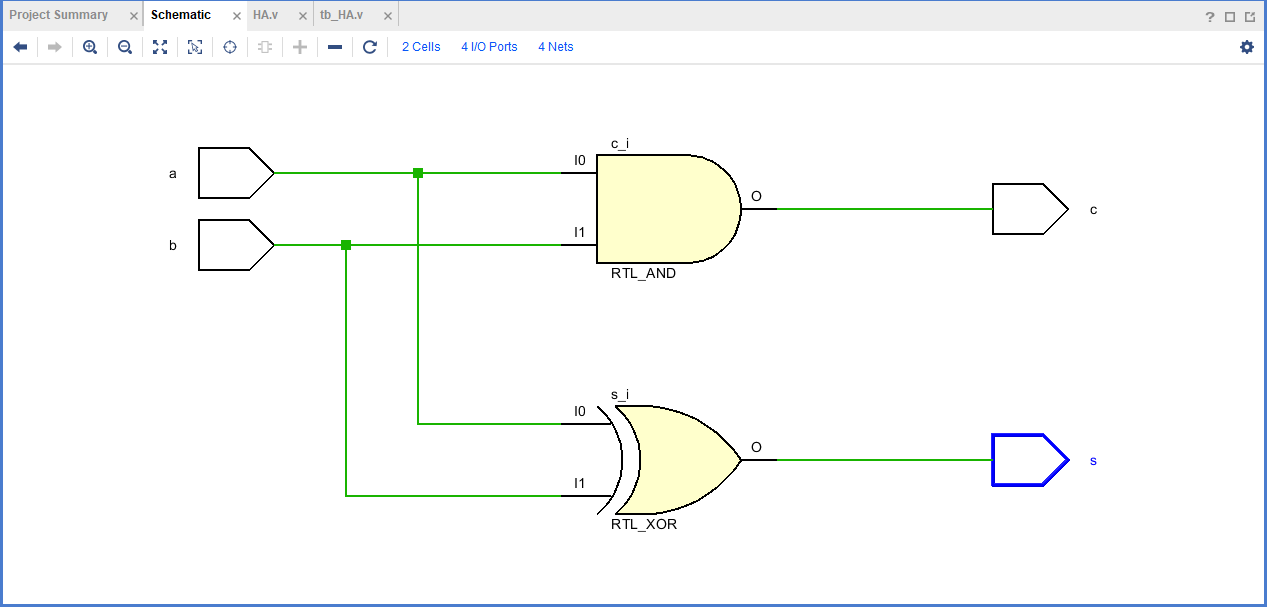
#1000

$finish;

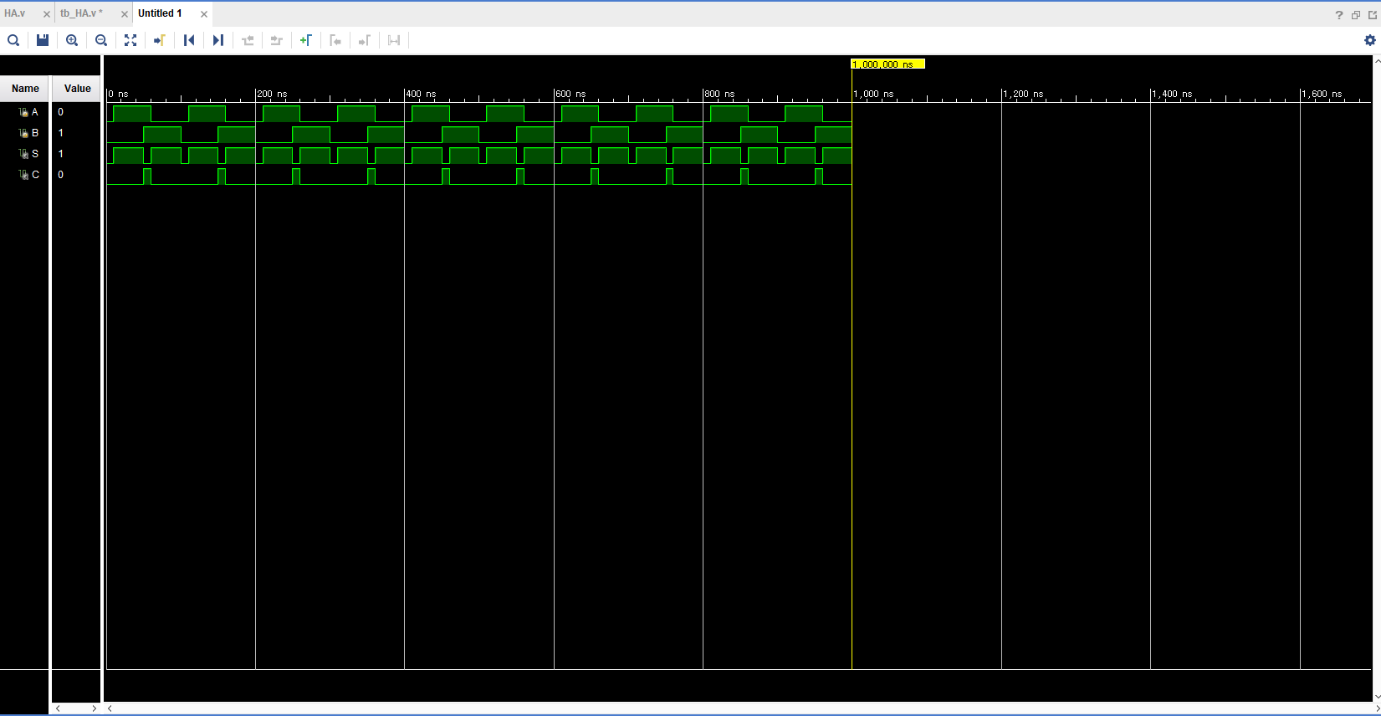
end

endmodule

**<Schematic>**



**<simulation>**



**<truth table>**

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**2-2) Full adder**

**<design source code>**

`timescale 1ns / 1ps

module FA(

input a,b,ci,

output co, s

);

assign s = (a^b)^ci,

co = (a&b)|(ci&(a^b));

endmodule

**<simulation source code>**

`timescale 1ns / 1ps

module tb\_FA;

reg A, B, Ci;

wire S, Co;

FA FAGATE(.a(A), .b(B), .ci(Ci), .s(S), .co(Co));

initial begin

A = 1'b0;

B = 1'b0;

Ci = 1'b0;

end

always @(A or B or C) begin

A <= #5 ~A;

B <= #20 ~B;

Ci <= #40 ~Ci;

end

initial begin

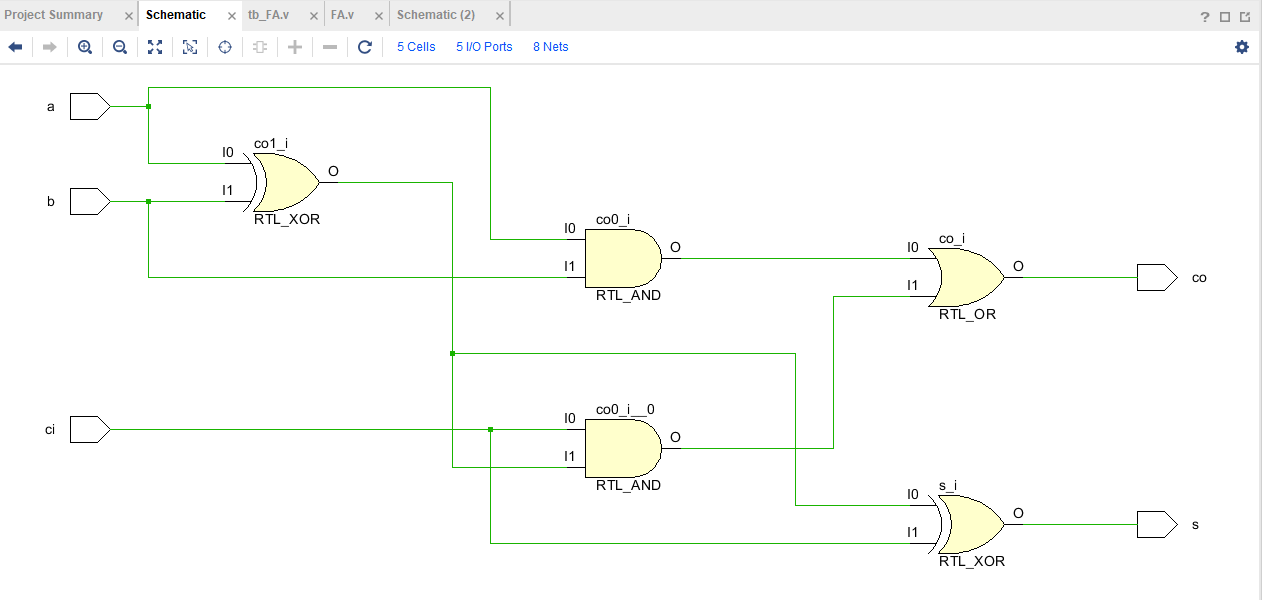
#1000

$finish;

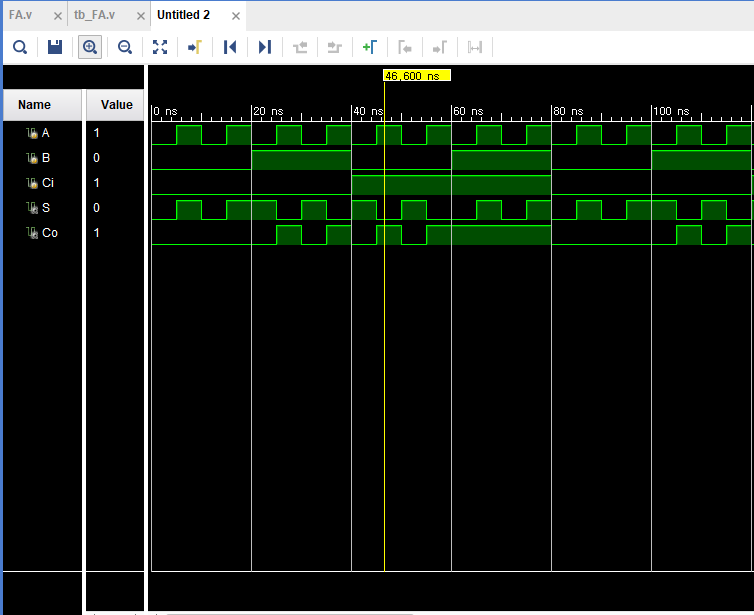
end

endmodule

**<Schematic>**



**<simulation>**



**<truth table>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| Ci(Cin) | B | A | S | Co(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 |

**3. Full subtractor 및 Half subtractor의 simulation 결과 및 과정 설명**

**3-1) Half subtractor**

**<design source code>**

`timescale 1ns / 1ps

module HS(

input a,b,

output s,c

);

assign s = a^b,

c = ~a&b;

endmodule

**<simulation source code>**

`timescale 1ns / 1ps

module tb\_HS;

reg A, B;

wire S, C;

HS HSGATE(.a(A), .b(B), .s(S), .c(C));

initial begin

A = 1'b0;

B = 1'b1;

end

always begin

A = #10 ~A;

B = #40 ~B;

end

initial begin

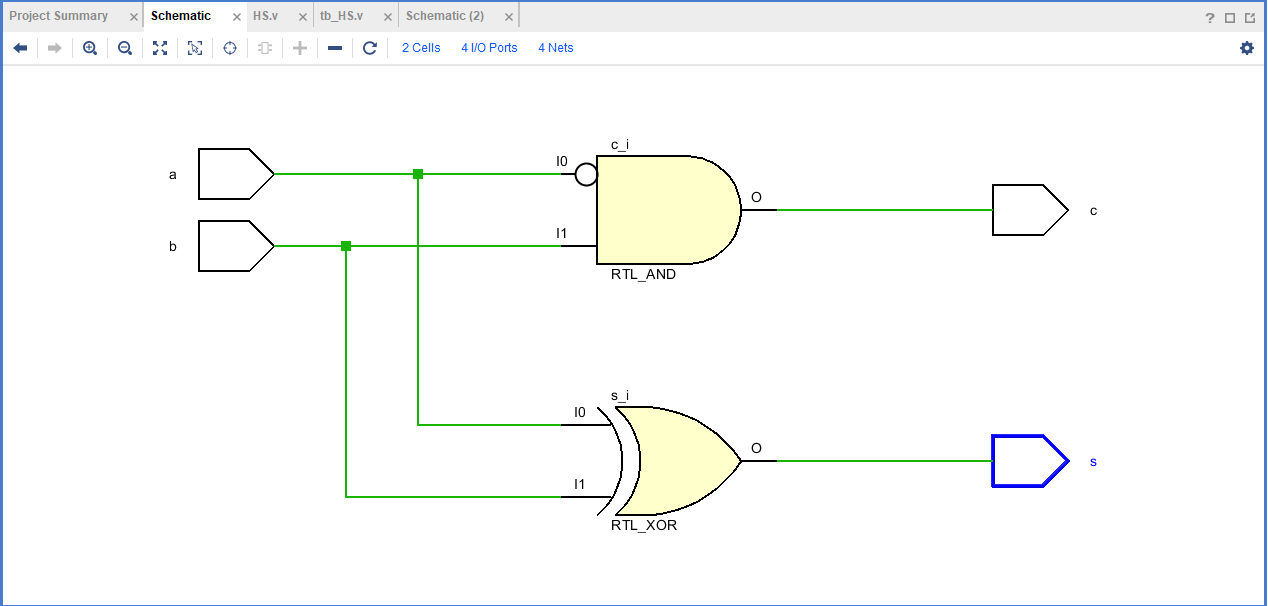
#1000

$finish;

end

endmodule

**<Schematic>**



**<simulation>**

텍스트, 전자기기, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**<truth table>**

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | C | S |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

**3-2) Full subtractor**

**<design source code>**

`timescale 1ns / 1ps

module FS(

input a,b,bn\_1,

output bn, d

);

assign d= a^b^(bn\_1),

bn = (~(a^b)&(bn\_1))|(~a&b);

endmodule

**<simulation source code>**

`timescale 1ns / 1ps

module tb\_FS;

reg A, B, Bn\_1;

wire D, Bn;

FS FSGATE(.a(A), .b(B), .d(D), .bn(Bn), .bn\_1(Bn\_1));

initial begin

A = 1'b0;

B = 1'b0;

Bn\_1 = 1'b0;

end

always @(A or B or Bn\_1) begin

A <= #5 ~A;

B <= #20 ~B;

Bn\_1 <= #40 ~Bn\_1;

end

initial begin

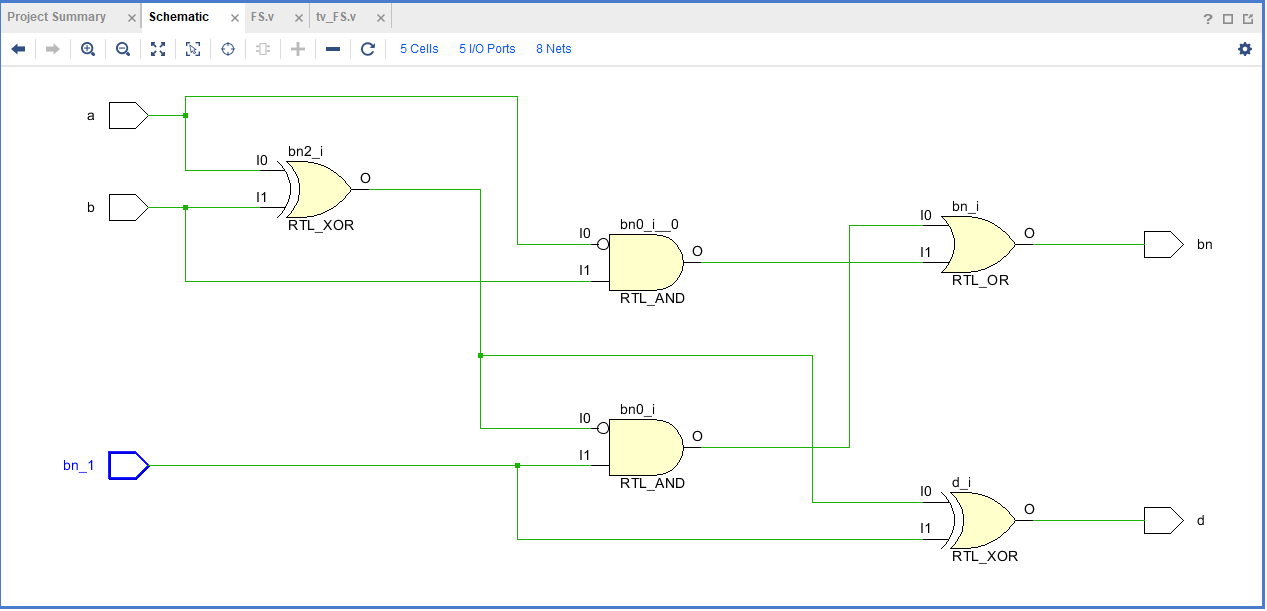
#1000

$finish;

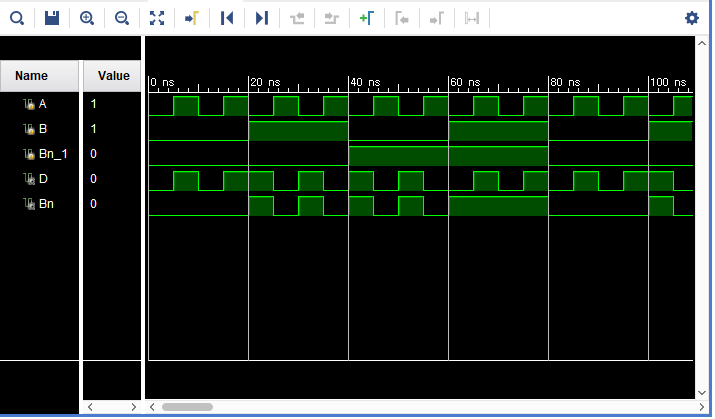
end

endmodule

**<Schematic>**

****

**<simulation>**



**<truth table>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| Bn\_1 | B | A | Bn | D |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**4. 8421(BCD)-2421 code converter 결과 및 과정 설명(SOP form, POS form, 카르노맵 포함, 진리표 작성)**

**<design source code>**

`timescale 1ns / 1ps

module BCD(

input a,b,c,d,

output p,q,r,s

);

assign p= ~((~a&~c&~d)|(~a&~b)),

q = ~((~a&~c&d)|(~a&~b)),

r = ~((~a&~b&~c)|(~a&b&c)|(~a&b&~d)),

s = d;

endmodule

**<simulation source code>**

`timescale 1ns / 1ps

module tb\_BCD;

reg A,B,C,D;

wire P,Q,R,S;

BCD BCDGATE(.a(A), .b(B), .c(C), .d(D), .p(P), .q(Q), .r(R), .s(S));

initial begin

A = 1'b0;

B = 1'b0;

C = 1'b0;

D = 1'b0;

end

always @(A or B or C or D) begin

A <= #20 ~A;

B <= #50 ~B;

C <= #70 ~C;

D <= #250 ~D;

end

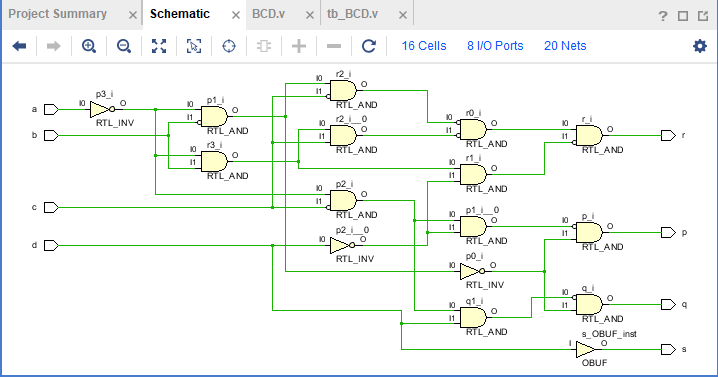
initial begin

#1000

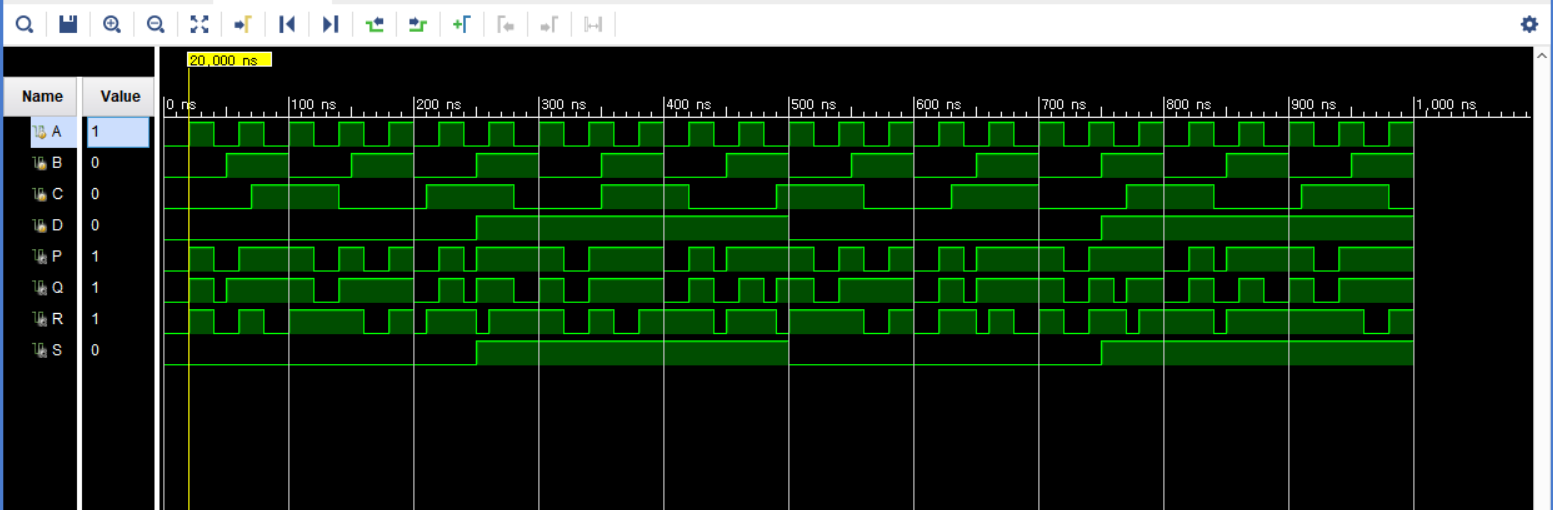
$finish;

end

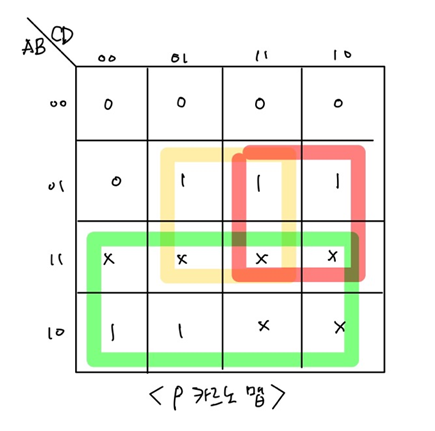
endmodule**<Schematic>**



**<simulation>**



**<카르노맵, SOP form, POS form>**



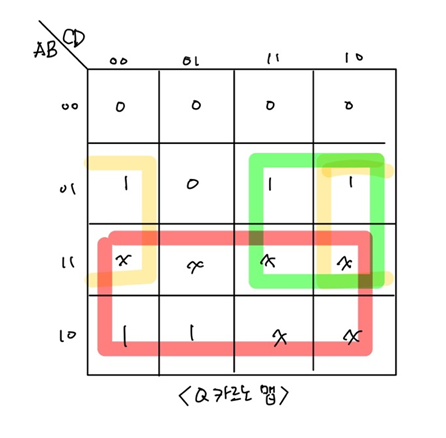
**P의 SOP form**

P = A+BC+BD

= A + B(C+D) 🡪 distributive

= (A+B)(A+B+C)

**POS form =>** (A+B)(A+B+C)



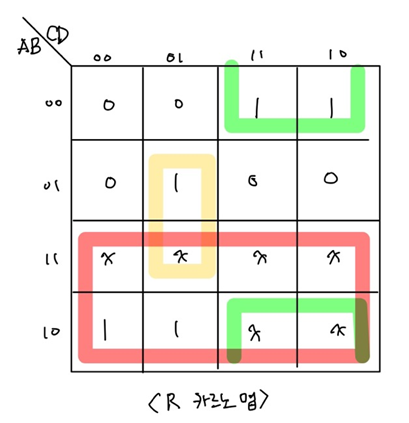
**Q의 SOP form**

Q = A+BC+BD’

= A+B(C+D’) 🡪 distributive

= (A+B)(A+C+D’) 🡪 distributive

**Q의 POS form =>** (A+B)(A+C+D’)



**R의 SOP form**

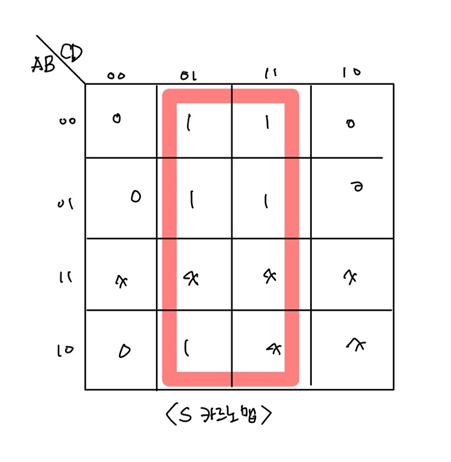
R = A+B’C+BC’D

= A+(B+C)(B’+C’D) 🡪 consensus

= (A+B+C)(A+B’+C’D) 🡪 distributive

= (A+B+C)(A+B’+C’)(A+B’+D) 🡪 distributive

**R의 POS form =** (A+B+C)(A+B’+C’)(A+B’+D)



**S의 SOP form**

S = D

**S의 POS form =>** D

**<truth table>**

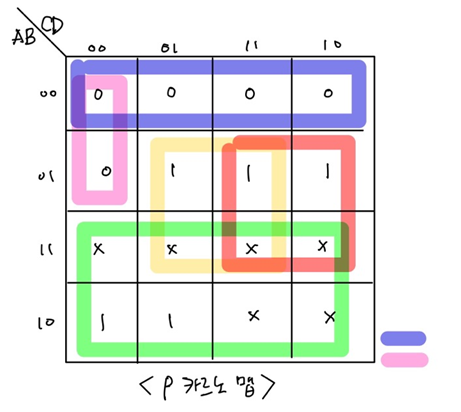
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| decimal | A | B | C | D | P | Q | R | S |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

**5. 결과검토 및 논의사항**

우선 Half adder와 Full adder 그리고 Half and Full subtractor 모두 각각의 구조를 Verilog 코딩한 다음 simulation 결과를 통해 Truth table을 작성했다. 결과적으로 Half adder, subtractor Boolean function과 Full adder, subtractor Boolean function와 결과를 각각 비교했을 때 결과가 일치하는 것을 확인했다. Logic circuit을 디자인하는 과정에서 드모르간 법칙을 이용해 NAND gate로 최대한 디자인했다. 그리고 8421 BCD에서 2421 BCD code로 변환하는 converter를 디자인하고자 카르노맵을 작성했고, 이를 통해 SOP, POS식을 얻고 minimization한 다음 function을 작성했다. 이후 드 모르간 법칙을 활용하면 최대한 NAND gate로 구성할 수 있도록 할 수 있음을 확인했다. 그리고 converter를 구현해서 최종적으로 simulation결과를 확인한 다음 FPGA로 동작시켜서 확인했다. 결과적으로 가령 8421 BCD code 0101가 입력됐을 때 1011이 출력되는 것(정상 작동)을 확인했다.

**6, 추가이론 조사 및 작성**

본 실험에서는 SOP식에서 switching algebra를 이용해 POS를 구했다. 하지만 POS 표현으로 간략화하는 다른 방법이 있다. 우선 카르노맵을 그린 후 POS Boolean function을 표준형으로 변환하여 각 항의 2진 값에 대응되거나 진리표에서 출력 값이 0이 된다면 셀에다가 0을 쓴다. 그리고 0을 가진 셀들을 2^n개 단위로 묶어서(SOP form 만들 때와 같은 방법) 합의 항을 구한 다음 서로 곱해서 간략화시킨 POS Boolean function을 구할 수 있다.



보라색 묶음과 분홍색 묶음을 통해 POS = (A+B)(A+C+D) 로 바로 구할 수 있다.