6주차 결과보고서

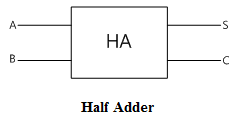
|  |
| --- |
| - Full Adder & Half Adder  - Full Subtracter & Half Subtracter  - Code Converter (8421(BCD)-2421)  - 결과 검토 |

20141196 김성희

**1. Full Adder & Half Adder**

**1-1. Half Adder**

- Half Adder는 아래 그림처럼 A, B input 2개를 받아서 S(Sum)과 C(Carry) ouput을 출력 해주는 회로다.



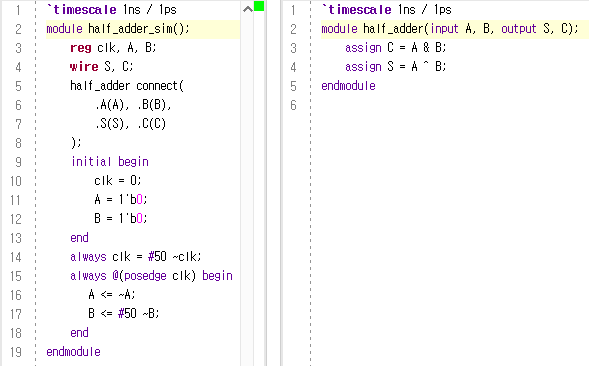
C:\Users\KSH\Desktop\그림판\제목 없음.pngSum과 Carry에 대한 진리표와 논리식은 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

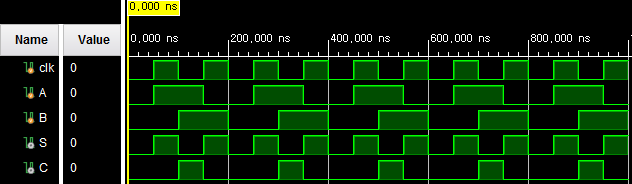
(논리식과 진리표 결과가 일치)

Verilog로 구현해 보자. (code, simulation, schematic, FPGA)

1. code ( ~ : NOT, & : AND, ^ : XOR )



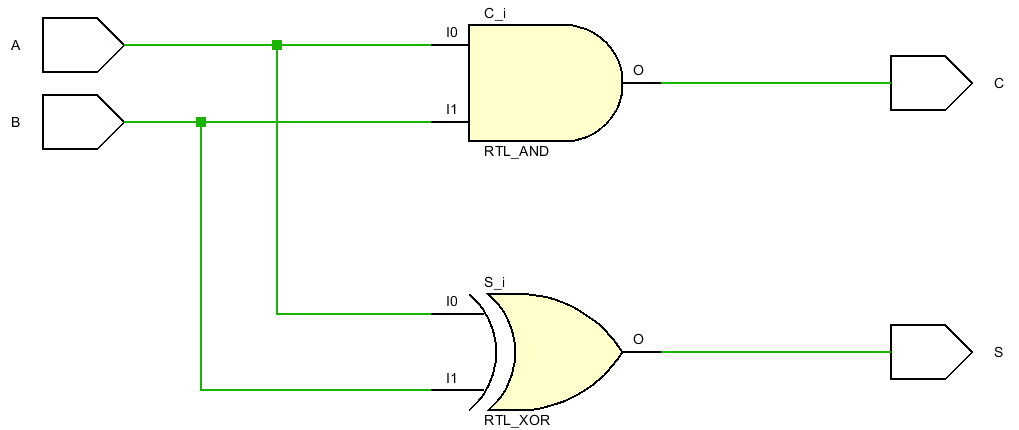
2. simulation



A, B = 2 inputs

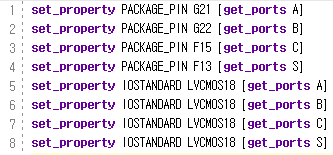
S = Sum, C = Carry

3. schematic



4. FPGA

- Design Sources경로에 code 짜기 -> Run Synthesis & Run Implementation -> Constraints경로에 .xdc 파일 추가하기 -> Open Implemented Design 클릭 후 Constraints Wizard 클릭(Define Target, .xdc파일 타겟 설정) -> Window 탭에서 I/O ports 클릭 -> 원하는 pin 선택 및 LVCMOS18(I/O Std) 선택 -> 저장 후 .xdc reload(아래 그림처럼 코드가 자동으로 짜서 나온다.) -> Generate Bitstream 클릭(Synthesis, Implementation도 자동으로 실행) 후에 Open Hardware Manager 클릭 -> Open Target 클릭 후 Auto Connect 클릭 -> Program Device 클릭 -> FPGA에 업로드 끝

 <- input ouput 변수를 I/O port와 연결

- F15, F13은 각각 LD(LED)1, 2를, G21, G22는 각각 MID, RIGHT switch를 의미한다.

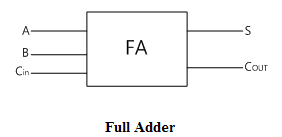
- MID, RIGHT switch 둘 다 누르지 않았을 때, LD1, 2 둘 다 안 켜진다.

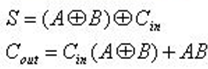
- MID, RIGHT switch 중 하나만을 선택하여 눌렀을 때, LD2만 켜진다. (S = AB인 경우)

- MID, RIGHT switch 둘 다 선택하여 눌렀을 때, LD1만 켜진다. (C = (!A & B) | (A & !B)인 경우)

**1-2. Full Adder**

- Full Adder는 아래 그림처럼 input으로 A, B Carry-in을 받아서 S(SUM)과 Cout(Carry-out)을 출력하는 회로다.



****다음은 FA의 진리표와 논리식이다.

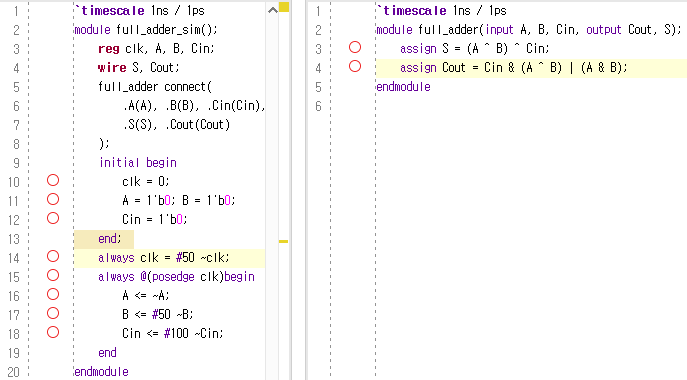
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **S** | **Cout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

\* Cout의 경우 Cin(A+B)+AB로 쓸 수 있으나 S에서의 재활용까지 고려하여 A XOR B를 사용.

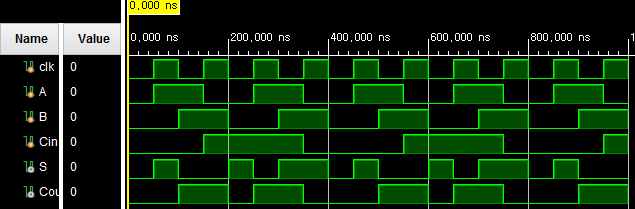
(논리식과 진리표 결과가 일치)

Verilog로 구현해 보자.(code, simulation, schematic, FPGA)

1. code ( ~ : NOT, & : AND, | : OR, ^ : XOR )



2. simulation

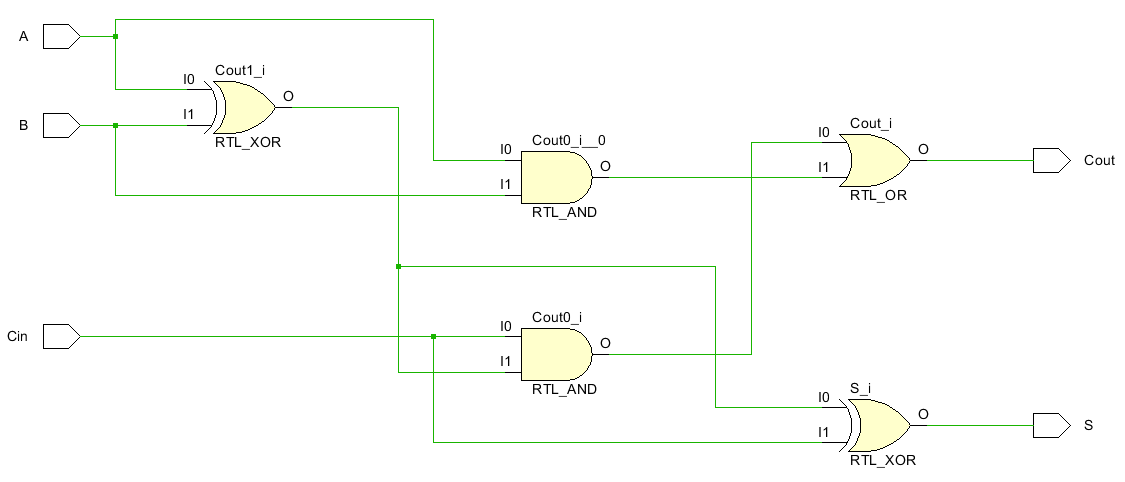


Input: A, B, Cin

Output: Cout, S

(Cout은 A,B,Cin중 2개 이상이 1인 경우, S는 홀수개의 input이 1인 경우 1의 출력값을 가진다.)

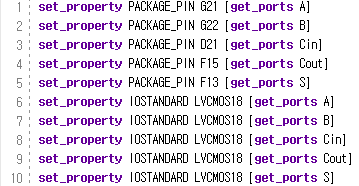
3.schematic



Cout1\_i (첫 번째 XOR gate)의 출력 값을 S와 Cout에서 공유한다.

4. FPGA

- Design Sources경로에 code 짜기 -> Run Synthesis & Run Implementation -> Constraints경로에 .xdc 파일 추가하기 -> Open Implemented Design 클릭 후 Constraints Wizard 클릭(Define Target, .xdc파일 타겟 설정) -> Window 탭에서 I/O ports 클릭 -> 원하는 pin 선택 및 LVCMOS18(I/O Std) 선택 -> 저장 후 .xdc reload(아래 그림처럼 코드가 자동으로 짜서 나온다.) -> Generate Bitstream 클릭(Synthesis, Implementation도 자동으로 실행) 후에 Open Hardware Manager 클릭 -> Open Target 클릭 후 Auto Connect 클릭 -> Program Device 클릭 -> FPGA에 업로드 끝

<-inputs, outputs 변수를 I/O port와 연결

- A는 MID, B는 RIGHT, Cin은 LEFT switch와 연결

- Cout은 LD1에, S는 LD2에 연결 (각각 1의 값이 할당되면 불이 들어온다.)

- 아무것도 안 눌렀을 때, LD1,2 모두 꺼짐.

- MID, RIGHT, LEFT switch 중 하나만을 눌렀을 때, LD2만 켜짐.

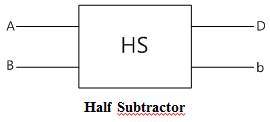
- MID, RIGHT, LEFT switch 중 2개만을 눌렀을 때, LD1만 켜짐.

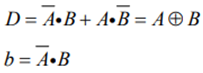
- MID, RIGHT, LEFT switch 중 3개 다 눌렀을 때, LD1, 2 둘 다 켜짐.

**2. Full Subtracter & Half Subtracter**

**1-1. Half Subtracter**

- Half Subtracter는 input A, B의 A-B를 계산하는 회로로, output D = (A-B < 0) ? (2+A-B) : A-B이고, b = (A-B < 0) ? 1 : 0이다.



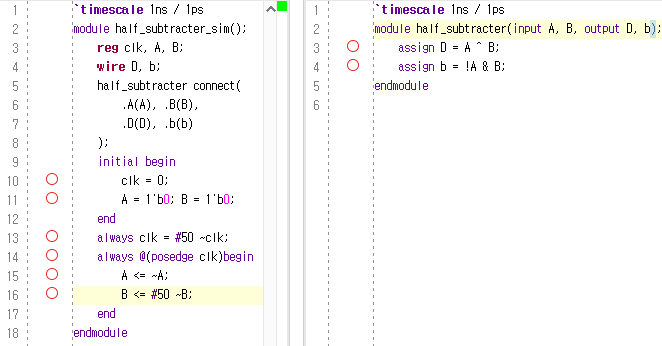
다음은 진리표와 논리식이다.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **D** | **b** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

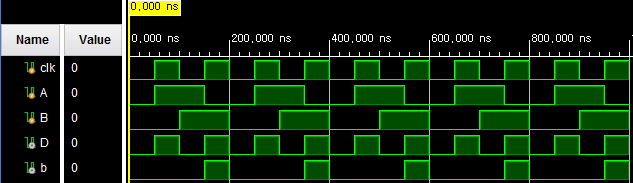
(논리식과 진리표 결과가 일치)

Verilog로 구현해 보자. (code, simulation, schematic, FPGA)

1.code ( ~, ! : NOT, & : AND, ^ : XOR )



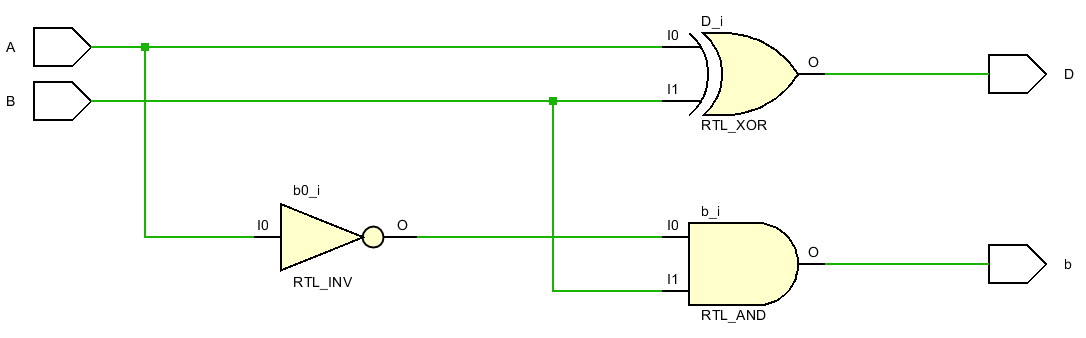
2.simulation



Input : A, B

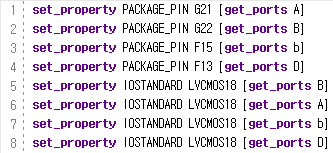
Output : D, b

D는 A와 B의 값이 다를 때, b는 A < B일 때 1의 값을 가진다.

3. schematic

4. FPGA

- Design Sources경로에 code 짜기 -> Run Synthesis & Run Implementation -> Constraints경로에 .xdc 파일 추가하기 -> Open Implemented Design 클릭 후 Constraints Wizard 클릭(Define Target, .xdc파일 타겟 설정) -> Window 탭에서 I/O ports 클릭 -> 원하는 pin 선택 및 LVCMOS18(I/O Std) 선택 -> 저장 후 .xdc reload(아래 그림처럼 코드가 자동으로 짜서 나온다.) -> Generate Bitstream 클릭(Synthesis, Implementation도 자동으로 실행) 후에 Open Hardware Manager 클릭 -> Open Target 클릭 후 Auto Connect 클릭 -> Program Device 클릭 -> FPGA에 업로드 끝

<- input, output 변수와 I/O port 연결

- A는 MID, B는 RIGHT switch와 연결

- b는 LD1, D는 LD2와 연결

- MID만 눌렀을 때 LD2가 켜진다.

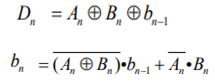
- RIGHT만 눌렀을 때 LD1과 LD2 둘 다 켜진다.

- 나머지의 경우 LD1, 2 둘 다 꺼진다.

**1-2. Full Subtracter**

- Full Subtracter는 아래 그림처럼 A와 B, bn-1을 입력하여 D = (A-B-bn-1  < 0) ? (2+A-B-bn-1) : (A-B- bn-1)와 bn = (A-B-bn-1  < 0) ? 1:0 를 출력하는 회로다.



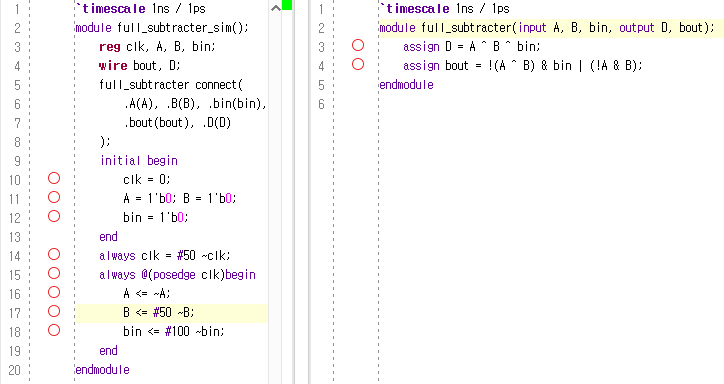
다음은 진리표와 논리식이다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **bn-1** | **D** | **bn** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

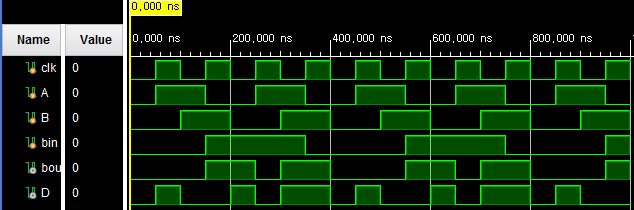
(논리식과 진리표 결과가 일치)

Verilog로 구현해 보자. (code, simulation, schematic, FPGA)

1. code ( ~, ! : NOT, ^ : XOR, & : AND, | : OR )



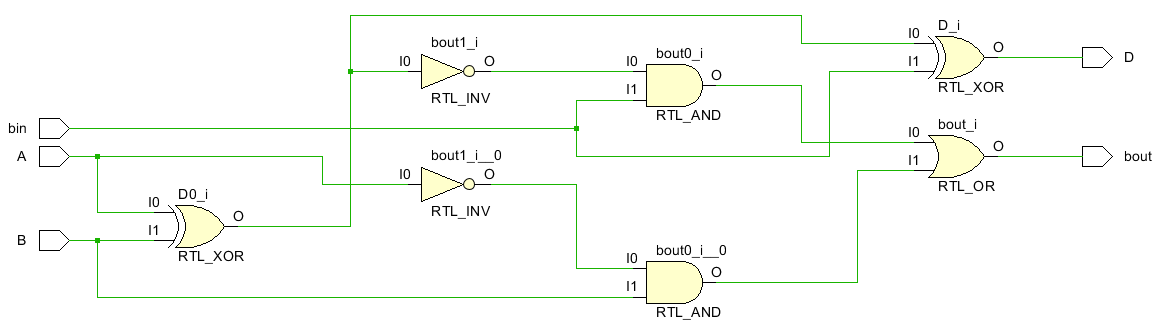
2. simulation



Input : A, B, bin

Output : bout, D

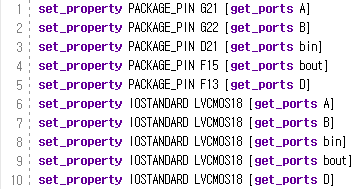
(D는 Input 중 홀수개가 1일 때, bout은 A와 B의 값이 같고 bin이 1이거나 A는 0 B는 1일 때 출력 값 1을 가진다.)

3. schematic

D0\_i (첫 번째 XOR gate)의 출력 값이 D와 bout의 출력 값을 계산할 때 공통으로 쓰인다.

4. FPGA

- Design Sources경로에 code 짜기 -> Run Synthesis & Run Implementation -> Constraints경로에 .xdc 파일 추가하기 -> Open Implemented Design 클릭 후 Constraints Wizard 클릭(Define Target, .xdc파일 타겟 설정) -> Window 탭에서 I/O ports 클릭 -> 원하는 pin 선택 및 LVCMOS18(I/O Std) 선택 -> 저장 후 .xdc reload(아래 그림처럼 코드가 자동으로 짜서 나온다.) -> Generate Bitstream 클릭(Synthesis, Implementation도 자동으로 실행) 후에 Open Hardware Manager 클릭 -> Open Target 클릭 후 Auto Connect 클릭 -> Program Device 클릭 -> FPGA에 업로드 끝

<- input, output 변수 I/O port 연결

- A, B, bin 각각 MID, RIGHT, LEFT switch와 연결

- bout, D 각각 LD1, LD2 연결

- RIGHT 또는 LEFT 버튼 한 개만 눌렀을 때 LD1, LD2 둘 다 켜진다.

- MID, RIGHT, LEFT 버튼 모두 눌렀을 때 LD1, LD2 둘 다 켜진다.

- MID 버튼 한 개만 눌렀을 때 LD1 한 개만 켜진다.

- RIGHT, LEFT 버튼 두 개 눌렀을 때 LD2 한 개만 켜진다.

- 나머지의 경우는 안 켜진다.

**3. 8421(BCD)-2421 Code Converter**

|  |  |  |
| --- | --- | --- |
| **Decimal** | **8421 BCD** | **2421 BCD** |
| **ABCD** | **XYZW** |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0010 |
| 3 | 0011 | 0011 |
| 4 | 0100 | 0100 |
| 5 | 0101 | 1011 |
| 6 | 0110 | 1100 |
| 7 | 0111 | 1101 |
| 8 | 1000 | 1110 |
| 9 | 1001 | 1111 |
| 10 | 1010 | xxxx |
| 11 | 1011 | xxxx |
| 12 | 1100 | xxxx |
| 13 | 1101 | xxxx |
| 14 | 1110 | xxxx |
| 15 | 1111 | xxxx |

8421 BCD를 2421 BCD로 변환한다.

<X의 K-map>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 0 | x | 1 |
| **01** | 0 | 1 | x | 1 |
| **11** | 0 | 1 | x | x |
| **10** | 0 | 1 | x | x |

<SOP> X = A + BD + BC

X = ( ( (C`D` )`B )`A` )` (NAND꼴)

<POS> X = (A+C+D)(A+B)

X = (( A + ( (C + D)` + B` )` )`)` (NOR꼴)

<Y의 K-map>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | x | 1 |
| **01** | 0 | 0 | x | 1 |
| **11** | 0 | 1 | x | x |
| **10** | 0 | 1 | x | x |

<SOP> Y = A + BD` + BC

Y = ( ( (C`D)`B )`A` )` (NAND꼴)

<POS> Y = (A+B)(A+C+D` )

Y = (( ((C | D`)` | B`)` | A )`)` (NOR꼴)

<Z의 K-map>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 0 | x | 1 |
| **01** | 0 | 1 | x | 1 |
| **11** | 1 | 0 | x | x |
| **10** | 1 | 0 | x | x |

<SOP> Z = A + B`C + BC`D

Z =( ((A`C`)`B`)`( ((C`D)`A`)`B)` )` (NAND꼴)

<POS> Z = (B`+C` )(A+B+C)(B`+D)

Z = (((C|D`)` | B`)` | ( ( (B|C)`)` | A)`)`

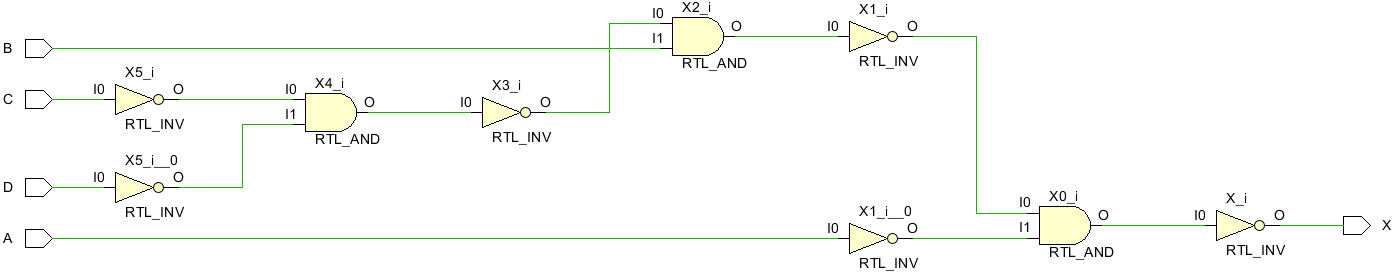
<W의 K-map>

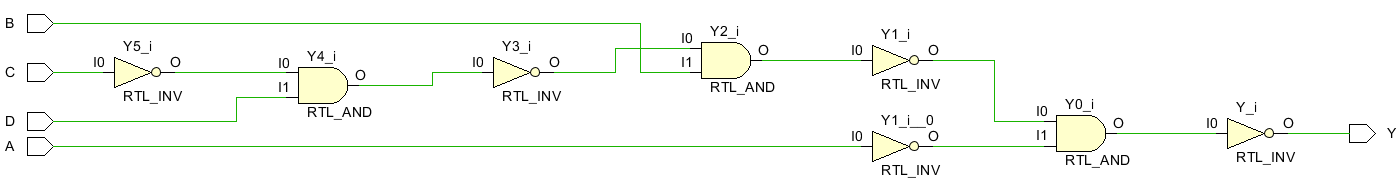
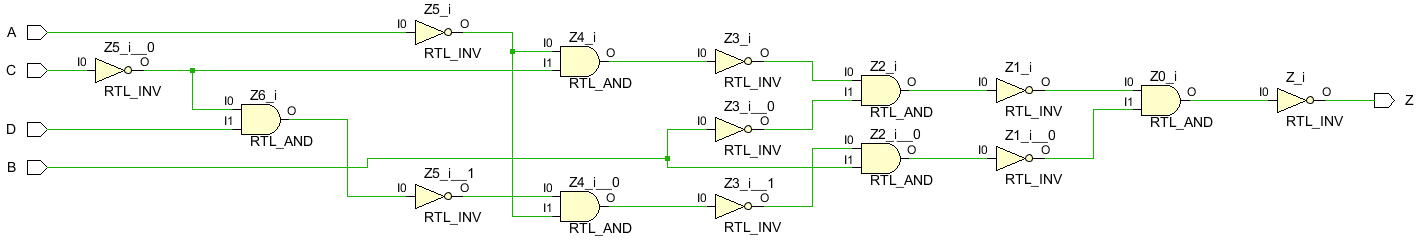
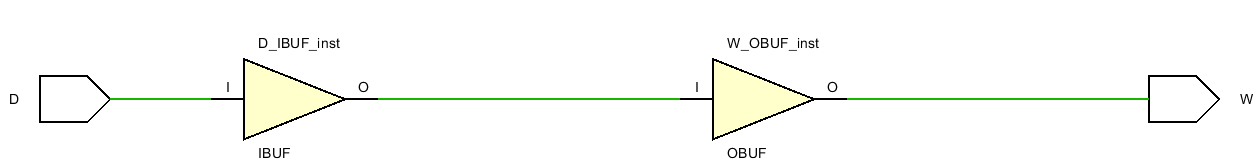
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 0 | x | 0 |
| **01** | 1 | 1 | x | 1 |
| **11** | 1 | 1 | x | x |
| **10** | 0 | 0 | x | x |

<SOP> W = D

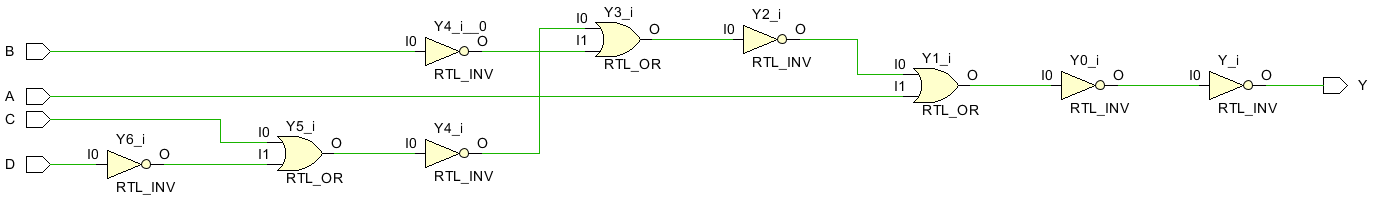
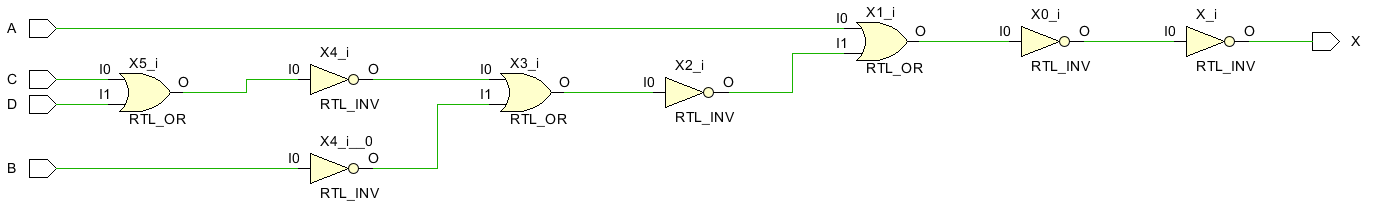
<POS> W = D

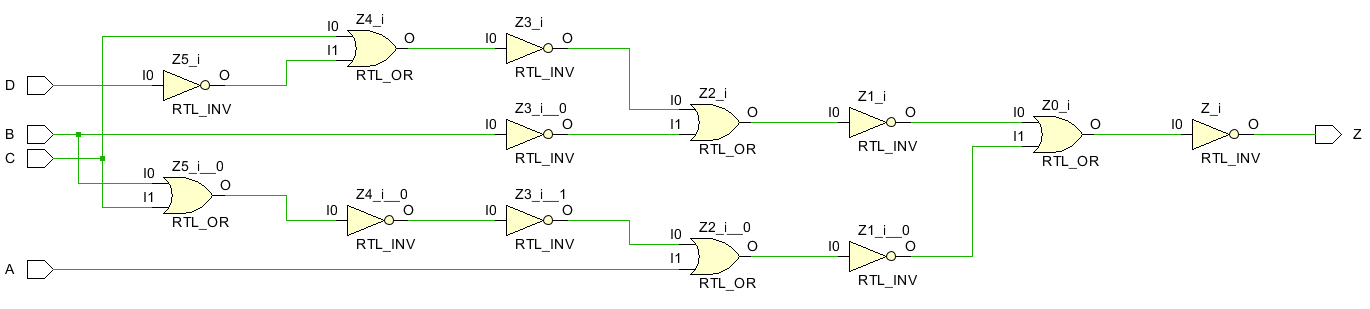
NAND, NOR형태로 논리회로를 구성해 보자.

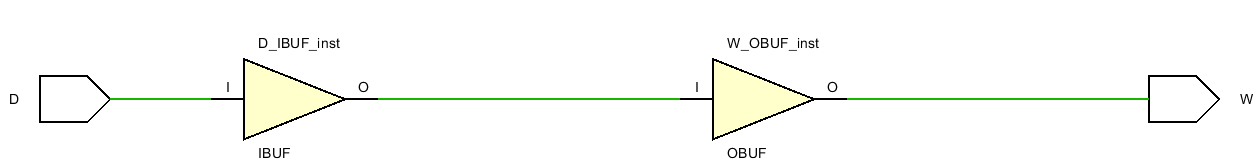
<NAND> (논리식은 K-map 옆 참조)



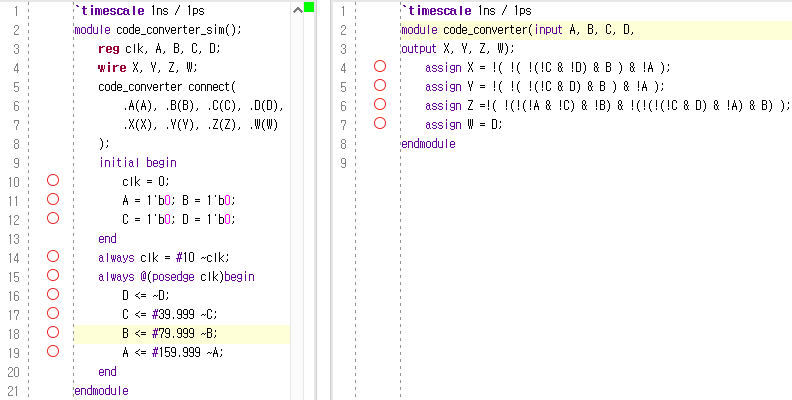
<NOR> (논리식은 K-map 옆 참조)



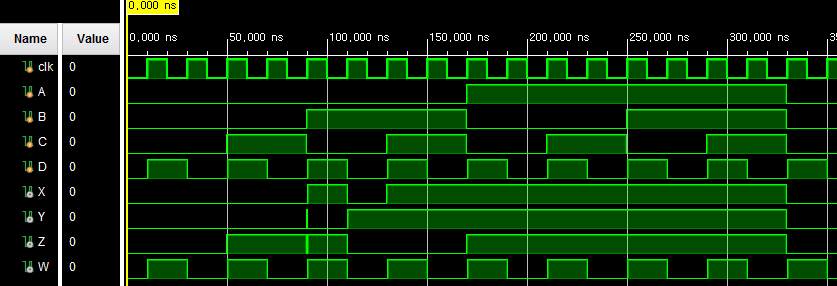


Verilog로 NAND형태를 구현해 보자. (code, simulation, schematic, FPGA)

1. code ( ~,! : NOT, & : AND ) ( !( A & B)꼴이 A NAND B이다. )



2. simulation



Input A, B, C, D

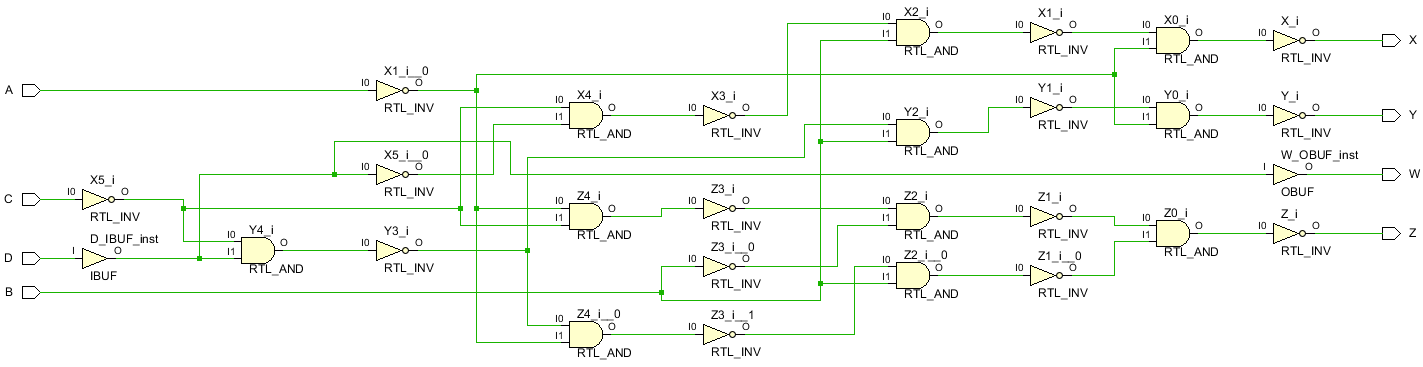
Output X, Y, Z, W

Y, Z부분 중간에 선은 code에서 Input 값을 x9.999ns 후에 바뀌도록 했기 때문이다.

210ns 이전까지 진리표에 나온 패턴 순서와 비슷(D를 제외하고 패턴이 같다. D는 0101..->1010…)

210ns부터는 Don’t Care값이므로 살펴 볼 필요가 없다.

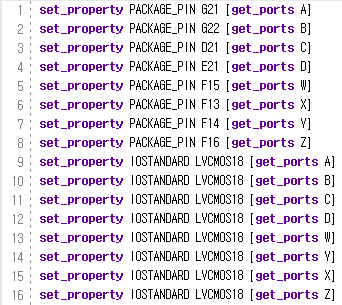
3. schematic



X, Y, Z, W 개별적인 schematic은 앞서 제시한 NAND 형태 논리회로 참조

4. FPGA

- Design Sources경로에 code 짜기 -> Run Synthesis & Run Implementation -> Constraints경로에 .xdc 파일 추가하기 -> Open Implemented Design 클릭 후 Constraints Wizard 클릭(Define Target, .xdc파일 타겟 설정) -> Window 탭에서 I/O ports 클릭 -> 원하는 pin 선택 및 LVCMOS18(I/O Std) 선택 -> 저장 후 .xdc reload(아래 그림처럼 코드가 자동으로 짜서 나온다.) -> Generate Bitstream 클릭(Synthesis, Implementation도 자동으로 실행) 후에 Open Hardware Manager 클릭 -> Open Target 클릭 후 Auto Connect 클릭 -> Program Device 클릭 -> FPGA에 업로드 끝

<- input, output 변수 I/O port에 연결

- input A, B, C, D는 각각 MID, RIGHT, LEFT, UP switch와 연결

- output X, Y, Z, W는 각각 LD1, LD2, LD3, LD4와 연결

- switch를 눌렀을 때 아래 표대로 불이 들어온다.

(A, B, C, D는 0 : switch no push, 1 : switch push이고 X, Y, Z, W는 0 : LED 꺼짐, 1 : LED 켜짐)

|  |  |
| --- | --- |
| **ABCD** | **XYZW** |
| 0000 | 0000 |
| 0001 | 0001 |
| 0010 | 0010 |
| 0011 | 0011 |
| 0100 | 0100 |
| 0101 | 1011 |
| 0110 | 1100 |
| 0111 | 1101 |
| 1000 | 1110 |
| 1001 | 1111 |
| 1010 | 1110 |
| 1011 | 1111 |
| 1100 | 1110 |
| 1101 | 1111 |
| 1110 | 1110 |
| 1111 | 1111 |

**4. 검토 및 기타 논의**

- K-map의 SOP 또는 POS minimize가 여러 개의 형태로 나올 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | x | 1 |
| **01** | 0 | 0 | x | 1 |
| **11** | 0 | 1 | x | x |
| **10** | 0 | 1 | x | x |
| **AB**  **CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | x | 1 |
| **01** | 0 | 0 | x | 1 |
| **11** | 0 | 1 | x | x |
| **10** | 0 | 1 | x | x |

위 map과 아래 map의 파란색 가로 직사각형 묶음을 보면 서로 다르게 묶였음을 알 수 있다. 그러나 둘 다 최소 POS를 충족함을 알 수 있다. 즉 여러 개의 형태가 존재한다. 이는 Don’t Care의 존재 때문으로 보인다.

- NAND gate혹은 NOR gate만으로 gate를 구성하는 것은 숙련자가 아닌 이상 상당히 껄끄럽다

Gate를 얼마나 쓰든 상관이 없다면 A = A \* 1, A = A + 0 등으로 (AB + CD)\_nand gate 와 (A+B)(C+D)\_NOR gate 꼴로 만들어 갈 수 있으나 최소 gate를 쓰는 조건이 붙는다면 상당히 껄끄러웠다.

Z = (((C|D`)` | B`)` | ( ( (B|C)`)` | A)`)` code converter의 Z output을 NOR gate꼴로 나타낼 때 NOR를 쓰기 위해 inverter를 두 번 쓴 것이 보인다, ((B|C)`)`. 최대한 적은 gate(트랜지스터)를 사용하고자 했지만 이것보다 최소한으로 사용할 수 없다는 증명을 할 수가 없었다.