컴퓨터공학실험2 6주차 결과 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. 실험 목적

2. Full Adder 및 Half Adder의 simulation 결과 및 과정 설명

3. Full Subtractor 및 Half Subtractor의 simulation 결과 및 과정 설명

4. 8421(BCD)-2421 Code converter의 simulation 결과 및 과정 설명

5. 결과 검토 및 논의 사항

**1. 실험 목적**

Adder와 Subtractor이 실제 회로에서 어떻게 구성되는지 알아본다. 이전 carry를 input으로 받는 Full Adder, borrow를 input으로 받는 Full Subtractor 또한 구현한다. 8421(BCD) code 를 2421 code로 변환할 때, Karnaugh map을 통해서 논리식 간소화(최소화)를 수행한 SOP, POS 형태로 표현해본다. 또한 NAND gate를 통해 SOP 형태를 표현한다.

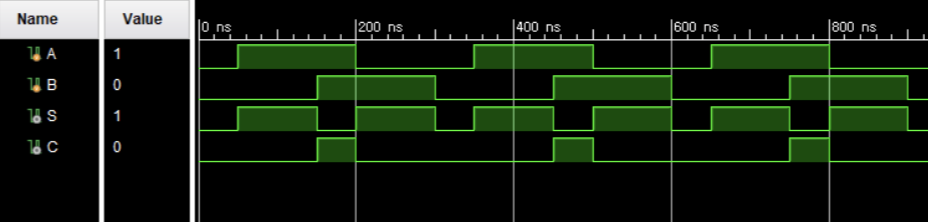
**2. Full Adder 및 Half Adder의 simulation 결과 및 과정**

회로에서 1bit Half Adder은 2-input, 2-output 의 형태를 가진다. input A, B에 대해서 더한 결과로는 00, 01, 10 3가지 결과가 나올 수 있는데, 이를 각각 경우에 맞춰서 output S, 또는 output C로 보내주는 역할을 수행한다. 이에 대한 논리식은 Sum = A’B + AB’ (XOR gate), Carry = AB (AND gate) 로 구현할 수 있으며, 진리표는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Half Adder | | | |
| input | | output | |
| A | B | S | C |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

<Half Adder의 진리표>

또한 simulation 결과는 아래와 같다.



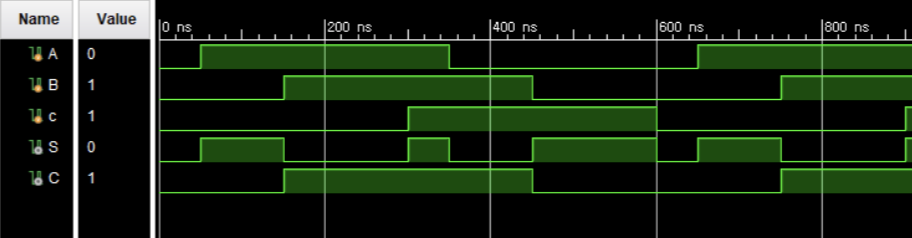
<Half Adder의 simulation 결과>

위와 다르게 Full Adder은 3-input, 2-output을 갖는 구조이다. input A, B, output S, C는 동일하지만, 이전 Full Adder에서 output으로 나온 C가 새로운 input(=input c)으로 들어가는 형태이다. 이렇게 회로를 구현한다면 1 bit를 넘어서 여러 bit의 연산에도 작용할 수 있는 Adder를 만들 수 있다는 장점이 존재한다. 이에 대한 논리 식은 S = A^B^c(^는 XOR gate), C = AB + Bc + cA 형태이다. 또한 진리표는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Full Adder | | | | |
| input | | | output | |
| A | B | c | S | C |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

<Full Adder의 진리표>

또한 simulation 결과는 아래와 같다.



<Full Adder의 simulation 결과>

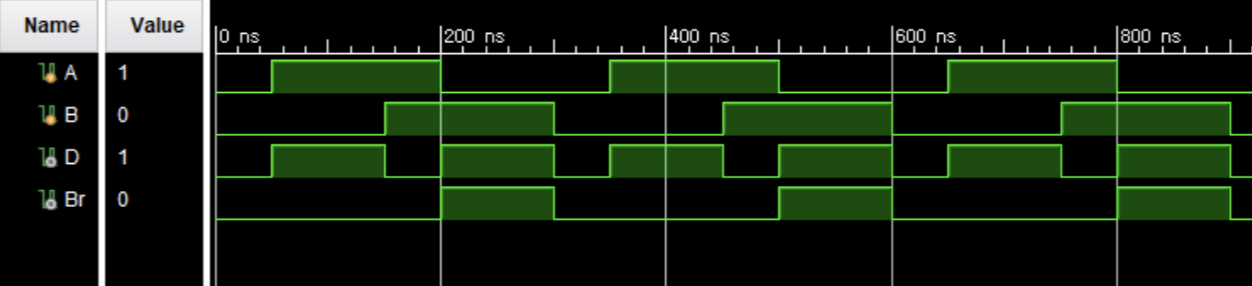
**3. Full Subtractor 및 Half Subtractor의 simulation 결과 및 과정**

우선 Half Subtractor은 위의 Half Adder와 동일하게 2-input, 2-output 형태이다. 이때 input A, B에 대해서, output D, Br을 갖게 되는데, 여기에서 D는 Adder의 S와 동일한 역할이며, Br은 0 - 1 인 형태일 때, 다음 weight에서 수를 빌림을 표시해주는 변수이다. 이에 대한 논리식은 D = A^B, Br = A’B 형태이다. 이에 대한 진리표는 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| Half Subtractor | | | |
| input | | output | |
| A | B | D | Br |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |

<Half Subtractor의 진리표>

또한 simulation 결과는 아래와 같다.



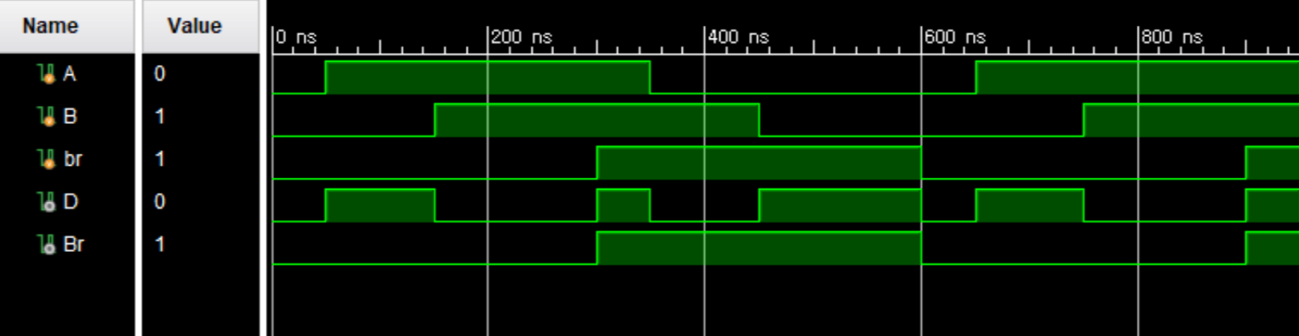
<Half Subtractor의 simulation 결과>

Full Subtractor 또한 Full Adder와 비슷하게 3-input, 2-output을 가지며, input에 br이 추가된 형태이다. 이때 실제 Subtract는 A – B – br 형태로 이루어지며, 각 경우에 맞게 output D, Br 정해진다. 이에 대한 논리식은 D = A Bbr + AB’br’ + A’Bbr’ + A’B’br, Br = A’B + A’br + Bbr이며, 진리표는 아래와 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Full Subtractor | | | | |
| input | | | output | |
| A | B | br | D | Br |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 |

<Full Subtractor의 진리표>

또한 simulation 결과는 아래와 같다.



<Full Subtractor의 simulation 결과>

**4. 8421(BCD)-2421 Code converter simulation 결과 및 과정**

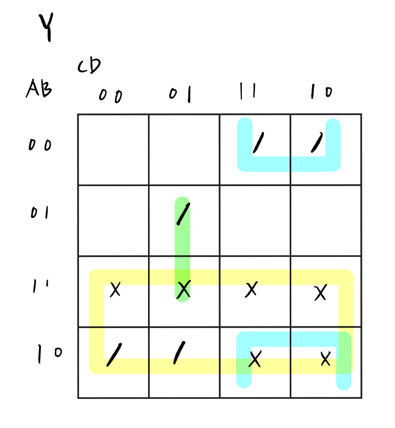
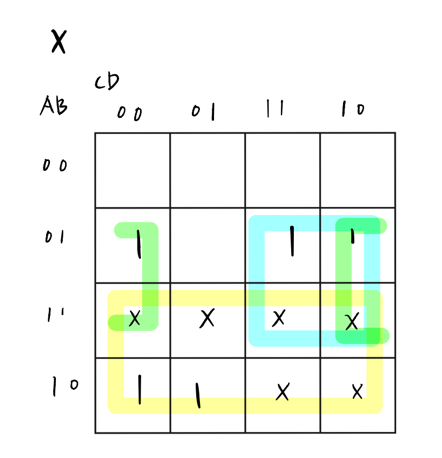
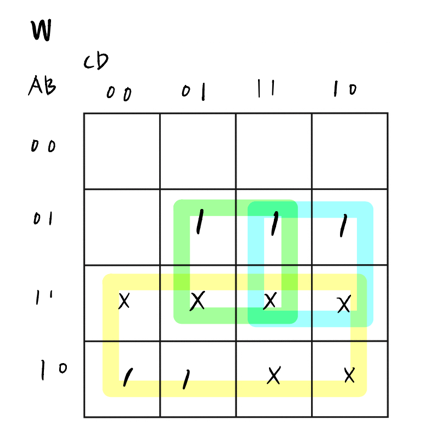
8421(BCD) code는 Binary Coded Decimal로, 10진수 숫자를 2진수로 바꿀 때 사용할 수 있는 하나의 방법이다. 2421 또한 8421 code와 비슷하게, 4bit 내에서 0~9까지의 수를 표현하는 한 방법이다. 이때 8421 code를 2421 code로 변환할 수 있는데, 이에 대한 진리표는 아래와 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8421(BCD)-2421 Code converter | | | | | | | |
| input | | | | output | | | |
| A | B | C | D | W | X | Y | Z |
| 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 |

<8421-2421 code converter 진리표>

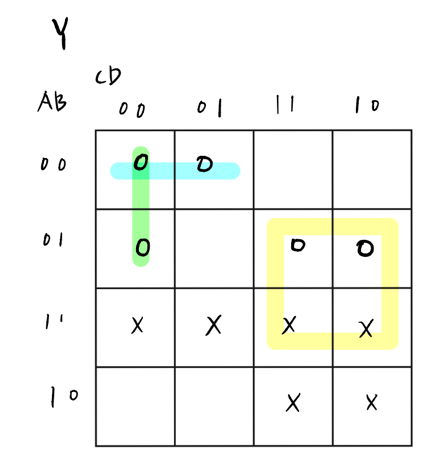
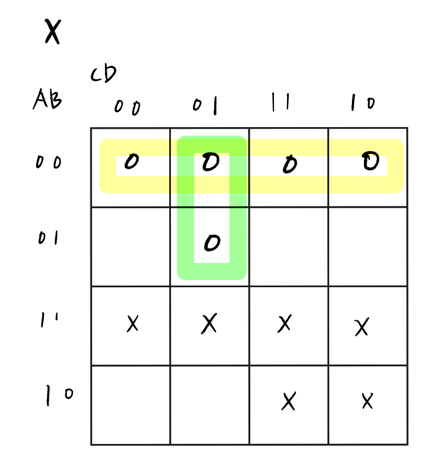
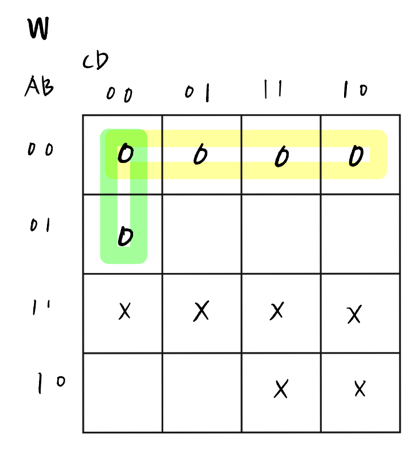
8421 code를 2421 code로 변환하는 한 방법으로는, Karnaugh map을 활용해 minimization을 거친 SOP, POS 형태로 표현할 수 있다.(또한 이는 회로로 또한 표현 가능하다.) 이에 대해 위 표에서 8421(BCD) code 의4 bit의 표현에서 1010, 1011, 1100, 1101, 1110, 1111 은 Karnaugh map에서 Don’t care term 형태가 됨을 파악할 수 있다. 이에 대한 Karnaugh map은 아래와 같다. (추가적으로 Z는 표를 통해서 D와 동일함을 간단하게 파악할 수 있다. 따라서 Z는 Karnaugh map을 거치지 않고, Z = D 형태로 표현한다.)

-SOP 표현에 대한 Karnaugh map



결과적으로 W = A + BD + BC, X = A + BD’ + BC, Y = A + BC’D + B’C, Z = D 이다.

-POS 표현에 대한 Karnaugh map



결과로 W = (A+C+D)(B+D), X = (A+B)(A+C+D’), Y = (B’+C’)(A+C+D)(A+B+C), Z = D 이다.

실습 시간에는 POS 형태로 Karnaugh map을 작성하고, 표현하며, 아래 형태로 verilog code를 작성했다.

(1) assign W = ~((~A) & (~(B&D)) & (~(B&C)));

(2) assign X = ~((~A) & (~(B&(~D))) & (~(B&C)));

(3) assign Y = ~((~A) & (~(B&(~C)&D)) & (~((~B)&C)));

(4) assign Z = D;

위에서 설명한 표현과는 약간 차이가 있다는 것을 파악할 수 있는데, SOP 표현은 NAND gate만을 이용해서 표현할 수 있다는 점을 토대로, verilog 표현에서 NAND gate만을 사용해서 표현한 것이다. 이렇게 표현했을 경우에, 한 gate만을 사용하기 때문에 실제 회로 구성에서도 비용 면에서 효율적일 수도 있다. 추가적으로 POS 표현은 NOR gate만을 이용해서 표현할 수 있는데, 이는 위 NAND 표현과 비슷하게 수행하면 되기 때문에, 이에 대한 내용은 생략한다.

**5. 결과 검토 및 논의 사항**

이번 실습을 통해서, 실제 컴퓨터에서 Adder, Subtractor이 어떻게 구성되는지에 대해서 파악하며, Karnaugh map을 활용한 conversion을 구현해낼 수 있었다. simulation 결과를 통해서도 Adder, Subtracter, 그리고 conversion 회로가 잘 작동함을 파악할 수 있었다. 이론적으로는 SOP expression은 NAND gate, POS expression은 NOR gate 만으로 구현할 수 있다는 점을 파악할 수 있었다.

이번 실습을 진행하면서 input carry, borrow, output Carry, Borrow를 단지 입력값으로의 input, 출력값으로의 output으로만 판단하고 simulation을 진행했는데, 실제 컴퓨터에서 Adder, Subtractor은 1 bit에서만 수행되지 않는 점을 파악할 수 있다. integer만 해도 4 byte(=32 bit)로, 실제 연산을 수행할 때 큰 bit 형태로 이루어짐을 파악할 수 있다. 이에 대해서도 구현해보면 좋을 것 같다는 생각이 들었다.