6주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20212022 이름: 이예준

**1.**

이번 실험의 목적은

Adder와 Subtracter의 개념 그리고 Code converter의 개념에 대해 이해하고,

Verilog를 사용하여 다양한 Adder, Subtracter 그리고Code converter 구현하고,

FPGA 보드를 이용해 Verilog로 구현된 회로의 동작을 확인하는 것이다.

**2.**

**-Full Adder**

스크린샷, 텍스트, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

\*Full Adder의 Simulation 결과

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| 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 |

\*Full Adder의 진리표

Full Adder는 3개의 Input(A, B, Cin )을 받아 2개의 Output(S, Cout)이 산출된다.

Cout은 현재 자릿수의 캐리(Carry)이고 Cin는 이전 자릿수에서 산출된 캐리(Carry)이다.

\*1과 1을 합했을 때 현재 bit범위에서 2를 표현할 방법이 없기 때문에

다음 자릿수 bit로 1을 보내는 것을 Carry라고 한다.

S는 A, B, Cin을 합했을 때 Carry로 넘어가지 않고 현재 자릿수에 있는 수를 산출한다.

S는 A와 B를 XOR연산하여 Carry로 넘어가지 않고 현재 자릿수에 남아 있는지, 그 합과

Cin도 XOR연산하여 Carry로 넘어가지 않고 현재 자릿수에 남아 있는지 확인하면서 구한다.

Cout은 A와 B를 AND연산하여 Carry가 있는지, Carry가 없고 현재 자릿수에 남아 있다면

이전 자릿수에서 넘어온 Cin과 AND연산하여 Carry가 있는지 확인한다.

이를 식으로 나타내면

이렇게 나타낼 수 있다.

**-Half Adder**

스크린샷, 텍스트, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

\*Half Adder의 Simulation 결과

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

\*Half Adder의 진리표

Half Adder는 2개의 Input(A, B)을 받아 2개의 Output(S, C)이 산출된다.

C는 현재 자릿수의 캐리(Carry)이고 S는 A, B을 합했을 때 현재 자릿수에 있는 수를

산출한다. S는 XOR연산하여 Carry로 넘어가지 않고 현재 자릿수에 남아 있는지 확인하고,

C는 A와 B를 AND연산하여 Carry가 있는지 확인한다.

이를 식으로 나타내면

이렇게 나타낼 수 있다.

**3.**

**-Full Subtracter**

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*Full Subtracter의 Simulation 결과

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| A | B | bin | D | bout |
| 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 |

\*Full Subtracter의 진리표

Full Subtracter는 3개의 Input(A, B, bin )을 받아 2개의 Output(S, bout)이 산출된다.

bout은 현재 자릿수의 Borrow이고 bin는 이전 자릿수에서 산출된 Borrow이다.

\*0에서 1을 뺄 때 음수를 표현할 방법이 없기 때문에 다음 자릿수 bit로 보내는 것을

Borrow라고 한다.

D는 A, B, bin을 뺐을 때 Borrow로 넘어가지 않고 현재 자릿수에 있는 수를 산출한다.

D는 A와 B를 XOR연산하여 Borrow로 넘어가지 않고 현재 자릿수에 남아 있는지, 그 차와

bin도 XOR연산하여 Borrow로 넘어가지 않고 현재 자릿수에 남아 있는지 확인하면서 구한다.

bout은 A의 보수와 B를 AND연산하여 Borrow가 있는지, Borrow가 없고 현재 자릿수에

남아 있다면 이전 자릿수에서 넘어온 bin과 AND연산하여 Borrow가 있는지 확인한다.

이를 식으로 나타내면

이렇게 나타낼 수 있다.

**-Half Subtracter**

스크린샷, 텍스트, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*Half Subtracter의 Simulation 결과

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | b | D |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

\*Half Subtracter의 진리표

Half Subtracter는 2개의 Input(A, B)을 받아 2개의 Output(D, b)이 산출된다.

b는 현재 자릿수의 Borrow이고 D는 A, B을 뺐을 때 현재 자릿수에 있는 수를 산출한다.

D는 A, B을 XOR연산하여 Borrow로 넘어가지 않고 현재 자릿수에 남아 있는지 확인하고,

b는 A의 보수와 B를 AND연산하여 Borrow가 있는지 확인한다.

이를 식으로 나타내면

이렇게 나타낼 수 있다.

**4.**

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

스크린샷, 디스플레이, 텍스트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*8421(BCD)-2421 Code Converter Simulation 결과

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In/Out | 8421(BCD) Code | | | | 2421 Code | | | |
| Value | a | b | c | d | x | y | z | t |
| 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 |
| 10 | 1 | 0 | 1 | 0 |  |  |  |  |
| 11 | 1 | 0 | 1 | 1 |  |  |  |  |
| 12 | 1 | 1 | 0 | 0 |  |  |  |  |
| 13 | 1 | 1 | 0 | 1 |  |  |  |  |
| 14 | 1 | 1 | 1 | 0 |  |  |  |  |
| 15 | 1 | 1 | 1 | 1 |  |  |  |  |

\*8421(BCD)-2421 Code Converter 진리표

8421코드는 가장 대표적인 BCD 코드로 각 자릿수의 bit마다 가중치를 둔다.

첫번째 bit는 20, 두번째 bit는 21, 세번째 bit는 22 그리고 마지막 네번째 bit는

23의 가중치를 가진다. 8421코드와 마찬가지로 2421코드도

첫번째 bit는 1, 두번째 bit는 2, 세번째 bit는 4 그리고 네번째 bit는 2의 가중치를 가진다.

따라서 8421코드는 0~15까지 나타낼 수 있고, 2421코드는 0~9까지 나타낼 수 있다.

위의 진리표를 보게 되면 입력값으로 a, b, c, d, 4개를 받아 x, y, z, t이 출력값으로 산출된다.

이를 이용해 카르노맵을 그려 출력식을 간소화할 수 있다.

SOP를 구할 때는 일반적으로 1을 묶어서 간소화식(F)를 구하는 방법으로 얻을 수 있지만,

POS를 구할 때는 살짝 다른 방법으로 구해야 한다.

그 방법은 0을 묶어서 F’을 구한 다음 드모르간의 법칙을 이용해 F를 구하여

POS를 얻는 방법이다.

**-x의 카르노맵**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(SOP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(POS)

**-y의 카르노맵**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 1 | 1 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(SOP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 1 | 1 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(POS)

**- z의 카르노맵**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 0 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(SOP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 1 | 0 | 0 |
| 11 |  |  |  |  |
| 10 | 1 | 1 |  |  |

(POS)

**-t의 카르노맵**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 |  |  |  |  |
| 10 | 0 | 1 |  |  |

(SOP)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ab cd | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 |  |  |  |  |
| 10 | 0 | 1 |  |  |

(POS)

**5.**

Adder와 Subtracter에 대해 Verilog 코드를 통한Simulation 결과와 FPGA보드 구현 결과를 봤을 때 출력값들이 모두 정상적으로 나오는 것을 확인할 수 있었다.

또한 Code Converter 역시 Simulation 결과를 봤을 때 정상적으로 나오는 것을

확인할 수 있었고, 카르노 맵을 통해 8421코드를 2421코드로 바꾸는 간소화식을

구할 수 있었다. 이번 실험에서 논의사항은 Converter를 통해 8421코드를 2421코드로

바꾸게 되는데, 각 코드의 특징에 대해 탐구해 보는 것이다.

8421코드의 경우 현재 사용되는 가장 대표적인 BCD코드이고, 순 2진수표현과 가장 유사하기 때문에 Encoding하기가 용이하다. 2421코드의 경우 각 자리의 2진수 0을 1로,

1을 0으로 바꿈으로써 보수를 간단히 얻을 수 있는 장점이 있다.

**6.**

가중치 코드인 BCD코드가 있다면 비가중치 코드인 그레이 코드도 존재한다.

그레이 코드는 가중치가 없어 연산에는 부적절하나 주로 전송, 입출력 장치,

아날로그-디지털 간 변환기 등에 쓰인다.

그레이 코드의 특징은 연속되는 코드들 간에 딱 하나의 bit만 변화하여 새로운 코드가 되며,

입력코드로 사용하면 오차가 적어지는 특징이 있다.

2진수 코드를 그레이 코드로 바꾸는 방법은 최상위 bit는 그대로 내려쓰고,

그 다음 그레이 bit는 앞의 2진수 bit와 그 다음 2진수 bit를 비교하여

같으면 0, 다르면 1을 내려쓴다(XOR연산).

|  |  |  |
| --- | --- | --- |
| 10진수 | 2진수 | 그레이 코드 |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

\*그레이 코드의 진리표