6주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231609 이름: 정희선

**1.**

가산기, 감산기, 부호 변환기 각각의 개념을 이해하고, verilog를 사용해 직접 구현해 본 뒤 이를 FPGA를 통해 실제로 구현된 회로가 동작하는 것을 확인하는 것이 이 실험의 목적이다.

**2.**

**2.1) Half Adder**

1. **Verilog**

*`timescale 1ns / 1ps*

*module practice(*

*input a,*

*input b,*

*output s,*

*output c*

*);*

*assign s = a^b;*

*assign c = a&b;*

*endmodule*

1. **Simulation**

텍스트, 소프트웨어, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

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

**3) Boolean 함수**

S=A’B+AB’ = A**⊕**B

C=AB

이론 상의 Boolean 함수와 동일한 결과를 도출한다.

**2.2) Full Adder**

1. **Verilog**

*`timescale 1ns / 1ps*

*module practice(*

*input a,*

*input b,*

*output s,*

*output c*

*);*

*assign s = a^b;*

*assign c = a&b;*

*endmodule*

1. **Simulation**

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

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| C | A | B | S | C |
| 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) Boolean 함수**

S= (A**⊕**B) **⊕** C\_in

C= C\_in(A**⊕**B)+AB

이론 상의 Boolean 함수와 동일한 결과를 도출한다.

**3.**

**3.1) Half Subtractor**

1. **Verilog**

*`timescale 1ns / 1ps*

*module boolean(*

*input a,*

*input b,*

*output d,*

*output borrow*

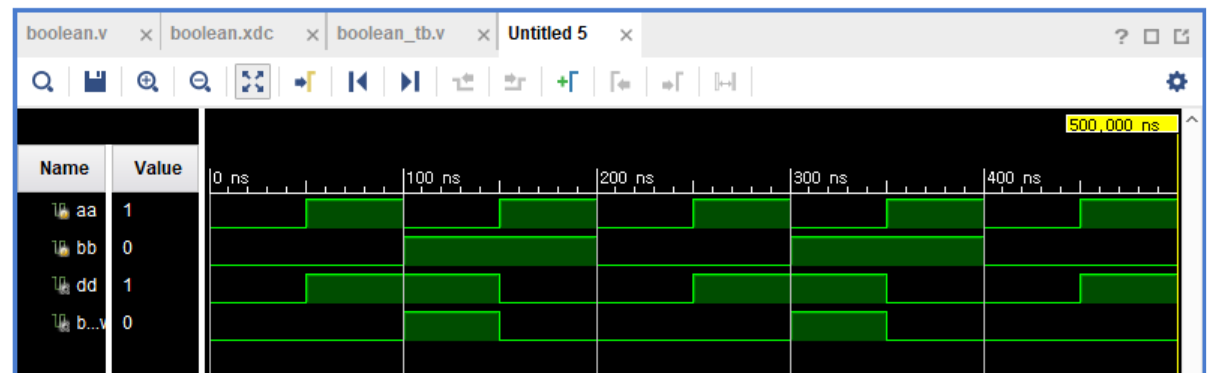
*);*

*assign d = a^b;*

*assign borrow = (~a)&b;*

*endmodule*

1. **Simulation**



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

**3) Boolean 함수**

D=A’B+AB’ = A**⊕**B

b=A’B

이론 상의 Boolean 함수와 동일한 결과를 도출한다.

**3.2) Full Subtractor**

1. **Verilog**

*`timescale 1ns / 1ps*

*module boolean(*

*input a,*

*input b,*

*input b\_n1,*

*output d,*

*output b\_n*

*);*

*assign d = a^b^b\_n1;*

*assign b\_n = (~(a^b)&b\_n1)|(~a&b);*

*endmodule*

1. **Simulation**

스크린샷, 소프트웨어, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| A\_n | B\_n | B\_n-1 | b\_n | D\_n |
| 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 |

**3) Boolean 함수**

D\_n= A\_n **⊕** B\_n **⊕** b\_n-1

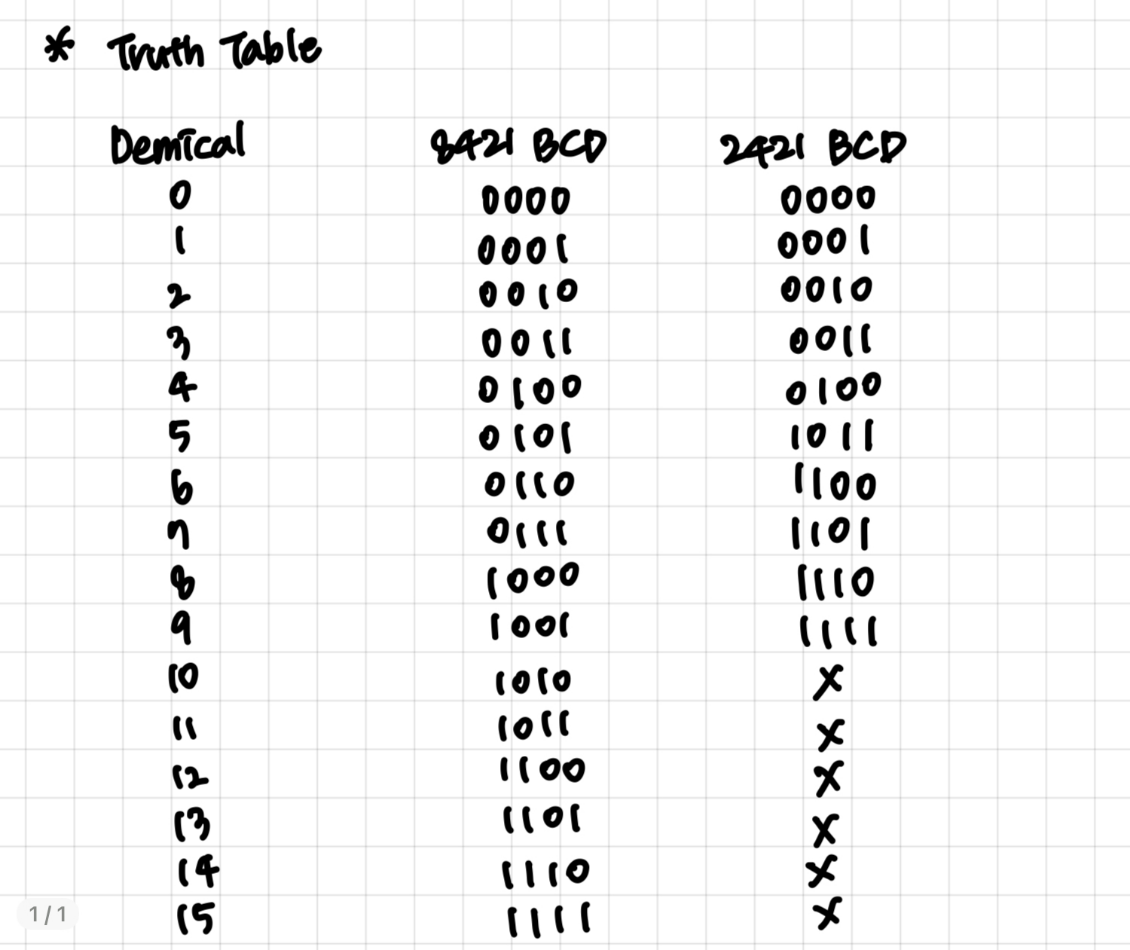
b\_n= () b\_n-1 + B\_n

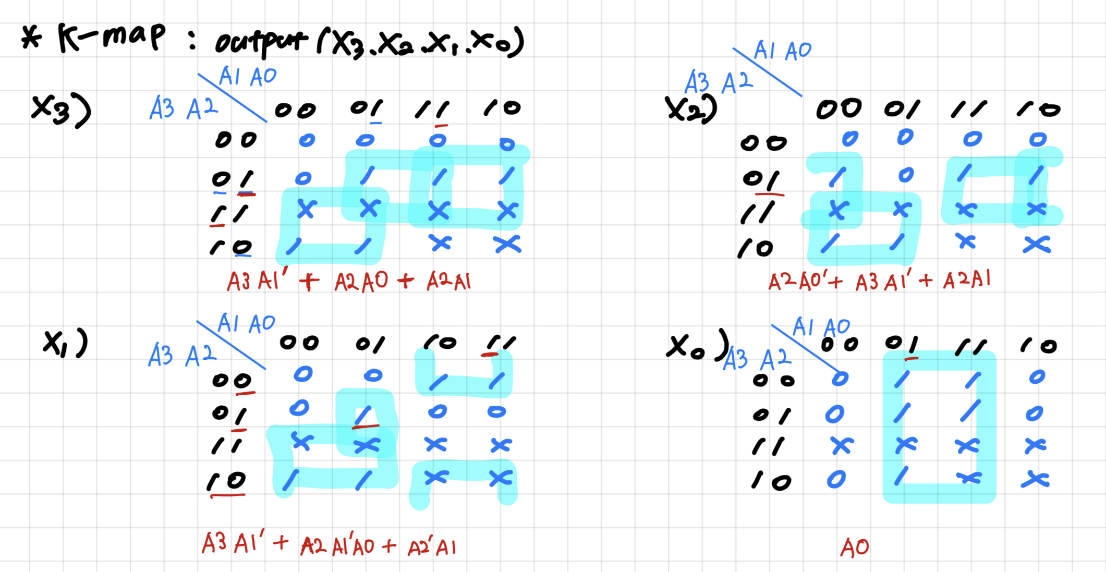
이론 상의 Boolean 함수와 동일한 결과를 도출한다.

**4.**

8421-2421 Code converter

1. **Truth Table**

****

1. **K-map**

*( truth table과 k-map은 실습시간에 직접 작성한 결과입니다. )*

**3) K-map minimization Boolean function**

|  |  |  |
| --- | --- | --- |
|  | **SOP** | **POS** |
| **X3** | **A3A1’+A2A0+A2A1** | **(A3+A1+A0)(A3+A2)** |
| **X2** | **A2A0’+A3A1’+A2A1** | **(A3+A1+A0’)(A3+A2)** |
| **X1** | **A3A1’+A2A1’A0+A2’A1** | **(A3+A1+A0)(A3+A2+A1)(A2’+A1)** |
| **X0** | **A0** | **A0** |

**4) Verilog**

*`timescale 1ns / 1ps*

*module boolean(*

*input a3,*

*input a2,*

*input a1,*

*input a0,*

*output b3,*

*output b2,*

*output b1,*

*output b0*

*);*

*assign b3 = (a3&~a1) | (a2&a0) | (a2&a1);*

*assign b2 = (a2&~a0) | (a3&~a1) | (a2&a1);*

*assign b1 = (a3&~a1) | (a2&~a1&a0) | (~a2&a1);*

*assign b0 = a0;*

*endmodule*

**5) Simulation**

**스크린샷, 텍스트, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명**

**6) NAND 형태의 Converter 구현**

위 Verilog 코드를 통해 알 수 있듯이, 현재 converter는 AND,OR,NOT gate를 사용해 구현되었다.

*assign b3 = (a3&~a1) | (a2&a0) | (a2&a1);*

*assign b2 = (a2&~a0) | (a3&~a1) | (a2&a1);*

*assign b1 = (a3&~a1) | (a2&~a1&a0) | (~a2&a1);*

*assign b0 = a0;*

우리는 이전의 실험에서 NAND gate만을 사용해서 AND,OR,NOT gate를 모두 구현할 수 있는 NAND gate의 범용성에 대해 배웠다. 따라서 위 Converter를 아래의 변형을 사용하면 NAND gate만을 사용해서 구현할 수 있다.

*A’ = A NAND A*

*A AND B = (A NAND B) NAND (A NAND B)*

*A OR B = (A’ NAND B’) = (A NAND A) NAND (B NAND B)*

**5.**

실험을 통해 full adder, half adder, full subtracter, half subtracter, code converter(8421->2421)를 각각 이해한 후 verilog로 구현해보고 동작을 simulation 해보았다. 먼저 full adder와 half adder의 simulation 결과는 boolean function을 통해 예측한 값과 일치했다. 두 개의 adder 모두 xor, and 게이틩 조합으로 설계되었다. 다음으로 full substrcter와 half subtracter의 simulation 결과 역시 boolen function을 통해 예측한 값과 일치했다. 마지막으로 code converter의 경우 simulation을 통해 8421에서 2421로 정확하게 변환되는 것을 확인할 수 있었다. 또한 NAND gate만을 사용하여 논리회로를 구성하는 방식을 검토하며 시간과 비용 측면에서 효율성이 가장 큰 NAND gate만을 사용하여 code converter를 구현하는 방법을 알아보았다. 추가적으로 binary code를 나타내는 방식이 8421과 2421말고도 다양하므로 이번 실험에서 구현한 code converter외에도 다양한 converter가 존재할 수 있다.

**6.**

**Ripple Carry Adder**: 여러 개의 full adder를 연결하여 다중 비트 덧셈을 수행하는 데 사용되는 adder이다.

**카르노 맵의 회로 최적화**: 카르노 맵은 단순히 복잡한 논리 식을 시각적으로 간단하게 표현하는 데 사용될 뿐만 아니라 SOP와 POS의 minimum form을 구현하는 데 사용될 수 있다. SOP와 POS를 표현하는 방식은 다양해서 그 minimum form을 구하기 위해서 정해진 논리적인 절차가 존재하지 없었으나, 카르노 맵에서 EPI를 찾은 뒤, 묶이지 않은 element에 대해 최소 개수의 PI를 찾은 후 그 묶음을 SOP와 POS로 나타내면 minimum form을 구할 수 있다. SOP를 구하는 방법은 앞서 이전의 실험 보고서에서 설명했듯이 1을 grouping 하는 과정으로 구할 수 있다. POS를 구하는 방법은 SOP를 구하는 방법을 약간 변형하면 되는데, F’에 대한 SOP를 구한 뒤 demorgan 법칙을 취하면 된다. F’에 대한 SOP를 구하는 과정은 F의 SOP를 구하는 과정과 유사한데, 단지 1이 아니라 0에 대해 grouping하면 된다.