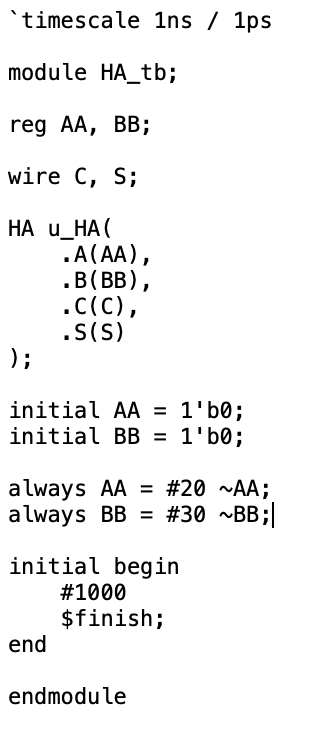
6주차 결과보고서

전공: 컴퓨터공학과/생명과학과 학년: 9학기 학번: 20181435 이름: 박다희

**1.**

이번 실험의 목적은 adder와 subtractor, code converter의 개념을 이해하고 다양한 adder와 다양한 subtracter 그리고 code converter를 verilog로 구현해보고 simulation결과를 확인하는 것이다. 또한 FPGA를 통해 verilog로 구현한 full adder, half adder, full subtractor, half subtractor, code converter의 동작을 확인해 본다.

**2.**

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

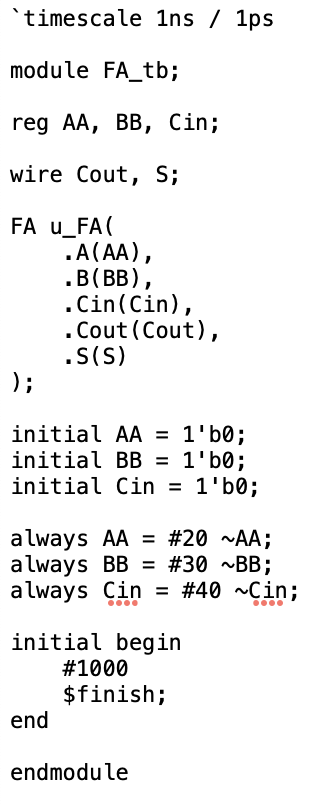
스크린샷, 텍스트, 전자 기기, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

도표, 평면도, 기술 도면, 디자인이(가) 표시된 사진

자동 생성된 설명

위의 사진은 차례대로 half adder의 test bench, design source, simulation, schematic diagram의 결과이다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

스크린샷, 디스플레이, 사각형, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

위의 사진들은 차례대로 full adder의 test bench, design source, simulation, schematic diagram의 결과이다. 이를 바탕으로 half adder와 full adder의 진리표를 작성하면 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | S(Sum) | C(Carry) |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B |  | S(Sum) |  |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

왼쪽의 표가 half adder의 진리표이고, 오른쪽의 표가 full adder의 진리표이다. Half adder는 두 개의 입력 값에 XOR gate를 실행한 값을 S라고 하므로 두 개의 입력 값 중에서 한 개의 입력값이 1일 때만 S는 1을 갖는다. C는 2개의 입력 값에 AND gate를 실행한 값으로 두 개의 입력 값이 모두 1인 경우에만 1이 된다. Full adder는 두 개의 입력 값에값까지 입력 하여 총 3개의 입력 값을 가지고 출력 값으로는 S값과 값으로 총 2개의 출력 값을 가진다. Full adder에서 S값은 세 개의 입력 값에 대한 XOR gate 실행 값이므로 세 개의 입력 값 중에서 1을 가지는 입력 값의 개수가 홀수 개이면 S값은 1을 갖는다. 값은 3개의 입력 값이 다 1이거나 값이 1이고, 나머지 두 개의 입력 값 중에서 한 개의 입력 값만 1을 가질 때 1을 갖는다.

**3.**

**텍스트, 영수증, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명**

**스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

위의 사진들은 차례대로 half subtractor의 test bench, design source, simulation, schematic diagram의 결과이다.

텍스트, 영수증, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

스크린샷, 디스플레이, 사각형, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명도표, 텍스트, 평면도이(가) 표시된 사진

자동 생성된 설명

위의 사진들은 차례대로 full subtractor의 test bench, design source, simulation, schematic diagram의 결과이다. 이를 바탕으로 half subtractor와 full subtractor의 진리표

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

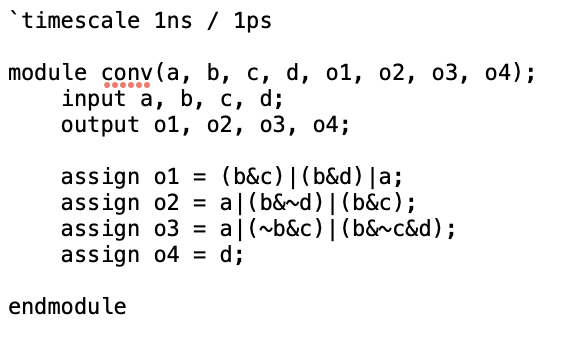
를 작성하면 다음과 같다.

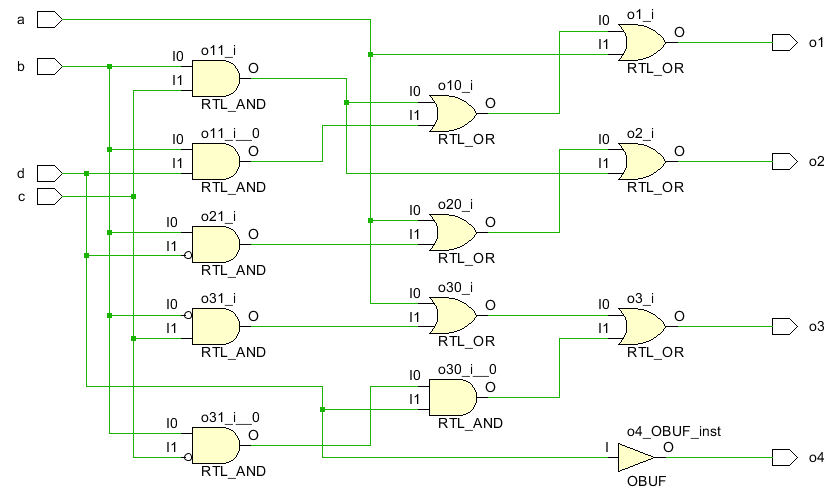
|  |  |  |  |
| --- | --- | --- | --- |
| A | B | D(Difference) | b(Borrow) |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |

왼쪽의 표가 half subtractor의 진리표고, 오른쪽의 표가 full subtractor의 진리표이다. Full Subtracter 는 2개의 입력 값과 bn값까지 총 세 개의 입력 값을 받고, D값과 b값을 출력하지만 Half Subtracter는 2개의 입력 값 만을 고려하며 D값과 b값을 출력 값으로 갖는다. Half subtractor의 경우 D의 값은 2개의 입력 값에 한 개씩 번갈아서 NOT gate실행하고 (A와 ~B), (~A와 B) 각각의 값을 AND 연산한 결과를 다시 OR 연산하였는데 이는 XOR 연산을 수행한 것과 결과가 같다. 따라서 두 개의 입력 값이 서로 다를 경우에만 D의 값이 1이 되고, B의 값은 A’B로 나타낼 수 있으므로 A가 0을 갖고 B가 1을 가질 경우에 1이 된다. Full subtractor 세 개의 입력 값에 XOR gate를 실행한 값으로 D값을 갖고, 세 개의 입력 값 중에서 1을 갖는 입력 값의 개수가 홀수이면 D의 값은 1이 된다. 또 다른 출력 값인 b는 bn값은 1이고 남은 두 개의 입력 값이 0이나 1로 같은 값을 가질 때나, A값과 B값이 각각 0과 1을 가질 때 1이 된다.

**4.** 8421(BCD)-2421 Code converter simulation 결과 및 과정에 대해서 설명하시오. (진리표 작성 및 카르노맵 SOP form, POS form 포함

텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명스크린샷, 사각형, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

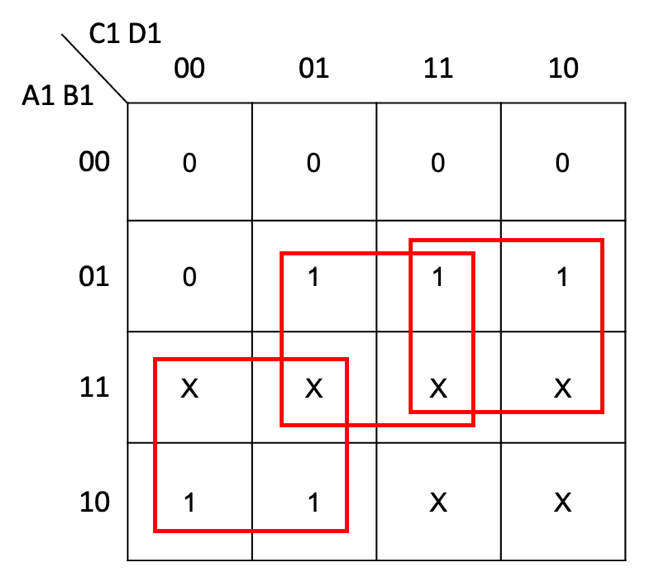
위의 사진들은 차례대로 code converter의 test bench, design source, simulation, schematic diagram의 결과이다. 이를 바탕으로 8421(BCD)code와 2421 code의 진리표를 작성하면 다음과 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8421 (BCD) CODE | | | | 2421 CODE | | | |
| a | b | c | d | O1 | O2 | O3 | O4 |
| 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 |

이러한 진리표를 바탕으로 verilog로 구현하기 위해 카르노 맵을 이용하여 각각의 출력 값에 대한 논리 함수를 구해보면 다음과 같다.

먼저 O1의 카르노맵의 결과와 SOP Form, POS Form은 차례대로 다음과 같다.

**c, d**



**a, b**

SOP form : bd + bc + a

POS form : (a’+c’+d’)(a’+b’)

O2의 카르노맵의 결과와 SOP Form, POS Form은 차례대로 다음과 같다.

도표, 텍스트, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

**a, b**

**c, d**

SOP form : ac’ + bd’ + bc

POS form : (a’+c’+d)(a’+b’)

O3의 카르노맵의 결과와 SOP Form, POS Form은 차례대로 다음과 같다.

**c, d**

c, d

도표, 텍스트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

**a, b**

a, b

SOP form : ac’ + b’c + bc’d

POS form : (a’+c’+d’)(a’+b’+c’)(b+c)

O4의 카르노맵의 결과와 SOP Form, POS Form은 차례대로 다음과 같다.

텍스트, 도표, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

**a, b**

**c, d**

SOP form : d

POS form : (c’+d’)(c+d’)

진리표를 바탕으로 카르노맵을 그려서 각각의 출력 값에 대한 논리 함수를 정리하였다. 10가지 종류의 8421 BCD code의 값들의 조합을 기준으로 4가지 종류의 2421 code에 대해 카르노맵을 그려 정리해주면 된다.

**5.**

Full adder, half adder, full subtractor, half subtractor, code converter가 어떤 논리 회로인지 알고 있기에 출력 값을 예측해볼 수 있었고, 각 논리 회로에 대한 진리표를 작성해서 카르노맵으로 정리해 코드로 구현할 수 있었다. Simulation결과를 통해 제대로 구현했는지 확인할 수 있었는데 만약 입력 값의 개수가 7개 이상이 된다면 카르노맵을 통해 논리 함수로 정리하는 것이 어려워져 canonical form을 구하는데 문제가 생긴다. 따라서 이런 경우에는 저번 주차에서 다룬 Quine-McCluskey 최소화 알고리즘을 이용하면 보다 효율적이다.

**6.**

보통 카르노맵을 통해 canonical form으로 논리 함수를 정리할 때에 SOP form으로 구한다. 따라서 이번에는 카르노맵을 통해 POS form으로 논리함수를 정리해보면 예를 들어 다음과 같은 진리표가 주어졌다고 가정해보자.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C | F | F’ |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

POS form을 구할 때는 출력 값 F를 0으로 만드는 입력 값들에 대한 합을 다 곱하면 되므로 (A+B+C)(A+B’+C)(A’+B+C) 라고 표현할 수 있고, 또는 카르노 맵을 통해 F’ 값의 SOP form의 논리함수를 구하여 보수를 취해줘도 F의 POS form을 구할 수 있다.