6주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

• 전,반가산기와 전,반감산기의 동작을 이해하고 verilog를 통해 구현한다.

• Code converter(8421 to 2421)의 동작을 이해하고 verilog를 통해 구현한다.

• 작성한 코드들을 바탕으로 simulation을 진행하여 회로의 동작을 확인한다.

• FPGA를 연결하여 회로의 동작을 확인한다.

**2.**

**2-1) Full Adder**

예비보고서에서 설명했듯 전가산기(Full Adder)는 두 개의 입력 비트를 더하는 가산기의 역할을 수행한다. 이때 반가산기와의 차이점은 단순히 두 입력 bit만 고려하는 것이 아니라 추가로 carry bit까지 고려하여 출력값을 계산하는 점이다. 출력은 결과값을 보여주는 S와 carry의 발생 여부를 보여주는 Cout으로 구성된다.

**S = (A ⊕ B) ⊕ Cin,**

**Cout = (A ⊕ B)Cin + AB**

출력과 입력의 boolean function은 위와 같다. S bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되므로 A, B, Cin의 XOR 연산을 통해 얻어진다. Cout bit는 입력 bit의 1의 개수가 2개 이상일 경우에만 1이 된다. 두 입력값 중 한 bit에 1이오고 Cin bit에 1이 오는 경우 혹은 두 입력값 모두에 1이 오는 경우 Cout bit가 1이 될 수 있다. 이를 Boolean function으로 표현하면 위와 같이 표현된다.

**그래픽이(가) 표시된 사진

자동 생성된 설명**

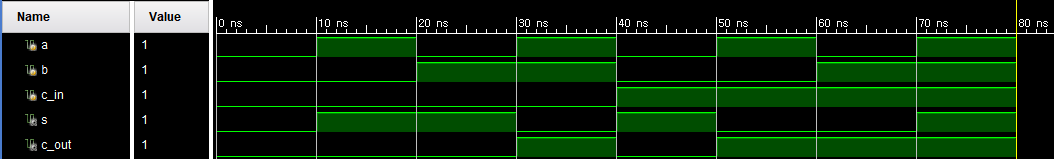
전가산기의 회로는 위와 같이 구성된다. Boolean function이 그대로 반영된 것을 알 수 있다.

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 영수증이(가) 표시된 사진

자동 생성된 설명

디자인 코드와 테스트 밴치 코드는 위와 같다. 가독성을 위해 테스트 밴치는 일부분만 첨부하였다. 전가산기의 boolean function을 그대로 구현했으며 XOR 연산은 ^ 문법을 이용했다. 입력 신호의 수가 3개이므로 각 신호를 10, 20, 40ns마다 변경시켜주어 최종적으로 80ns에 모든 신호의 변화를 확인할 수 있도록 설계하였다. 시뮬레이션 결과와 진리표는 아래와 같다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **Cin** | **B** | **A** | **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 |

시뮬레이션 결과에서 S bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되고 Cout bit는 입력 bit의 1의 개수가 2개 이상일 경우에만 1이 되는 것을 확인할 수 있다. 또한 진리표 역시 같은 결과를 표시하고 있다.

**2-2) Half Adder**

반가산기(Half Adder) 역시 두 개의 입력 비트를 더하는 가산기의 역할을 수행한다. 이때 전가산기와의 차이점은 carry bit가 아닌 단순히 두 입력 bit만 고려하여 출력값을 계산하는 점이다. 전가산기와 동일하게 출력은 결과값을 보여주는 S와 carry의 발생 여부를 보여주는 C로 구성된다.

**S = A ⊕ B**

**C = AB**

출력과 입력의 boolean function은 위와 같다. 전가산기와 달리 Cin을 고려할 필요가 없어져 회로의 복잡도가 낮아졌다. 전가산기와 같은 맥락으로 S bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되므로 A, B 의 XOR 연산을 통해 얻어진다. Cbit는 입력 bit의 1의 개수가 2개 이상일 경우에만 1이 된다. 따라서 두 입력값 모두에 1이 오는 경우 Cbit가 1이 될 수 있다. 이를 Boolean function으로 표현하면 위와 같이 표현된다.

**로고이(가) 표시된 사진

자동 생성된 설명**

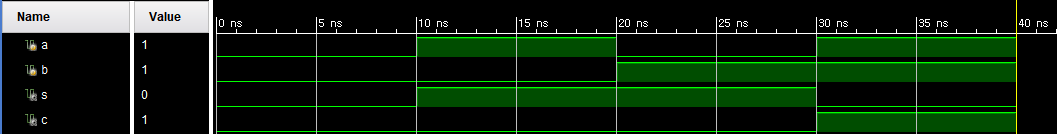
반가산기의 회로는 위와 같이 구성된다. Boolean function이 그대로 반영된 것을 알 수 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

디자인 코드와 테스트 밴치 코드는 위와 같다. 가독성을 위해 테스트 밴치는 일부분만 첨부하였다. 반가산기의 boolean function을 그대로 구현했으며 XOR 연산은 ^ 문법을 이용했다. 입력 신호의 수가 2개이므로 각 신호를 10, 20ns마다 변경시켜주어 최종적으로 40ns에 모든 신호의 변화를 확인할 수 있도록 설계하였다. 시뮬레이션 결과와 진리표는 아래와 같다.



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

시뮬레이션 결과에서 S bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되고 Cbit는 입력 bit의 1의 개수가 2개 이상일 경우에만 1이 되는 것을 확인할 수 있다. 또한 진리표 역시 같은 결과를 표시하고 있다.

**3.**

**3-1) Full Subtractor**

예비보고서에서 설명했듯 전감산기(Full Subtractor)는 두 개의 입력 비트를 빼는 감산기의 역할을 수행한다. 이때 반감산기와의 차이점은 단순히 두 입력 bit만 고려하는 것이 아니라 추가로 borrow bit까지 고려하여 출력값을 계산하는 점이다. 출력은 결과값을 보여주는 D와 버림의 발생 여부를 보여주는 Br로 구성된다.

**D = (A ⊕ B) ⊕ Br0**

**Br =** **~(A ⊕ B)Br0 + ~AB**

출력과 입력의 boolean function은 위와 같다. D bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되므로 A, B, Br0의 XOR 연산을 통해 얻어진다. Brbit는 뺄셈을 당할 입력 bit인 A의 값이 빼는 값인 B와 Br0의 합 보다 작은 경우에만 1이 된다. 두 입력값에 같은 값이 들어가고 Br0에 1이 오는 경우 혹은 A에는 0 B에는 1값이 오는 경우에만 Brbit가 1이 될 수 있다. 이를 Boolean function으로 표현하면 위와 같이 표현된다.

**도표, 라인, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명**

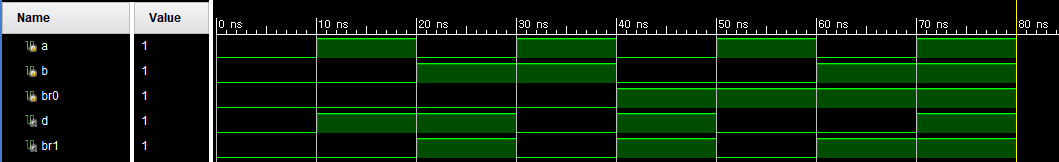
전감산기의 회로는 위와 같이 구성된다. Boolean function이 그대로 반영된 것을 알 수 있다.

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명 텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

디자인 코드와 테스트 밴치 코드는 위와 같다. 가독성을 위해 테스트 밴치는 일부분만 첨부하였다. 전가산기의 boolean function을 그대로 구현했으며 XOR 연산은 ^ 문법을 이용했다. 또한 Br의 연산식이 복잡하므로 연산 우선순위를 주의하며 구현했다. 입력 신호의 수가 3개이므로 각 신호를 10, 20, 40ns마다 변경시켜주어 최종적으로 80ns에 모든 신호의 변화를 확인할 수 있도록 설계하였다. 시뮬레이션 결과와 진리표는 아래와 같다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **An** | **Bn** | **bn-1** | **bn** | **Dn** |
| 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 |

시뮬레이션 결과에서 D bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되는 것과 Brbit는 뺄셈을 당할 입력 bit인 A의 값이 빼는 값인 B와 Br0의 합 보다 작은 경우에만 1이 되는 것을 확인할 수 있다. 또한 진리표 역시 같은 결과를 표시하고 있다.

**3-2) Half Subtractor**

반가산기(Half Adder) 역시 두 개의 입력 비트를 빼는 감산기의 역할을 수행한다. 이때 전가산기와의 차이점은 borrow bit가 아닌 단순히 두 입력 bit만 고려하여 출력값을 계산하는 점이다. 전감산기와 동일하게 출력은 결과값을 보여주는 S와 borrow의 발생 여부를 보여주는 Br로 구성된다.

**D = A ⊕ B**

**Br = ~AB**

출력과 입력의 boolean function은 위와 같다. 전감산기와 달리 Br0를 고려할 필요가 없어져 회로의 복잡도가 낮아졌다. 전감산기와 같은 맥락으로 D bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되므로 A, B 의 XOR 연산을 통해 얻어진다. 또한 A에는 0 B에는 1값이 오는 경우에만 Brbit가 1이 될 수 있다. 이를 Boolean function으로 표현하면 위와 같이 표현된다.

**도표, 라인, 그림, 스케치이(가) 표시된 사진

자동 생성된 설명**

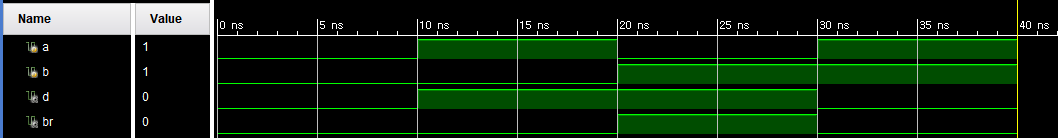
반감산기의 회로는 위와 같이 구성된다. Boolean function이 그대로 반영된 것을 알 수 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

디자인 코드와 테스트 밴치 코드는 위와 같다. 가독성을 위해 테스트 밴치는 일부분만 첨부하였다. 반감산기의 boolean function을 그대로 구현했으며 XOR 연산은 ^ 문법을 이용했다. 입력 신호의 수가 2개이므로 각 신호를 10, 20ns마다 변경시켜주어 최종적으로 40ns에 모든 신호의 변화를 확인할 수 있도록 설계하였다. 시뮬레이션 결과와 진리표는 아래와 같다.



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

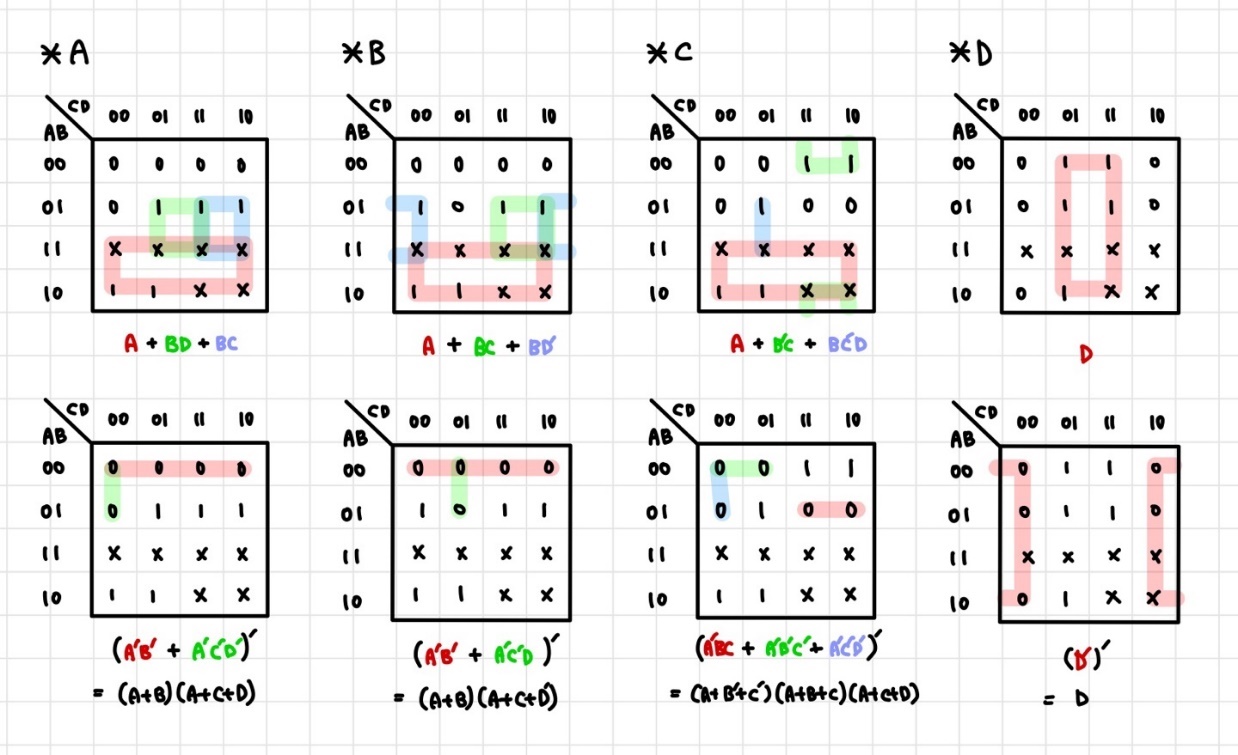
시뮬레이션 결과에서 D bit는 입력 bit의 1의 개수가 홀수일 경우에만 1이 되는 것과 bbit는 A의 값이 B의 값보다 작은 경우에만 1이 되는 것을 확인할 수 있다. 또한 진리표 역시 같은 결과를 표시하고 있다.

**4.**

**4-1) Truth Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 8421 (BCD) CODE | | | | 2421 CODE | | | |
| A1 | B1 | C1 | D1 | A2 | B2 | C2 | D2 |
| 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 |

**4-2) 카르노맵**

****

**4-3) 구현**

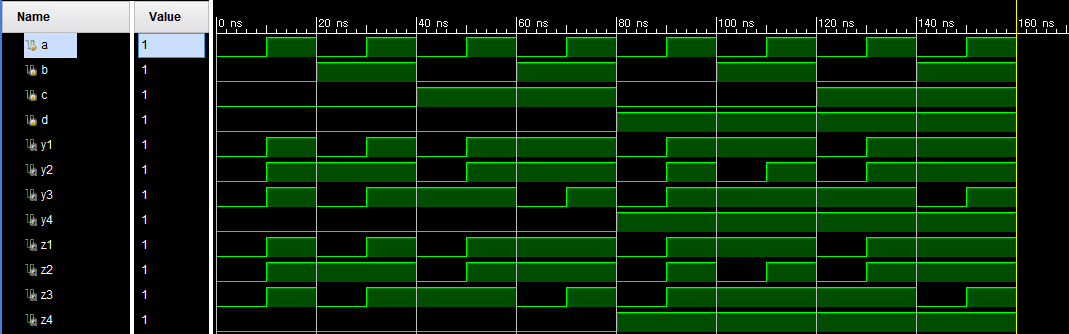
진리표를 바탕으로 카르노맵을 그리고 SOP, POS 두 형태로 나타내어 8421 to 2421 code converter의 boolean function을 구현했다. 이를 바탕으로 디자인 코드를 아래와 같이 작성했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

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

자동 생성된 설명

카르노 맵을 통해 구한 Boolean function을 그대로 사용했고 입력 신호의 수가 4개이므로 각 신호를 10, 20, 40, 80ns마다 변경시켜주어 최종적으로 160ns에 모든 신호의 변화를 확인할 수 있도록 설계하였다. 시뮬레이션 결과는 아래와 같다.



**5.**

**5-1) 가산기와 감산기**

반가산기와 반감산기의 경우 하위 비트에서 Carry와 Borrow를 고려하지 않는다. Boolean function을 구현할 때 이러한 특성이 잘 나타났다. 또한 이 때문에 2비트 이상의 연산에서 정확한 결과를 얻기 위해서는 전가산기와 전감사기를 사용해야 한다. 이 때 전가산기와 전감산기는 각 각 반가산기와 반감산기 두 개를 이용하여 구현할 수 있다는 것을 schematic을 통해 확인할 수 있다.

**5-2) Converter**

정해진 논리식이 없다면 진리표를 만들고 그를 바탕으로 카르노맵을 그려 직접 논리식을 도출해야 한다. 8421 to 2421 converter 역시 이 과정을 거쳤으며, 추가적으로 SOP와 POS 두 형태를 구했는데 이들의 결과값은 당연히 같아야 한다. 이를 시뮬레이션 결과를 통해 확인할 수 있었다.

**6.**

**6-1) NAND, NOR 게이트의 활용**

NAND와 NOR만을 활용하여 회로 구성 시 비용이 크게 절감된다. 그러나 NAND와 NOR 게이트 만을 사용하기 위해선 기존 회로를 변형할 필요가 있다. 이때 5주차 실습의 드모르간 법칙을 적극 활용할 수 있다. 드모르간 법칙에 따라

**A + B = (A’ x B’)’**

**A x B = (A’ + B’)’**

가 성립한다. 즉, OR 게이트는 NAND로 AND 게이트는 NOR로 변환이 가능하다. 그림으로 나타내면 아래와 같다.

도표, 스케치, 그림, 라인 아트이(가) 표시된 사진

자동 생성된 설명

**6-2) 다양한 디지털 코드**

8421코드와 2421코드 외에 다양한 디지털 코드가 존재한다.

1) 5421 코드

네번째 비트가 5의 가중치를 갖게 되는 디지털 코드로 주로 카드 결제 시스템에 사용되는 코드 체계이다. 코드를 통해 카드의 유효성을 확인하고 정보를 인식할 수 있다.

2) Excess -3

BCD 코드의 변형으로 십진수 숫자를 나타내기 위해 3을 더하고 그 결과를 4 비트 이진 코드로 표현한 코드이다. 예를 들어 0011이 0을 나타내고 1100이 9를 나타낸다.

3) 그레이 코드

아날로그-디지털 변환기 혹은 I/O 장치의 코드로 주로 쓰이는 코드이다. 연속되는 수에서 하나의 비트만을 변화시킨다. 예시는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| 0000 (0) | 0001 (1) | 0011 (2) |
| 0010 (3) | 0110 (4) | 0111 (5) |
| 0101 (6) | 0100 (7) | 1100 (8) |