6주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

**1. 목적**

가산기(Adder)와 감산기(Subtractor)에 대해 이해한다. 진리표를 작성하고, k-map을 적용시켜본다. K-map으로 최소화된 불 함수를 구한다. 이를 바탕으로 각 full / half 감가산기를 직접 코드로 구현해, FPGA보드에서 동작해 본다. 또한, BCD converter를 이해하고, 이를 바탕으로 8421 to 2421 코드 컨버터를 위와 같은 방법으로 불 함수를 구하고 이를 코드로 구현한다. 마찬가지로, 구현한 코드를 FPGA보드를 통해 동작해 본다.

**2. Full Adder / Half Adder**

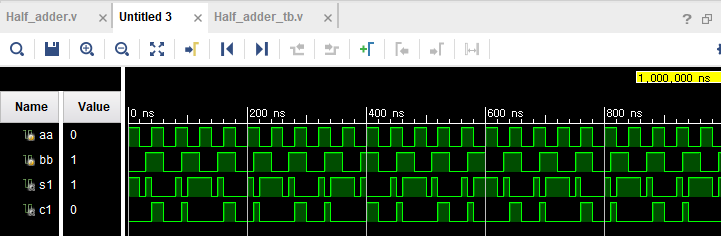
2-1**.** Half Adder

텍스트이(가) 표시된 사진

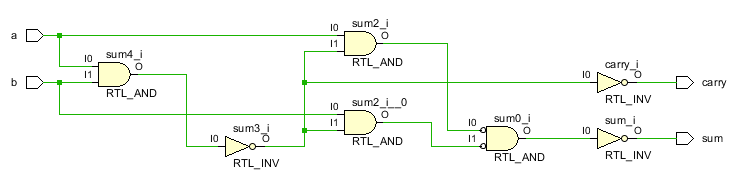
자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명

왼쪽 코드는 실습 때 배운, Half Adder 회로 그대로 코딩한 것이다. 오른쪽의 경우 NAND게이트로 변환하여 코딩한 것이다. 테스트 배치파일을 바꾸지 않았을 때 시뮬레이션 결과가 모두 동일하게 작동함을 알 수 있다. Sum의 경우 기존 A XOR B로 표현되었는데, XOR게이트는 (A’B) + (AB’)이다. 여기서, sum = A(A’+B’) + B(B’ + A’) 로 바꿀 수 있다. ( AA’ = 0, BB’ = 0 ) 이제, 드 모르간 법칙을 이용하여, sum’ = (A’ + AB)(B’ + AB) => sum = ((A’ + AB)(B’ + AB))’ 이다. 또한, (A’ + AB)’ = A(A’+B’) 이고 (A’ + AB) = (A(A’+B’))’ = (A(AB)’)’ 이다. 복잡하지만, 최종적으로 sum = ( (A(AB)’)’(B(AB)’)’ )’ 이 되어 NAND로 표현이 가능하다! Carry의 경우엔 AB이므로 쉽게 ~(AB)’ 로 표현 가능하다. Sum의 경우 A와 B가 서로 다를 때 1을 출력하며, Carry의 경우 모두 1일 때 1을 출력한다. 스캐메틱과 진리표는 아래와 같다.



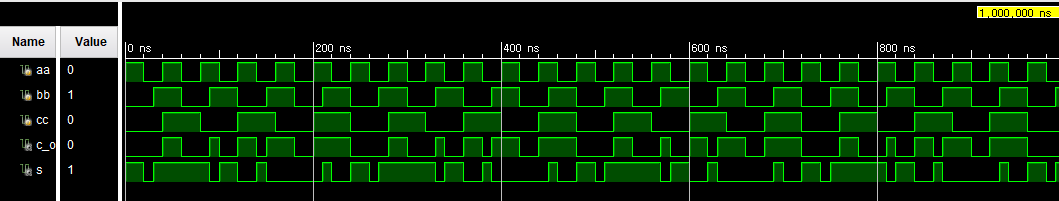
시뮬레이션 결과와 불 함수를 비교해보면 동일함을 알 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

2-2. Full Adder

텍스트이(가) 표시된 사진

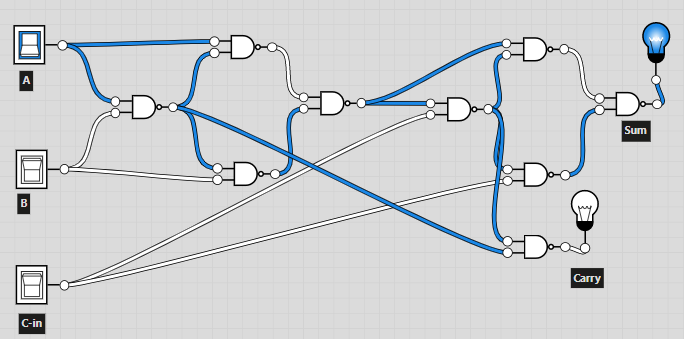
자동 생성된 설명



실습 때 배운 Full adder의 회로를 보고 코딩을 하게 되면 왼쪽 코드와 같이 가능하다. Sum의 경우 앞서 반가산기와는 조금 다르게, A XOR B XOR C-in 으로 표현된다. Carry의 경우 AB + C (A XOR B)로 표현 가능하다. 이렇게 코딩한 시뮬레이션 결과, 불 함수와 일치하게 동작하였다. 이 시뮬레이션을 바탕으로 작성한 진리표는 아래와 같다. Sum의 경우 A와 B, C-in 에서 1의 개수가 홀수라면 1일 출력하며 짝수라면 0을 출력한다. C-out의 경우 입력에서 1의 개수가 2이상이라면 1을 출력한다. 앞선 예시처럼 전 가산기 역시 모두 NAND게이트만을 사용해 구현 가능하다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | C-in | Sum | C-Out |
| 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 |

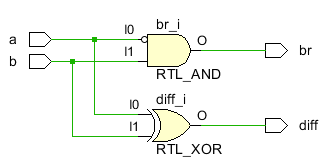
아래 그림은 NAND게이트만을 사용해 전가산기를 구현한 회로이며, 사이트에서 직접 그렸다. <https://logic.ly/demo/>



**3. Full Subtractor / Half Subtractor**

3-1. Half Subtractor

텍스트이(가) 표시된 사진

자동 생성된 설명 

테이블이(가) 표시된 사진

자동 생성된 설명

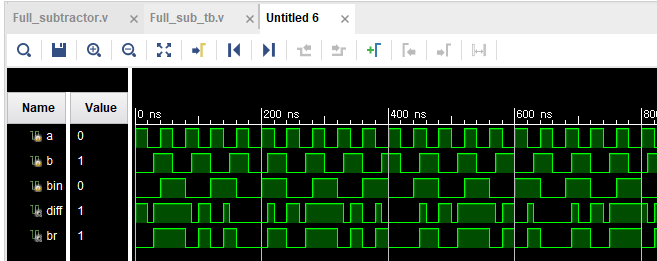
가산기와 마찬가지로 회로를 보고 코딩을 하게되면, 왼쪽과 같이 가능하다. Diff의 경우 신기하게도 가산기와 동일한 A XOR B가 나온다. Br의 경우 빌림수로 부르며, (~A) & B로 표현된다. 시뮬레이션 결과와 불 함수는 동일하며, 이때 진리표는 아래와 같이 작성가능하다. A와 B의 Diff는 A또는 B가 1일 때 1이을 출력하며, 빌림수 Br의 경우 A는 0이고 B가 1일 때 1을 출력한다. NAND만을 사용한 구현이 가능하다.

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | Diff | Br |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

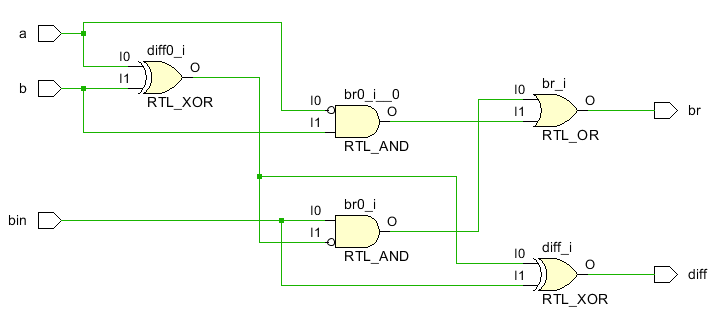
3-2. Full Subtractor

텍스트이(가) 표시된 사진

자동 생성된 설명



실습 때 배운 Full Subtractor의 회로를 보고 코딩을 하게 되면 왼쪽 코드와 같이 가능하다. Diff의 경우 앞서 가산기와 동일하게, A XOR B XOR Bin 으로 표현된다. Br의 경우 조금 다르다. Bin (A XOR B)’ + A’B 로 표현 가능하다. 이렇게 코딩한 시뮬레이션 결과, 불 함수와 일치하게 동작하였다. 이 시뮬레이션을 바탕으로 작성한 진리표는 아래와 같으며, diff의 경우 A와 B, Bin 에서 1의 개수가 홀수라면 1일 출력하며 짝수라면 0을 출력한다. Br의 경우 A-B 혹은 A-Bin이 -1이라면 1을 출력한다. 앞선 예시처럼 전 가산기 역시 모두 NAND게이트만을 사용해 구현 가능하다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Bin | Diff | Br |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

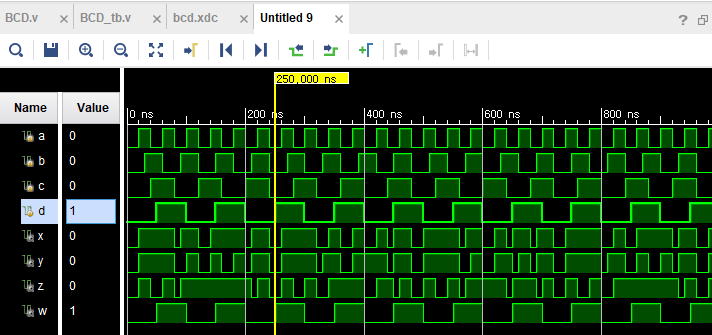
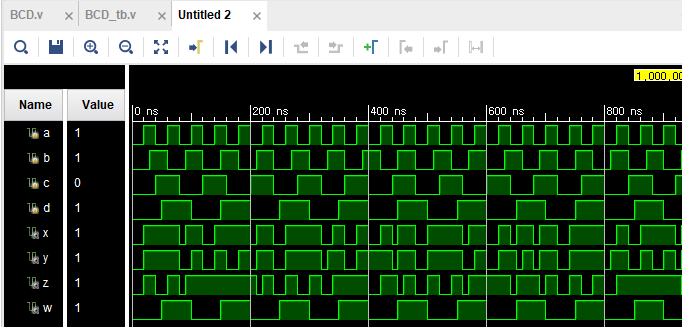
각 가산기와 감산기의 불 함수는 k-map을 활용해 예비보고서에서 이미 구해봤으므로, 결과 보고서에서의 자세한 설명은 생략한다.

**4. 8421(BCD) – 2421 Code converter**

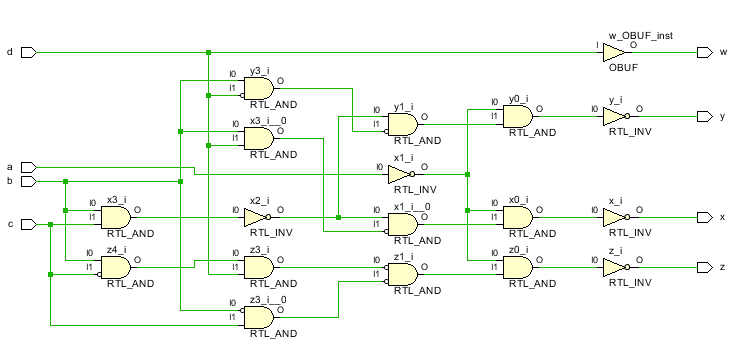
**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

****

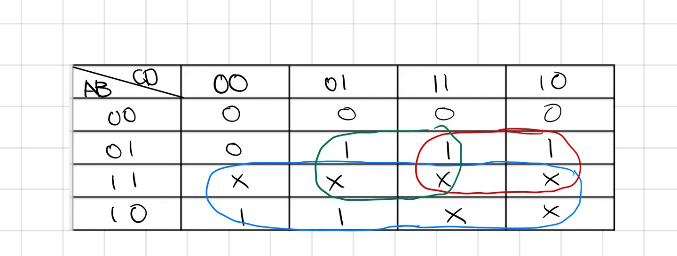
왼쪽코드는 카르노 맵을 통해 구한 BCD converter이며, 오른쪽은 이 불 함수를 nand게이트로 바꾸어 구현한 BCD converter이다. 둘의 테스트 밴치 파일 코드는 동일하며, 이때 둘의 시뮬레이션 결과가 완벽하게 동일하다. 이 시뮬레이션 결과를 확인해 보면, 아래 나올 진리표와 완벽하게 동일하다. 실습시간에 배웠듯이 BCD converter는 0100(2) 까지는 동일한 값을 가진다. 입력값 a,b,c,d가 각각 0101일 때 서로 달라지며 이때, x,y,z,w는 각각 1,0,1,1을 출력한다. 아래는 스케메틱이다.

****

8421(BCD) -2421 converter의 진리표는 아래와 같다. 10진수로 10이 넘어가게 되면, 모두 무관항 처리를 해준다. 이 진리표를 바탕으로 각 W,X,Y,Z의 카르노맵을 작성할 수 있다. (코드에서는 X,Y,Z,W 순이다.)

테이블이(가) 표시된 사진

자동 생성된 설명

먼저 W의 카르노 맵을 작성하면 아래 그림과 같다. 카르노 맵 작성에 자세한 사항은 앞선 주차에서 설명하였으므로 생략하도록 한다. 무관항 x의 경우 필요에 따라 0이나 1로 선택할 수 있다. 따라서 파란색 묶음, 빨간색 묶음, 초록색 묶음이 가능하다.

파란색 묶음의 불 식은 A, 초록색 묶음은 BD, 빨간색 묶음은 BC 이다. 따라서, W = A + BD + BC로 나타 낼 수 있다. 이를 다시 NAND게이트를 사용해 변환하게 되면, BD + BC = ((BD)’)’+ ((BC)’)’ 로 바꿀 수 있다. 드모르간 법칙에 의해서, ((BD)’(BC)’)’ 가 된다. 즉, (B NAND D) NAND (B NAND C) 이다. 같은 방법으로 A + ((BD)’(BC)’)’ = (A’((BD)’(BC)’)’)’이 되고, 즉, ~A NAND ((B NAND D) NAND (B NAND C)) 이다. 이는 SOP이므로, POS를 구하기 위해선, W’을 구하면 된다.

카르노 맵 상에서, 0을 묶고 구하게 되면, W’ = A’B’ + A’C’D’ 이므로, W = (A’B’ + A’C’D’)’ = (A+B)(A+C+D) 이다.

테이블이(가) 표시된 사진

자동 생성된 설명 다음으로, X의 카르노 맵도 동일한 방법으로 작성할 수 있다. 마찬가지로, 빨간색, 초록색, 파란색 3가지의 묶음이 가능하다. 우선 파란색 묶음은 위와 동일하게 A, 빨간색 묶음도 위와 동일학게 BC, 초록색 묶음의 경우 BD’ 이다.

따라서, X = A + BC + BD’ 으로 나타 낼 수 있다. NAND게이트로 변환하게 되면 X = ~A NAND ( B NAND C) NAND ( B NAND ~D) 이다.

또한, X’ = A’B’ + A’C’D 이므로, X = (A’B’ + A’C’D)’ = (A+B)(A+C+D’) 이다.

테이블이(가) 표시된 사진

자동 생성된 설명 다음으로, Y의 카르노 맵이다. 3가지의 묶음이 가능하다. 파란색 묶음은 A가 된다. 빨간색 묶음은, B’C이며 초록색 묶음은, BC’D 이다. 따라서, Y = A + B’C + BC’D 이다. 이를 NAND게이트로 변환하게 되면, ~A NAND ( B NAND ~C NAND D ) NAND ( ~B NAND C ) 이다.

Y’ = A’C’D’ + A’B’C’ + BC 이므로, Y = (A’C’D’ + A’B’C’ + BC)’ = (A+C+D) (A+B+C)(B’+C’) 이다.

테이블이(가) 표시된 사진

자동 생성된 설명마지막으로, Z의 카르노 맵이다. 가장 단순하며, 진리표로도 알 수 있다. Z = D와 동일하다.

**5. 결과 및 논의**

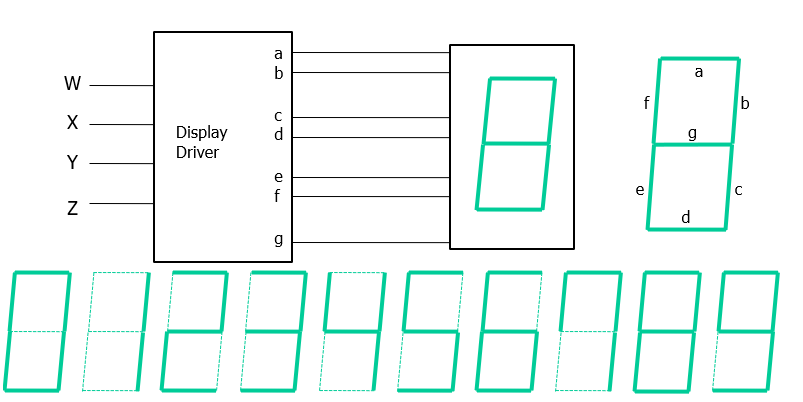
정리해서 말하자면, 우선 전가산기나 전감산기는 두 개의 반가산기/감산기로 이루어졌다. 또한, 이런 1-bit 감가산기를 적절히 연결한다면 n-bit에 대한 가감산기도 만들 수 있다. 이때, 가산기의 경우 A+B나 B+A나 상관 없지만, 감산기의 경우 A-B와 B-A는 다르므로 회로의 구현이나 코드 구현에서 조심해야한다. 실습을 통해 이런 감가산기의 불 표현식과 시뮬레이션 결과가 동일함을 최종적으로 확인할 수 있었다.

BCD 컨버터의 경우 주로 사용하는 8421뿐 아니라 2421 등 여러 종류의 BCD가 존재함을 알 수 있었다. 또한, 감가산기와는 다르게 정해진 불 표현식이 존재하지 않고 카르노 맵을 어떻게 작성하는지 따라 개인의 표현식이 달라질 수 있으며, 카르노 맵을 사용하면 and와 or로 이루어진 회로를 구할 수 있다. 여기서, 앞선 주차에서 배운 드모르간 법칙을 활용하면 손 쉽게 BCD컨버터를 효율적인 NAND게이트 회로로 구현할 수 있다. NOR로의 변환은 직접 해보지는 않았지만, 적절한 식의 변형을 통해 NOR게이트 회로도 구현할 수 있을 것이다. 최종적으로, 카르노 맵을 통해 구한 불 표현식이나, 이를 NAND게이트로 변환한 식 모두 의도한 8421 to 2421 컨버터의 기능을 했다. 또한, SOP나 POS 모두 NAND게이트로 변환한 식이 동일한 결과를 보인다.

**6. 추가 이론**

BCD를 활용한 7-segment display가 있다. 이는 다음 그림과 같이 7개의 발광 다이오드 (LED)를 통해 10진수를 표현한다. 예를 들어, W,X,Y,Z가 모두 0이라면 이는 십진수로 0이다. 따라서, 화면에 0을 출력해야 한다. 이때, 0을 출력하기 위해서는 a,b,c,d,e,f가 1이면 되고, g는 0이면 된다. 이런식으로 0000 ~ 1001에 대한 진리표를 작성할 수 있다. 보통, 10~15는 무관항 혹은 0으로 생각하고 진리표를 작성한다.

작성된 진리표를 바탕으로 k-map을 사용하면 7개의 output인 a,b,c,d,e,f,g에 대한 불 표현식을 얻을 수 있다. 더불어 이를 바탕으로 논리회로의 구현이 가능하다.



**7 참고문헌**

디지털 회로개론 chap4 Designing Combinational Systems. 김주호 교수