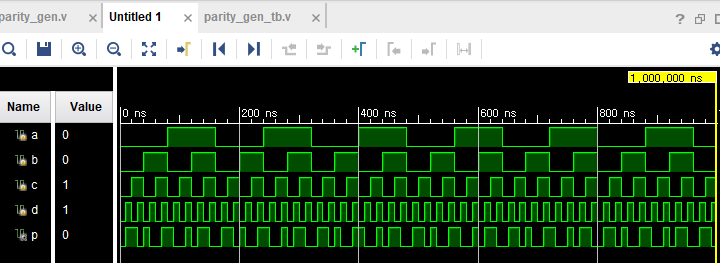
7주차 결과보고서

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

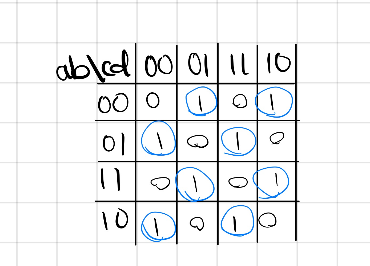
**1. Even parity bit generator 및 checker**

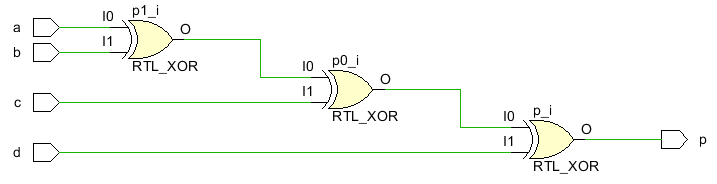
1.1 Generator

**텍스트이(가) 표시된 사진

자동 생성된 설명**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | p |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

4-bit를 송신할 때 짝수 패리티 생성기는 입력되는 1의 개수를 짝수로 맞추기 위한 p를 생성한다. 이에 1의 개수에 맞춰서 진리표를 간단하고 쉽게 작성할 수 있다. 작성된 진리표를 바탕으로 k-map을 그리면 오른쪽 그림과 같다. 얼핏 보면, 모두 단일로 묶여 있어 굉장히 복잡한 표현식을 가질 것 같다. 하지만, 이는 A XOR B의 k-map을 생각해보면 쉽게 알 수 있다. 또한, 패리티 생성기의 특징을 잘 생각해보면 k-map을 그려보지 않아도 XOR의 게이트를 가질 것임을 예측할 수 있다. 따라서, 짝수 패리티 생성기는 아래 스케메틱과 같이 3개의 XOR 게이트로 그려진다. 코드를 사용한 시뮬레이션 결과 역시 확인해보면 a,b,c,d의 1이 되는 경우가 홀수일 때 p가 1이 되어 전체 1의 개수를 짝수로 만듬을 알 수 있다.

****

또한, XOR게이트는 4개의 nand게이트로 이루어진다. 예로 다음과 같이 표현 가능하나 알아보기 힘드며 복잡하다.

(a nand b nand c nand ~ d) nand (a nand b nand ~ c nand d) nand (a nand ~ b nand c nand d) nand (a nand ~ b nand ~ c nand ~ d) nand (~ a nand b nand c nand d) nand (~ a nand b nand ~ c nand ~ d) nand (~ a nand ~ b nand c nand ~ d) nand (~ a nand ~ b nand ~ c nand d)

참고로, SOP minterms로 나타내면,

p = a’b’c’d + a’b’cd’ + a’bc’d’ + a’bcd + abc’d + abcd’ + ab’c’d’ + ab’cd 이때 적절히 묶어주면, 다음과 같은 식을 얻을 수 있다.

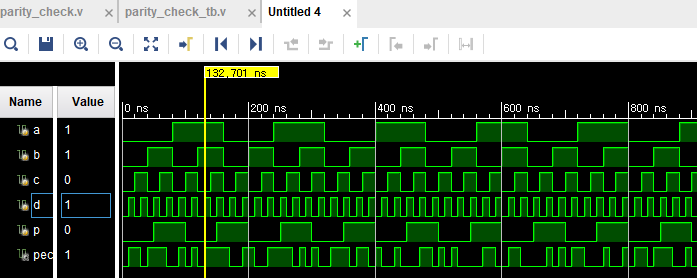
= a’b’(c’d + cd’) + a’b(c’d’ + cd) + ab(c’d + cd’) + ab’(c’d + cd)

= (a’b’+ ab )(c xor d) + (a’b + ab’)(c xor d)’

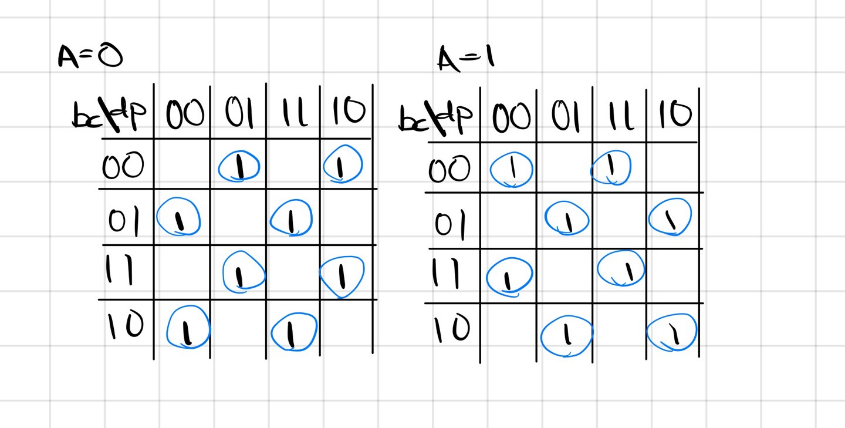
= (a xor b) xor (c xor d)

1.2 Checker

텍스트이(가) 표시된 사진

자동 생성된 설명

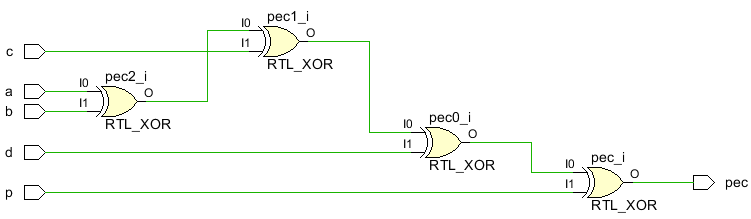
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | d | p | pec |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

4-bit 짝수 패리티 생성기를 이용해 송신하게 되면 최종적으로 수신받는 bit는 5-bit 이다. 또한, 짝수 패리티 검사기의 경우 수신받는 1의 개수가 짝수인지 판별한다. 이때, 정상적으로 짝수개의 1이 들어왔다면 pec은 0을 출력하며, 홀수가 들어왔다면 오류를 의미하는 0을 출력한다. 이를 바탕으로 5개의 입력에 대한 진리표를 작성할 수 있다. 작성된 진리표를 이용해 k-map 그려보면 왼쪽그림과 같다. 이때 k-map은 주로 4개의 input에 대해 작성하므로, A=0과 A=1의 경우 두가지로 나누어 작성하였다. A=0일 때는 앞선 짝수 패리티 생성기와 동일한 k-map이므로 XOR을 이용해 나타낼 수 있다. 반면, A=1일 때는 XOR과 NOT의 관계가 있으므로 XNOR을 이용해 나타낼 수 있다.

Pec = a’(b XOR c XOR d XOR p) + a(b XOR c XOR d XOR p)’ 이므로 이는 (a) XOR (b XOR c XOR d XOR p) 로 나타낼 수 있다. 결론적으로 짝수 패리티 검사기는 아래의 스케메틱처럼 4개의 XOR을 이용해 나타낼 수 있다. 또는 이 외에도 SOP minterms으로 표현한뒤 적절하게 잘 묶어주면 마찬가지로 XOR로 표현 가능하다.

PEC= A’B’C’D’P+A’B’C’DP’+A’B’CD’P’+A’B’CDP+A’BCD’P+A’BCDP’+A’BC’D’P’+A’BC’DP

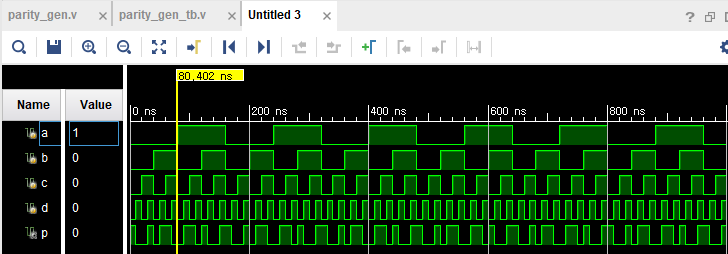
시뮬레이션과 비교했을 때, a,b,c,d,p의 1의 개수가 짝수 일 때 pec의 값이 0으로 출력되며 반대로 홀수일 경우 1을 출력함을 알 수 있다.

****

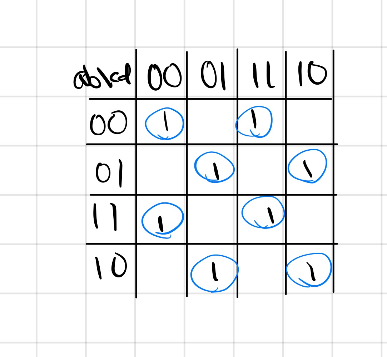
**2. Odd parity bit generator 및 checker**

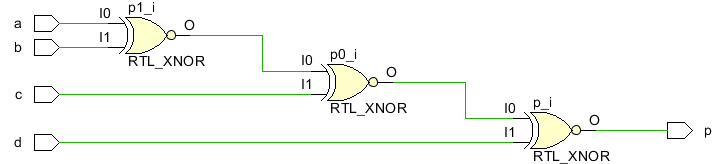
2,1 Generator

텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | p |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

 4-bit를 송신할 때 홀수 패리티 생성기는 입력되는 1의 개수를 홀수로 맞추기 위한 p를 생성한다. 이에 1의 개수에 맞춰서 진리표를 작성할 수 있다. 작성된 진리표를 바탕으로 k-map을 그리면 오른쪽 그림과 같다. 앞서 구한 짝수 패리티의 k-map과 완벽하게 NOT의 관계에 있음을 알 수 있다. 따라서 이는 A XOR B의 NOT임을 생각해보면 쉽게 알 수 있다. 또한, 홀수 패리티 생성기의 특징을 잘 생각해보면 k-map을 그려보지 않아도 XNOR의 게이트를 가질 것임을 예측할 수 있다. 따라서, 홀수 패리티 생성기는 아래 스케메틱과 같이 3개의 XNOR 게이트로 그려진다. 코드를 사용한 시뮬레이션 결과 역시 확인해보면 a,b,c,d의 1이 되는 경우가 짝수일 때 p가 1이 되어 전체 1의 개수를 홀수로 만듬을 알 수 있다.



또한, 3개의 XNOR말고도 3개의 XOR과 마지막에 NOT을 붙여준 회로의 구현도 가능하다. K-map을 사용해 표현하면,

p = a’b’c’d’ + a’b’cd + a’bc’d + a’bcd’ + abc’d’ + abcd + ab’c’d + ab’cd’

= a’b’(c’d’ +cd) + a’b(c’d + cd’) + ab(c’d’ + cd) + ab’(c’d + cd’)

= (a’b’ + ab)(c’d’ + cd) + (a’b + ab’)(c’d + cd’)

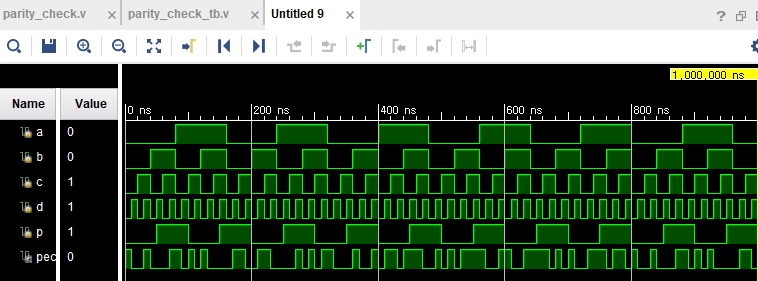
(\* A = a XNOR B, B = c XNOR d )

= (a XNOR b)(c XNOR d) + (a XOR b)(c XOR d) = AB + A’B’ = A XNOR B

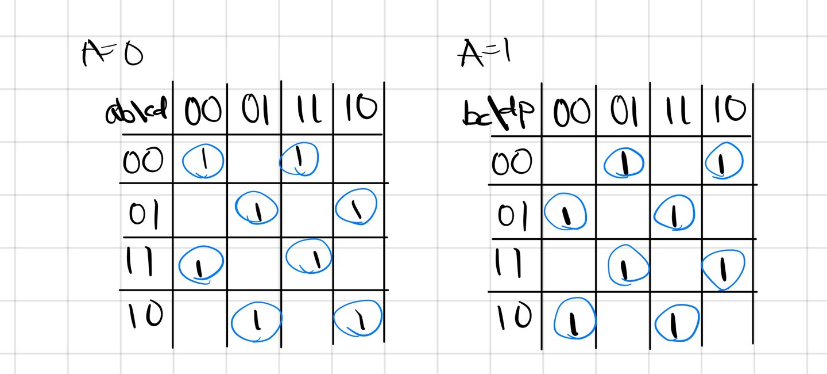
= (a XNOR b) XNOR (c XNOR d) 또는 (a XOR b XOR c XOR d)’

2.2 Checker

텍스트이(가) 표시된 사진

자동 생성된 설명 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | d | p | pec |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

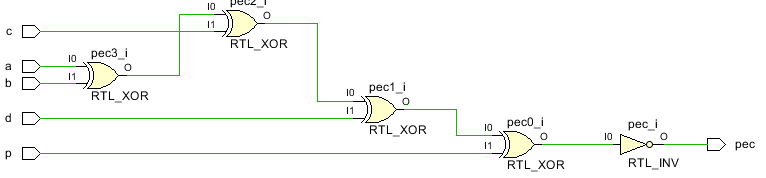
마찬가지로 4-bit 홀수 패리티 생성기를 이용해 송신하게 되면 최종적으로 수신받는 bit는 5-bit 이다. 또한, 홀수 패리티 검사기의 경우 수신받는 1의 개수가 홀수인지 판별한다. 이때, 정상적으로 홀수개의 1이 들어왔다면 pec은 0을 출력하며, 짝수가 들어왔다면 오류를 의미하는 0을 출력한다. 이를 바탕으로 5개의 입력에 대한 진리표를 작성할 수 있다. 작성된 진리표를 이용해 k-map 그려보면 왼쪽그림과 같다. 이때 앞선 방법과 동일하게 k-map을 A=0과 A=1의 경우 두가지로 나누어 작성하였다. A=0일 때는 앞선 홀수 패리티 생성기와 동일한 k-map이므로 XNOR을 이용해 나타낼 수 있다. 반면, A=1일 때는 XOR을 이용해 나타낼 수 있다.

Pec = a(b XOR c XOR d XOR p) + a’(b XOR c XOR d XOR p)’

= aB + a’B’ = (a XNOR B) = a XNOR (b XOR c XOR d XOR p) = ~( a XOR b XOR c XOR d XOR p) 또는, a’ XNOR ( b XNOR c XNOR d XNOR p) 가 성립한다.

결론적으로 홀수 패리티 검사기는 아래의 스케메틱처럼 4개의 XOR과 1개의 NOT을 이용해 나타낼 수 있다. 또는 이 외에도 SOP minterms으로 표현한뒤 적절하게 잘 묶어주면 마찬가지로 XOR로 표현 가능하다.

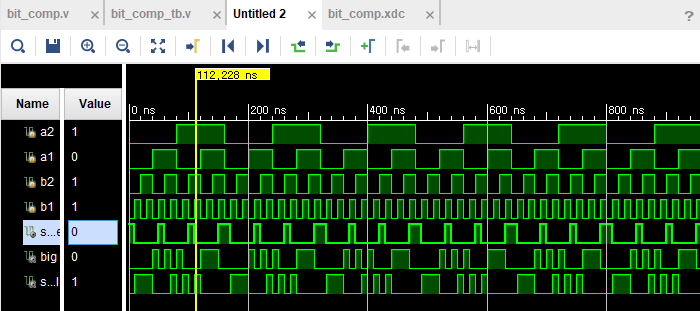
시뮬레이션과 비교했을 때, a,b,c,d,p의 1의 개수가 홀수 일 때 pec의 값이 0으로 출력되며 반대로 짝수일 경우 1을 출력함을 알 수 있다.



**3. 2-bit compartor**

**텍스트이(가) 표시된 사진

자동 생성된 설명**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 입력 | | 출력 | | |
| A | B | A=B | A>B | A<B |
| A1A2 | B1B2 | Same | Big | Small |
| 00 | 00 | 1 | 0 | 0 |
| 01 | 0 | 0 | 1 |
| 10 | 0 | 0 | 1 |
| 11 | 0 | 0 | 1 |
| 01 | 00 | 0 | 1 | 0 |
| 01 | 1 | 0 | 0 |
| 10 | 0 | 0 | 1 |
| 11 | 0 | 0 | 1 |
| 10 | 00 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 |
| 10 | 1 | 0 | 0 |
| 11 | 0 | 0 | 1 |
| 11 | 00 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 |
| 11 | 1 | 0 | 0 |

2-bit 비교기는 앞선 주차에서 배운 1-bit 비교기의 직렬연결으로 구현할 수 있다. 그리나, 이번 실습에서는 진리표를 작성하고 이를 k-map을 사용해 함수로 표현했다.

텍스트, 낱말맞추기게임이(가) 표시된 사진

자동 생성된 설명

A와 B가 같기 위해서는 A1 = B1 이며 A2 = B2 이어야 한다. K-map에서 첫번째 그림에 해당한다.

이를 함수 F라 하면, F = A1’A2’B1’B2’ + A1’A2B1’B2 + A1A2B1B2 + A1A2’B1B2’이 된다.

적절하게 묶어 주게 되면, A1’B1’(A2’B2’ + A2B2) + A1B1(A2B2 + A2’B2’)

= (A2B2 + A2’B2’)(A1’B1’+A1B1) 이 되고, 이는 (A1 XNOR B1)(A2 XNOR B2)가 된다.

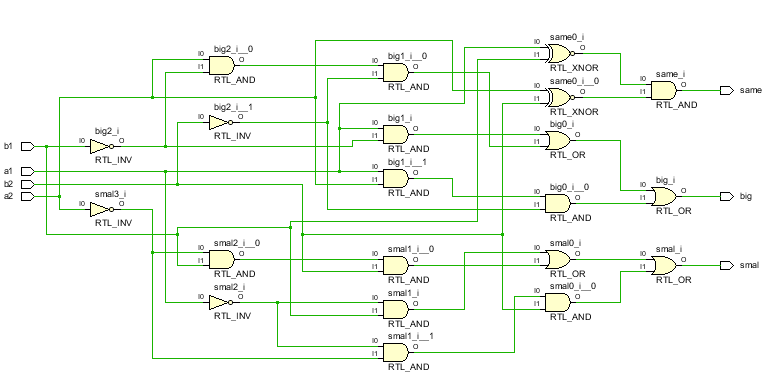
시뮬레이션 결과와 진리표는 역시 동일한 결과를 나태내며, 이는 k-map을 사용해 구한 함수 F가 맞다는 것을 의미한다.

다음으로 A > B의 진리표를 이용해 k-map을 작성하게 되면 그림의 2번째와 같다. 마찬가지로, 이를 함수 F라 할 때, F는 3가지 묶음으로 표현할 수 있다.

먼저 가장 큰 파란색 묶음의 경우 A1B1’ 이 된다. 빨간색 묶음의 경우엔 A2B1’B2’ 이 되며, 마지막으로 초록색 묶음의 경우 A1A2B2’ 이 된다. 따라서 최종적으로 A > B의 불 표현식은 F = A1B1’ + A2B1’B2’ + A1A2B2’ 이다. 시뮬레이션과 진리표는 동일한 결과를 출력하며 함수 F의 성립을 보장한다.

마지막으로 A < B일 때 진리표를 이용해 k-map을 작성하면 그림의 마지막과 같다. 이를 함수 F라 할 때, F 역시 3가지의 묶음으로 표현할 수 있다.

먼저 파란색 묶음의 경우 A1’B1 으로 나타낼 수 있으며, 빨간색 묶음의 경우 A1’A2’B2로 나타낼 수 있다. 초록색 묶음은 A2’B1B2 이므로 최종적으로 A < B의 함수 F표현식은 F = A1’B1 + A1’A2’B2 + A2’B1B2 이다. 시뮬레이션과 진리표의 결과가 동일하므로 역시 표현식이 맞다는 것을 의미한다.

****

**4. 결과 및 논의사항**

짝수와 홀수 패리티 비트 생성기 및 검사기를 k-map을 사용해 비교적 손쉽게 구현이 가능했다. 특히, k-map에서 복잡해보일 수 있는 경우였음에도 XOR게이트를 이용해 간단하게 표현이 가능했다. 짝수와 홀수 패리티 비트는 서로 NOT의 관계에 있음을 실험적으로 검증했고, 또한 불 대수로 표현했을 때 역시 NOT의 관계에 있음을 알 수 있었다. N-bit에 대한 생성기 및 검사기도 어렵지 않게 구현이 가능하다.

또한, 생성기와 검사기는 비슷한 회로를 가진다. 이를 통해 검사기는 생성기의 역할도 가능함을 알 수 있다. 위와 같은 생성기 회로들에서 패리티 비트 p = 0 으로 놓는다면, A XOR 0 = A 이기 때문에 생성기의 역할을 할 수 있음을 알 수 있다.

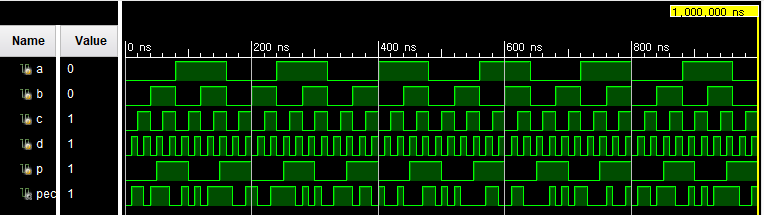
또한, 생각해보면 홀수 패리티는 짝수에 NOT게이트를 추가한 경우다. 효율을 생각해본다면 짝수 패리티를 사용하는 것이 좋을 것이다.

2-bit는 앞서 말했듯 1-bit의 직렬연결로도 구현이 가능했지만, 이번 실험에서는 진리표와 k-map을 사용해 직접 함수를 구하고 회로를 구현했다. 이를 다시 활용한다면 n-bit 비교기의 구현도 어렵지 않을 것이다. 특히, A=B를 구하는 회로는 매우 간단하다. XNOR은 서로 동일한 입력이 들어올 경우 1을 출력하므로, n-bit에 대해서 F = (A\_n-1 XNOR B\_n-1 ) && (A\_n-2 XNOR B\_n-2) && … (A\_0 XNOR B\_XNOR)로 구현될 것이다.

**5. 추가 이론**

앞선 실행을 진행하면서 알게 된 사실이 있다. 홀수 패리티 생성기의 경우 ~(a xor b xor c xor d) = a xnor b xnor c xnor d 가 성립함을 불 표현식을 사용해 보였다. 이에 따라 검사기에서 마찬가지로 (a xor b xor c xor d xor p)’ = a xnor b xnor c xnor d xnor p 가 드모르간 법칙에 의해 성립할거라 생각했다.

하지만, 이는 예상과는 다르게 짝수 패리티 검사기의 결과가 나오게 되었다. 시뮬레이션 결과는 다음과 같았다.



(a xor b xor c xor d xor p)’ = a’ xnor b xnor c xnor d xnor p의 식이 성립한다. 이는 a XOR b 이 사실은 ab’ + a’b 로 되어있기 때문에 발생하는 문제이다. XOR에 대해 드모르간을 적용할 때에는 조심해서 사용해야 한다. 드모르간 법칙은 and와 or에 대해서 적용되기 때문에 여러 개의 xor에 대한 드모르간 법칙은 때로 xnor가 되지 않을 수 있다. 따라서, 홀수 패리티 검사기를 구현할 때는 짝수 패리티 검사기에 NOT을 붙여 회로를 구성하는 것이 깔끔하고 안전할 것이다.

또한, 예비보고서에도 말했듯 단순한 패리티 생성기 및 검사기로는 2-bit에러를 찾을 수 없다. 이를 보완하기 위해 2차원 패리티 검사기가 있다. 2차원의 경우 수평(LRC), 수직(VRC)으로 패리티를 추가하면 된다.