컴퓨터공학실험2 7주차 결과 보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**0. 목차**

1. Even Parity bit generator 및 checker의 simulation 결과 및 과정 설명

2. Odd Parity bit generator 및 checker의 simulation 결과 및 과정 설명

3. 2-bit binary comparator의 simulation 결과 및 과정 설명

4. 결과 검토 및 논의 사항

5. 추가 이론 조사 및 작성

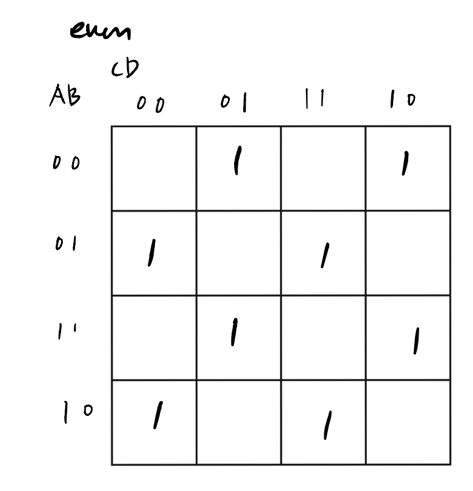
**1. Even Parity bit generator 및 checker의 simulation 결과 및 과정**

이번 실습에서는 parity bit generator/checker, 2-bit binary comparator을 verilog를 통해 구현하며, FPGA로 동작 검증을 수행하는 실습을 진행했다. 이때 parity bit generator로는 even parity bit generator을 구현했으며, checker로는 odd parity bit checker를 구현했다.

우선 Even Parity bit generator은 실습에서 진행한 형태로, input A, B, C, D에 대해 output X = A^B^C^D; 형태로 XOR gate만을 이용해서 even parity bit generator을 제작할 수 있다. 또한 이에 대한 Truth table과 k-map은 아래와 같다.

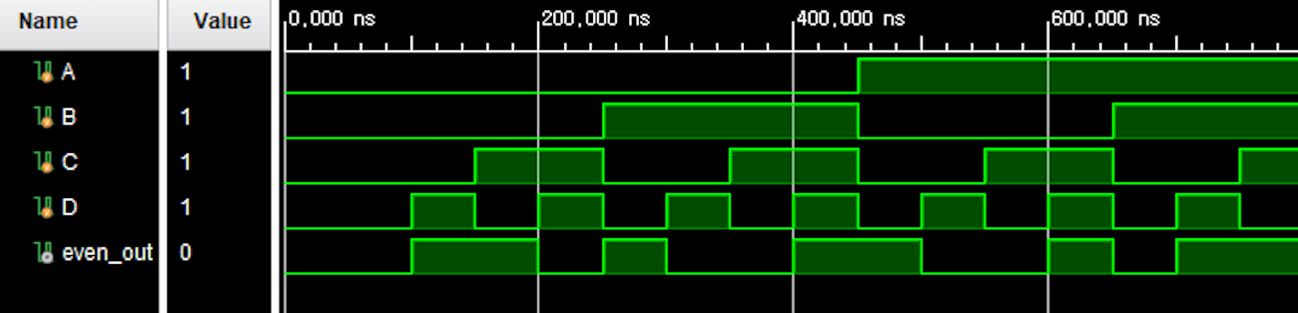
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Even Parity bit generator | | | | |
| input | | | | output |
| A | B | C | D | Out |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 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 |

<Even Parity bit generator의 truth table>



<Even Parity bit generator의 k-map>

Parity bit generator의 simulation 결과는 아래와 같다.



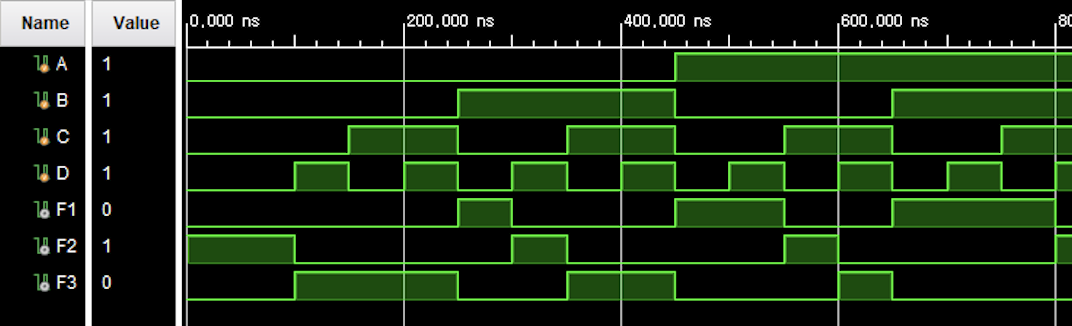
<Even Parity bit generator의 simulation 결과>

Even Parity bit checker은 실습 시간에서는 직접 구현하지 않았지만, 간단하게 Odd parity checker에 NOT gate를 결과에 추가해주며 구현해낼 수 있다. 식으로 표현하면, output X= ~(A^B^C^D^P); 형태로 만들어낼 수 있다. 이에 대한 Truth table은 아래와 같으며, even parity bit generater을 통해서 checker 또한 k-map에서 사각형으로 묶이는 부분이 많이 없을 것임을 파악할 수 있다. k-map보다는 그냥 식으로의 표현이 훨씬 효율적일 것이다. 따라서 k-map은 생략한다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Even Parity bit checker | | | | | |
| input | | | | | output |
| A | B | C | D | P | Out |
| 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 |

<Even Parity bit checker의 truth table>

이에 대한 simulation 결과는 아래와 같다.



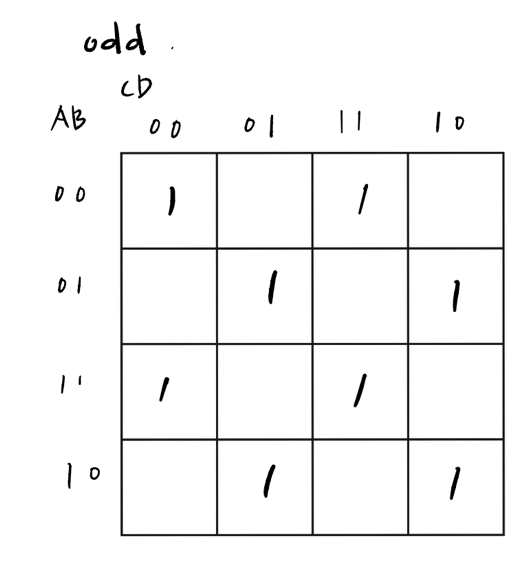
<Even Parity bit checker의 simulation 결과>

**2. Odd Parity bit generator 및 checker의 simulation 결과 및 과정**

Odd parity generator 또한 Even Parity checker를 구현하는 것과 비슷하게 만들어낼 수 있다. Even parity generator에 NOT gate를 추가해주는 형태이며 그 식은 output X = ~(A^B^C^D); 형태가 됨을 쉽게 판단할 수 있다. 이에 대한 Truth table과 k-map은 아래와 같다.

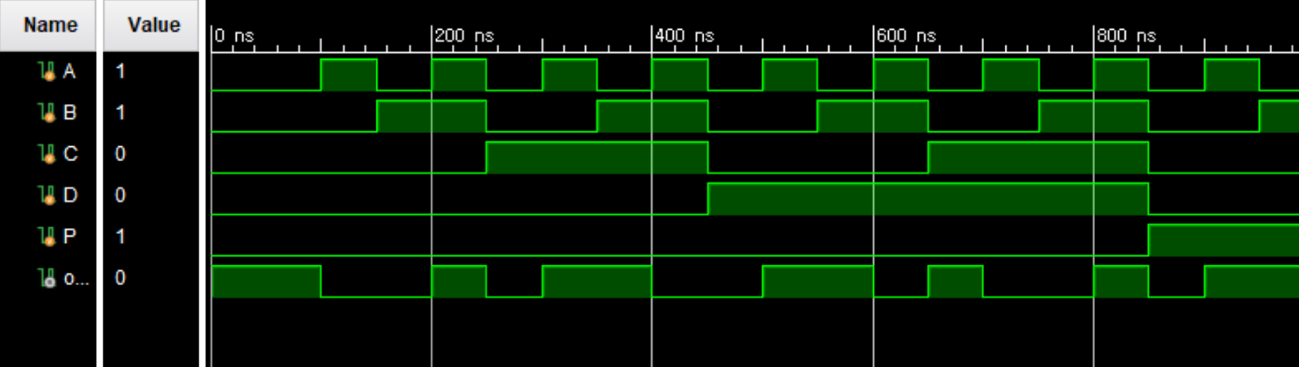
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Odd Parity bit generator | | | | |
| input | | | | output |
| A | B | C | D | Out |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 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 |

<Odd Parity bit generator의 truth table>



<Odd Parity bit generator의 k-map>

또한 simulation 결과는 아래와 같다.



<Odd Parity bit generator의 simulation 결과>

Odd parity checker은 이번 실습에서 직접 구현한 parity checker로, Even parity generator와 동일하게 XOR gate만을 이용해서 만들어낼 수 있다. XOR gate를 이용하면 input의 High 개수가 홀수라면, output이 High이 되며, 짝수라면 output이 Low가 되게 회로를 구현할 수 있기 때문이다. 식은 output X = A^B^C^D^P; 이며 Truth table은 아래와 같다. 또한 even parity bit checker와 동일한 이유로 k-map은 생략한다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Odd Parity bit checker | | | | | |
| input | | | | | output |
| A | B | C | D | P | Out |
| 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 |

<Odd Parity bit checker의 truth table>

또한 simulation 결과는 아래와 같다.



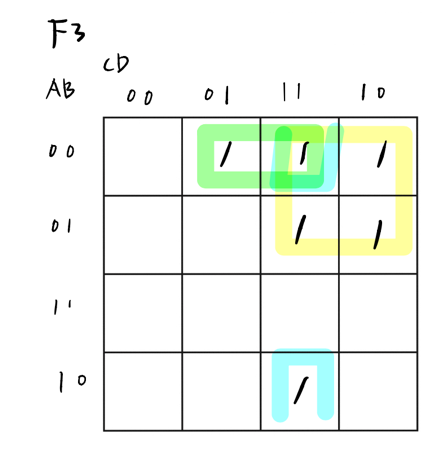
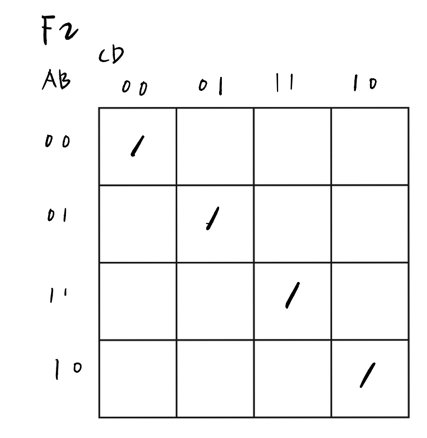
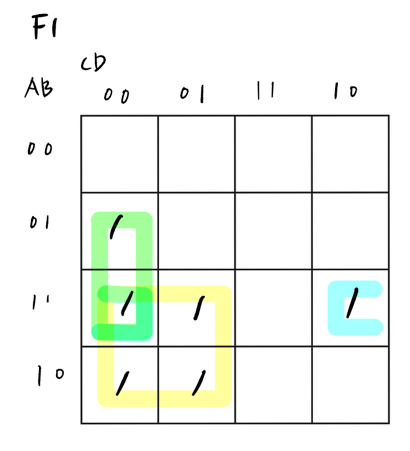
<Odd Parity bit checker의 simulation 결과>

**3. 2-bit binary comparator의 simulation 결과 및 과정**

이전 실습에서 1-bit binary comparator을 구현한 적이 있다. input으로 2개의 1bit 를 받아 input A, B에 대한 3가지 output을 판별하는 형태였다. 이와 비슷하게 2-bit binary comparator은 2개의 2bit input을 받아 A, B에 대해서 A>B, A=B, A<B 가 각각 참인지 거짓인지를 알려주는 역할을 수행하는 회로라고 할 수 있다. 각 output에 적절한 식은 Truth table, k-map을 통해서 구현하였으며, 이는 아래와 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2-bit binary comparator | | | | | | |
| input | | | | output | | |
| A | B | C | D | F1 | F2 | F3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 2 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

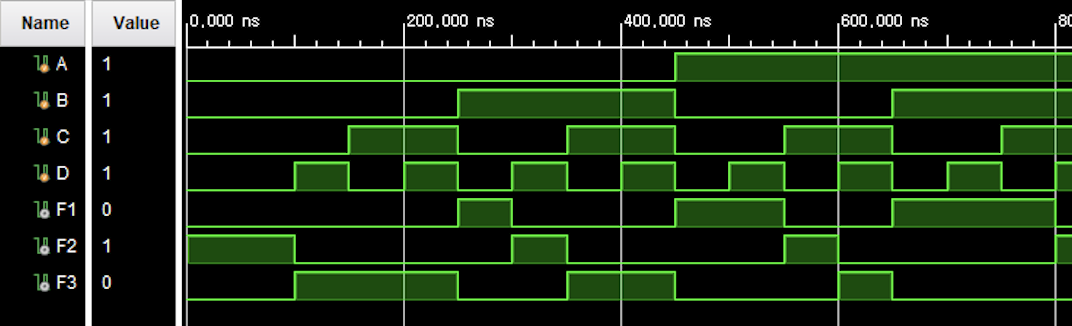
<2-bit binary comparator의 truth table>



<2-bit binary comparator의 k-map>

위 결과를 통해서 F1, F3에 대한 k-map으로의 minimization은 효율적이지만, F2의 표현에서는 별로 효율적이지 않음을 파악할 수 있다.

또한 simulation 결과는 아래와 같다.



<2-bit binary comparator의 simulation 결과>

**4. 결과 검토 및 논의 사항**

이번 실습에서는 XOR gate를 활용한 parity bit generator/checker을 직접 구현하고, 2-bit binary comparator을 직접 구현하며 1-bit binary comparator과 다르게 k-map을 통해서 comparator을 구현해볼 수 있었다. 또한 FPGA를 사용하며 verilog를 통해 구현해낸 함수를 각 Pin과 알맞은 형태로 연결하며, 옳은 동작을 수행하는지 검증해볼 수 있었다. 각 pin을 통해 input을 조절하며, LED를 활용해서 output을 주고받는 형태로 동작 검증을 수행했다.

이번 실습을 진행하며, 계속 이론적으로만 알고 있었던 FPGA를 직접 사용하며 동작 검증을 수행하면서, 사실 이해가 가지 않는 부분이 많았는데(왜 verilog를 사용하는지, reg, 실제로는 구현해낸 input이 어떻게 작용하는지 등), 이를 해소할 수 있는 좋은 기회가 되었다.

또한 이번 실습을 진행하면서 input을 switch 형태의 pin에 연결해서 직접 물리적으로 주는 형태를 만들었는데, 이전까지 진행했던 실습에서 검증을 할 때는, verilog에서 simulation을 활용해서 자동으로 주어지는 input을 토대로 output을 판별하며, 이를 timing diagram으로 구현해서 검증을 진행했었다. 이 형태를 FPGA에서도 적용해서 수행할 수 있는지에 대한 궁금증이 생겼다.

**5. 추가 이론 조사 및 작성**

위에서 이야기한 것처럼, FPGA에서도 simulation을 수행하는 것처럼 자동으로 input을 줄 수 있는지에 대해서 알아보자. 이를 파악하기 위해서 생각해본 것이 실생활에서 존재하는 전자시계이다. 전자시계 또한 7-Segment Display가 존재하는 하나의 회로라고 볼 수 있는데, FPGA를 이용한다면 이를 구현해낼 수 있을 것이며, 결국 자동으로 input을 계속해서 주어지는 형태라는 것을 파악할 수 있다. 여기에서 중요한 점은 회로로 시계를 구현할 때 system clock 을 이용해서 구현을 한다는 것이다. 이를 위해서는 clock counter 등 다양한 회로가 필요하기 때문에 아직은 알지 못하지만, 물리적인 input 없이도 실행하는 system이 존재할 수 있을 것임은 파악할 수 있다.