9주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20211547 이름: 신지원

**1.**

- Active high

Active high 는 입력 신호가 1일 때 동작하는 것을 말한다. 따라서 이때의 decoder 코드는 AND gate 를 사용하여 구현하였다. D0 은 둘 다 0인지, D1은 (a,b) 중 후자가 더 큰지, D2는 전자가 더 큰지, D4는 둘이 같은 지를 출력한다. 아래는 AND gate 로 짠 코드에 따른 시뮬레이션이다.

스크린샷, 소프트웨어, 컴퓨터, 디스플레이이(가) 표시된 사진

자동 생성된 설명

둘 다 0일 때 D0에 신호가, bb가 더 클 때 D1에 신호가, aa 가 더 클 때 D2에 신호가, 둘 다 1일 때 D4에 신호가 1로 출력되는 것을 볼 수 있다. 이에 따른 진리표와 카르노맵은 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In aa | In bb | Out d0 | Out d1 | Out d2 | Out d3 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

번호, 텍스트, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

- Active low

Active low 는 입력 신호가 0일 때 동작하는 것을 말한다. 따라서 이때의 decoder 코드는 NAND gate 를 사용하여 구현하였다. D0 은 둘 다 0인지, D1은 (a,b) 중 후자가 더 큰지, D2는 전자가 더 큰지, D4는 둘이 같은 지를 출력한다. 아래는 NAND gate 로 짠 코드에 따른 시뮬레이션이다.

스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

testbench 의 주기를 적절하게 입력하지 못하여 모든 경우의 수를 볼 수 없었지만, 입력값이 aa는 0으로 bb는 1로 후자가 더 크기 때문에 D1에 해당되는 출력값은 0으로, 나머지는 모두 1로 출력되어야 active low 를 잘 구현하였다고 할 수 있다. 위 simulation 코드 또한 마찬가지로 잘 구현되었다. 이에 따른 진리표와 카르노맵은 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In aa | In bb | Out d0 | Out d1 | Out d2 | Out d3 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

도표, 번호, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

**2.**

Encoder 에 대한 코드는 OR gate 로 구현 가능하다. 실습에서 구현한 코드에 따른 시뮬레이션은 아래와 같다.

스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

아래 진리표와 카르노맵은 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input aa | Input bb | Input cc | Input dd | Output e0 | Output e1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

자동 생성된 설명

**3.**

4 to 2 Encoder 는 input 4개를 제외하고 다른 입력 형태를 정의하지 않았다. 따라서 본인이 정의한 aa,bb,cc,dd 의 4개의 input을 제외하고 나머지를 입력하는 것은 오류이다. 따라서 실습에서 말씀해주신 것처럼 다른 입력 형태는 Don’t care로 처리해주었다.

**4.**

priority encoder 를 구현하기 위해선 출력에 대해 Don’t care 처리했던 부분도 고려를 해주어야 한다. 고려한 바에 따른 시뮬레이션은 아래와 같다.

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

이러한 시뮬레이션을 구현하기 위해선 아래와 같이 진리표와 카르노맵을 작성하였다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input aa | Input bb | Input cc | Input dd | Output e0 | Output e1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | X | 0 | 1 |
| 0 | 1 | X | X | 1 | 0 |
| 1 | X | X | X | 1 | 1 |

e0 = aa | (~bb && cc )

e1 = aa | bb

**5.**

BCD 를 구현하기 위해서는 진리표와 카르노맵을 작성한 뒤 시뮬레이션으로 확인해볼 수 있기 때문에 논리식이 굉장히 복잡하였다. 또한 입력이 많기 때문에 testbench 에서 주기를 작성하는 것도 어려움이 있었다. 최대한 모든 주기를 고려하여 아래 시뮬레이션 코드를 작성하였다.

스크린샷, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In w | In x | In y | In z | Out aa | Out bb | Out cc | Out dd | Out ee | Out ff | Out gg | Out hh | Out ii |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

쇼지, 낱말맞추기게임이(가) 표시된 사진

자동 생성된 설명

**6.**

예비보고서에서 작성하였던 것처럼 컴퓨터가 읽을 수 있도록 2진수로 부호화 해줄 때 필요한 것이 Encoder이며, 다시 10진수로 사람이 읽을 수 있도록 바꿔주는 것이 decoder의 역할이다. 우선 Encoder의 응용은 엘레베이터의 버튼을 누를 때, 버튼 자체는 10진수로 되어있지만 엘레베이터가 움직이기 위해서는 2진수로 변경해주어야 한다. 따라서 표준화, 보안, 처리 속도 향상 등을 위하여 다른 형태나 형식으로 변환하면서 응용 가능하다. decoder의 응용은 주로 장치나, 회로가 있고 우리가 공부하는 알고리즘도 이에 해당한다.

**7.**

스크린샷이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In a0 | In b0 | In aa | In bb | In cc | In dd | Out f |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |

**8.**

아래는 실습에서 구현한 1 to 4 line deMUX 의 시뮬레이션이다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

이를 이용하여 4 to 16 line deMUX 를 수행하고 결과로 나타내었다. 출력은 a~p, 입력은 w~z 로 구현하였다. 하지만 너무 길어 출력의 시뮬레이션만 캡쳐하였다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | x | y | z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**9.**

실습 이후 추가로 Pirority Encoder 의 회로를 구성하기 위하여 입력에 우선순위를 두었고 그에 따라 회로를 설계하였다. 따라서 모든 입력에 대한 Encoder 가 작동할 수 있는 논리회로를 구성하였다. 4 to 16 line deMUX 의 회로도 직접 카르노맵을 그려보면서 Verilog 코드를 구현해보았다.

추가적으로 아쉬웠던 부분은 입력이 많아지면 많아질 수록 그에 따른 주기를 두기가 더 어려워진다는 것이었다. 모든 경우의 수를 볼 수 있도록 적당한 주기를 주는 것이 실습에서 가장 어렵고, 해결해야 할 문제인 것 같다.

**10.**

디멀티플렉서에 대하여 더 자세히 알아보고자 한다. 멀티플렉서는 MUX 라고 표기하며 n 개의 선택선의 조합에 의해 선택된 입력선 중에서 하나를 선택하는 것이다. 비슷한 맥락으로 디멀티플렉서란 한 개의 입력을 어떤 출력단으로 보낼지 선택하는 논리 회로를 말한다. 지난 실습에서 배웠던 내용이다.

추가로 2-input Demultiplexer 는 A0, A1 의 출력 중 하나를 선택하여 입력값으로 내보내는 회로를 말한다. 또한 1 to 8 Demultiplexer 는 8개의 출력핀 중에 하나를 선택하는 것을 말한다. 이렇듯 실습에서 학습한 종류 말고도 다양한 종류의 Input 을 달리하여 응용할 수 있다.