9주차 결과보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20191583 이름: 김태곤

**1. 2 to 4 Decoder 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함,[AND,NAND GATE])**

**1) AND GATE**

**1)) Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **input** | | **output** | | | |
| **A** | **B** | **D0** | **D1** | **D2** | **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** |

**2)) Karnaugh Map**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **0** | **0** |

**D0 = A’B’**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **0** | **1** |
| **1** | **0** | **0** |

**D1 = A’B**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **0** | **0** |
| **1** | **1** | **0** |

**D2 = AB’**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **0** | **0** |
| **1** | **0** | **1** |

**D3 = AB**

**3)) Source Code**

**텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4)) Schematic**

**도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명**

**5)) Simulation**

**소프트웨어, 텍스트, 멀티미디어 소프트웨어, 라인이(가) 표시된 사진

자동 생성된 설명**

**6)) 결과 및 과정**

2 to 4 AND GATE Decoder는 각 출력 Dn이 해당하는 입력 비트의 AND 조합이 참일 때만 1이 된다. 때문에 D0는 A’과 B’, D1은 A’과 B, D2는 A와 B’, D3는 A와 B의 AND연산을 통해 결과값을 출력하였다. A는 200, B는 100마다 값을 변화시켜 알아보기 쉽게 simulation을 하였다. 그 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 2 to 4 AND GATE Decoder의 변화를 Simulation을 통해 확인할 수 있었다.

**2) NAND GATE**

**1)) Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **input** | | **output** | | | |
| **A** | **B** | **D0** | **D1** | **D2** | **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** |

**2)) Karnaugh Map**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **0** | **1** |
| **1** | **1** | **1** |

**D0 = (A’B’)’**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |

**D3 = (AB’)’**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **1** | **0** |
| **1** | **1** | **1** |

**D1 = (A’B)’**

|  |  |  |
| --- | --- | --- |
| **B**  **A** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **1** | **0** |

**D4 = (AB)’**

**3)) Source Code**

**텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4)) Schematic**

**도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명**

**5)) Simulation**

**소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**6)) 결과 및 과정**

2 to 4 NAND GATE Decoder는 출력이 입력 비트의 NAND 조합에 따라 결정된다. 즉, 입력이 모두 참일 때 해당 출력은 거짓(0)이 되고, 나머지 경우에는 참(1)이 된다. 때문에 D0는 A’과 B’, D1은 A’과 B, D2는 A와 B’, D3는 A와 B의 NAND연산을 통해 결과값을 출력하였다. A는 200, B는 100마다 값을 변화시켜 알아보기 쉽게 simulation을 하였다. 그 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 2 to 4 NAND GATE Decoder의 변화를 Simulation을 통해 확인할 수 있었다.

**2. 4 to 2 Encoder 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함)**

**1) Truth Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **input**  **output** | | | | **output** | |
| **A** | **B** | **C** | **D** | **E0** | **E1** |
| **1** | **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **1** | **0** |
| **0** | **0** | **0** | **1** | **1** | **1** |

**2) Karnaugh Map**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **X** | **1** | **X** | **1** |
| **01** | **0** | **X** | **X** | **X** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **X** | **X** | **X** |

**E0 = C+D(0000은 invalid 0으로 처리)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **X** | **1** | **X** | **0** |
| **01** | **1** | **X** | **X** | **X** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **X** | **X** | **X** |

**E1 = B + D(0000은 invalid 0으로 처리)**

**3) Source Code**

**텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진

자동 생성된 설명**

**4) Schematic**

**도표, 평면도이(가) 표시된 사진

자동 생성된 설명**

**5) Simulation**

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**6) 결과 및 과정**

4 to 2 Encoder는 4개의 입력을 받아 2개의 출력으로 압축하는 논리 회로이다. 이 회로는 4개의 입력 중 하나만 활성화되어 있을 때(즉, 하나의 입력만 1이고 나머지는 '0) 그 입력의 위치를 2비트 이진 코드로 출력한다. 이를 Truth Table로 작성하고, K-map을 통해 코드를 추출하였다. Truth Table을 작성할 때 A가 1인 경우 0001을 나타낸다. 0000의 경우 invalid값으로 0을 처리해 값을 얻은 결과 E0는C와 D의 OR연산, E1은 B와 D의 OR연산을 통해 결과값을 출력하였다. A,B,C,D는 각각 800,400,200,100마다 변화시켜 알아보기 쉽게 simulation을 하였다. 그 결과 800, 400, 200, 100을 살펴보면 각각 1개의 참의 값이 입력된 결과를 확인할 수 있다. 이는 truth table과 동일한 simulation 결과를 얻었고, 올바르게 4 to 2 Encoder 의 변화를 Simulation을 통해 확인할 수 있었다.

**3. 4 to 2 Encoder에서 입력 형태 4가지를 제외한 나머지 입력 형태는 무엇을 뜻하는지 설명하시오.**

4 to 2 Encoder는 통상적으로 네 개의 입력 중 하나만이 활성화되어야 정상적으로 동작하는 논리 회로이다. 이는 입력 중 하나만이 1이고 나머지 세 개는 0인 경우를 의미한다. 이러한 Encoder는 입력된 데이터의 위치(활성화된 비트의 위치)를 나타내는 2비트 이진 코드로 축약한다. 그러나, 네 개의 입력 중 둘 이상이 동시에 1로 설정되거나 모든 입력이 0인 경우, 이는 인코더의 표준 동작에 속하지 않는 입력 형태가 된다. 그러나 실제 시스템에서는 다양한 입력 상태가 발생할 수 있으며, 이러한 모든 경우를 처리할 수 있는 회로가 필요할 때가 있다. 이때 유용하게 사용되는 것이 바로 Priority Encoder 이다. Priority Encoder를 사용하면 4가지 형태가 아닌 모든 입력 형태(16가지)에 대하여 출력 값을 얻을 수 있다. Priority Encoder는 입력이 여러 개 동시에 활성화되었을 때, 가장 높은 우선순위의 입력을 기반으로 출력을 결정한다.

**4. 4 to 2 encoder의 4가지 형태가 아닌 모든 입력 형태(16가지)에 대하여 동작되는 priority encoder의 논리 회로를 구성하여라.**

우선순위 인코더는 입력이 여러 개 동시에 활성화되었을 때, 가장 높은 우선순위의 입력을 기반으로 출력을 결정한다. 이를 통해 16가지 가능한 입력 상태 모두에 대응할 수 있다. 우선순위 인코더는 일반적으로 다음과 같은 규칙을 따른다.

D에 가장 높은 우선순위를 부여하고, 그 다음이 C, B, A 순이다.

둘 이상의 입력이 동시에 활성화되면, 가장 높은 우선순위의 입력에 해당하는 출력이 생성된다.

이를 바탕으로 Truth Table을 작성하면 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **input**  **output** | | | | **output** | |
| **D** | **C** | **B** | **A** | **E0** | **E1** |
| **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **0** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** | **1** |
| **0** | **1** | **0** | **0** | **1** | **0** |
| **0**  **0** | **1** | **0** | **1** | **1** | **0** |
| **0** | **1** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** |
| **1** | **0** | **0** | **1** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **0** | **1** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **1** | **1** |
| **1** | **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** | **1** | **1** |

우선순위 인코더에서 16가지 입력 상태 각각에 대해 예상되는 출력을 알아보면 다음과 같다. 0000의 경우 출력 정의가 되지 않는다. 따라서 Default값으로 0을 설정하였다. 0011은 우선순위에 따라 B가 선택되었고, 0101, 0110, 0111은 우선순위에 따라 C가 선택되었다. 이로인해 0011은 01의 출력값을 가지게 되었고, 0101, 0110, 0111은 10의 출력값을 가지게 되었다. 1001부터 1111까지는 모두 우선순위에 따라 D가 선택되었고, 11의 출력값을 가지게 된다. 이를 바탕으로 K-map을 구성하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **1** | **1** |
| **01** | **0** | **1** | **1** | **1** |
| **11** | **0** | **1** | **1** | **1** |
| **10** | **0** | **1** | **1** | **1** |

**E0 = C + D**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **1** | **0** |
| **01** | **1** | **1** | **1** | **0** |
| **11** | **1** | **1** | **1** | **0** |
| **10** | **0** | **1** | **1** | **0** |

**E1 = BC’+D**

이를 논리 회로로 구성하면 다음과 같이 나타낼 수 있다.

도표, 라인, 디자인이(가) 표시된 사진

자동 생성된 설명

**5. BCD to Decimal decoder 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함)**

**1) Truth Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **input** | | | | **output** | | | | | | | | | |
| **A** | **B** | **C** | **D** | **D0** | **D1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** | **D8** | **D9** |
| **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **0** |
| **0** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **1** | **0** | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |

**강의 자료에 나온 74HC42의 Output은 1~9로 구성되어 있지만 보다 일반적인 상황으로 output을 D0~D9으로 구성하였다. D1~D9만 생각하면 강의 자료와 동일한 구성에도 같은 결과가 나온다.**

**2) Karnaugh Map**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **1** | **0** | **0** | **0** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D0 = A’B’C’D’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **1** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D2 = B’CD’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **1** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D4 = BC’D’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **0** | **0** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D1 = A’B’C’D**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **1** | **0** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D3 = B’CD**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **0** | **1** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D5 = BC’D**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **0** | **0** | **0** | **1** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D6 = BCD’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **1** | **0** | **X** | **X** |

**D8 = AD’**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **0** | **0** | **1** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **0** | **X** | **X** |

**D7 = BCD**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CD**  **AB** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **0** | **0** | **0** | **0** |
| **11** | **X** | **X** | **X** | **X** |
| **10** | **0** | **1** | **X** | **X** |

**D9 = AD**

**3) Source Code**

**텍스트, 스크린샷, 디스플레이, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4) Schematic**

**도표, 텍스트, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명**

**5) Simulation**

**스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**6) 결과 및 과정**

BCD(Binary Coded Decimal) to Decimal 디코더는 4비트 BCD 입력(A, B, C, D)을 받아 해당하는 십진수를 표현하는 10개의 출력(D0~D9)으로 압축하는 논리 회로이다. BCD는 각 십진수 자릿수를 4비트 이진수로 표현하여, 십진수의 숫자 0에서 9까지를 0000부터 1001까지의 이진 코드로 변환한다. 따라서 Output은 D0~D9으로 구성하였다. 이 Decoder는 각 십진수 값을 해당하는 출력만을 활성화시켜 나타낸다. 예를 들어, 입력이 0000인 경우 D0가 활성화되고(1000000000), 나머지 D1~D9는 비활성화된다(0). 이를 Truth Table로 작성하고, K-map을 통해 코드를 추출하였다. Truth Table은 순서대로 각 Output이 한 개만1이 되도록 간단하게 구성하였다. 이를 K-map으로 나타내고 간소화를 시켰다. D0의 경우 A’B’C’D’의 AND연산을 통해 얻을 수 있고, D1의 경우 A’B’C’D의 AND 연산을 통해 얻을 수 있다. 모든 결과는 AND연산으로 구성하였다. D2의 경우 B’CD’, D3는 B’CD, D4는 BC’D’, D5는 BC’D, D6는 BCD’, D7은 BCD, D8은 AD’, D9은 AD의 AND 연산으로 나타내었다. A,B,C,D는 각각 800,400,200,100마다 변화시켜 알아보기 쉽게 simulation을 하였다. 그 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 BCD to Decimal Decoder의 변화를 Simulation을 통해 확인할 수 있었다. 이러한 구성을 통해 BCD to Decimal Decoder는 정확하게 십진수 값을 디지털 형태로 디코딩할 수 있으며, 각 입력 값에 대한 정확한 출력을 보장한다.

**6. Encoder와 decoder의 주요 응용에 대하여 설명하시오.**

인코더와 디코더는 디지털 시스템에서 데이터를 변환하고, 처리하며, 전달하는 데 중요한 역할을 하는 기본적인 논리 회로이다. 이들은 통신, 데이터 압축, 데이터 처리 등 다양한 응용 분야에서 활용된다.

1) Encoder

1)) 키보드 인코딩

컴퓨터 키보드와 같은 입력 장치에서는 수많은 키를 간결하게 전자적으로 식별하기 위해 인코더가 사용된다. 각 키는 고유의 코드로 변환되어, 컴퓨터 시스템이 사용자의 입력을 정확하게 인식하고 처리할 수 있도록 한다. 사용자가 누른 키보드의 키를 원본으로 해당 키에 대응하는 고유의 스캔 코드로 변환한다. 키 입력을 스캔 코드로 변환하여 컴퓨터가 처리할 수 있도록 전달한다.

2)) 디지털 회로의 선택 신호 생성

멀티플렉서, 데멀티플렉서와 같은 디지털 스위칭 회로에서 인코더는 선택 신호를 생성하는 데 사용된다. 이는 다수의 입력 또는 출력 라인 중에서 특정 라인을 선택할 수 있게 해준다. 다수의 입력 라인이 특정 입력 라인을 선택하는 신호로 변환된다. 입력에 따라 특정 신호 라인을 활성화하는 선택 신호 생성한다.

3)) 주소 인코딩

메모리 주소를 할당하거나 특정 네트워크 주소를 지정하는 데 있어 인코더는 입력 정보를 압축된 형태로 변환하여 처리 효율성을 높인다.. 데이터 소스 또는 기기가 각 데이터 소스에 할당된 주소 코드로 변환한다. 데이터 소스를 특정 주소로 매핑하여 데이터 라우팅을 관리한다.

4)) 통신 시스템

인코더는 데이터 전송을 위해 원시 데이터를 신호 혹은 코드로 변환하는 데 사용되며, 이는 데이터를 보다 효율적으로 전송하고 오류를 감소시키는 데 도움을 준다. 전송할 데이터를 암호화된 데이터로 변환한다. 데이터를 암호화하여 안전하게 전송한다.

2) Decoder

1)) 디스플레이 디코딩

세그먼트 디스플레이나 기타 시각적 디스플레이 장치에서 디코더는 받은 코드를 구체적인 디스플레이 출력 신호로 변환한다. 예를 들어, 디지털 시계나 계산기에서 숫자를 디스플레이할 때 사용된다. 디지털 코드를 숫자나 문자를 표시하는 시각적 신호로 변환한다. 디지털 코드를 해석하여 세그먼트 디스플레이에 표시한다.

2)) 메모리 주소 디코딩

컴퓨터 시스템에서 디코더는 메모리 주소를 디코딩하여 특정 메모리 셀이나 레지스터를 선택할 수 있게 한다. 이를 통해 CPU는 메모리의 정확한 위치에 접근하여 데이터를 읽거나 쓸 수 있다. CPU로부터의 메모리 주소 요청을 특정 메모리 셀로 변환한다. 주소 요청을 해석하여 특정 메모리 셀에 접근을 허용한다.

3)) 데이터 라우팅

네트워킹 장비에서 디코더는 수신된 데이터 패킷의 목적지 주소를 해석하여, 올바른 출력 포트로 데이터를 전송하는 데 사용된다. 네트워크로 수신된 데이터 패킷을 목적지 주소로 변환한다. 패킷의 목적지 주소를 해석하여 적절한 네트워크 경로로 전송한다.

4)) 디지털 신호 복원

통신에서 디코더는 전송 과정 중에 변경되거나 왜곡된 디지털 신호를 원래의 데이터로 복원하는 데 사용된다. 이는 데이터의 정확성과 신뢰성을 보장하는 데 중요하다. 왜곡되거나 변경된 디지털 신호를 원본 데이터의 복원된 형태로 변환한다. 수신된 신호를 해석 및 수정하여 원래의 데이터로 복원한다.

**7. 4 to 1 line MUX 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(code, Truth table 작성)**

**1) Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **input** | | | | | | **output** |
| **S0(b)** | **S1(a)** | **A** | **B** | **C** | **D** | **O** |
| **0** | **0** | **0** | **X** | **X** | **X** | **0** |
| **0** | **0** | **1** | **X** | **X** | **X** | **1** |
| **0** | **1** | **X** | **0** | **X** | **X** | **0** |
| **0** | **1** | **X** | **1** | **X** | **X** | **1** |
| **1** | **0** | **X** | **X** | **0** | **X** | **0** |
| **1** | **0** | **X** | **X** | **1** | **X** | **1** |
| **1** | **1** | **X** | **X** | **X** | **0** | **0** |
| **1** | **1** | **X** | **X** | **X** | **1** | **1** |

**2) Source Code**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**3) Schematic**

**도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명**

**4) Simulation**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

**5) 결과 및 과정**

4 to 1 라인 MUX는 하나의 출력 신호(O)와 2개의 선택 입력(S0, S1)을 사용하여 4개의 입력(A, B, C, D) 중 하나를 선택하는 논리 회로이다. 이 회로는 선택 입력에 따라 하나의 입력 신호만을 출력으로 전달한다. 이는 각 출력이 특정 입력 조합에만 반응하도록 설계되어 있다. 예를 들어, 선택 입력이 S0=0, S1=0인 경우 A가 활성화되어 출력(O)에 나타나며(1), 나머지 B, C, D는 비활성화된다(0). Truth Table을 보면 S0와 S1이 0일때 O는 A의 값과 동일하다. S0=0, S1=1일 때는 B의 값과 동일하며, S0=1, S1=0일때는 C의 값과 동일하다. S0=1, S1=1일 때는 D의 값과 동일하다. 따라서 O의 값은 (S0’,S1’,A의 AND연산), (S0’,S1,B의 AND연산), (S0,S1’,C의 AND연산), (S0,S1,D의 AND연산)을 OR연산 해준 것과 같다. 해당 결과를 simulation을 통해 알아보기 위해 A,B,C,D는 각각 40,80,160,320마다 값을 변화시켰고, S0와 S1은 각각 10, 20마다 값을 변화시켰다. 그 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 4 to 1 line MUX의 변화를 Simulation을 통해 확인할 수 있었다.

**8. 1 to 4 line deMUX 를 이용하여 4 to 16 decoder를 수행하고 결과를 나타내시오. (코드, Truth table 작성)**

**1) 1 to 4 line deMUX**

**1)) Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **input** | | | **output** | | | |
| **E** | **A** | **B** | **O1** | **O2** | **O3** | **O4** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**강의자료에 나온 구조에서는 O2와 O3가 바뀐 상황이지만 실습에서는 위 Truth Table의 결과가 나오도록 구성하였다. 즉 0010, 0100을 0100, 0010으로 나타내었다.**

**2)) Source Code**

**텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**3)) Schematic**

**도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명**

**4)) Simulation**

**소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**5)) 결과 및 과정**

1 to 4 line deMUX는 하나의 입력 신호(E)와 2개의 선택 입력(A, B)을 사용하여 4개의 출력(O1, O2, O3, O4)으로 데이터를 분배하는 논리 회로이다. 이 회로는 하나의 입력 신호를 기반으로 선택 입력에 따라 하나의 출력만을 활성화한다. 이는 각 출력이 하나의 특정 입력 조합에만 반응하도록 설계되어 있다. 예를 들어, 입력이 E=1이고 선택 입력이 A=0, B=0인 경우 O1이 활성화되며 (1), 나머지 O2, O3, O4는 비활성화된다 (0). Truth Table을 보면 E가 0일때 Output 1,2,3,4는 모두 0의 결과를 가졌다. E가 1일 때는 AND GATE 2 to 4 Decoder와 동일한 결과를 가졌다. 따라서 AND GATE 2 to 4 Decoder와 동일하게 구성하되, E의 값을 AND연산으로 추가하여 구성하였다. O0는 A’과 B’, O1은 A’과 B, O2는 A와 B’, O3는 A와 B의 AND연산을 하였고, 모든 Output에 E값을 추가 AND 연산을 진행하였다. E는 400, A는 200, B는 100마다 값을 변화시켜 알아보기 쉽게 simulation을 하였다. 그 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 1 to 4 line deMUX의 변화를 Simulation을 통해 확인할 수 있었다.

**2) 4 to 16 decoder**

**1)) Truth Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| input | | | | output | | | | | | | | | | | | | | | |
| A | B | C | D | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 | D13 | D14 | D15 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**2)) Source Code**

**텍스트, 스크린샷, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명**

**3)) Schematic**

**텍스트, 도표, 평면도, 개략도이(가) 표시된 사진

자동 생성된 설명**

**4)) Simulation**

**스크린샷, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**5)) 결과 및 과정**

이전에 구현했던 DeMUX를 사용하면 4 to 16 Decoder를 더 쉽게 구현할 수 있다. 4 to 16 디코더의 구현을 위해, 먼저 1 to 4 DeMUX들을 기본 구성 요소로 사용한다. 각 DeMUX는 한 개의 입력과 두 개의 선택 신호를 받아 네 개의 출력을 제공한다. 이 구성에서는 총 5 개의 1 to 4 DeMUX를 사용하여, 각각의 DeMUX가 4개의 출력을 제어할 수 있도록 한다.

입력 A3, A2, A1, A0은 모든 DeMUX에 공통적으로 적용된다. A3와 A2는 상위 DeMUX의 선택 신호로 사용되며, A1과 A0은 각 DeMUX의 입력으로 전달된다. 이러한 방식으로, 각 DeMUX는 입력에 따라 네 개의 출력 중 하나를 선택하며, 전체적으로 16개의 출력이 생성된다.

이를 Verilog로 구현하기 위해 먼저 DeMUX module을 선언하였다. input으로는 A,B와 E를 받았다. 각각에 대한 Output O0~O3는 앞서 구현한 DeMUX와 동일하게 구성하였다. 이후 4 to 16 decoder를 구현하기 위해 4개의 input A0~A3를 받고, 16개의 output y0~y15를 출력하는 Decoder module을 구성하였다. 처음 입력신호 E를 N으로 생각하고, 출력되어지는 4개의 bit 역시 N으로 생각하여, 4개의 시그널 비트를 디코더에서 입력으로 받는, 4 to 16 디코더를 구현하였다. 즉 Decoder에 N0~N3의 wire를 선언하였고, DeMUX(A3,A2,1,N0,N1,N2,N3)를 하여 처음 DeMUX를 실행시켰다. 이후 DeMUX 4개를 각각의 결과에 맞게 (A1,A0,N0,Y0,Y1,Y2,Y3), (A1,A0,N1,Y4,Y5,Y6,Y7)... 구성하여 최종 4 to 16 Decoder를 구현하였다.

이를 simulation을 통해 확인한 결과 truth table과 동일한 simulation 결과를 얻었고, 올바르게 4 to 16 Decoder using 1 to 4 MUX의 변화를 Simulation을 통해 확인할 수 있었다.

**9. 결과 검토 및 논의 사항.**

이번 프로젝트에서는 다양한 디지털 회로의 설계와 시뮬레이션을 통해 여러 필수적인 논리 요소들의 동작을 검증하였다. 주요 구성 요소들의 설계와 결과는 다음과 같다.

2-to-4 디코더는 두 개의 입력 신호를 받아 네 개의 출력을 제어하며, 설계된 진리표와 K-map을 활용하여 AND 및 NAND 게이트를 사용한 최소화된 논리식을 도출하였다. 시뮬레이션 결과는 진리표와 일치하며, 모든 가능한 입력에 대해 정확한 출력을 보여주었다.

4-to-2 인코더는 네 개의 입력 중 하나만이 활성화되어야 하며, 해당 입력을 두 비트의 출력으로 변환한다. K-map을 사용하여 진리표에 따른 논리식을 간소화하였고, 인코더의 시뮬레이션은 모든 입력 조합에 대해 예상된 출력을 정확하게 나타냈다.

우선순위 인코더는 표준 4-to-2 인코더에서 처리하지 못하는 여러 입력이 활성화된 상태를 관리하며, 모든 가능한 입력 조합에 대한 논리 회로를 구성하였다. 이 회로는 복수의 입력이 활성화되었을 때 정확한 처리를 보장하며, 시스템의 신뢰성을 높인다.

BCD to Decimal 디코더는 BCD 입력을 받아 10개의 출력 중 하나를 활성화한다. 진리표와 K-map을 통해 각 출력에 대한 최소화된 논리식을 도출하였고, 시뮬레이션 과정은 모든 유효한 BCD 값에 대해 정확한 출력을 보여주며, 디코더 설계의 정확성을 확인할 수 있었다.

4-to-1 멀티플렉서는 선택 입력에 따라 한 개의 입력을 출력으로 전달하며, 각 입력 조건에 대한 코드와 진리표를 작성하여 설계된 논리를 시뮬레이션으로 검증하였다. 멀티플렉서의 유연성과 정확성은 시뮬레이션을 통해 입증되었으며, 다양한 입력 상황에서도 안정적으로 작동함을 확인하였다.

1-to-4 디멀티플렉서를 이용한 4-to-16 디코더는 네 개의 1-to-4 디멀티플렉서를 사용하여 구성되었다. 각 디멀티플렉서는 하위 디멀티플렉서의 활성화 입력으로 동작하며, 최종 출력은 16개의 독립된 채널을 제공한다. 이 구조는 각 입력의 우선순위에 따라 데이터를 정확하게 라우팅하는 데 매우 효과적이며, 복잡한 디지털 시스템에서 중요한 역할을 한다.

이 프로젝트를 통해 얻은 결과와 경험은 보다 복잡한 디지털 시스템의 설계 및 구현에 필요한 이론과 실습 지식을 확장하는 기반이 될 것이다.

**10. 추가 이론 조사 및 작성.**

2 to 4 디코더를 사용하여 3 to 8 디코더를 구현할 수 있다. 2 to 4 디코더는 두 개의 입력(A1 및 A0)과 네 개의 출력(Y3부터 Y0)을 가진다. 반면, 3 to 8 디코더는 세 개의 입력(A2, A1, A0)과 여덟 개의 출력(Y7부터 Y0)을 가진다.

도표, 스케치, 기술 도면, 라인이(가) 표시된 사진

자동 생성된 설명

보다 높은 순서의 디코더를 구현하기 위해 필요한 하위 순서 디코더의 수를 다음 공식을 사용하여 찾을 수 있다.

필요한 하위 디코더 수 = m2/m1 = 8/4 = 2이다.

따라서 하나의 3 to 8 디코더를 구현하기 위해 두 개의 2 to 4 디코더가 필요하다.

입력 A1과 A0은 각각의 2 to 4 디코더에 병렬로 적용된다. 입력 A2의 보수는 하위 2 to 4 디코더의 Enable, E에 연결되어 출력 Y3부터 Y0을 얻는다. 이들은 하위 네 개의 min terms이다. 입력 A2는 상위 2 to 4 디코더의 Enable, E에 직접 연결되어 출력 Y7부터 Y4를 얻는다. 이들은 상위 네 개의 min terms이다.

도표, 라인, 기술 도면, 평면도이(가) 표시된 사진

자동 생성된 설명

이어서 4 to 16 디코더 는 다음과 같이 구현할 수 있다. 3 to 8 디코더는 세 개의 입력(A2, A1, A0)과 여덟 개의 출력(Y7부터 Y0)을 가진다. 반면 4 to 16 디코더는 네 개의 입력(A3, A2, A1, A0)과 열여섯 개의 출력(Y15부터 Y0)을 가진다. 이때 필요한 하위 디코더 수는 m2/m1 = 16/8 = 2이다. 따라서 하나의 4 to 16 디코더를 구현하기 위해 두 개의 3 to 8 디코더가 필요하다. 입력 A2, A1, A0은 각각의 3 to 8 디코더에 병렬로 적용된다. 입력 A3의 보수는 하위 3 to 8 디코더의 Enable, E에 연결되어 출력 Y7부터 Y0을 얻는다. 이들은 하위 여덟 개의 min terms다. 입력 A3는 상위 3 to 8 디코더의 Enable, E에 직접 연결되어 출력 Y15부터 Y8을 얻는다. 이들은 상위 여덟 개의 min terms다.

도표, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명