9주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**2 to 4 Decoder**

**1) Active high**

* 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 |

위는 2개의 입력(A, B)을 받아 4개의 출력(D0, D1, D2, D3)을 내보내는 Active high 방식으로 구현한 2 to 4 Decoder의 진리표를 작성한 것이다. 이 디코더는 입력이 '00'이면 첫 번째 출력을 1로 활성화하고, '01'이면 두 번째 출력을 1로 활성화하고, '10'이면 세 번째 출력을 1로 활성화하며, '11'이면 네 번째 출력을 1로 활성화한다.

* K-map

|  |  |
| --- | --- |
| 텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진  자동 생성된 설명 |
|  |  |
| 텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진  자동 생성된 설명 |
|  |  |

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 각 출력에 대한 Boolean function을 얻을 수 있었다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module dec(  input a, b,  output d0, d1, d2, d3  );  assign d0 = ~a&~b;  assign d1 = ~a&b;  assign d2 = a&~b;  assign d3 = a&b;  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Verilog 코드를 작성한 결과이다. dec 모듈은 2개의 입력(a, b)과 4개의 출력(d0, d1, d2, d3)을 가지고 있다. 출력 d0는 ~a&~b, 출력 d1은 ~a&b, 출력 d2는 a&~b, 출력 d3는 a&b를 연산한 결과이다.

* Schematic Diagram

텍스트, 도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b의 값을 바꾸면 그에 따라 출력 d0, d1, d2, d3의 값이 변화하는 것을 확인할 수 있다. 입력이 '00'이면 d0가 1을 출력하고, '01'이면 d1이 1을 출력하고, '10'이면 d2가 1을 출력하고, '11'이면 d3가 1을 출력한다. 이때 1로 활성화된 출력을 제외한 나머지 출력들은 0을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2) Active low**

* 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 |

위는 2개의 입력(A, B)을 받아 4개의 출력(D0, D1, D2, D3)을 내보내는 Active low 방식으로 구현한 2 to 4 Decoder의 진리표를 작성한 것이다. 이 디코더는 입력이 '00'이면 첫 번째 출력을 0으로 활성화하고, '01'이면 두 번째 출력을 0으로 활성화하고, '10'이면 세 번째 출력을 0으로 활성화하며, '11'이면 네 번째 출력을 0으로 활성화한다. 이는 Decoder를 Active high 방식으로 구현한 경우의 출력값을 invert한 결과와 같다.

* K-map

|  |  |
| --- | --- |
| 도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진  자동 생성된 설명 | 도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진  자동 생성된 설명 |
|  | *(A’B)’* |
| 도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진  자동 생성된 설명 | 도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진  자동 생성된 설명 |
|  |  |

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 각 출력에 대한 Boolean function을 얻을 수 있었다. Active high 방식의 각 출력에 대한 Boolean function에 NOT을 취해 드모르간 법칙을 적용한 결과와 같다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module dec(  input a, b,  output d0, d1, d2, d3  );  assign d0 = a|b;  assign d1 = a|~b;  assign d2 = ~a|b;  assign d3 = ~a|~b;  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Verilog 코드를 작성한 결과이다. dec 모듈은 2개의 입력(a, b)과 4개의 출력(d0, d1, d2, d3)을 가지고 있다. 출력 d0는 a|b, 출력 d1은 a|~b, 출력 d2는 ~a|b, 출력 d3는 ~a|~b를 연산한 결과이다.

* Schematic Diagram

도표, 텍스트, 평면도이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b의 값을 바꾸면 그에 따라 출력 d0, d1, d2, d3의 값이 변화하는 것을 확인할 수 있다. 입력이 '00'이면 d0가 0을 출력하고, '01'이면 d1이 0을 출력하고, '10'이면 d2가 0을 출력하고, '11'이면 d3가 0을 출력한다. 이때 0으로 활성화된 출력을 제외한 나머지 출력들은 1을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2.**

**4 to 2 Encoder**

* Truth Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | |
| **A** | **B** | **C** | **D** | **E0** | **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 |

위는 4개의 입력(A, B, C, D)을 받아 2개의 출력(E0, E1)을 내보내는 4 to 2 Encoder의 진리표를 작성한 것이다. 이 인코더는 입력이 '0001'이면 출력으로 '00'을 내보내고, '0010'이면 출력으로 '01'을 내보내고, '0100'이면 출력으로 '10'을 내보내고, '1000'이면 출력으로 '11'을 내보낸다.

* K-map

|  |  |
| --- | --- |
| 텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진  자동 생성된 설명 |
|  |  |

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 각 출력에 대한 Boolean function을 얻을 수 있었다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module enc(  input a, b, c, d,  output e0, e1  );  assign e0 = a|b;  assign e1 = a|c;    endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Verilog 코드를 작성한 결과이다. enc 모듈은 4개의 입력(a, b, c, d)과 2개의 출력(e0, e1)을 가지고 있다. 출력 e0는 a|b, 출력 e1은 a|c를 연산한 결과이다.

* Schematic Diagram

도표, 라인, 그래프, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d의 값을 바꾸면 그에 따라 출력 e0, e1의 값이 변화하는 것을 확인할 수 있다. 입력이 '0001'이면 e0가 0, e1이 0을 출력하고, '0010'이면 e0가 0, e1이 1을 출력하고, '0100'이면 e0가 1, e1이 0을 출력하고, '1000'이면 e0가 1, e1이 1을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**3.**

Ecoder에서 입력 형태는 입력 비트 중 단 하나의 비트만 1의 값을 가지고 나머지 비트들은 0의 값을 가지거나(active-high), 단 하나의 비트만 0의 값을 가지고 나머지 비트들은 1의 값을 가지도록(active-low) 구성되어야 한다. 이러한 형태를 제외한 입력 값들이 **나머지 입력 형태**에 해당된다. 이러한 형태는 Encoder 구현 과정에서 고려할 대상이 아니기에 Don’t Care 형태로 지칭된다. 이는 해당 형태의 입력에 대한 출력값이 Encoder에서 정의되지 않거나 중요하지 않을 때 사용되는 용어이다. 데이터 전송 과정에서의 오류 등으로 발생할 수 있으며, 이러한 입력 형태의 경우 출력값이 명확하게 정의되지 않아서 처리가 어려울 수 있다. 따라서 이러한 입력 형태를 무시하거나 Don't Care로 취급하여 Encoder에서 특별한 출력값을 반환하지 않을 수 있다. 반면 4번 문항에서 알아볼 Priority Encoder와 같이 우선순위가 부여된 Encoder의 경우에는 "나머지 입력 형태"에 대해서도 출력값을 반환할 수 있다. 이 경우, 특정한 우선순위에 따라 나머지 입력 형태에 대한 출력값을 정의해야 한다.

**4.**

**4 to 2 Priority Encoder**는 4개의 입력 비트를 입력으로 받아 입력 값에서 가장 높은 우선순위를 가지는 비트의 값에 따라 두 개의 출력 비트를 반환하는 방식으로 논리 회로를 구현할 수 있다.

* Truth Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | | |
| **A** | **B** | **C** | **D** | **E0** | **E1** | **NR** |
| 0 | 0 | 0 | 0 | X | X | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | X | 0 | 1 | 0 |
| 0 | 1 | X | X | 1 | 0 | 0 |
| 1 | X | X | X | 1 | 1 | 0 |

위와 같이 don’t care를 사용해 4 to 2 Priority Encoder의 진리표를 작성할 수 있다. 이 인코더는 A가 1일때, B ~ D는 신경쓰지 않는다. A가 0이고, B가 1일 때 C, D는 신경 쓰지 않는다. A, B가 0이고, C가 1일때, D는 신경 쓰지 않는다. 출력 NR은 request가 없다는 것을 의미하며 입력이 모두 0인 경우에 1이 된다. 이때 출력 E0와 E1은 dont care로 나타낸다.

* K-map

|  |  |
| --- | --- |
|  |  |
|  |  |

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 각 출력에 대한 Boolean function을 얻을 수 있었다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module prienc(  input a, b, c, d,  output e0, e1, NR  );  assign e0 = a|b;  assign e1 = a|(~b&c);  assign NR = ~a&~b&~c&~d;    endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Verilog 코드를 작성한 결과이다. prienc 모듈은 4개의 입력(a, b, c, d)과 3개의 출력(e0, e1, NR)을 가지고 있다. 출력 e0는 a|b, 출력 e1은 a|(~b&c), 출력 NR은 ~a&~b&~c&~d를 연산한 결과이다.

* Schematic Diagram

도표, 라인, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이는 4 to 2 encoder의 4가지 형태가 아닌 모든 입력 형태(16가지)에 대하여 동작되는 priority encoder의 논리 회로를 나타낸다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d의 값을 바꾸면 그에 따라 출력 e0, e1, NR의 값이 변화하는 것을 확인할 수 있다. 이 우선순위 인코더는 입력이 '0000'인 경우 NR은 1을 출력한다. 다음으로 입력이 '0001'인 경우에는 '00', 입력이 '001'로 시작하는 경우에는 '01', 입력이 '01'로 시작하는 경우에는 '10', 입력이 '1'로 시작하는 경우에는 '11'을 출력한다. 이를 통해 4 to 2 Priority Encoder의 논리 회로가 의도한 대로 잘 동작한다는 것을 확인할 수 있다.

**5.**

**BCD to Decimal decoder**

* Truth Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | | | | | | | | | |
| **A0** | **A1** | **A2** | **A3** | **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 |
| 1 | 0 | 1 | 0 | X | X | X | X | X | X | X | X | X | X |
| 1 | 0 | 1 | 1 | X | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 0 | 1 | X | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X | X |

위는 4개의 입력(A0~A3)을 받아 10개의 출력(D0~D9)을 내보내는 BCD to Decimal decoder의 진리표를 작성한 것이다. 이 디코더는 입력으로 BCD 코드(0000~1001)을 받아 입력의 십진수 값(0~9)에 해당하는 출력 라인 중 하나를 활성화하여 십진수 출력을 생성하는 역할을 한다. 입력이 9 초과(1010~1111)인 경우는 BCD 코드에서 고려 대상이 아니기에 Don’t care로 처리한다.

* K-map

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 각 출력에 대한 Boolean function을 얻을 수 있었다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module bddec(  input a0, a1, a2, a3,  output d0, d1, d2, d3, d4, d5, d6, d7, d8, d9  );  assign d0 = ~a0&~a1&~a2&~a3;  assign d1 = ~a0&~a1&~a2&a3;  assign d2 = ~a1&a2&~a3;  assign d3 = ~a1&a2&a3;  assign d4 = a1&~a2&~a3;  assign d5 = a1&~a2&a3;  assign d6 = a1&a2&~a3;  assign d7 = a1&a2&a3;  assign d8 = a0&~a3;  assign d9 = a0&a3;    endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Verilog 코드를 작성한 결과이다. bddec 모듈은 4개의 입력(a0~a3)과 9개의 출력(d0~d9)을 가지고 있다. 입력 a0~a3은 BCD 코드의 각 자리수를 나타내고, 출력 d0~d9는 십진수 0~9를 나타낸다. 각 출력에는 카르노맵을 통해 구한 boolean식을 할당해주었다.

* Schematic Diagram

텍스트, 도표, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a0, a1, a2, a3의 값을 바꾸면 그에 따라 출력 d0~d9의 값이 변화하는 것을 보여준다. 이를 통해 0000~1001 까지의 입력에 대해 BCD to Decimal decoder가 올바른 출력을 나타내는 것을 확인할 수 있었다.

**6.**

**Encoder**는 데이터를 더 적은 비트 수로 나타내어 데이터를 압축하는 데 사용된다. 예를 들어, 특정 패턴을 인코딩하여 더 간단한 형태로 변환함으로써 데이터를 효율적으로 표현한다. 또한 디지털 신호를 해석하고 인코딩하여, 데이터를 처리하거나 전송하는 데 사용된다. 센서 정보를 디지털 신호로 변환하는 데에도 활용된다.

**Decoder**는 인코딩된 정보를 해독하여 의미 있는 데이터로 변환하는 데 사용된다. 예를 들어, 통신 시스템에서 전송된 정보를 해석하는 데 활용된다. 또한 여러 신호 중 특정한 신호를 선택하거나 구분하여 다시 원래의 형태로 돌리는 데에도 사용된다.

정리하자면, Encoder와 Decoder는 주로 정보를 변환하거나 해독하는 데 사용되며, 데이터 처리, 통신, 디지털 시스템 설계 등 다양한 분야에서 핵심적인 역할을 한다.

**7.**

**4 to 1 line MUX**

도표, 기술 도면, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

* Truth Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Select** | | **Input** | | | | **Output** |
| **a** | **b** | **A** | **B** | **C** | **D** | **E** |
| 0 | 0 | 1 | X | X | X | 1 |
| 0 | 0 | 0 | X | X | X | 0 |
| 0 | 1 | X | 1 | X | X | 1 |
| 0 | 1 | X | 0 | X | X | 0 |
| 1 | 0 | X | X | 1 | X | 1 |
| 1 | 0 | X | X | 0 | X | 0 |
| 1 | 1 | X | X | X | 1 | 1 |
| 1 | 1 | X | X | X | 0 | 0 |

위는 2개의 Select input(a, b)과 4개의 입력(A, B, C, D)을 받아 1개의 출력(E)을 내보내는 4 to 1 line MUX의 진리표를 작성한 것이다. 입력의 경우의 수가 64개나 되기 때문에 don’t care를 뜻하는 X를 이용하여 나타냈다. 위 진리표는 Select input a, b가 00이면 입력 A, B, C, D 중 A에 의해, 01이면 B에 의해, 10이면 C에 의해, 11이면 D에 의해 출력 값이 결정되는 것을 의미한다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module mux(  input a, b, A, B, C, D,  output e  );  assign e = (A&~a&~b)|(B&a&~b)|(C&~a&b)|(D&a&b);    endmodule |

위는 그림 (A)의 구조에 따라 Verilog 코드를 작성한 결과이다. mux 모듈은 6개의 입력(a, b, A, B, C, D)과 1개의 출력(E)을 가지고 있다. 출력 e는 (A&~a&~b) | (B&a&~b) | (C&~a&b) | (D&a&b)를 연산하여 구할 수 있다.

* Schematic Diagram

도표, 평면도, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, A, B, C, D의 값을 바꾸면 그에 따라 출력 e의 값이 변화하는 것을 확인할 수 있다. Select input이 00인 경우, 입력 A의 값을 출력하고, 01인 경우, 입력 B의 값을 출력하고, 10인 경우, 입력 C의 값을 출력하고, 11인 경우, 입력 D의 값을 출력한다. 이를 통해 4 to 1 line MUX가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**8.**

**1) 1 to 4 line deMUX**

다음은 1 to 4 line deMUX를 이용한 4 to 16 decoder를 구현하기에 앞서 실습시간에 진행했던 1 to 4 line deMUX의 시뮬레이션 과정 및 결과를 설명한 것이다.

도표, 기술 도면, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

* Truth Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Select** | | **Input** | **Output** | | | |
| **a** | **b** | **F** | **A** | **B** | **C** | **D** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

위는 2개의 Select input(a, b)과 1개의 입력(F)을 받아 4개의 출력(A, B, C, D)을 내보내는 1 to 4 line deMUX의 진리표를 작성한 것이다. deMUX는 Select input a, b가 00이면 출력 A, B, C, D 중 A에, 01이면 B에, 10이면 C에, 11이면 D에 입력 F의 값을 전달한다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module dmux(  input a, b, F,  output A, B, C, D  );  assign A = F&~a&~b;  assign B = F&a&~b;  assign C = F&~a&b;  assign D = F&a&b;    endmodule |

위는 그림 (A)의 구조에 따라 Verilog 코드를 작성한 결과이다. dmux 모듈은 3개의 입력(a, b, F)과 4개의 출력(A, B, C, D)을 가지고 있다. 출력 A는 F&~a&~b, 출력 B는 F&a&~b, 출력 C는 F&~a&b, 출력 D는 F&a&b를 연산하여 구할 수 있다.

* Schematic Diagram

도표, 텍스트, 평면도, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

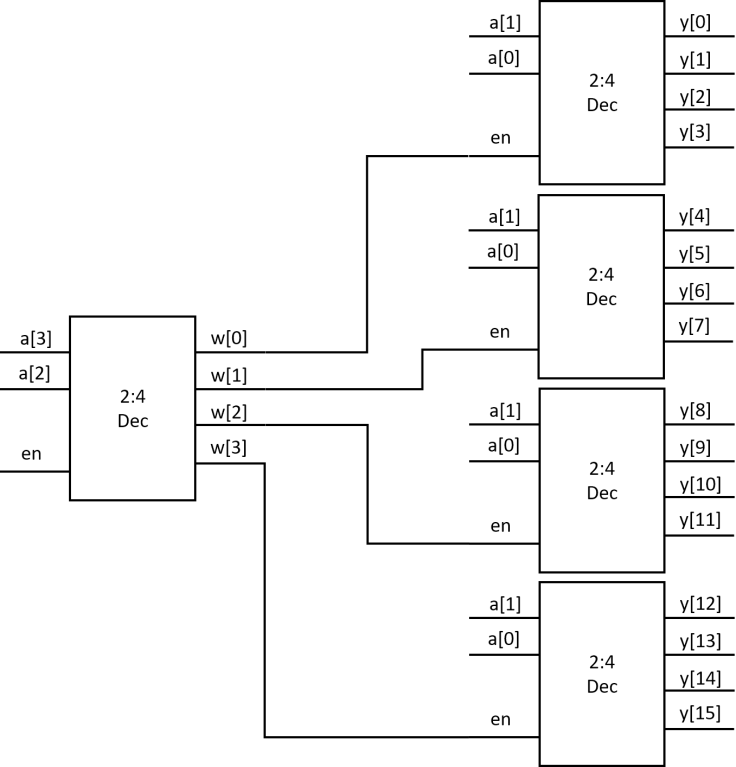
스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, F의 값을 바꾸면 그에 따라 출력 A, B, C, D의 값이 변화하는 것을 확인할 수 있다. Select input이 00인 경우, 출력 A에 입력 F의 값을 출력하고, 01인 경우, 출력 B에 입력 F의 값을 출력하고, 10인 경우, 출력 C에 입력 F의 값을 출력하고, 11인 경우, 출력 D에 입력 F의 값을 출력한다. 이를 통해 1 to 4 line deMUX가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2)** **1 to 4 line deMUX를 이용한 4 to 16 decoder**

Demux의 입력 데이터(F)를 Decoder의 enable input처럼 취급하면 해당 Demux는 enable input이 있는 Decoder와 동일하게 동작한다. 따라서, 1 to 4 deMux를 이용해 4-to-16 Decoder를 구현하는 방법은 다음 그림과 같이 2 to 4 decoder로 구현하는 방법과 유사하다. 앞서 구현한 1 to 4 line Demux의 입력 데이터(F)는 2 to 4 decoder의 enable 입력(en)과 같은 역할을 수행하기에, 그대로 사용할 수 있다.



[그림] 4 to 16 decoder using 2 to 4 decoder

* Truth Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | | | | | | | | | | | | | | | |
| **A3** | **A2** | **A1** | **A0** | **Y0** | **Y1** | **Y2** | **Y3** | **Y4** | **Y5** | **Y6** | **Y7** | **Y8** | **Y9** | **Y10** | **Y11** | **Y12** | **Y13** | **Y14** | **Y15** |
| 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 |

위는 4개의 입력(A0~A3)을 받아 16개의 출력(Y0~Y15)을 내보내는 4 to 16 decoder의 진리표를 작성한 것이다. 이 디코더는 입력이 '0000'에서 '1111'까지 변하면 각각 0 번째에서 15 번째 출력을 활성화 한다.

* Verilog 코딩

|  |
| --- |
| **Design Source Code** |
| `timescale 1ns / 1ps  module dmux(  input a, b, F,  output A, B, C, D  );  assign A = F&~a&~b;  assign B = F&a&~b;  assign C = F&~a&b;  assign D = F&a&b;    endmodule  module dec416(  input a0, a1, a2, a3,  output y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15  );    wire en0, en1, en2, en3;  dmux dm(a1, a0, 1, en0, en1, en2, en3);  dmux dm0(a3, a2, en0, y0, y1, y2, y3);  dmux dm1(a3, a2, en1, y4, y5, y6, y7);  dmux dm2(a3, a2, en2, y8, y9, y10, y11);  dmux dm3(a3, a2, en3, y12, y13, y14, y15);  endmodule |

위는 1 to 4 line deMUX를 이용한 4 to 16 decoder의 Verilog 코드를 작성한 결과이다. dmux 모듈은 4개의 입력(a0, a1, a2, a3)과 16개의 출력(y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15)을 가지고 있다. 네 개의 라인인 en0, en1, en2, en3을 사용하여 네 개의 dmux 모듈을 호출하고 있다.

* Schematic Diagram

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 의도한 대로 잘 설계된 것을 확인할 수 있다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a0, a1, a2, a3의 값을 바꾸면 그에 따라 출력 y0~y15의 값이 변화하는 것을 확인할 수 있다. 입력을 0000에서 1111로 바꾸면 그에 대응하여 출력 y0에서 y15가 순차적으로 활성화된다. 이를 통해 1 to 4 line deMUX를 이용한 4 to 16 decoder의 논리 회로가 정상적인 4 to 16 decoder와 같이 잘 동작한다는 것을 확인할 수 있다.

**9.**

* 2 to 4 Decoder: 두 개의 입력 A, B를 받아 네 가지 입력 형태(00, 01, 10, 11)에 따라 네 개의 출력선(D0, D1, D2, D3) 중 하나에서 1이 출력되는 것을 확인할 수 있었다. 이는 Active-high 방식의 경우이고, Active-low 방식의 경우 0을 출력한다.
* 4 to 2 Encoder: 네 개의 입력 A, B, C, D를 받고, 네 가지 입력 형태(0001, 0010, 0100, 1000)에 따라 출력 E0, E1이 00, 01, 10, 11 중 하나를 출력하는 것을 확인할 수 있었다.
* 4 to 2 Priority Encoder: 네 개의 입력 비트 A, B, C, D 중 가장 높은 우선순위를 가지는 비트의 값에 따라 두 개의 출력 비트 E0, E1를 반환하는 방식을 통해 모든 입력 형태에 대해 동작하는 우선순위 인코더를 구현했다.
* BCD to Decimal Decoder: BCD 코드(0000~1001)의 각 자리수를 4개의 입력(A0~A3)으로 받아 입력의 십진수 값(0~9)에 해당하는 출력 라인 중 하나를 활성화하여 십진수 출력(D0~D9)을 생성하는 것을 확인할 수 있었다.
* 4 to 1 Line MUX: 두 개의 select input 값 a, b에 따라 네 개의 입력(A, B, C, D) 중 하나를 출력(E)하는 것을 확인할 수 있었다.
* 1 to 4 Line Demux: 두 개의 select input 값 a, b에 따라 네 개의 출력(A, B, C, D) 중 하나를 선택해 입력(F)을 전달하는 것을 확인할 수 있었다.
* 1 to 4 line deMUX를 이용한 4 to 16 decoder: 4개의 입력(A0~A3)을 받아 16가지 입력 형태(0000~1111)에 따라 16개의 출력선(Y0~Y15) 중 하나에서 1이 출력되는 것을 확인할 수 있었다.

위의 모든 회로의 시뮬레이션 결과가 의도한대로 동작하는 것을 확인해 구현이 올바르게 이루어졌음을 알 수 있었다. 4 to 2 Priority Encoder는 실습에서 다루지 않아 보고서 작성 시 직접 회로를 구성해봤는데, 실습 시간에서 다룬 내용들이 우선순위 인코더 구현 시에 도움이 되어 생각보다 수월하게 구현할 수 있었다.

**10.**

**Multiplexer(Mux)와 Demultiplexer(Demux)**는 디지털 회로 설계에서 사용되는 중요한 구성 요소로, 각각의 **주요 응용**은 다음과 같다.

Multiplexer (Mux)

1. 데이터 전송 및 선택: Mux는 여러 입력 중에서 하나의 입력을 선택하여 출력으로 전송하는 데 사용된다.
2. 주소 및 데이터 버스: CPU와 기타 하드웨어 간의 통신에서 주소 및 데이터를 전송하는 데 사용된다. 주로 여러 주소 중 하나를 선택하여 데이터를 전송하는 데 활용된다.
3. 시그널 및 채널 선택: 여러 입력 중 하나를 선택하여 특정한 시그널이나 채널을 선택하는 데 사용된다.

Demultiplexer (Demux)

1. 데이터 분배 및 선택: Demux는 단일 입력을 여러 출력 중 하나를 선택하여 전송하는 데 사용된다. 주로 하나의 입력을 여러 출력으로 분배하고, 데이터를 선택하거나 분배하는 데 활용된다.
2. 비트 및 신호 분배: 하나의 비트 또는 신호를 여러 부분으로 분해하여 각각의 부분을 선택하는 데 사용된다. 데이터를 분해하고 각 부분에 전달하는 데 활용된다.
3. 시그널 및 채널 분배: 하나의 시그널이나 채널을 여러 개로 분배하여 다양한 출력 장치에 전송하는 데 활용된다. 특히 통신 시스템에서 주로 사용된다.