1. 9주차 예비보고서

|  |
| --- |
| - Decoder  - Encoder  - Multiplexer  - DeMultiplexer  - Priority Encoder  - 기타 |

**20141196 김성희**

1. Decoder
   1. **가. 기본 개념**

- Decoder란 한글로 표현하면 해석기정도로 표현할 수 있다. 즉 입력 값을 어떤 방법으로 해석해서 출력 값을 내놓는 회로를 의미한다.

- 어떤 방식:

1. 입력 값으로 0~3, 0~7, 0~15, … 을 이진수로 표현한다. 즉 각각 2개, 3개, 4개, … 의 비트를 가지는 이진수를 입력 값으로 받는다.

2. 출력 값은 입력 값의 경우의 수만큼 존재한다. 예를 들어 0~3의 2개의 비트를 가지는 입력 값은 4개의 출력 값을 가진다.

3. 출력 값은 한 개를 제외하고 모두 0을 가진다. 그 한 개는 1의 값을 가지며 입력 값의 십진수에 맞춰 정해진다. 예를 들어 입력 값이 0(00)이면 첫 번째 출력 값이 1이고, 1(01)이면 두 번째 출력 값이 1을 갖는다.

- 다음 표를 보면 이해가 쉬울 것이다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **N** | **X** | **Y** | **F0** | **F1** | **F2** | **F3** |
| **0** | **0** | **0** | **1** | **0** | **0** | **0** |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** |
| **2** | **1** | **0** | **0** | **0** | **1** | **0** |
| **3** | **1** | **1** | **0** | **0** | **0** | **1** |

<- FN=1, 나머지는 0인 것을 볼 수 있다.

<표1>

- 이번 실험(8주차 실험)에서 공부한 7-segment display도 사실 Decoder와 비슷한 원리를 가진다. 입력 값이 0~15인 4개의 비트이며 출력 값은 A,B,C,~,G,DP,X,~,X는 16개 비트의 출력 값이다. 다만 Decoder와는 다르게 0->100…000, 1->010…000과 같은 규칙을 가지지는 않는다.

- Decoder를 간단하게 표현하면 n-bit를 2­n-bit로 바꿔 주는 회로다.

**나. 구현 방법**

**- 구현 방법은 7-segment display와 같다. 위 ‘가. 기본 개념’의 표1처럼 표를 그리고 그에 대한 카르노맵을 그려 논리식을 구한다. (사실 카르노맵을 구할 필요도 없이 F0=X’Y’, F1=X’Y, F2=XY’, F3=XY 와 같이 바로 구할 수 있다.)**

1. Encoder

**가. 기본 개념**

- Encoder를 한글로 표현하면 암호기로 표현할 수 있다. 그리고 Decoder와 같은 방식을 이용해 2n-bit를 n-bit로 바꿔 주는 회로다. (Encoder는 Decoder의 입력 값과 출력 값이 서로 뒤바뀐 회로다.)

- Encoder와 Decoder는 밀접한 관계를 가지고 있다. 그 이유는 다음 그림을 보면 이해할 수 있다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **F0** | **-->** | **4 X 2**  **Encoder** |  | **2 X 4**  **Decoder** | **-->** | **F0** |
| **F1** | **-->** | **---------🡪** | **-->** | **F1** |
| **F2** | **-->** | **---------🡪** | **-->** | **F2** |
| **F3** | **-->** |  | **-->** | **F3** |

- Encoder에 대한 진리 표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F0** | **F1** | **F2** | **F3** | **X** | **Y** |
| **1** | **0** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **1** | **0** |
| **0** | **0** | **0** | **1** | **1** | **1** |

논리식은 다음과 같다.

X = F2 OR F3

Y = F1 OR F3

- Encoder를 사용하는 이유는 사실 ‘암호’ 보다는 다른 장치가 잘 알아들을 수 있게 바꿔 주기 때문이다. 예를 들어 F0~F3의 버튼이 있다고 할 때, 어떤 버튼을 눌렀는지 컴퓨터가 알아듣기 쉽게 0,1,2,3번의 번호를 부여하는 것이다. (Decoder는 반대로 사람이 읽을 수 있게 바꾸는 역할을 한다.)

**나. 구현 방법**

위 논리식으로 구현한다. (사실 OR게이트로 간단하게 구현 가능하므로 따로 IC회로를 제작하지 않는다.)

1. Multiplexer(MUX)
2. 가. 기본 개념

- 앞의 Decoder와 Encoder는 특정 법칙을 이용해 값을 바꿔 주는 것이 원리인 회로였다. 그러나 MUX의 키워드는 ‘변화’가 아니라 ‘선택’에 있다. 즉 MUX는 여러 개의 입력 값 중 하나의 입력 값을 선택하여 출력 값으로 내놓는 회로다.

- 다음 표와 그림을 보면 이해하기 쉬울 것이다. (4X1 MUX : 4개의 입력 중 1개 선택)

표

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Y** |
| **0** | **0** | **i0** |
| **0** | **1** | **i1** |
| **1** | **0** | **i2** |
| **1** | **1** | **i3** |

|  |
| --- |
| **i0** |
| **i1** |
| **i2** |
| **i3** |

**4X1 MUX**

**Y**

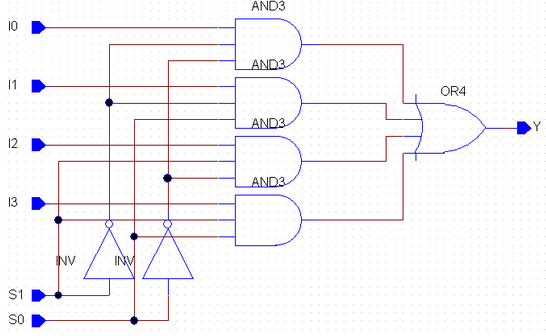
|  |
| --- |
| **S0** |
| **S1** |

\* 왼쪽 그림은 논리상 입력 4개 중 1개를 선택하는 회로지만 실제로는 6개의 입력이 필요하다. 그 이유는 오른쪽 표에 나온다. 2개 비트는 4개의 수를 만들 수 있기 때문이다. 즉 n개의 비트 중 1개를 선택하는 nX1 MUX는 n개 비트 이외에 [log2n]개의 비트를 추가로 입력해야 한다.

1. 나. 구현 방법

다음과 같은 논리식을 통해 구현하면 된다. Y = i0S0**‘**S1**‘** + i1S0S1**‘** +i2S0**‘**S1+ i3S0S1

실제 회로도는 아래 그림과 같다.



1. DeMultiplexer(DEMUX)
2. 가. 기본 개념

- encoder와 decoder의 관계와 같다. Multiplexer의 방식을 반대로 진행하는 회로다.

|  |
| --- |
| **i0** |
| **i1** |
| **i2** |
| **i3** |

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Y** |
| **0** | **0** | **i0** |
| **0** | **1** | **i1** |
| **1** | **0** | **i2** |
| **1** | **1** | **i3** |

1X4

DEMUX

**F**

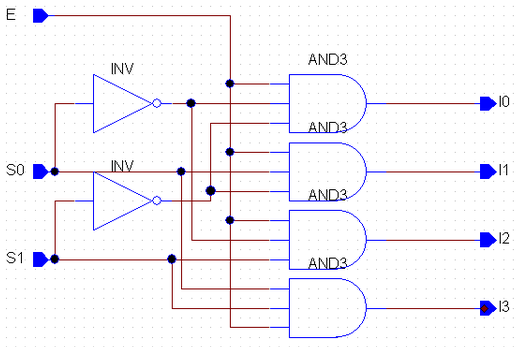
|  |  |
| --- | --- |
| **S0** | **S1** |

\* F가 1일 때, S0 S1의 값에 따라 i0, i1, i2, i3 의 값이 정해진다. 00이면 i0가 1, 01이면 i1이 1, 10이면 i2가 1, 11이면 i3이 1이 되는 형식이다.

\* F를 없앤다면 2X4 Decoder와 동일하다는 것을 알 수 있다. **DEMUX**와 **Decoder**의 **차이점**은 물리적으로는 F의 유무이고 논리적으로는 ‘선택’과 ‘변화’의 차이다. 즉 DEMUX는 많은 입력 장치 중 하나의 입력 장치를 출력하고 Decoder는 많은 입력 장치에서 많은 출력 장치를 출력한다.

나. 구현 방법

i**0** = FS**0’**S**1’** i**1** = FS**0**S**1’** i**2** = FS**0’**S**1** i**3** = FS**0**S**1** 이와 같은 식을 통해 회로를 구성하면 다음과 같은 회로가 나온다.



1. Priority Encoder
2. 가. 기본 개념

- 일반 적인 Encoder와 동일한 방식이다. 단 차이점은 말그대로 우선순위를 주는 것이다. 다음 표를 보도록 하자.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **F0** | **F1** | **F2** | **F3** | **X** | **Y** |
| **1** | **0** | **0** | **0** | **0** | **0** |
| **X** | **1** | **0** | **0** | **0** | **1** |
| **X** | **X** | **1** | **0** | **1** | **0** |
| **x** | **X** | **x** | **1** | **1** | **1** |
| **0** | **0** | **0** | **0** | **X** | **X** |

위 표에서 알 수 있듯이 Encoder와는 달리 X표시가 된 입력이 보일 것이다. F1이 1이면 F0가 0이든 1이든 무조건 XY는 01 값을 가진다. 그리고 F2가 1이라면 F0, F1이 어떤 값을 가지더라도 반드시 X Y가 10 값을 가진다. 마찬가지로 F3이 1이라면 F0, F1, F2가 어떤 값을 가지든 XY는 11이다. 즉 F0 < F1 < F2 < F3 순으로 우선순위를 부여했다는 의미이다. (F3이 제일 높고 F0가 제일 낮다.)

- 왜 priority encoder를 사용하는가?

그 이유는 2개이상의 1을 가지는 입력이 잘못 들어왔을 때 어떻게 처리해야 하는가에 대한 의문에서 시작되었다. 우선순위를 부여함으로써 2개이상의 1이 존재해도 XY의 값이 충분히 잘 튀어 나온다.

1. 나. 구현 방법

역시 카르노맵을 통해 논리식을 구한다. (왼 : X, 오 : Y)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **F0F1**  **F2F3** | **00** | **01** | **11** | **10** |
| **00** | **0** | **0** | **0** | **0** |
| **01** | **1** | **1** | **1** | **1** |
| **11** | **1** | 1 | **1** | **1** |
| **10** | **1** | 1 | **1** | **1** |

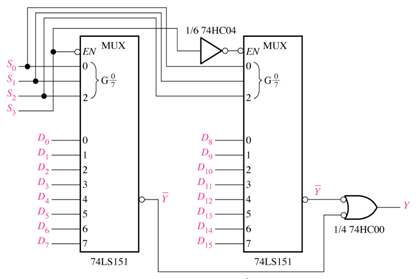
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **F0F1**  **F2F3** | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **1** | **0** |
| **01** | **1** | **1** | **1** | **1** |
| **11** | **1** | 1 | **1** | **1** |
| **10** | **0** | 0 | **0** | **0** |

X = F2 + F3 Y = F1F2**’** +F3

1. 기타

- Decoder, Encoder, MUX, DEMUX는 Enable 입력을 추가하는 것으로 확장이 가능하다.

- 예를 들어 8X1 MUX의 경우 2개를 사용하여 16X1 MUX를 만들 수 있다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S3** | **S2** | **S1** | **S0** | **Y** |
| 0 | 0 | 0 | 0 | D0 |
| 0 | 0 | 0 | 1 | D1 |
| 0 | 0 | 1 | 0 | D2 |
| 0 | 0 | 1 | 1 | D3 |
| 0 | 1 | 0 | 0 | D4 |
| 0 | 1 | 0 | 1 | D5 |
| 0 | 1 | 1 | 0 | D6 |
| 0 | 1 | 1 | 1 | D7 |
| 1 | 0 | 0 | 0 | D8 |
| 1 | 0 | 0 | 1 | D9 |
| 1 | 0 | 1 | 0 | D10 |
| 1 | 0 | 1 | 1 | D11 |
| 1 | 1 | 0 | 0 | D12 |
| 1 | 1 | 0 | 1 | D13 |
| 1 | 1 | 1 | 0 | D14 |
| 1 | 1 | 1 | 1 | D15 |

첫 번째와 두번째 MUX 모두 S0~S2의 값으로 각각 D0~7, D8~15 에서 하나씩 값이 선택된다. 그리고 S3으로 Enable을 표시하여 EN이 0이면 선택 값을 0으로, 1이면 그대로 보낸다. 즉 S3이 0이면 첫번째 MUX에서 선택을, S3이 1이면 두번째 MUX에서 선택을 한다.

(마지막에 Y1 OR Y2를 해주면 되는데 NAND게이트가 기본 게이트 이기에 OR 게이트를 NAND 게이트로 바꿔주었다.)