9주차 결과보고서

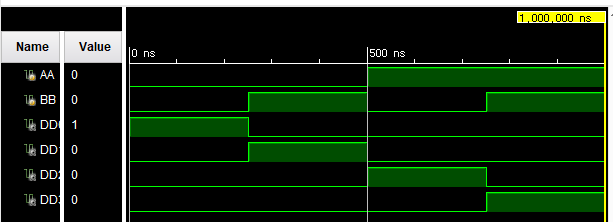
전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **2 to 4 Decoder의 결과 및 simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함.[AND,NAND Gate])**
2. AND gate

테이블이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

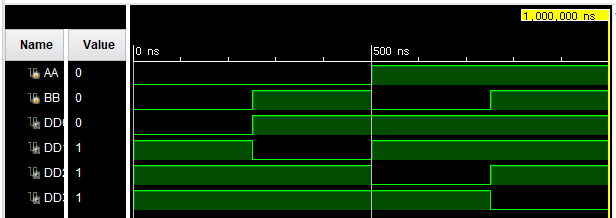


1. NAND gate

텍스트이(가) 표시된 사진

자동 생성된 설명 테이블이(가) 표시된 사진

자동 생성된 설명



2 to 4 decoder란 2 bit로 인코딩 된 데이터를 받아 원래의 데이터로 decoding해주는 기계를 의미한다. 따라서 2개의 input으로 나타낼 수 있는 4가지 경우를 이용하여 4가지 output bit에 대하여 active high 또는 active low를 이용하여 원래의 데이터를 표현한다. 따라서 이를 진리표로 작성하면 다음과 같다.

1. Active high

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | 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 |

1. Active low

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | 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 |

이번 실습에서는 active high를 이용하여 회로를 구성했다. Boolean식을 계산하기 위해 카르노 맵을 작성해보자.

-active high

1. D0

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 D0=~A&~B이다.

1. D1

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 D1=~A&B이다.

1. D2

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 D2=A&~B이다.

1. D3

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 D3=A&B이다.

위의 Boolean 식들은 AND게이트롤 통해 회로를 구성했다. 이를 NAND게이트를 이용하여 나타내려면 Active low decoder를 구현해야 한다. 실습에서는 진행하지 않았지만 active low의 카르노 맵 또한 구현해보자.

-active low

1. D0

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 ~D0=~A&~B이다.

1. D1

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 ~D1=~A&B이다.

1. D2

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

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 ~D2=A&~B이다.

1. D3

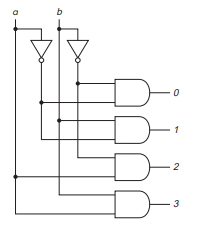
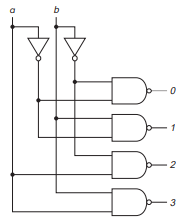
|  |  |  |
| --- | --- | --- |
| A  B | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

1이 하나이므로 1에 해당하는 항에 대한 Boolean 식을 작성하면 ~D3=A&B이다.

위 식들을 정리하여 나타내면 D0=~(~A&~B), D1=~(~A&B), D2=~(A&~B), D3=~(A&B)이고 이는 NAND게이트로 구현할 수 있다.

또한 구한 boolean식을 Verilog 코드로 구성하여 simulation 결과를 확인했다. 위의 simulation 결과와 직접 작성한 진리표가 일치하는 것을 확인할 수 있었다.

Schematic을 통해 구성한 회로를 비교해보자.

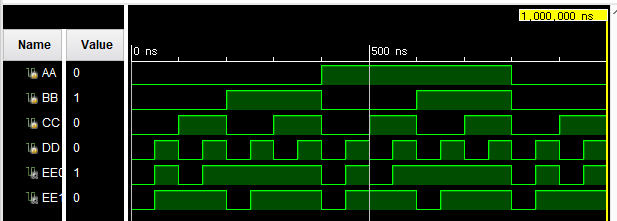
두 가지 형태의 decoder는 각각 bit의 어떤 상태를 유효한 데이터로 사용할 것인 것에 대해서 서로 다르며 그 외 기능적인 부분은 동일하다.

1. **4 to 2 Encoder의 결과 및 simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함.)**

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명



4 to 2 encoder는 데이터의 압축 또는 보안의 이유로 데이터의 양을 줄이는 것을 의미한다. 4 to 2는 4가지의 bit를 2개로 줄인다는 의미로 input으로 4개의 bit가 들어오고 output으로 2개의 bit가 나온다. encoding에서 사용하는 input bit에 대하여 0001, 0010, 0100, 1000처럼 4개 중 어떤 bit에 active한 값이 들어오냐에 따라 회로를 구성하며 나머지 case에 대해서는 don’t care이다. 따라서 이를 truth table로 작성하면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E0 | Out E1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |

이를 k-map으로 작성해보자.

1. E0

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

0을 제외하고 don’t care와 1을 가장 큰 implicant로 묶어 2, 3열 8개, 2, 3행 8개를 묶어 E0=B|D로 나타낼 수 있다.

1. E1

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

0을 제외하고 don’t care와 1을 가장 큰 implicant로 묶어 2, 3행 8개, 3, 4행 8개를 묶어 E0=C|D로 나타낼 수 있다.

구한 boolean식을 통해 Verilog 코드를 구현하여 simulation 결과를 확인했다. 나머지 경우를 제외하고 주어진 0001, 0010, 0100, 1000의 경우만 확인하면 작성한 truth table과 동일하게 결과가 나타난 것을 알 수 있다.

1. **4 to 2 Encoder에서 입력 형태 4가지를 제외한 나머지 입력 형태는 무엇을 뜻하는지 설명하시오.**

위의 2번에서 4가지 입력 형태를 제외한 나머지 경우는 don’t care로 처리하여 회로를 구성했다. 이는 input으로 4가지 형태만 들어온다는 가정하에 회로를 구성한 것으로 만약 나머지 형태의 input도 들어온다면 회로의 빈틈이 생기게 된다. 예를 들어 위 2번의 Verilog 코드로 나온 simulation 결과를 확인하자. 이는 나머지 경우에 대해서도 모두 나타내고 있다. 하나의 경우로 C와 D가 1인 경우를 보자. 이때 output으로 11이 나타난다. 이는 하나의 input에 대해 하나의 output으로 encoding하지 못해 후에 decoding시 어떤 데이터가 송신되었는지 정확하게 판단할 수 없는 치명적인 단점을 가지고 있다. 따라서 이를 해결하기 위해, priority encoder를 사용한다.

1. **4 to 2 Encoder의 4가지 형태가 아닌 모든 입력 형태(16가지)에 대하여 동작되는 priority encoder의 논리 회로를 구성하여라.**

Priority encoder는 모든 16가지의 입력 형태에 대해 encoding을 진행하기 위한 회로이다. 기본적인 동작 원리는 이름 그대로 priority, 우선순위를 부여하는 것이다. 각 자릿수마다 우선 순위를 부여하여 가장 높은 우선순위에 active가 발견되면 하위 순위는 어떤 값이 나오든 don’t care하는 것이다. 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out E0 | Out E1 |
| 0 | 0 | 0 | 0 | X | X |
| 1 | 0 | 0 | 0 | 0 | 0 |
| X | 1 | 0 | 0 | 0 | 1 |
| X | X | 1 | 0 | 1 | 0 |
| X | X | X | 1 | 1 | 1 |

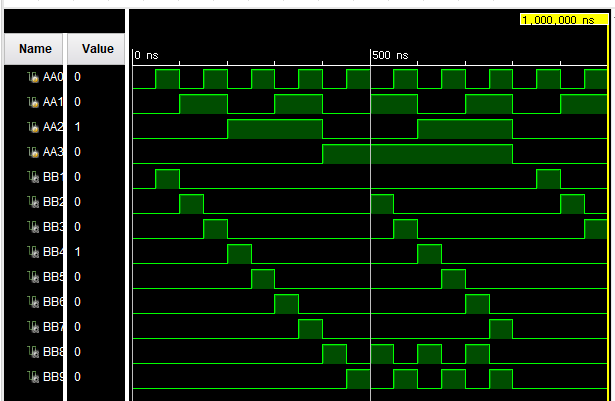
X친 곳의 data는 0이 오든, 1이 오든 신경 쓰지 않고 상위 비트의 active를 확인하여 encoding한다. 따라서 모든 X에 값을 부여하여 모든 경우의 수를 표시하면 16가지 입력 형태 모두 표현할 수 있다. 이외, 0000의 input의 경우 active한 bit가 없으므로 이는 encoding하지 않고 valid bit라는 output을 하나 추가하여 이 경우에만 0을 출력하고 나머지 경우는 1을 출력하도록 한다. 이 valid bit는 encoding의 작동 유무를 판단해주는 output이다. 각 output에 대한 Boolean식은 E0=D|C, E1=D | ~C&B이다.

1. **BCD to Decimal decoder의 결과 및 simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함.)**

테이블이(가) 표시된 사진

자동 생성된 설명 테이블이(가) 표시된 사진

자동 생성된 설명



BCD to Decimal decoder도 decoder의 한 종류로 encoding된 데이터를 원래의 데이터로 decoding하는 기능을 가지고 있다. 이름 그대로 binary coded decimal를 decimal하게 바꾸는 것으로 4가지의 input데이터를 통해 9가지의 output을 나타낸다. 이때 4가지 input에 대해서 각 자릿수마다 8421값을 부여하여 output의 값을 나타낸다. 예를 들어, 0001은 output B1로, 0010은 output B2로 decoding된다. 표현해야할 값은 총 10개이고 9개의 output을 사용한다. 진리표로 나타내면 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

10개의 입력형태를 제외한 나머지 6개의 입력에 대한 값들은 모두 don’t care이다. Boolean 식을 구하기 위해 k-map을 작성해보자.

1. B1

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 1 자체이다. 따라서 B1=~A&~B&~C&D

1. B2

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 4행 1, 4열이다. 따라서 B2=~B&C&~D이다.

1. B3

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 3행 1, 4열이다. 따라서 B3=~B&C&D

1. B4

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 1행 2, 3열이다. 따라서 B4=~B&~C&~D이다.

1. B5

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 2행 2, 3열이다. 따라서 B5=B&~C&D이다.

1. B6

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 4행 2, 3열이다. 따라서 B6=B&C&~D이다.

1. B7

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 3행 2, 3열이다. 따라서 B7=B&C&D이다.

1. B8

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 1행 3, 4열이다. 따라서 B8=A&~C&~D이다.

1. B9

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

0을 제외하고 1과 don’t care를 가장 크게 묶은 implicant는 2행 3, 4열이다. 따라서 B9=A&~C&D이다.

이를 Verilog 코드를 통해 구현하여 simulation결과를 확인하면 기존에 작성했던 truth table과 결과가 일치하는 것을 알 수 있었다.

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

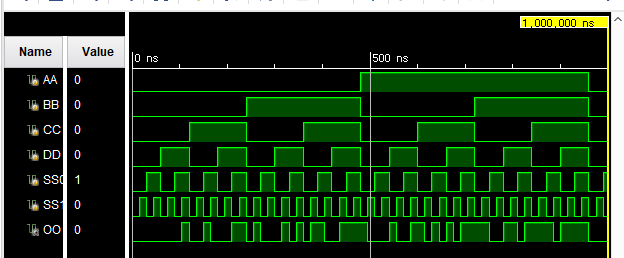
Encoder와 decoder는 데이터의 수를 줄이거나 늘리는 기능을 한다. 이는 데이터를 압축하는 기능에 더불어 데이터가 변환되어 데이터 보안의 기능도 수행할 수 있다. 9주차 팀의 발표처럼 데이터 통신에서 보안의 기능을 위해 사용된다. 예시로 manchester code가 있다. Manchester code는 데이터와 clock signal을 통해 만들어진다. Clock signal은 일정 시간마다 반복하여 signal을 바꾸는 clock의 역할을 하며 clock과 data를 이용하여 XOR게이트를 통해 manchester코드가 만들어진다. Encoding된 manchester코드를 decoding할 때는 역 연산을 통해 manchester code가 양의 부호를 띄면 clock과 반대의 값을 데이터로, 음의 부호를 띄면 clock과 같은 값을 데이터로 정하면 된다. 또한 보안 외에 동영상 파일을 압축하거나 음성 신호를 처리할 때도 encoding과 decoding이 사용된다.

1. **4 to 1 line MUX 의 결과 및 simulation 과정에 대해서 설명하시오. (code, truth table 작성)**

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명



4 to 1 line MUX는 4가지 input에서 mux회로를 통해 하나의 값을 output으로 정하여 나타내는 것이다. 4 to 1 line mux에서는 선택할 input 4개 이외에 선택의 과정을 수행하기 위한 2개의 input이 함께 들어간다. 2개의 input을 사용하여 4가지의 경우의 수를 만들어 선택을 수행할 수 있다. 이 2가지 input에 대한 진리표를 작성하면 다음과 같다.

|  |  |  |
| --- | --- | --- |
| In S0 | In S1 | Out |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

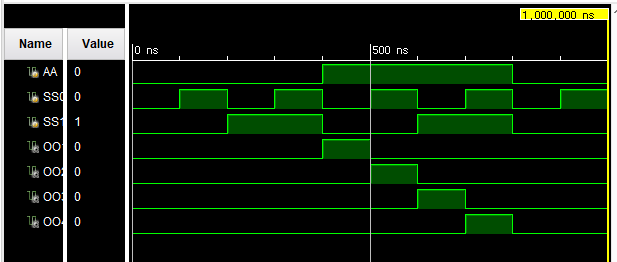
S0와 S1을 selector input으로 선택을 수행하는 input이다. 위와 같이 S0와 S1으로 나타낼 수 있는 4가지의 경우의 수를 통해 A, B, C, D 중 하나의 input을 선택하게 된다. 선택을 진행한 후에 A, B, C, D의 데이터를 output으로 출력한다. 회로의 구성은 S0와 S1을 A, B, C, D를 AND게이트로 묶어 선택 과정과 출력을 진행하고 각 A, B, C, D에 대한 게이트를 다시 OR게이트로 묶어 최종 output을 결정한다. A에 대한 회로를 예시로 나타내면 A&~S0&~S1이다. 최종 output O에 대한 Boolean 식은 O=A&~S0&~S1 | B&~S0&S1 | C&S0&~S1 | D&S0&S1이다. 이를 코드로 작성하여 simulation 결과를 확인하면 작성한 truth table과 일치하게 결과가 나타난 것을 확인할 수 있다.

1. **1 to 4 line deMUX를 이용하여 4 to 16 decoder를 수행하고 결과를 나타내시오. (code, truth table 작성)**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명



1 to 4 line demux는 mux와 반대의 기능을 하는 회로이다. 4가지의 output 중 하나를 선택하여 input하나에 대한 동일한 값을 출력한다. Mux와 마찬가지로 선택을 하기 위한 select input이 필요하다. S0와 S1을 통해 어떤 output을 선택하여 출력할지 결정하고 선택한 output에 input의 값을 출력한다. 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| In A | In S0 | In S1 | Out O1 | Out O2 | Out O3 | Out 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 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

예를 들어, S0가 1, S1이 0인 경우를 보자. 이 때, output 중 O2가 선택되고 A의 값을 결과로 출력한다. 이를 boolean 식으로 나타내면 O1의 경우 O1=A&~S0&~S1이다. 나머지도 구해보면 O2=A&S0&~S1, O3=A&~S0&S1, O4=A&S0&S1이다.

이를 verilog코드로 구성하여 simulation결과를 확인하면 작성한 truth table과 결과가 일치하는 것을 확인할 수 있다.

1. **결과 검토 및 논의 사항.**

각각 encoder와 decoder, mux와 demux를 원하는 기능을 수행할 수 있도록 진리표를 직접 작성하고 k-map을 통해 boolean식을 구했다. 이를 Verilog 코드로 작성하여 simulation결과를 확인하고 FPGA를 통해 동작을 검증했다. Simulation 결과와 직접 작성한 truth table이 일치하는 것을 확인할 수 있었고 FPGA로 동작을 검증한 결과 원하는 기능을 수행하는 것을 확인할 수 있었다.

1. **추가 이론 조사 및 작성.**

실습에서 직접 구성한 encoder와 decoder, mux와 demux를 병렬적으로 연결하면 더 많은 input과 output에 대해 같은 기능을 수행하는 회로를 구성할 수 있다. Encoder의 경우 4 to 2 encoder를 병렬로 연결하여 8 to 3 encoder를 구성할 수 있다. 또한 decoder의 경우도 2 to 4 decoder를 병렬로 연결하여 3 to 8 decoder를 만들 수 있다.