9주차 결과보고서

전공 : 국제한국학과 학년 : 4학년 학번 : 20181202 이름 : 김수미

**1. 2 to 4 Decoder 의 결과 및 Simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함, AND/NAND 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

|  |  |
| --- | --- |
| D0 | D1 |
|  |  |
| D2 | D3 |
|  |  |

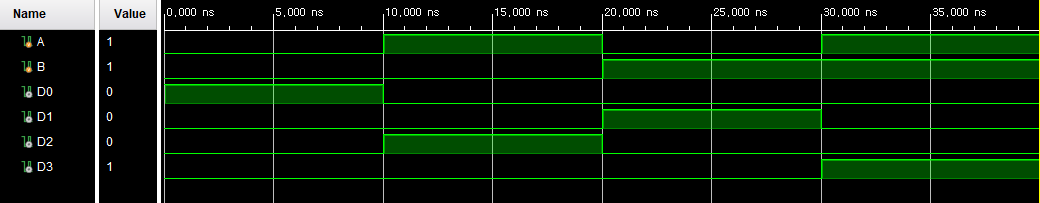
3) Source Code

|  |  |
| --- | --- |
| Using AND gate | |
| Design Source | Test Bench |
|  |  |
| Using NAND gate | |
| Design Source | Test Bench |
|  |  |

4) Schematic

|  |
| --- |
| Using AND gate |
|  |
| Using NAND gate |
|  |

5) Simulation



6) 결과 및 과정

실습에서 구현한 2 to 4 decoder는 N bit의 입력이 주어질 때 2^N bit 의 출력을 반환하며, 이 출력 bit는 단 하나의 bit만 1이고 나머지는 0 이거나(active high), 단 하나의 bit만 0이고 나머지는 1의 형태를 띤다(active low). 실습에서는 active high 방식의 decoder를 구현했다. A,B가 하나의 2bit 입력으로 들어오며 A를 MSB로 간주하여 A,B의 값이 00으로 들어오면 D0, D1,D2,D3의 출력은 1000, 01이면 0100, 10이면 0010, 11이면 1111이 되도록 설계했다.

**2. 4 to 2 Encoder 의 결과 및 Simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

1) Truth Table

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

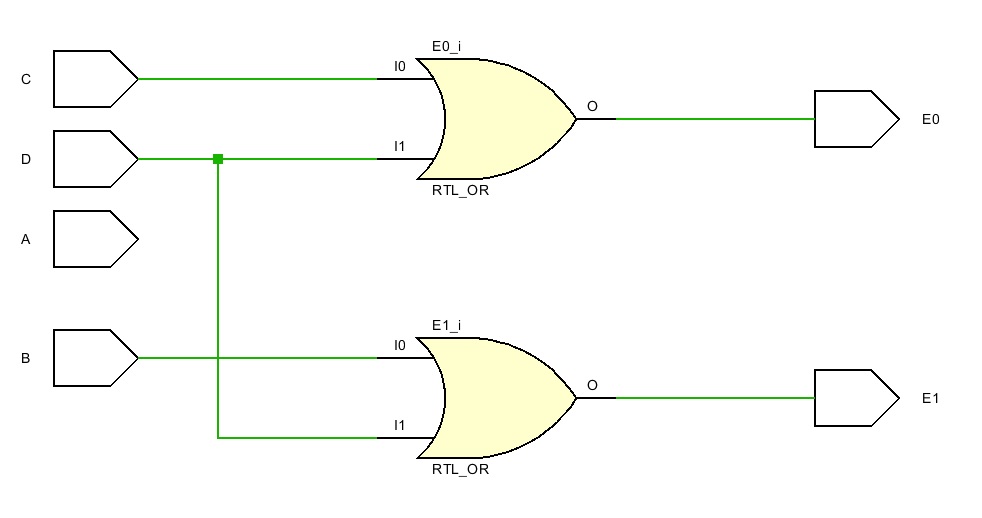
2) Karnaugh Map

|  |  |
| --- | --- |
| E0 | E1 |
|  |  |

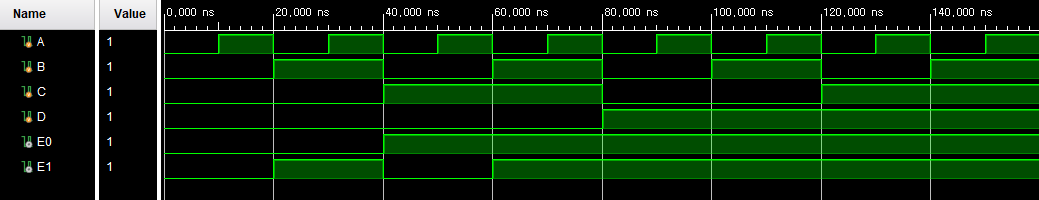
3) Source Code

|  |  |
| --- | --- |
| Design Source | Test Bench |
|  |  |

4) Schematic



5) Simulation



6) 결과 및 과정

실습에서 구현한 4 to 2 encoder는 Decoder에서 출력값이었던 것을 입력값으로 받아 Decoder에서 입력값이었던 것을 출력값으로 반환한다. 즉 4bit의 입력값을 A가 MSB인 ABCD라고 할 때, 0001의 입력값에 대해서는 11의 출력을, 0010에 대해서는 10, 0100에 대해서는 01, 1000에 대해서는 00의 출력값을 반환한다.

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

Ecoder에서 입력 형태는 여러개의 입력 bit 중 단 한개의 bit만 1의 값을 가지고 나머지는 0의 값을 가지거나, 단 한개의 bit만 0의 값을 가지고 나머지는 1의 값을 가지는 형태를 띠어야 한다. 즉 이러한 형태를 띠지 않는 값들이 위에서 이야기하는 ‘나머지 입력 형태’에 해당하며 이러한 입력 값은 데이터 전송 과정에서의 오류 등으로 발생할 수 있다. 일반적인 Encoder의 경우 이러한 입력 값은 Don’t Care로 처리되지만, 이러한 값들에 대해서도 출력값을 반환할 수 있도록 설계된 priority encoder를 사용하면 입력 형태 4가지를 제외한 나머지 입력 형태에 대해서도 출력값을 반환받을 수 있다.

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

1) Truth Table

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

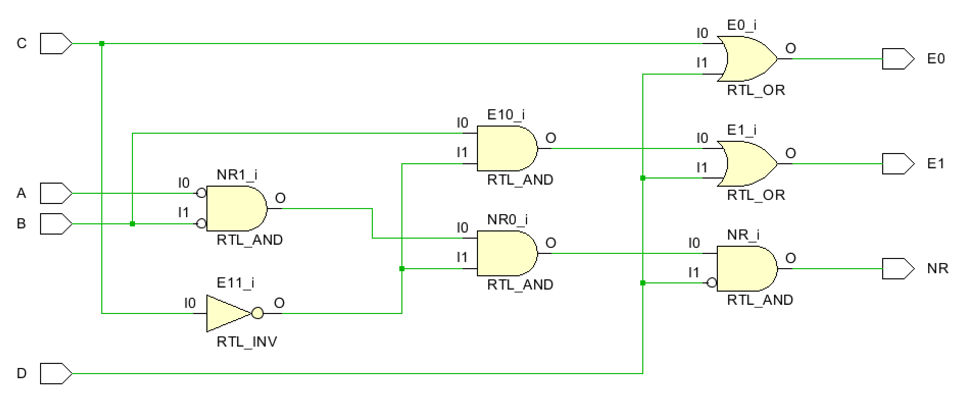
2) Karnaugh Map

|  |  |
| --- | --- |
| E0 | E1 |
|  |  |

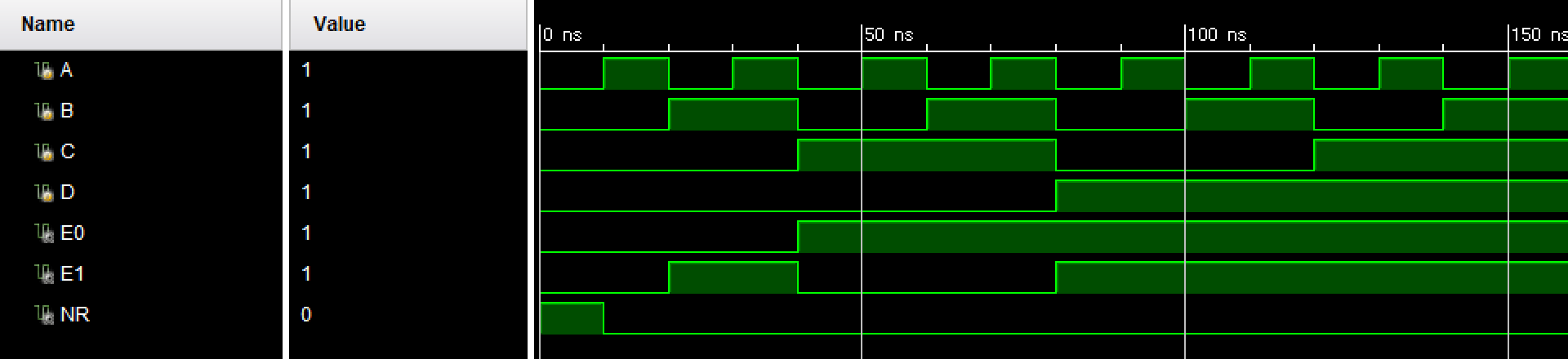
NR은 A,B,C,D가 모두 0인 경우에만 1의 값으로 set되기 때문에 굳이 Karnaugh map을 그리지 않아도 NR = (~A)(~B)(~C)(~D) 의 형태로 논리식을 세울 수 있다.

3) Source Code

|  |  |
| --- | --- |
| Design Source | Test Bench |
|  |  |

4) Schematic

5) Simulation



6) 결과 및 과정

설계한 Priority Encoder는 D>C>B>A 순으로 Priority가 높다. Priority가 높다는 것은, 예를 들어 ABCD의 입력 값이 0101로 들어왔을 때, B와 D의 값이 모두 1로 setting되어 있으나 D의 priority가 B보다 높기 때문에 0101의 입력을 0001로 간주한다는 의미이다. 이러한 방식을 이용하면 입력 bit가 단 한개의 값만 1또는 0으로 setting된 형태가 아니더라도 Encoder의 입력으로 사용하여 출력 값을 반환 받을 수 있다.

**5. BCD to Decimal decoder 의 결과 및 Simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

1) Truth Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | | | Output | | | | | | | | | |
| A8 | A4 | A2 | A1 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 |
| 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X |
| 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 |
| 1 | 0 | 1 | 0 | X | X | X | X | X | X | X | X | X |
| 1 | 0 | 1 | 1 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 0 | 1 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 1 | 0 | X | X | X | X | X | X | X | X | X |
| 1 | 1 | 1 | 1 | X | X | X | X | X | X | X | X | X |

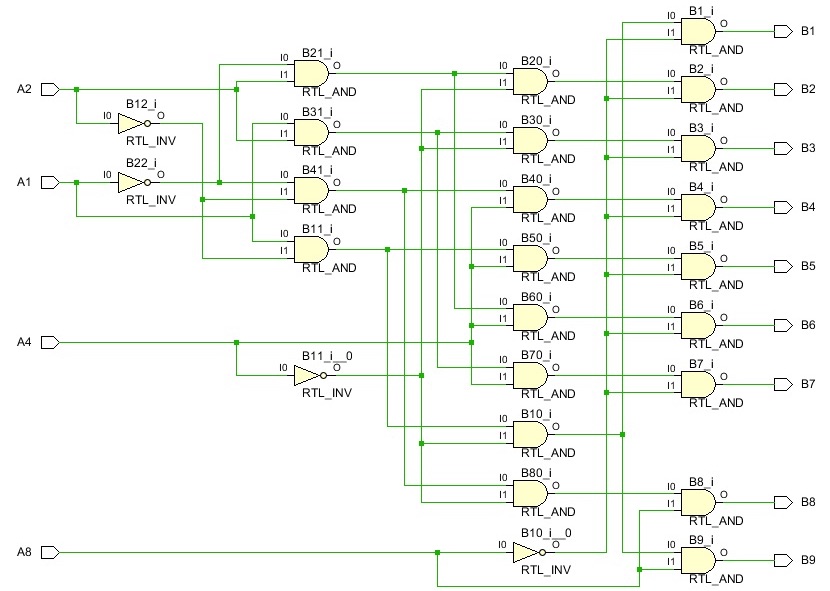
2) Karnaugh Map

|  |  |  |
| --- | --- | --- |
| B1 | B2 | B3 |
|  |  |  |
| B4 | B5 | B6 |
|  |  |  |
| B7 | B8 | B9 |
|  |  |  |

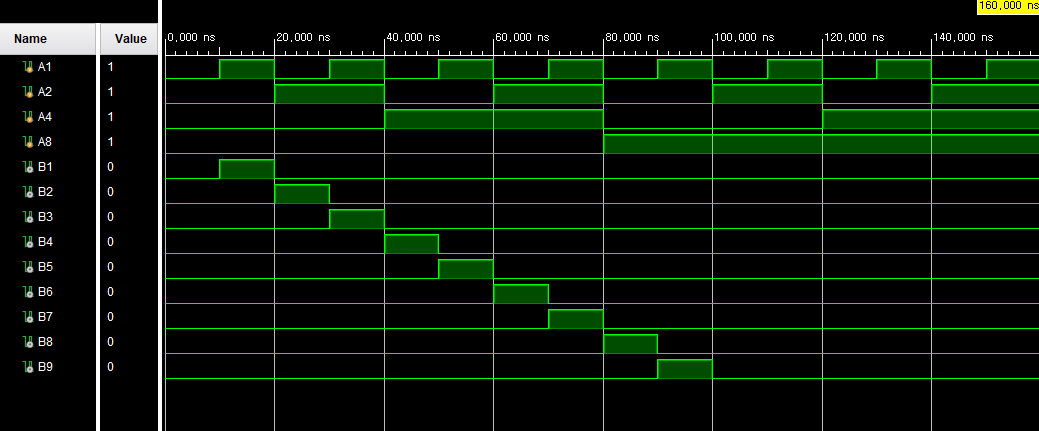
3) Source Code

|  |  |
| --- | --- |
| Design Source | Test Bench |
|  |  |

4) Schematic



5) Simulation



6) 결과 및 과정

실습에서 구현한 BCD to Decimal Decoder는 8421 BCD 코드를 입력 값으로 받아 그에 대응되는 출력 값을 반환하는 회로이다. 4bit의 입력 값은 각각 8,4,2,1의 값을 가지는데 예를 들어 0110의 값이 입력으로 들어온 경우, 4,2의 자리 bit 값이 set 된 것이므로 해당 입력 값은 6을 의미하게 된다. 출력값은 총 9bit로 이루어져 있으며 9개의 bit중 단 한 개만 1로 set되어 출력값으로 반환된다. 각 bit는 차례대로 1,2,3,4,5,6,7,8,9를 의미하는데, 예를 들어 0110이 입력 값으로 들어오면 이는 6의 값을 의미하는 것이므로, 000001000의 출력값을 반환하게 되는 것이다.

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

1) Encoder의 주요 응용



2) Decoder의 주요 응용



+) 9주차 발표자료 제작에 사용한 자료를 활용하였기 때문에, 저와 함께 자료조사 및 발표를 진행한 팀원과 해당 내용이 겹칠 수도 있습니다.

**7. 4 to 1 line MUX 의 결과 및 Simulation 과정에 대해서 설명하시오. (code, Truth table 작성)**

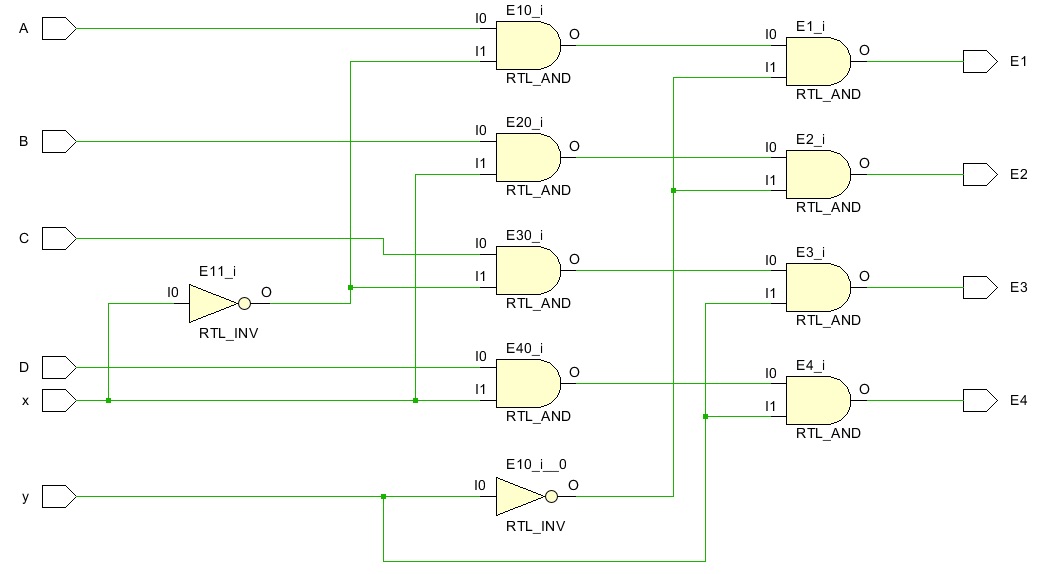
1) Truth Table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data | | | | Select | | Output | | | |
| A1 | B1 | C1 | D1 | InA | InB | A2 | B2 | C2 | D2 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

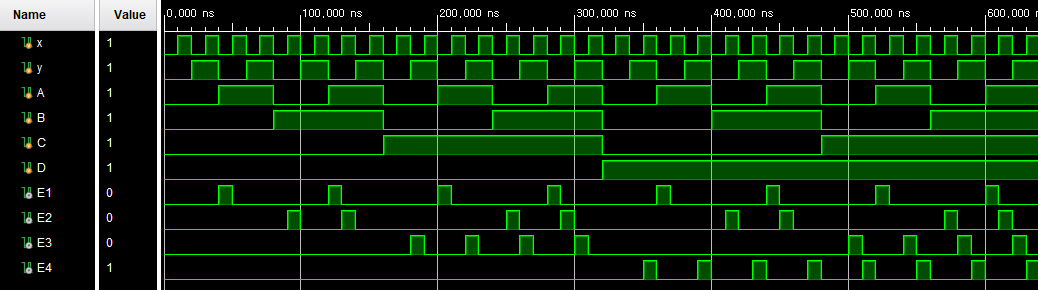
2) Source Code

|  |  |
| --- | --- |
| Design Source | Test Bench |
|  |  |

3) Schematic



4) Simulation



5) 결과 및 과정

실습에서 구현한 4 to 1 multiplexer(MUX)는 2bit의 select signal과 A,B,C,D 4개의 입력값을 가지며, 입력값이 1로 setting된 상태에서 해당 입력 신호에 올바르게 대응되는 select signal이 들어와야 해당 입력값이 출력값으로 선택된다. 예를 들어 ABCD의 입력값이 1000으로 들어왔다는 것은 A의 값이 입력으로 들어왔다는 의미이며, A에 대응되는 select signal은 00이므로 1000의 입력값과 00의 select signal 이 들어와야 출력값 역시 1000으로 반환된다. 입력값이 1000으로 들어왔는데 select signal이 00이 아닌 01이나 11등의 값으로 들어왔다면, 출력값은 1000으로 반환되지 않고 0000의 값을 반환할 것이다.

**8. 1 to 4 line deMUX 를 이용하여 4 to 16 decoder를 수행하고 결과를 나타내시오.**

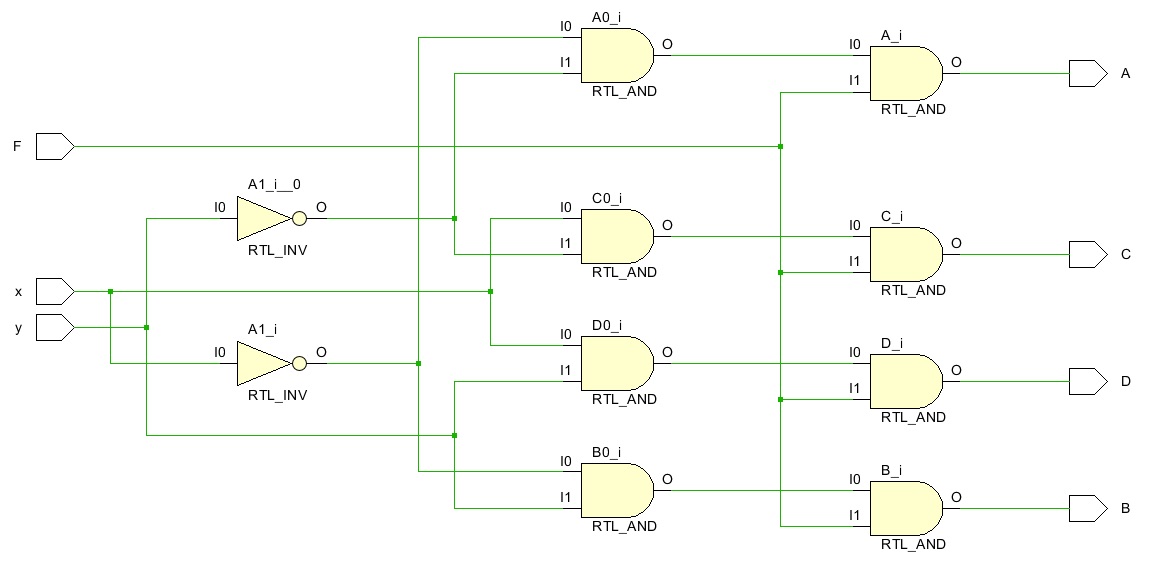
1) Truth Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Select | | Data | Output | | | |
| InA | InB | A | Out1 | Out2 | Out3 | Out4 |
| 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 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

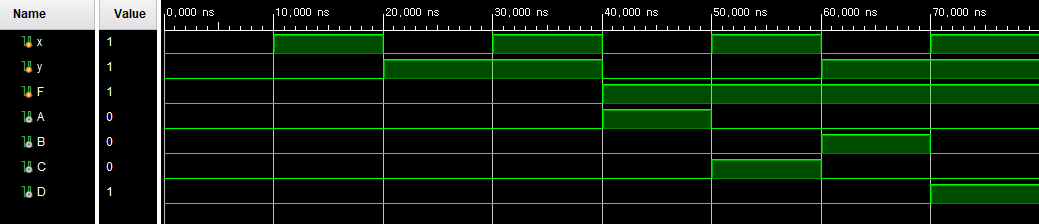
2) Source Code

|  |  |
| --- | --- |
| Design Source | Test Bench |
|  |  |

3) Schematic



4) Simulation



5) 결과 및 과정

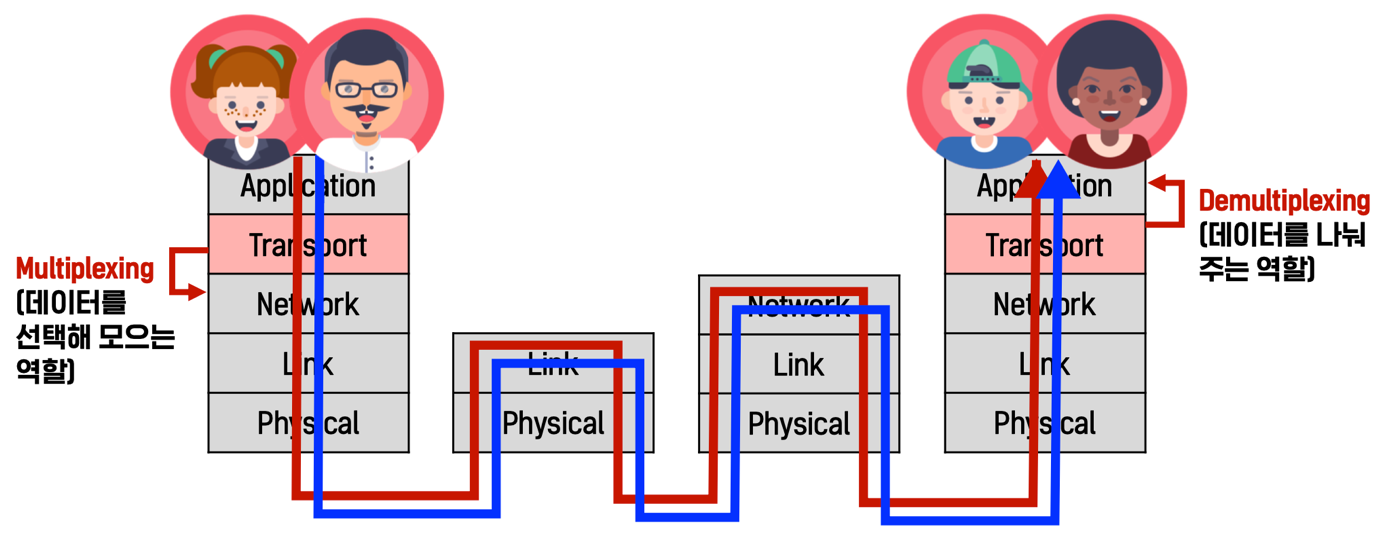
실습에서 구현한 1 to 4 demultiplexer의 경우 2bit의 select signal과 1bit의 data 값을 가지는데, select signal이 어떻게 들어오는지에 따라 4bit의 출력값에서 단 1개의 bit만 선택되어 1로 set되는데, data 값이 1로 set 되어 있을 때에만 이 동작이 수행되고 data값이 0으로 setting 되어 있을 때는 select signal값에 상관 없이 출력값은 0000으로 반환된다.

**9. 결과 검토 및 논의 사항.**

실습 시간에 진행하지 않은 부분을 결과보고서를 작성하면서 Verilog 코드를 새롭게 작성하여 Vivado에서 작동시킬 필요가 있었는데, Vivado 프로그램이 맥북 OS 환경에서는 지원되지 않아 가상머신을 설치하고 Vivado 프로그램을 설치하느라 애를 먹었다. 맥북 환경에서도 원활하게 가동되는 Vivado와 비슷한 프로그램을 찾아봐야 할 필요성을 이번 결과보고서를 작성하면서 느끼게 되었다.

**10. 추가 이론 조사 및 작성.**

1) 데이터 통신 분야에서의 MUX/deMUX의 활용



데이터 통신 분야에서의 multiplexing, demultiplexing이라 함은 Network Protocol Layers 중 Transport Layer 에서 이루어지는 작업을 의미하며 multiplexing은 application layer 의 소켓으로 부터 정보를 모아 만든 Header 를 데이터에 추가해 생성한 Segment 를

Network Layer 로 전달하는 작업을, demultiplexing은 transport layer segment의 데이터를 올바른 소켓으로 전달하는 작업을 의미한다.

+) 9주차 발표자료 제작에 사용한 자료를 활용하였기 때문에, 저와 함께 자료조사 및 발표를 진행한 팀원과 해당 내용이 겹칠 수도 있습니다.