디지털 논리2 및

컴퓨터 공학 기초 실험2   
보고서

과제제목: Register file, Counter & Shifter

실험일자: 2017년 09월 29일 (금)

제출일자: 2017년 10월 11일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금 5, 6, 7

학 번: 2012722028

성 명: 장한별

1. 제목 및 목적
   1. 제목

Register file, Counter & Shifter

* 1. 목적

1. **Register File**

Register에 대해 보다 정확히 이해하고, 지난 실습 시간에 구현한 32bits register를 8개를 instance한 register file을 구현한다. 이때, 구현하는 register file은 각 register에 address를 할당하여 해당 주소값의 정보를 읽는 read와 주소값에 정보를 쓰는 write의 기능을 수행하는 register file을 구현한다.

1. **Counter & Shifter**

FSM을 보다 정확히 이해하고, Moore FSM과 Mealy FSM의 개념과 특징과 차이점, 장단점 등에 대해 이해하고, 공부한다. 이를 통해서 counter의 개념을 이해하고, FSM을 이용하여 5-way Counter를 설계하고 구현한다. 또한 flip-flop과 combinational logic을 이용하여 sequential logic인 shifter를 설계하고 구현할 수 있도록 한다.

1. 원리(배경지식)
2. **queue**

|  |
| --- |
| <queue> |
|  |
| 위 그림은 queue 자료구조를 그림으로 나타낸 그림이다. Queue의 특징은 FIFO(First In First Out) 즉, 먼저 들어온 데이터가 먼저 나가는 구조이다. 따라서 1,2,3,4,5 이렇게 데이터가 들어왔으면 빠져나갈 때 1,2,3,4,5의 순서대로 빠져 나가게 된다. |

1. **stack**

|  |
| --- |
| <stack> |
|  |
| 위 그림은 stack 자료구조를 그림으로 나타낸 그림이다. Stack의 특징은 FILO(First in Last Out) 즉, 먼저 들어온 데이터가 가장 마지막으로 나가게 되는 구조이다. 따라서 1,2,3,4,5의 순서대로 데이터가 들어왔다면 나가는 순서는 5,4,3,2,1의 순서로 데이터가 나가게 된다. |

1. **Register file**

|  |
| --- |
| <Register file> |
|  |
| 위 그림은 registerfile의 structural specification이다. Write logic은 wAddr을받아 8개의 register중 한 개를 선택하고 we값이 1로 set되면 그 선택한 register를 사용하게 된다.  또한 8 32-bit registers는 write logic에의해 선택된 data를 저장하게 되고, read logic은 rAddr를 받아 register를 선택하여 그 register안에 저장되어있는 값을 출력하게되는 구조이다. |

1. **Moore FSM 과 Mealy FSM**

|  |
| --- |
| <Moore FSM과 Mealy FSM의 장단점> |
|  |
| <Moore FSM> |
|  |
| <Mealy FSM> |
| 위의 첫 번째 그림은 Moore FSM의 circuit이고, 위의 두 번째 그림은 Mealy FSM의 circuit이다. Moore FSM은 출력이 오직 현재상태에 의해서 결정되는 FSM이고, Mealy FSM은 출력이 현재상태 뿐만 아니라, 입력의 영향도 받는 circuit이다. Moore FSM의 경우 현재 상태에 의해서 output이 결정되기 때문에 state diagram을 이해 하고 , 구성하기가 비교적 쉽다. 반면에 Mealy FSM의 경우 output이 현재 상태뿐만이 아니라 Input값의 영향도 받기 때문에 직관적으로 바로 이해하고 구성하는 데에는 조금 어려움이 있을수 있다.  하지만 같은 동작을 할 때 Moore FSM의경우 Mealy FSM보다 더 많은 state를 사용 해야 하므로 bit수가 좀더 많아진다는 단점이 있다. |

1. **Ring Counter**

|  |
| --- |
| <ring counter> |
|  |
| 위 그림은 4-bits ring counter의 circuit이다. 링 카운터는 하나의 플립플롭에만 1이 들어가고 나머지는 0인 상태이다. 이때 이 1이 그다음 플립플롭으로 들어가게되면 1을 받은 플립플롭만 1인상태가 되고 나머지는 모두 0인상태가 되는 회로이다. 따라서 n자리의 링카운터는 n자리의 상태를 counter할수 있다. 마지막 플립플롭의 출력이 첫 플립플롭의 입력으로 다시 들어가는 고리모양의 카운터를 ring counter라고 한다. |

1. 설계 세부사항
2. Register File
3. Write operation

<one-hot-encoding write operation>

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs( wAddr ) | | | Outputs( q ) | | | | | | | |
| [2] | [1] | [0] | [7] | [6] | [5] | [4] | [3] | [2] | [1] | [0] |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |
| --- |
| <Write operation> |
|  |
| 처음 그림은 write operation을 one-hot encoding 방법으로 decoding을 한 table이고, 위 그림은 write operation의 circuit 이다.3-to-8 decoder는 wAddr의 값을 받아 8개의 register중 하나를 선택하고 이 선택한 register의 decoder의 출력와 en 신호의 and gate로 묶어 내보낸다. 위와 같이 one hot encoding을 한 이유는 8개의 bit중 선택한 register의 비트값만 1로 set하고 나머지는 0을 만듬으로써 en 신호와 and하기 떄문에 나머지 레지스터에는 0이 들어가고 하나의 register의 값만 저장할 수 있게 하기 위함이다. |

1. Read operation

|  |
| --- |
| <Read operation> |
|  |
| 위 그림은 Read operation 의 circuit이다. 먼저 Input 값으로 write operation과 register를 거친 값이 read operation의 input으로 들어가게된다. 그에따라 8개의 32bit register 가 input으로 들어가게되고 rAddr의 신호에 따라서 8-to-1 MUX에서 선택하여 하나의 Register 값만을 rData, 즉 output으로 내보내게 된다. |

1. Register file

|  |
| --- |
| <Register file> |
|  |
| 위 그림은 register file의 circuit이다. input으로는 wAddr, wData, we, clk, rAddr, reset까지 총 6개의 input이 들어오게 되고, rData라는 1개의 output값을 가지게 된다. 처음 input이 들어오면 write logic에 의해서 어떤 register주서에 값을 입력할지 결정하게 되고 그에 따라서 결정된 register에 wData값을 저장하게 된다. 이후 8개의 register중 rAddr의 값에 따라서 어떤 주소값에서 즉 어떤 register값을 read할지 read logic에서 결정하여 해당 값을 rData로 출력하게되는 구조이다. |

i) I/O Description

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | clk | 1-bit | Clock |
| reset\_n | 1-bit | Reset |
| we | 1-bit | Write enable |
| wAddr | 32-bits | Write address |
| rAddr | 32-bits | Read address |
| wData | 32-bits | Write data |
| Output | rData | 32-bits | Read data |
| Wire | to\_reg | 8-bits | Connect data |
| from\_reg | 32-bits | Connect data |

ii)Module Description

|  |  |  |
| --- | --- | --- |
| Classification | Name | Description |
| Module | Register\_file | Register file |
| Instance | U0\_write\_operation | Write operation |
| U1\_register32\_8 | 32bits register 8 |
| U2\_read\_operation | Read operation |

1. Counter & Shifter
2. cnt5

|  |  |  |
| --- | --- | --- |
| <state diagram> | Encoding states( Binary encoding) | |
|  | zero | 3’b000 |
| One | 3’b001 |
| Two | 3’b010 |
| three | 3’b011 |
| four | 3’b100 |

|  |  |  |  |
| --- | --- | --- | --- |
| <I/O Description> | | | |
| Port | Name | Bandwidth | Description |
| Input | clk | 1-bit | clk |
| reset\_n | 1-bit | Reset |
| inc | 1-bit | Increase |
| Output | cnt | 3-bit | counter |

cnt5를 설계하기 위하여 먼저 state diagram을 그리고, 그 이후 input과 output을 define하여 설계하였다. inc값이 1로 들어오면 다음 state로 0이 들어오면 그 전 state로 돌아가는 구조로 설계하였고 reset이 0이 들어오면 모든 값이 0으로 초기화 되는 구조이다. 각 state 를 encoding하는 방법으로는binary encoding을 사용하였다. 또한 input과 output은 위 표대로 구성하였다.

1. shifter8

|  |
| --- |
| image6.jpeg |

<state diagram>

1. Module configuration

|  |  |  |
| --- | --- | --- |
| 구분 | name | description |
| Top module | Shifter8 | 8-bits shifter |
| Sub module | mx4 | 1-bit 4-to-1 mx |
| Sub module | LSL8 | 8-bits logic shift left module |
| Sub module | LSR8 | 8-bits logic shift right module |
| Sub module | ASR8 | 8-bits logic shift right module |

1. I/O Description

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | clk | 1-bit | Clock |
| reset\_n | 1-bit | Reset |
| op | 3-bits | Opcode |
| shamt | 2-bits | Shift amount |
| d\_in | 8-bits | Input data |
| Output | d\_out | 8-bits | Output data |
| wire | do\_next | 8-bits | Connect data |

1. Define Parameter(Binary Encoding)

|  |  |
| --- | --- |
| NOP | 3’b000 |
| LOAD | 3’b001 |
| LSL | 3’b010 |
| LSR | 3’b011 |
| ASR | 3’b100 |

Shifter8을 설계하기 위해서 먼저 top module과 submodule을 나누어 instance하는 설계를 먼저 하였고, 그다음 각 모듈에 필요한 input과 output을 정리한 뒤 필요한 값에 따라 설계하였다. shifter8은 combinational circuit logic에서 다음 state를 계산하여 register에 넣어준다. 이때 필요한 opcode와 shamt값을 받아 요청한만큼의 shift를 하게 된다. 이때 clk 값도 넣어주게 되는데 clk이 rising edge 일 때 작동하도록 설계하였다.

1. 설계 검증 및 실험 결과
2. Register File
   1. 시뮬레이션 결과

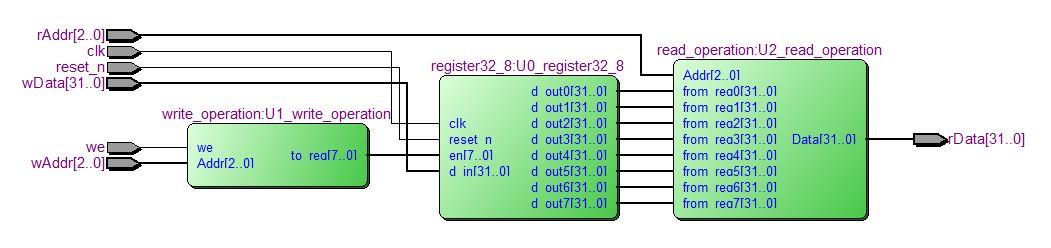


**<Register file 의 WaveForm>**

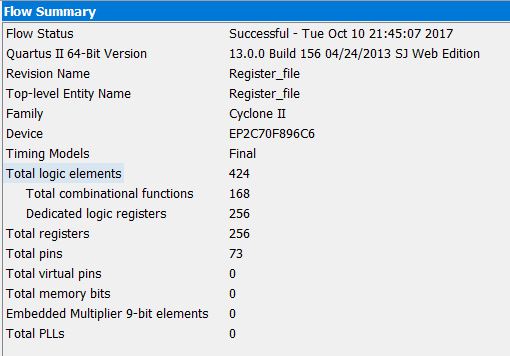
위 그림은 Register file의 waveform 이다. wAddr의 레지스터를 불러들여서 해당하는 값들을 저장하고 rAddr의 값 즉 저장했던 주소값들을 다시 불러들여서 저장했던값들이 정확히 저장이 되었음을 확인 할 수 있다. 001에는 ff00ff00를 저장하였고 011에는 0000ffff를 저장하였다. 이후 001과 011을 불러들였을 때 ff00ff00과 0000ffff가 출력됨을 확인할 수 있다.

주소값을 불러들였을때를 확인하기 위해 010, 10진수로 2 인부분을 확인하면 어떠한 값도 저장되어 있지 않음을 확인할 수 있고, 그 이후 어떠한 값도 출력되지 않음을 확인할 수 있다.

* 1. 합성(synthesis) 결과

**<Register File 의 RTL MAP Viewer>**

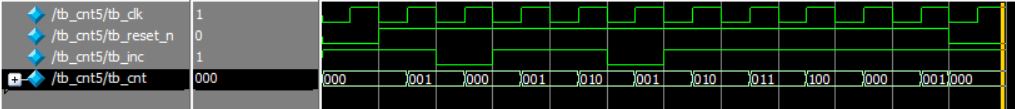
위 그림은 Register\_file의 RTL Viewer이다. Write\_operation, register32\_8, read\_operation 을 instance하여 설계하였음을 확인할 수 있고, input으로는 clk,reset\_n, wData는 register로 들어가고, we와 wAddr은 write\_operation에, rAddr은 read\_operation에 들어감을 확인할 수 있다. Output은 rData이다..

****

**<Register File 의 Flow Summary>**

위 그림은 Register File 의 Flow summary 이다. Total logic elements 는 424, Total pins 는 73임을 확인할 수 있다.

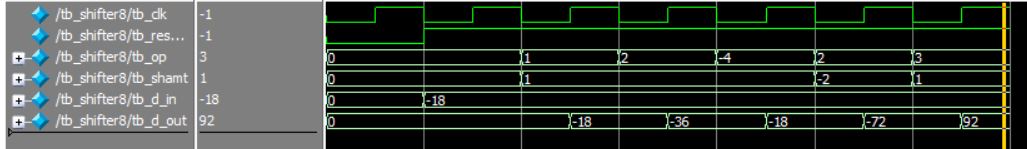
1. Counter & Shifter
   1. 시뮬레이션 결과
      1. cnt5



**<cnt5 의 Wave Form>**

위 그림은 cnt5의 wave form 이다. Clk이 rising edge일 때 tb\_inc의 값이 1일때를 보면 tb\_cnt즉 cout값이 000에서 001 010 011 100순으로 state로 생각하자면 0에서 4까지 변하는 것을 확인할 수 있다.

* + 1. shifter 8

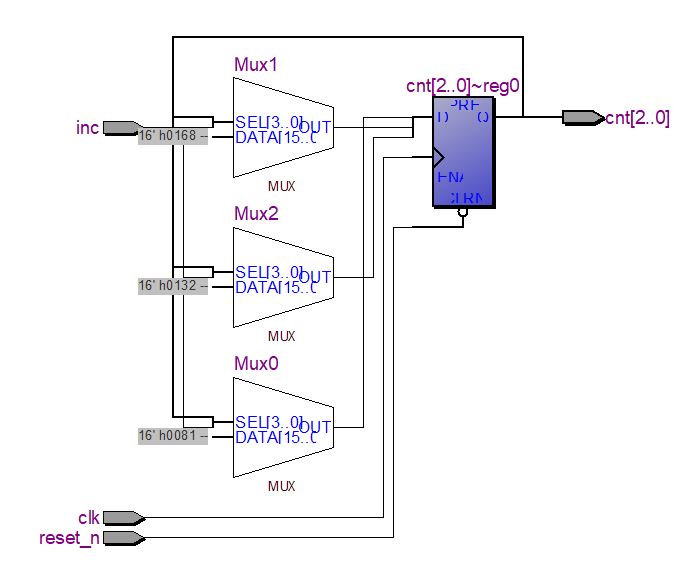


**<shifter 8 의 WaveForm>**

위 그림은 shifter8의 waveform 이다. Reset이 1로 set된다음 clk이 rising edge일 때 작동하도록 설계하였고, opcode를 정하여 shift할 방향을 정하고 shamt의 정도 크기 만큼 shift를 하게끔 설계하였다. 위 결과 화면에 보이듯이 d\_in이 들어오고 clk이 rising edge일 때 opcode와 shamt에 따라서 output인 d\_out이 결정되고 있다.

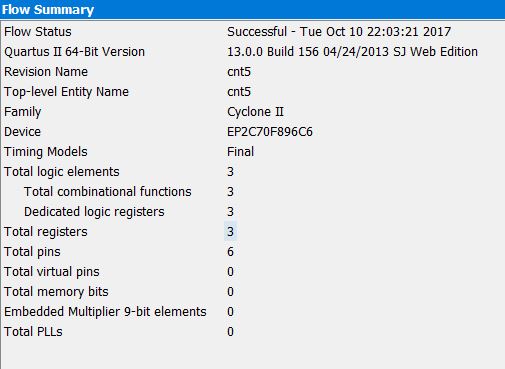
ASR 과 LSR 의 차이점을 비교해보면, 둘다 개념은 나누기의 개념이지만 LSR은 shift amount만큼 오른쪽으로 shift시킨후 빈 공간을 0으로 채우지만 ASR은 shift amount만큼 오른쪽으로 shift시킨후 빈 공간을 이전의 MSB로 채운다. 위의 결과화면에서도 보여지듯이 처음 11101110(2) 였던 수를 010(LSL)하여 11011100(2)를 얻었다. 이때 011(LSR)을 하게 되면 01101110(2)이 나오겠지만 LSR을하여서 이전의 MSB로 채워져 11101110(2)인 그전 값이 나온 것을 확인할 수 있다.

* 1. 합성(synthesis) 결과
     1. cnt5



**<cnt5 의 RTL MAP Viewer>**

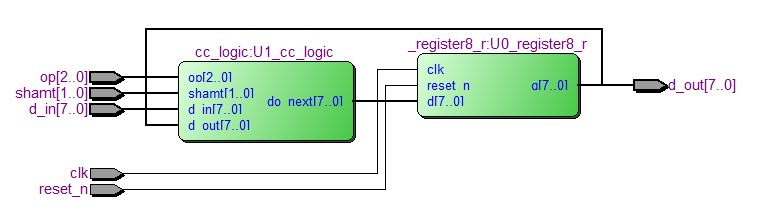
위 그림은 cnt5의 RTL Viewer이다. inc값이 mux로 들어가고 있고 그에따른 상태를 결정하여 register로 들어가고 있따. 이때 clk과 reset\_n도 같이 register에 들어가고 있다. 이때 나오는 결과값은 다시 input으로 들어가 다음 상태를 결정하는데 영향을 미친다.



**<cnt5 의 Flow Summary>**

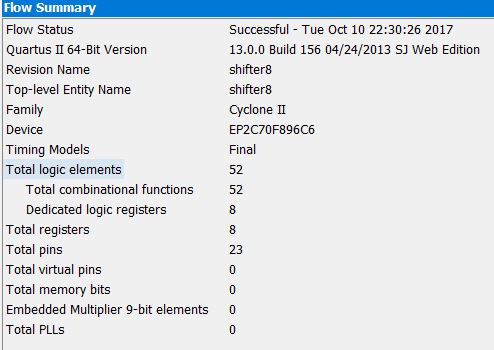
위 그림은 cnt5 의 Flow Summary 이다. Total logic elements 는 3, Total registers 는 3, Total pins 는 6 임을 확인할 수 있다.

* + 1. Shifter 8



**<Shifter 8 의 RTL MAP Viewer>**

위 그림은 shifter8의 RTL Viewer이다. cc\_logic과 \_register8\_r이 instance되어 설계되었음을 확인할 수 있고, input으로는 opcode, shift amount ,d\_in, clock, reset이 들어가고 있음을 확인할 수 있다. 이때 output으로 나오는 d\_out은 cc\_logic의 input으로 다시 들어가고 있음을 확인할 수 있다.



<Shifter 8 의 Flow Summary>

위 그림은 Shifter 8 의 Flow summary 이다. Total logic elements 는 52, Total registers 는 8, Total pins 는 23임을 확인할 수 있다.

1. 고찰 및 결론
   1. 고찰

이번 실습시간에는 register file, Counter 와 Shifter를 설계하고 구현하였다. Register file 이란 register들을 이용하여 데이터들을 주소값에 저장하고 그 데이터를 필요할 때 주소값을 불러들여서 해당 값을 출력하는 구조이다. 처음 개념을 들을 때는 내용자체가 어려운 부분이 아닌 것 같았지만 구현을 하는데는 조금 어려움이 있었다. 첫째 주소값을 register에 값에 저장한다는 내용을 직관적으로 처음에 이해할 수 없엇지만 코드를 짜면서 이해할 수 있게 되었다. 또한 3-to-8 decoder 의 의미였다. 굳이 3비트로 받은 값들을 8비트로 바꾸어 내보내는지 처음에는 잘 이해가 되지않았지만 one-hot encoding의 엄청난 장점을 깨닫게 되었다.

선택한 하나의 값만 1로 만들고 나머지를 다 0으로 만들어서 비교하는 비트를 최소화 하는 것 이었다. 만약 binary encoding으로 했다면 각 비트마다 다 비교를 해야됐기 때문에 회로가 조금 복잡해 질 것 같다. 또한 output과 output reg의 차이점과 유용함도 배웠고, wire을 선언할 때 2차원 배열식으로 32비트의 wire을 한번에 8개를 선언하는 방법에 대해서도 배울 수 있는 실습이었다. 이런 file의 개념이 n-bit의 register들의 덩어리라고 조교님께서 수업시간에 얘기해주었다. File의 개념을 Verilog에 접목시켜서 보다 정확히 이해할 수 있게 되었고, 이런 address의 개념을 익힘으로써 앞으로 Verilog를 통해 구현할 수 있는 범위가 넓어진 것 같다.

cnt5 를 구현할 때는 크게 어려움이 없었지만 Shifter8부분중 본인은 ASR과 LSR의 차이가 잘 이해가 되지 않았고, test bench의 입력 값을 계속 다르게 주면서 waveform을 확인하면서 그 차이를 이해할 수 있게 되었다. ASR과 LSR모두 Shift right 즉 나누기를 하는 부분에서는 같지만 LSR은 shift right한 뒤 빈 공간을 0으로 채워 넣는 반면, ASR은 shift right한 뒤 빈 공간을 이전의 MSB로 채워 넣는 그런 차이가 있었다. 따라서 허용범위를 뛰어넘는 연산을 할 때는 LSR보다 ASR이 보다 효과적일것으로 생각이 든다.

* 1. 결론

실습에서 구현한 Counter 와 조사한 Ring counter 를 비교하자면, Load\_counter는 값을 저장해 가면서 사용하는 것이다. 즉, 중간에 값을 새로 저장할 수도 있지만, ring counter는 이전 값을 받아서 출력하기 때문에 값을 중간에 새로 저장하는 것이 불가능하다. 따라서 ring counter는 값을 변화시킬 필요가 없이 반복, 순환적인 동작을 할 때에 사용하는 것이 좋다. Ring counter 는 load\_counter 보다 다른 input이 필요하지 않고 clock pulse에 따라 동작하기 때문에 시간이나 size를 더 줄일 수 있기 때문이다. ring counter를 사용하는 예로는 커피자판기가 있다. 커피자판기는 동전이 들어오기 전까지 대기 상태에 있다가 동전이 들어온 후 로 선택된 커피에 필요한 재료를 순차적으로 내보낸 후에 다시 초기상태로 돌아가는 것이 ring counter의 원리라고 할 수 있다.

Barrel Shifter - Barrel shifter는 한 개의 연산으로 데이터 공간 안에 있는 다수의 비트를 좌측이나 우측으로 shift 시킬 수 있는 shifter이다. 즉 데이터들을 한번의 clock cycle안에서 비트 이동을 시킬 수 있는 shifter이다. 데이터의 크기가 커질수록 barrel shifter를 구현 하는 데에 더 많은 multiflexer가 필요하다는 단점이 있지만 한번의 clock 주기의 시간 안에 데이터들의 비트 이동을 할 수 있으므로 시간 단축을 하는데 에는 뛰어난 효과가 있다. n bit의 길이를 가지는 register를 n-bit만큼 shift시키려면 필요한 multiplexer의 개수는 n개이고, 1bit씩 넘어가므로 필요한 bandwidth는 n-1이다.

1. 참고문헌

공진흥 / 컴퓨터공학기초실험2 / 새빛관303호(광운대학교) / 2017년.

이준환 / 디지털논리회로2 / 참빛관B101호(광운대학교) / 2017년.