컴퓨터 공학 기초 실험2 보고서

실험제목: Loadable Counter & Register file

실험일자: 2014년 10월 06일 (목)

제출일자: 2014년 10월 13일 (목)

학 과: 컴퓨터공학과

담당교수: 이혁준 교수님

실습분반: 목요일 0, 1, 2

학 번: 2013722095

성 명: 최재은

1. 제목 및 목적
   1. 제목

Shifter & Counter / Register file

* 1. 목적

Shifter와 counter를 직접 구현해 봄으로써 flip-flop과 combinational logic을 익힌다.

이를 통해 bit stream이 shift되는 원리를 이해하고 직접 구성해 봄으로써 이를 익힌다.

또한 이를 구현하기 위해서 counter라는 개념을 공부해보고 이해하여 구성된 모듈들이

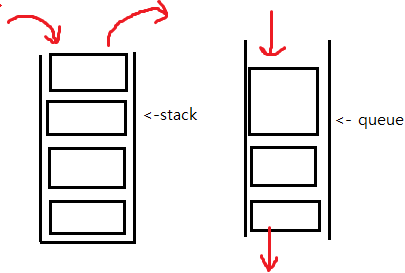
Sequential 하게 작동하는지를 확인하여 본다

1. 원리(배경지식)

5 way Counter module은 다음 진행할 state를 계산하고 어떠한 입력 값에 의해 특정 state를 진행하게 해주는 모듈이다. 초기에 reset\_n값을 통해서 입력 값을 받아들일지 말지를 선택하며 reset\_n이 1이 되면 클럭에 따른 상태천이를 진행한다. 양방향성을 특징으로 갖고 있으며 inc 입력에 따라서 그 진행 방향을 바꿀 수 있다. 이를 응용하여 어떤 state에서도 모든 state로 진행할 수 있는 전방위성 Counter module을 구성한다. 입력 값은 shifter를 통해서 bit stream을 움직이는데 이때 사용된 shifter는 LSL, LSR, ASR이다.

LSL은 입력된 shift amount만큼 bit stream을 왼쪽으로 이동시키는데 이때 한 칸 이동될 때마다 그 값들은 2배가 된다. LSR은 bit stream을 오른쪽으로 이동시키며 한 칸 이동할 때마다 그 값들은 1/2배가 된다. ASR은 bit stream을 오른쪽으로 이동시키며, 이동될 때 MSB를 복사하여 다시금 MSB로 설정한다. 즉, MSB가 바뀌지 않는다. Register file은 입력 데이터를 레지스터에 저장하고 다시 그에 저장된 값을 읽어 들이는 module로 특정 입력변수의 값에 따라서 입력 값을 저장하거나 저장하지 않을지를 선택할 수 있으며 이를 지나온 데이터는 Register에 저장이 된다. wAddr라는 입력변수의 값에 따라서 데이터를 저장할 register를 선택하는데 이 register들은 32bit를 저장할 수 있도록 구현되어있다. 이에 flip-flop을 사용하였다. register선택에 있어서 wAddr은 decoder를 통해 8bit의 값으로 변환되며 이 변환된 값들은 register의 주소 값 역할을 한다. 또한 rAddr이라는 변수의 값을 통해서 읽어 들일 register의 주소를 찾아가 그에 저장된 값들을 읽어 들여 출력해줄 수 있다. 크게 3개의 모듈(write\_operation, register, read\_operation)으로 구분되어있다.

Mealy FSM은 현재 상태와 입력 값에 따라서 출력이 영향을 받고 Moore FSM은 현재상태에 의해서만 출력이 영향을 받는다는 점에서 차이가 있다. Moore FSM은 설계가 직관적이어서 이해하기 쉽지만, Mealy형은 설계가 직관적이지 않아 이해하기 어렵다. Moore FSM은 Mealy형 보다 더 안정적이지만 Mealy형보다 더 많은 state를 갖게 된다. Ring counter란 state의 구성이 마치 원처럼 구성되어있고 양방향으로 상태를 천이할 수 있는 counter이며 이를 통해 링 안을 순차 순환할 수 있다. Stack과 Queue는 자료구조의 형태로 stack은 무언가를 쌓는다는 의미를 갖는다. 순서대로 자료를 쌓아서 보관하고 사용하는데 데이터가 먼저 들어올수록 늦게 빠져나가게 된다. Queue는 먼저 들어온 데이터가 먼저 빠져나가는 FIFO의 형태를 띄고 있다. 아래와 같은 그림으로 설명 할 수 있다.



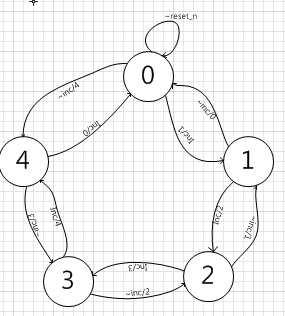
1. 설계 세부사항

1) 5-way Counter

- 양방향성을 특징으로 갖고 있으며, inc 에 따라서 진행 방향을 조정할 수 있습니다.

- reset\_n에 의해서 시작여부를 조정할 수 있습니다.

- 다음과 같이 원의 형태로 표현됩니다.



- 다음과 같은 동작을 합니다.

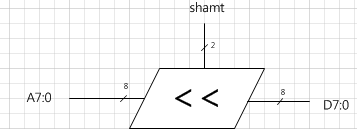
|  |  |  |
| --- | --- | --- |
| Reset\_n | Inc | Next |
| 0 | x | 0 |
| 1 | 0 | Prev\_state |
| 1 | Next\_state |

2) LSL8

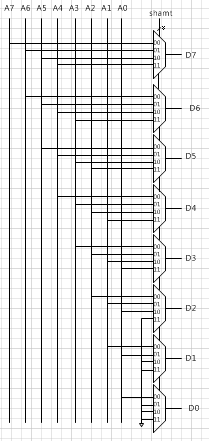
- bit stream 전체를 왼쪽으로 최대 3만큼 shift 시킨 후, 빈 공간은 0으로 채웁니다.

- 4bit mux 8개를 instance하여 값들을 선택합니다.

- 다음과 같이 간단하게 표현할 수 있습니다.



- 다음과 같은 형태로 구성됩니다.



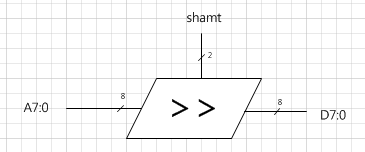
- 위에서 보는 바와 같이 빈 공간들이 0(GND)으로 채워집니다.

3) LSR8

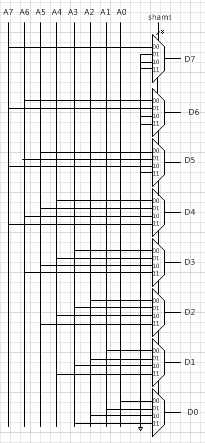
- bit stream 전체를 오른쪽으로 최대 3만큼 shift 시킨 후, 빈 공간은 0으로 채웁니다.

- 4bit mux 8개를 instance하여 값들을 선택합니다.

- 다음과 같이 간단하게 표현될 수 있습니다..



- 아래와 같은 형태로 구성됩니다.

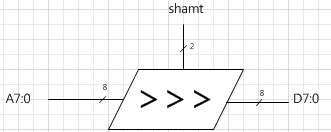


4) ASR8

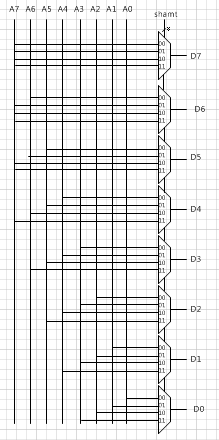
- bit stream 전체를 오른쪽으로 최대 3만큼 shift 시킨 후, 빈 공간은 이전의 MSB로 채웁니다.

- 4bit mux 8개를 instance하여 값들을 선택합니다.

- 다음과 같이 간단하게 표현될 수 있습니다..



- 다음과 같은 형태로 구성됩니다.

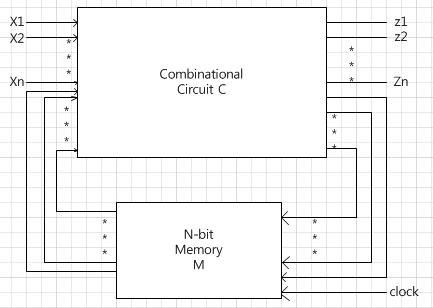


5) shifter8

- 위에서 설명한 LSL, LSR, ASR을 이용하여 register에 저장되어있는 bit를 이동시키는 module

- 8bit loadable shifter로써 다음과 같은 signal을 입력으로 받음

|  |  |
| --- | --- |
| Signal | Description |
| Reset\_n | Active low에 동작하는 reset signal. Register를 0으로 초기화함. |
| op | Shift를 시키기 위한 명령어로써 다음의 명령어를 가짐.  - NOP : No operation(현재의 값을 그대로 출력)  - Load : 입력된 data를 출력  - LSL : bit stream을 최대 3칸 만큼 왼쪽으로 이동  - LSR : bit stream을 최대 3칸 만큼 오른쪽으로 이동  - ASR : LSR과 같은 동작을 하되, 이전의 MSB를 복사하여 MSB로 저장 |
| shamt | Shift amount (2bit) |



- 위에서 보듯이 flip flop의 기능이 있는 순차회로로 표현된다.

- 그 출력은 현재의 입력 값과 플립 플롭에 저장되어있는 값에 의해 결정된다.

- 모듈의 구성은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 구분 | 이름 | 설명 |
| Top module | shifter8 | 8bit loadable shifter의 top module |
| Sub module | LSL8 | 8bit logical shift left module (top module에서 instance) |
| LSR8 | 8bit logical shift right module (top module에서 instance) |
| ASR8 | 8bit logical shift left module (top module에서 instance) |
| mx4 | 1bit 4-To-1 multiplexer(LSL8, LSR8, ASR8에서 각각 instance) |
| \_register\_8\_r | 입력 받은 8bit data값을 기억하는 register(reset\_n이 0이면  초기화 됨) |
| cc\_logic | 입력 받은 3bit opcode에 따라서 next state의 값을 결정 |

- 사용된 인자의 구성은 다음과 같습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module  이름 | 구분 | 이름 | 비트 수 | 설명 |
| shifter8 | Input | clk | 1 | Clock |
| reset\_n | 1 | Active low에 동작하는 reset 신호로 값이 인가되면 register의 값을 0으로 초기화 |
| op | 3 | operation으로 NOP, LOAD, LSL, LSR ASR 중 하나를 선택 |
| sharmt | 2 | Shift amount로, 2 bit이기 때문에 최대 3까지의 shift를 수행가능 |
| d\_in | 8 | op에서 LOAD가 인가되었을 때, d\_in을 통해 들어온 값을 register에 저장하기 위한 입력 |
| output | d\_out | 8 | Register의 값을 출력 |
| LSL8 | Input | d\_in | 8 | Data in |
| LSR8 | input | shamt | 2 | Shift amount |
| ASR8 | Output | d\_out | 8 | Data out |
| mx4 | input | d0 | 1 | Multiplexer의 1번째 data입력 |
| d1 | 1 | Multiplexer의 2번째 data입력 |
| d2 | 1 | Multiplexer의 3번째 data입력 |
| d3 | 1 | Multiplexer의 4번째 data입력 |
| s | 1 | Multiplexer의 선택 신호 |
| output | y | 1 | Multiplexer의 결과 |

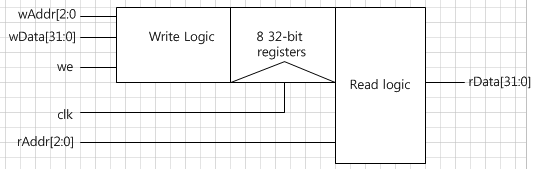
6) Register file

- 입력 받은 데이터값을 특정 주소를 갖는 레지스터에 저장하고, 그 주소 값을 통해서

저장된 데이터를 다시 불러 읽을 수 있는 module

- 크게 Write logic, register, Read logic으로 구분되며, 이 실험에서는 32bit를 저장할 수 있는 8개의 register를 구현하였습니다.

- 전체적인 구조는 아래의 그림과 같습니다.



- 사용된 모듈의 구성은 아래와 같습니다.

|  |  |  |
| --- | --- | --- |
| 구분 | 이름 | 설명 |
| Top module | Register\_file | Register file의 top module |
| Sub module | Register32\_8 | 8개의 32bit register module(top module에서 instance) |
| Write\_operation | Write address decode module(top module에서 instance) |
| Read\_operation | Select register using read address(top module에서 instance) |

- 사용된 인자들의 구성은 다음과 같습니다.

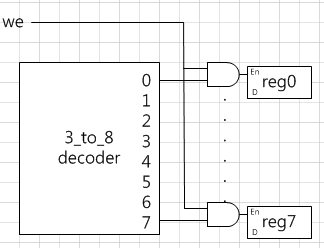
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module 이름 | 구분 | 이름 | bit | 설명 |
| Register\_file | input | Clk | 1 | Clock |
| Reset\_n | 1 | Active low에 동작하는 reset 신호로 값이 인가되면 register 값을 0으로 초기화 |
| We | 1 | Write enable |
| wAddr | 3 | Write address |
| rAddr | 3 | Read address |
| wData | 32 | Write data |
| output | rData | 32 | Read data |
| Register32\_8 | Input | Clk | 1 | clock |
| Reset\_n | 1 | Active low에 동작하는 reset 신호로 값이 인가되면 register 값을 0으로 초기화 |
| en | 1 | Register enable |
| D\_in | 32 | Data in |
| output | D\_out0~d\_out7 | 32 | Register data out(8개) |
| Write\_operation | input | we | 1 | Write enable |
| Addr | 3 | Write address |
| output | To\_reg | 8 | Selected register enable signal |
| Read\_opearation | input | From\_reg0~  from\_Reg7 | 32 | 8 registers’ data |
| Addr | 3 | Read address |
| output | Data | 32 | Data out |

(1) write\_operation

- 사용자로부터 wAddr을 받아서 8개의 register 중 한 개를 선택한 뒤 we이 1일 때

Register에 값을 씁니다.

- 다음과 같은 형태로 표현될 수 있습니다.



- 입력 받은 opcode를 decoder를 통해서 8bit 주소 값으로 변환합니다.

- we가 1일 때, and gate를 통해서 8개의 bit값을 각각 비교하여 to\_reg(output)을

출력합니다.

(2) register32\_8

- 32bit 기억소자 8개를 instance하여 구성하였기 때문에 8개의 출력 값을 따로 갖습니다.

- reset\_n이 0일 때는 값을 받아들이지 않습니다.

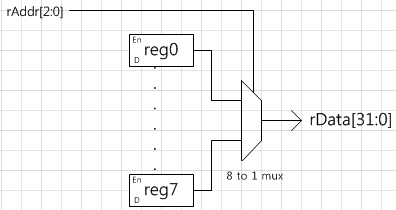
- 8bit en 변수를 통해서 어떤 레지스터에 저장할 지를 선택하도록 되어있습니다.

(3) read\_operation

- 8 to 1 mux를 사용하였고 8개 register에서의 출력 값들을 입력으로 받고 있습니다.

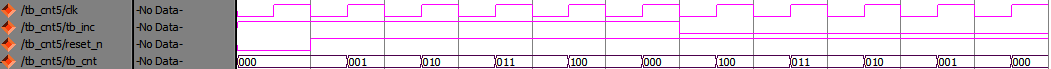
- rAddr값을 통해서 어떤 레지스터를 출력할지를 선택합니다.

- 아래의 그림과 같이 표현될 수 있습니다.



1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

1) 5-way counter



- 위와 같은 결과를 얻었으며 55ns에서 inc값을 0으로 하여 진행방향을 바꾸었습니다.

- 예상했던 것과 같은 결과를 확인하였습니다.

2) LSL8



- shamt를 3으로 주어 위와 같은 결과를 얻었으며 예상과 같은 결과를 확인하였습니다.

3) LSR8



- shamt를 3으로 주어 위와 같은 결과를 얻었으며, 예상과 같은 결과를 확인하였습니다.

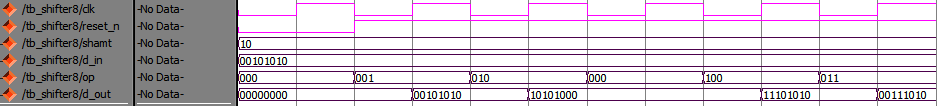
4) ASR8



- shamt를 3으로 주어 위와 같은 결과를 얻었으며, 이동간 빈 공간에는 이전의 MSB인

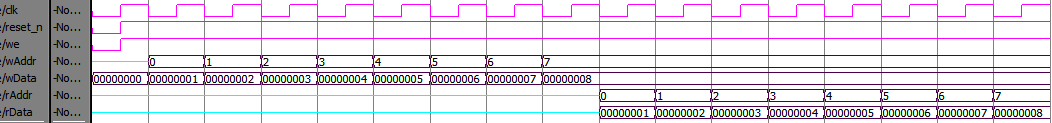
1이 들어가있음을 확인할 수 있습니다.

5) shifter8



- 위의 waveform을 확인하면 reset\_n이 1이 되고, 0opcode가 001일 때의 rising edge에서 값이 들어가고, 010일 때 LSL 2bit이 수행되었습니다. 이후 000으로 주어 상태 값을 유지하고 100일 때 ASR 2bit을 수행합니다. 011일 때 LSR 2bit이 수행되었음을 확인할 수 있습니다.

6) Register \_file

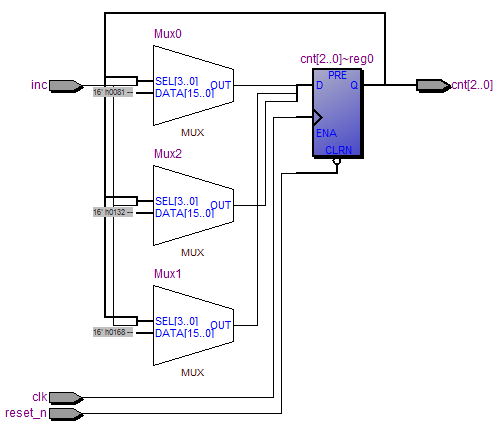


- 위와 같은 결과를 얻었으며 0~7까지 입력된 값이 90ns부터 출력되는 것을 확인 할 수 있었습니다. 결과는 예상과 동일하였습니다.

* 1. 합성(synthesis) 결과

1) 5-way counter

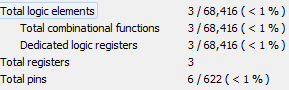
<RTL Viewer>



- reset\_n과 clk가 flip-flop에 연결되었음을 확인할 수 있고, 이를 통해서 next state를

기억해 두는 것을 확인 할 수 있습니다.

<Flow summary>

- 총 3개의 logic이 사용되었습니다.

- behavior 방식으로 구현하면서 3개의

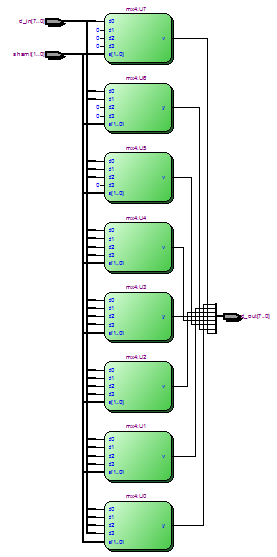
Register를 사용하였습니다.

- 1bit input clk, reset\_n, inc와 3bit output

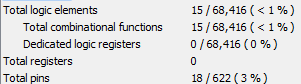
총 6개의 pin이 사용되었습니다.

2) LSL8

<RTL Viewer>



- 8개의 mux를 이용해 해당 shift를 진행하면서 빈 공간에는 0이 들어가는 것을 확인할 수 있었습니다.

<Flow summary>

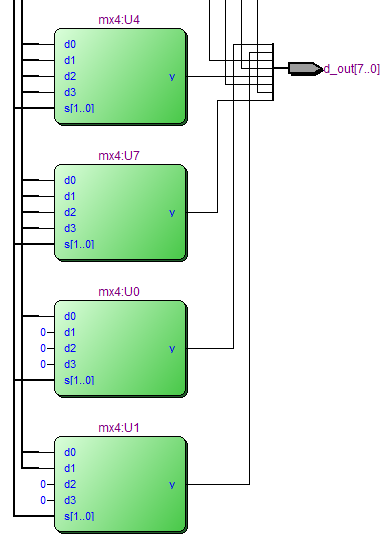
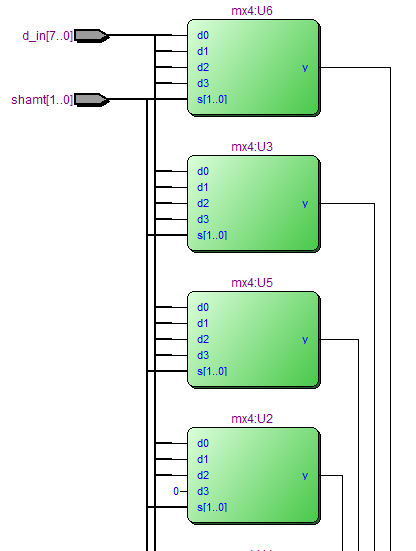
- 총 15개의 logic이 사용되었습니다.

- 2bit shamt와 8bit input d\_in, output d\_out

총 18개의 pin이 사용되었습니다.

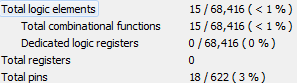
3) LSR8

<RTL Viewer>



- 위에서 확인할 수 있듯이 8개의 mux를 이용해 shift를 하고 빈 공간에 0이 들어감을 확인할 수 있었습니다.

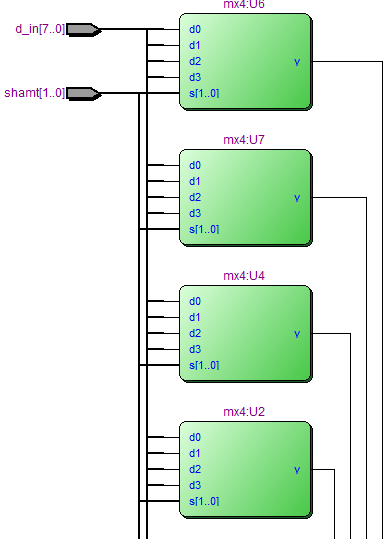
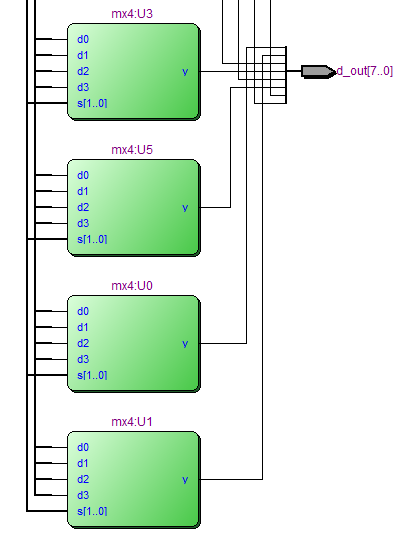
<Flow summary>

- 총 15개의 logic이 사용되었습니다.

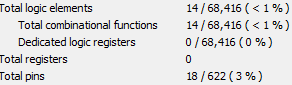
- 2bit shamt와 8bit input d\_in, output d\_out

총 18개의 pin이 사용되었습니다.

4) ASR8

<RTL Viewer>

- 위의 두 shifter와는 다르게 빈 공간에 0이 아닌 MSB가 들어감을 확인할 수 있습니다.

<Flow summary>

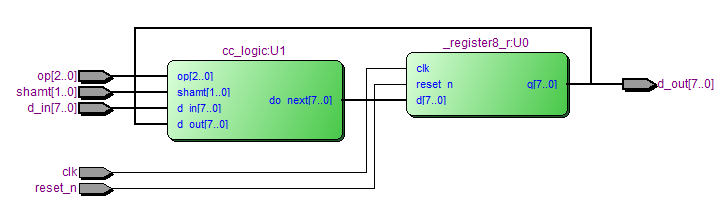
- 총 14개의 logic이 사용되었습니다,

- 2bit shamt와 8bit input d\_in, output d\_out

총 18개의 pin이 사용되었습니다.

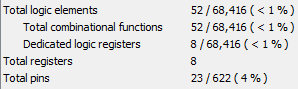
5) shifter8

<RTL Viewer>



- register에서의 output이 cc\_logic의 input 값으로 들어가는 것을 확인할 수 있습니다.

<Flow summary>

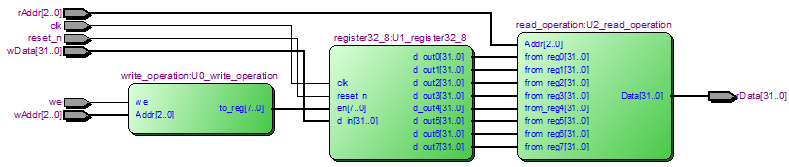
- 총 52개의 logic이 사용되었음을 확인할 수

있으며, input으로 3bit op, 2bit shamt, 8bit

D\_in, clk, reset\_n, output으로 8bit d\_out,

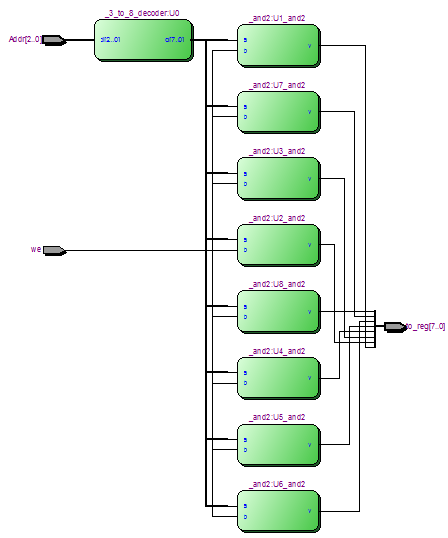
총 23개의 pin이 사용됨을 확인하였습니다.

6) Register\_file



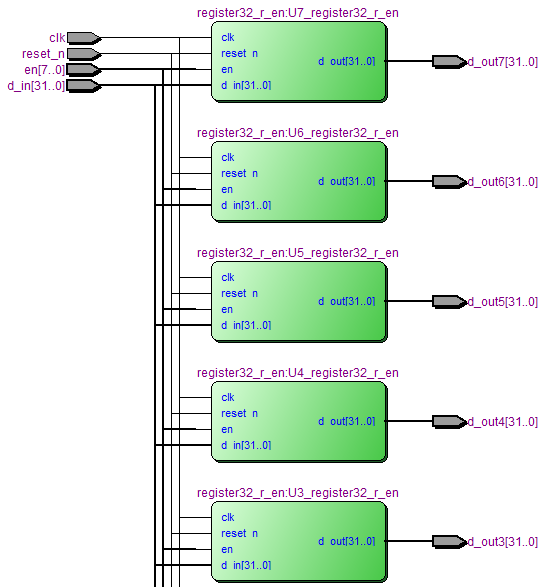
- 순차적으로 write\_operation module, register32\_8 module, read\_operation module에 값이 들어가는데 각각의 통제변수를 통해 값들이 입출력 됨을 확인할 수 있습니다.

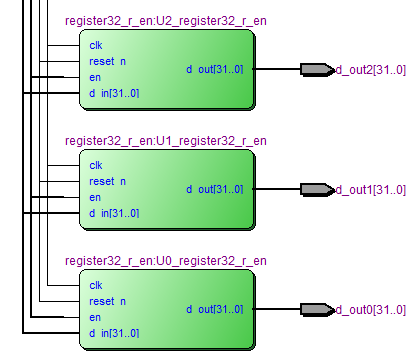
7) write\_operation



- 위와 같은 형태로 표현이 되는데 and2 gate를 사용하여 decoding한 주소 값을 출력합니다.

8) Register32\_8



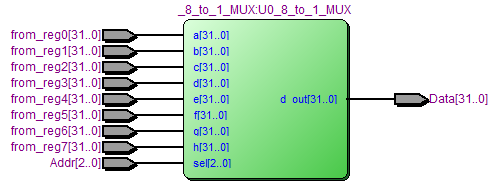


- 위에서 보는 바와 같이 32bit를 저장할 수 있는 register가 8개 있습니다.

- 입력값은 en 변수에 의해 입력받은 주소를 갖는 register에 저장이 되며, 이들은

각각의 출력변수를 통해 그 값들을 출력해주게 됩니다.

9) read\_operation



- register32\_8 module에서 나온 출력 값들이 모두 mux의 입력 값으로 들어가고, mux는 rAddr을 입력으로 받아서 출력해줄 data를 선택합니다.

1. 고찰 및 결론
   1. 고찰 및 결론

Barrel shifter란 한 개의 연산으로 데이터 워드 내에 있는 다수의 비트를 이동하거나 회전시킬 수 있는 하드웨어 장치입니다. 워드 크기가 커질수록 필요한 멀티플렉서가 더

많아지지만 1클럭 사이클에 비트 이동을 끝낼 수 있기 때문에 시간을 단축시켜 성능 향상에 크게 기여합니다. , n-bit를 가지는 register를 n bit만큼 shift하고자 할 때 4bit mux에 3개의 mux가 사용되고 n-bit shifter에서는 4bit mux가 n개 만큼 소요 됩니다.

그러므로 총 3n개의 멀티플렉서가 필요합니다. 이번 실험을 진행하면서 각 모듈이 instance하는 것들이 많아서 뭔가 머리가 복잡했었습니다. 전체적인 그림을 보면서 대강이나마 정리를 한 다음에서야 조금씩 진행을 나갔는데, 뭔가 퍼즐이 맞는 것처럼 어느 순간 연결이 되는 것이 신기하고 기분이 좋았던 기억이 납니다. Always를 쓰면서 조금 헷갈리는 부분이 있었는데 이번 실험을 진행하면서 해결하였습니다.

1. 참고문헌

Wikipedia