컴퓨터 공학 기초 실험2 보고서

실험제목: Shifter & Counter

실험일자: 2020년 10월 16일 (금)

제출일자: 2020년 10월 23일 (금)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 5,6,7

학 번: 2019202009

성 명: 서여지

1. 제목 및 목적

A. 제목

Shifter & Counter

B. 목적

비트를 이동시키는 shifter와 입력에 따라 수를 증가 혹은 감소시키는 counter를 flip-flop과 combinational logic 부분으로 분리하여 설계한다. shifter와 counter모두 register를 포함하기 때문에 회로 전체는 sequential logic이다. cnt5에 대한 적절한 testbench를 작성하고, 결과를 이해한다.

2. 원리(배경지식)

1) Moore FSM과 Mealy FSM의 장단점

Finite state machine은 회로의 output logic 부분에서 current state뿐만 아니라 FSM의 input신호가 함께 사용되는지 여부에 따라 Mealy FSM과 Moore FSM으로 구분할 수 있다. 이때 input 신호가 사용되지 않는 것이 Moore FSM이고, 사용되는 것이 Mealy FSM이다.

Mealy FSM은 같은 문제를 해결하는 Moore FSM과 비교했을 때, 일반적으로 더 적은 state를 갖는 형태로 구현할 수 있다. 이때 문제 해결에 사용되는 회로를 더 작게 설계하여 비용을 줄일 수 있는 장점이 있다. 그러나 input신호가 register를 거치지 않고 바로 output회로에 연결되기 때문에 input의 값의 변화에 따라 결과값이 흔들리는 glitching이 발생할 수 있다는 단점이 나타난다.

2) counter - ring counter

counter는 펄스 신호에 따라 순서대로 state가 변화하는 register이다. 특히 ring counter는 가장 처음의 state와 마지막 state가 이어져 순환하는 형태로 state가 변화하는 counter이다. data를 one-hot encoding으로 나타내면, 각각의 bit를 저장하는 register의 정보가 한 bit씩 shifting되어 순환하는 것처럼 보인다. 이것을 ring counter의 data가 rotation한다고 표현한다. 이번 실험에서 설계하는 ring counter는 data를 load하여 수를 증가시키거나, 감소시킬 수 있는 counter이다.

3) shifter

Shifter는 register의 정보를 좌우로 일정한 bit 만큼 이동시킬 수 있는 회로이다. 이번 실습에서 구현한 shifter는 세 가지 방법으로 data를 이동시킬 수 있다. Logical shift left는 DATA를 왼쪽으로 shamt만큼 이동시킨다. 오른쪽에 새로 생성되는 bit는 0

으로 채운다. 예를 들어 0011을 1bit만큼 LSL하면, 0110이 된다. Logical shift right는 LSL과 반대로 data를 오른쪽으로 이동시키며, 왼쪽에 새로 생성되는 bit를 0으로 채운다. 0011을 1만큼 LSR하면 0001이다. 마지막으로 arithmetic shift right는 LSR과 마찬가지로 data를 오른쪽으로 shifting하지만, 왼쪽에 새로 채워지는 bit가 0이 아닌 data의 signbit이다. 1000을 2bit ASR하면 1110이 된다.

3. 설계 세부사항

A. shifter8

shifter8 module은 LOAD, LSL, LSR, ASR 기능을 갖는다. 따라서 각각의 기능을 하는 state와 아무 명령도 입력되지 않았을 때의 NOP state까지 총 5개의 state를 이용하여 구현한다. clk, reset_n 입력과 명령어인 op, shift할 크기인 shamt, LOAD할 정보인 d_in을 input으로 받고, 결과인 d_out을 출력한다. 총 5개의 state를 갖으므로, 5종류 이상의 state를 나타낼 수 있는 3bit로 encoding한다. 각각의 state는 모두 op 입력에 따라 변경되고, LSL, LSR, ASR 동작은 shamt을 이용한 MUX로 연산한다. always문에서 입력값이나 d_lsl, d_lsr, d_asr의 값이 변경되었을 때 다음 state를 update한다.

각 state의 Encoding은 다음과 같고, op코드와 일치한다. 다음 state는 op코드에 의해서만 결정되므로 다음과 같이 정리할 수 있다.

state	Encoding
NOP	000
LOAD	001
LSL	010
LSR	011
ASR	100

current state	opcode	op	Next state
Xxx	NOP	000	000
Xxx	LOAD	001	001
Xxx	LSL	010	010
Xxx	LSR	011	011
xxx	ASR	100	100

B. cntr8

cntr8 module은 LOAD기능과 reset상태인 IDLE를 갖고, inc 입력값에 따라 수를 증가시키거나 감소시킨다. 수의 증가와 감소는 2개의 state를 이용하여 계산하므로 모두 6개의 state를 이용하여 counter를 구현한다. 6개 이상의 state를 나타낼 수 있는 3bit binary encoding을 사용한다. next state는 현재 state와 inc값에 의해 결정되고, output은 os_logic module에서 cla8 module을 instance하여 구한다.

state	Encoding
IDLE_STATE	000

current state	Reset_n	load	inc	Next state
xxx	0	x	x	000

LOAD_STATE	001
INC_STATE	010
INC2_STATE	011
DEC_STATE	100
DEC2_STATE	101

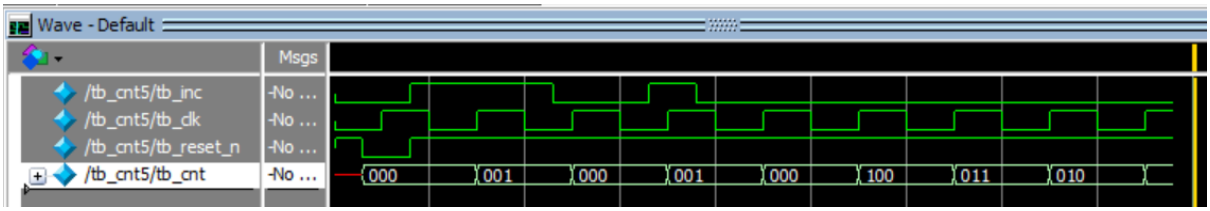
xxx	1	1	x	001
INC가 아닌 나머지 STATE	1	0	1	INC
010	1	0	1	INC2
DEC가 아닌 나머지 STATE	1	0	0	DEC
100	1	0	0	DEC2

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

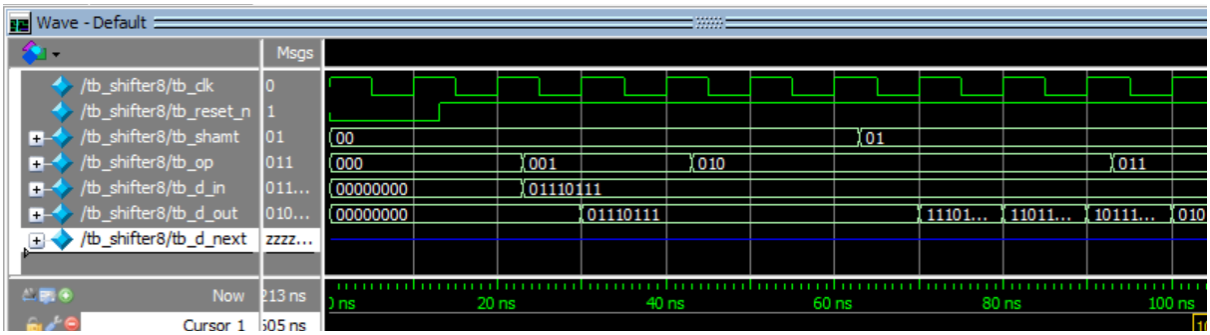
1) cnt5

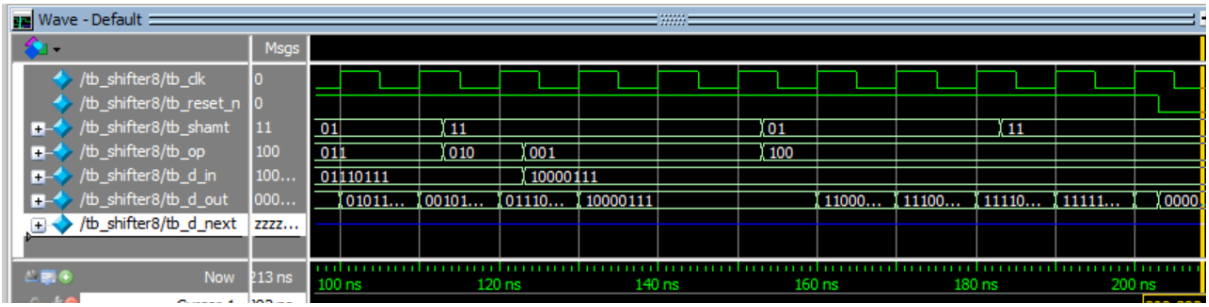
testbench를 작성하여 시뮬레이션 하였다. reset이 0일 때 0으로 리셋되었고, inc가 1인 동안 증가, 0인동안 감소하는 것을 확인할 수 있다. 000에서 감소하는 경우 100으로 연결되는 것도 확인가능하다.



2) shifter

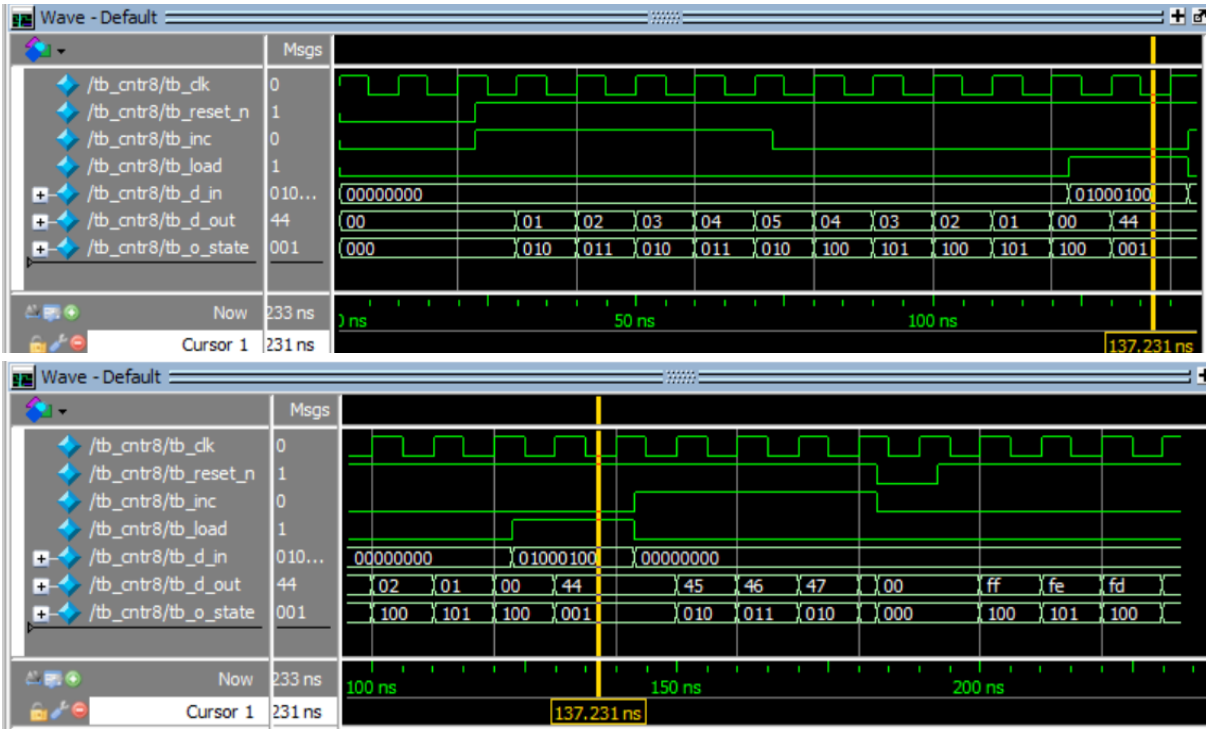
testbench는 실습자료에 제시된 case를 이용하여 작성하였다. 제시된 결과와 일치하는 것을 확인할 수 있다. clk의 rising edge에서 연산 결과가 update된다.





3) counter

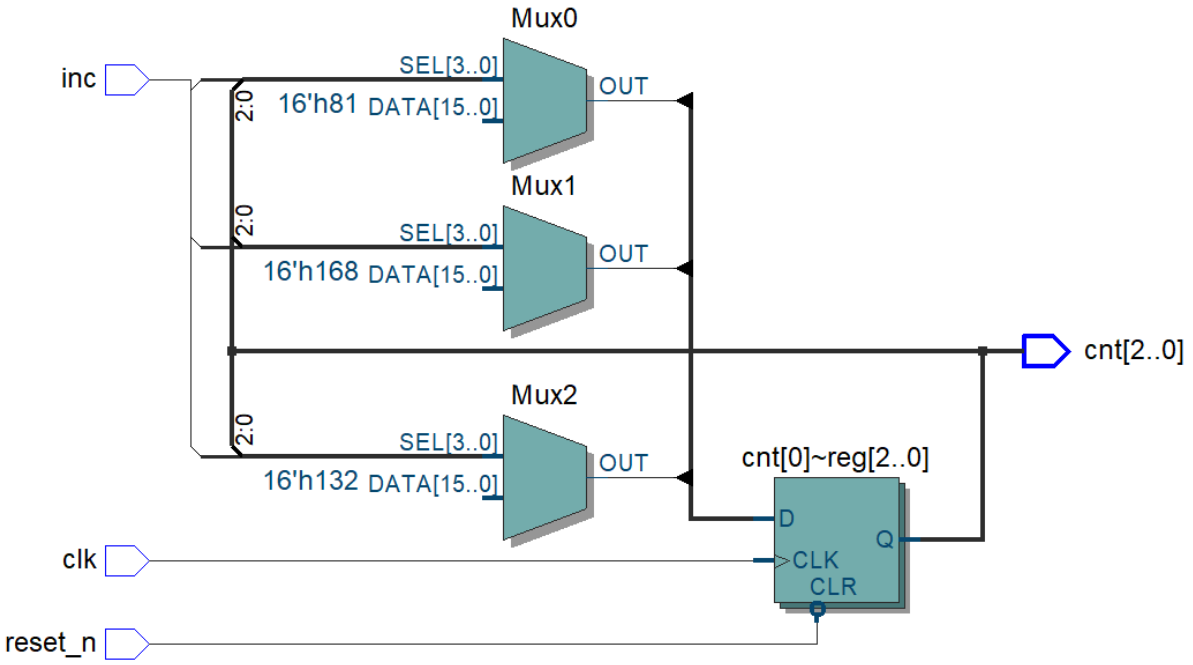
testbench는 실습자료에 제시된 case를 이용하여 작성하였다. 제시된 결과와 일치하는 것을 확인할 수 있다. clk의 rising edge에서 연산 결과가 update된다.



B. 합성(synthesis) 결과

1) cnt5

register에 저장된 결과가 다시 MUX의 input으로 입력되어 다음 계산을 실행하는 구조임을 확인할 수 있다.

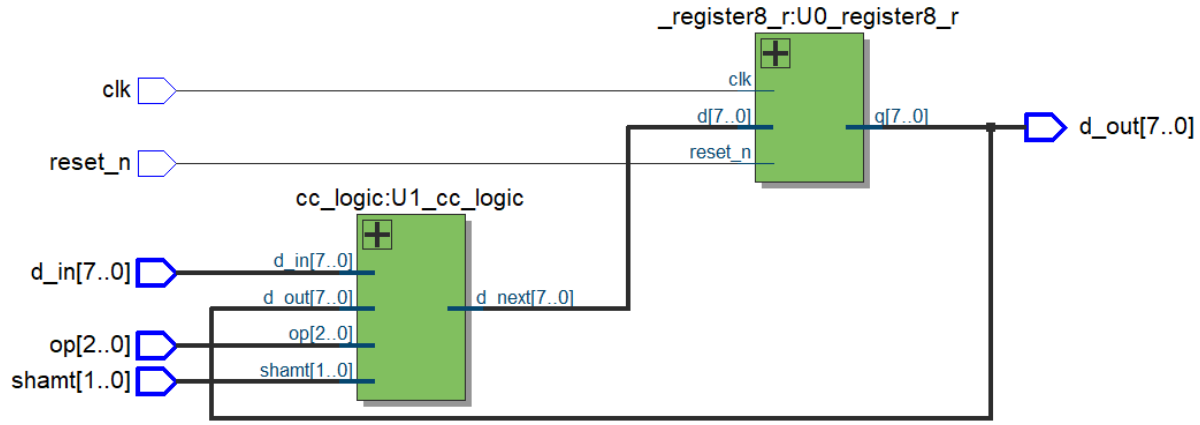


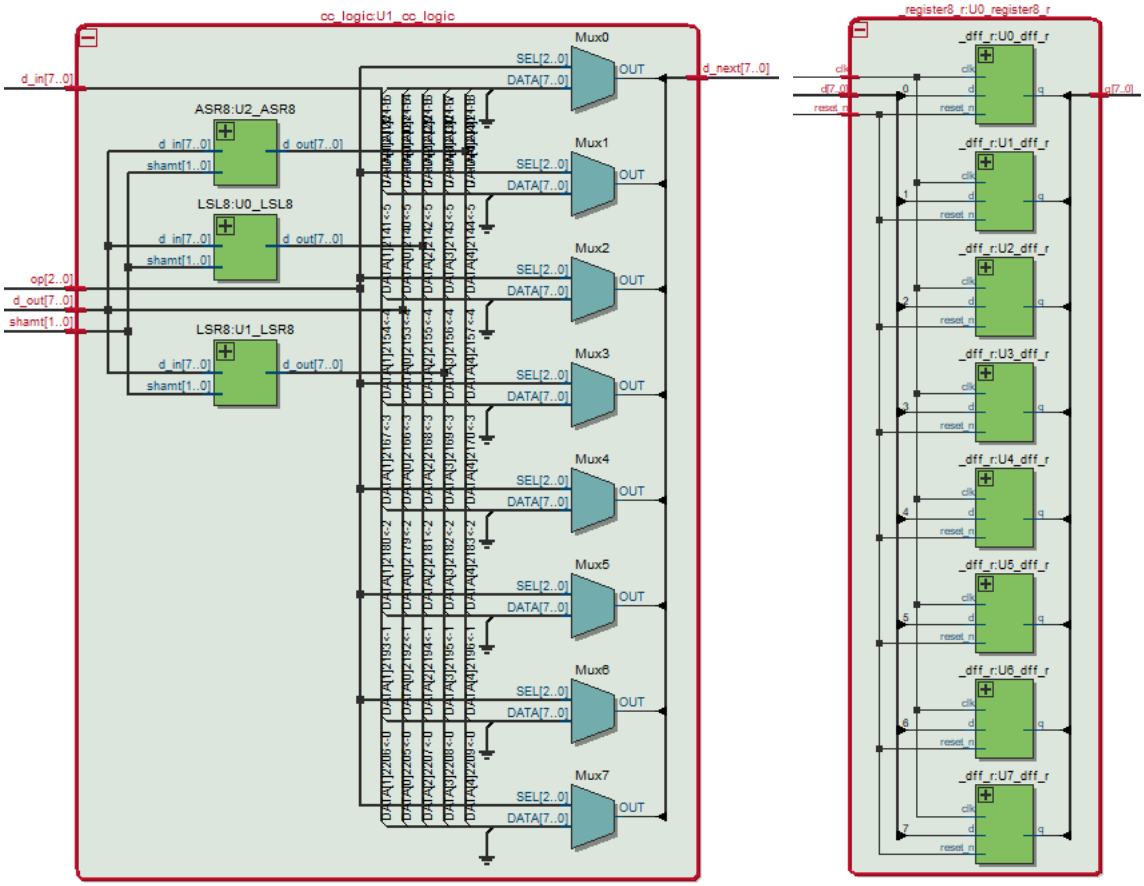
input과 output에 각각 3개의 pin을 이용하여 총 6개의 pin이 사용된 것을 확인할 수 있다.

Flow Summary	
Flow Status	Successful - Sun Oct 18 10:17:04 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cnt5
Top-level Entity Name	cnt5
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	2 / 41,910 (< 1 %)
Total registers	3
Total pins	6 / 499 (1 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

2) shifter8

shifter8은 output logic이 따로 존재하지 않고, register에 저장된 정보가 그대로 출력된다. cc_logic의 구조에서 mux를 이용하여 lsl, lsr, asr연산을 하는 것을 확인할 수 있다.



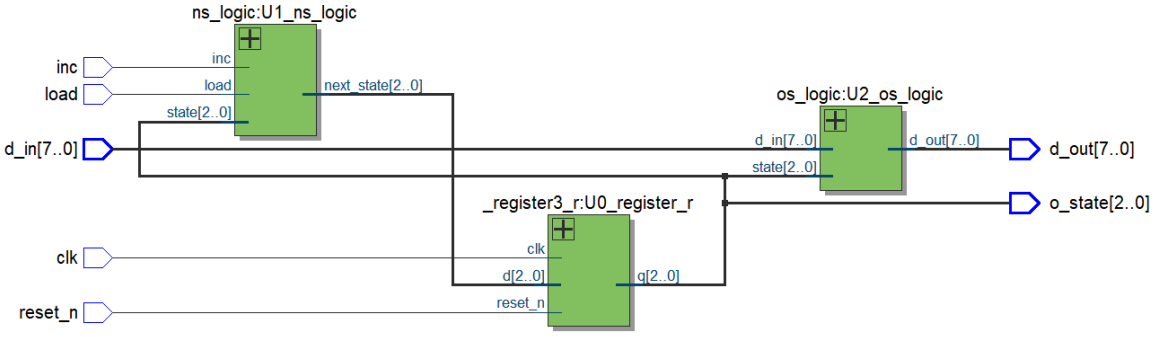


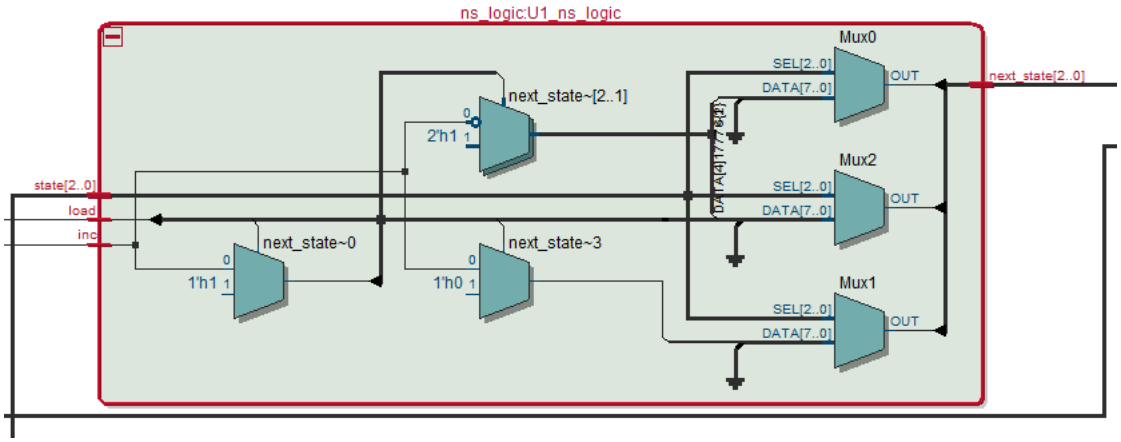
8bit의 d_in, 3bit의 op, 2bit의 shamt, 1bit의 clk, reset_n을 이용하여 15개의 pin이 이용되었고, output으로 8개의 pin이 이용되어 23개의 pin이 이용되었다. 8개의 register가 사용되었다.

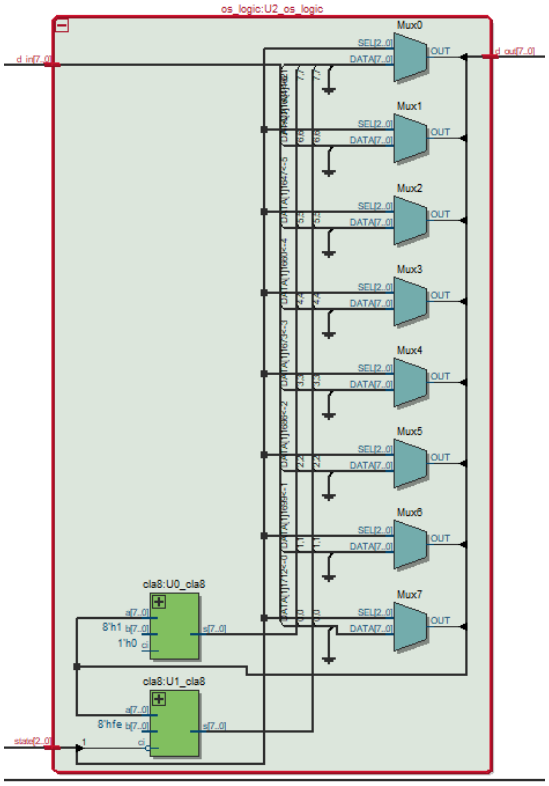
Flow Summary	
Flow Status	Successful - Sun Oct 18 12:56:03 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	shifter8
Top-level Entity Name	shifter8
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	20 / 41,910 (< 1 %)
Total registers	8
Total pins	23 / 499 (5 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

3) counter

ns_logic에서 회로의 다음 state가 결정되고, os_logic에서 count가 계산되는 FSM형태로 구현한 것을 확인할 수 있다. inc, load, d_in, clk, reset_n 입력이 각각 하나의 pin을 이용하고, 데이터의 입력인 d_in이 8개, 출력인 d_out이 8개를 이용한다. 마지막으로 o_state가 3개의 pin을 사용하여 총 23개의 pin이 이용되었다. register는 3개 사용되었다.







Flow Summary	
Flow Status	Successful - Sun Oct 18 15:41:32 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cntr8
Top-level Entity Name	cntr8
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	15 / 41,910 (< 1 %)
Total registers	3
Total pins	23 / 499 (5 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

shifter와 counter를 설계하는 과정에서 공통적으로 output logic을 작성하는 과정에서 data input에 d_in을 연결하는 바람에 계산 결과가 비정상적으로 출력되었다. 이렇게 연결하는 경우 register에서 clk의 주기에 따라 state가 변하기 때문에, logic자체는 실행이 되지만, 계산에 이용되는 data가 동일한 값이기 때문에 output이 변화하지 않는다. 해당 부분의 코드를 이전 결과값에 대하여 연산을 반복하는 것으로 바꾸어 문제를 해결하였다. 나타난 현상이 마치 state가 변하지 않는 것처럼 보였기 때문에 오류를 발견하는 것이 까다로웠다.

B. 결론

이번 과제는 shifter와 counter를 설계하여 구현하는 과제였다. shifter와 counter의 code는 지금까지의 실습에서 작성했던 structural implement 프로그램과 다른 구조를 띈다는 점이 흥미로웠다. 상대적으로 단순하고, 프로그램의 흐름을 파악하기가 더 쉬웠다고 생각한다.

C. loadable counter와 ring counter의 장단점, 응용분야

loadable counter는 외부에서의 input을 이용하여 count할 수 있다는 것이 가장 큰 차별점이고, 장점이라고 생각한다. load기능이 없을 때 원하는 수부터 값을 증가 혹은 감소시키기 위해서는 해당하는 수만큼 count한 뒤에 counter를 이용하는 방법으로 계산을 진행해야겠지만, load기능을 이용하면 해당 과정을 생략하고 바로 원하는 수를 구할 수 있다. 이러한 장점은 counter의 사용영역 전체에서 광범위하게 응용될 수 있다고 생각한다. 동시에 여러 개의 count를 전환하여 이용하는 상황에서, 현재 count값을 외부에 저장했다가 load하여 이용하는 것에 loadable counter를 이용할 수 있을 것이다.

ring counter는 state가 정해진 순서대로 바뀌며 전체가 순환한다는 장점이 있다. 또한 one-hot encoding을 이용하는 경우에는 따로 decoding할 필요가 없다는 것도 장점이다. 그러나 이 경우에 각 bit마다 register를 사용해야하기 때문에 회로의 크기가 커지고, delay가 커질 수 있다는 단점이 존재한다. ring counter는 지정된 state의 가장 마지막에 도달하면 다음에는 다시 최초의 state로 돌아가게 된다는 점을 이용하여 시계에 응용할 수 있을 것이다. 예를 들면 59초가 마지막 state가 되고, 다음 state는 최초의 0초가 되는 것이다.

D. barrel shifter에 대해 조사하고, n-bit의 길이를 가지는 register를 n-bit만큼 shift 시키고자 할 때 필요한 multiplexer의 bandwidth와 수에 대하여 논하시오.

Barrel shifter는 여러 개의 multiplexer를 이용하여 shifting 계산을 한번의 cycle에 실행할 수 있는 회로이다. n bit 길이의 register를 shift하는 barrel shifter에 필요한 multiplexer의 수는 n $\log_2(n)$으로 제시된다. bandwidth란 일정 시간동안 전달할 수 있는 데이터의 양을 나타낸다.

6. 참고문헌

이준환교수님, 디지털논리회로2 강의자료, 광운대학교 컴퓨터정보공학과,2020

이준환교수님, 컴퓨터공학기초실험2 강의자료, 광운대학교 컴퓨터정보공학과,2020

David Money Harris 외1인, Digital Design and Computer Architecture, Elesvier

Johnson Ring Counter, https://www.electronics-tutorials.ws/sequential/seq_6.html

barrel-shifter, https://www.techopedia.com/definition/17863/barrel-shifter