컴퓨터 공학 기초 실험2 보고서

실험제목: Shifter & Counter

실험일자: 2022년 10월 18일 (화)

제출일자: 2022년 10월 20일 (목)

학 과: 컴퓨터정보공학부

담당교수: 공영호 교수님

실습분반: 화요일 0,1,2

학 번: 2019202050

성 명: 이강현

1. 제목 및 목적

A. 제목

Shifter & Counter

B. 목적

Flipflop과 combinational logic을 이용해 sequential logic인 shifter과 counter을 설계해본다. FSM의 동작 방식을 이해하여 Mealy Machine, Moore Machine의 차이점을 이해하고 FSM인 Counter와 Shifter를 Verilog로 구현해본다.

2. 원리(배경지식)

<Finite State Machine(FSM)>

유한 상태 기계(FSM)란 하나의 상태만 취할 수 있는 기계를 말한다. 유한 상태 기계는 한번에 하나의 상태를 가질 수 있으며 현재 상태는 외부로부터 입력을 받으면 조건에 따라 정해진 다른 상태로 변할 수 있다. 이러한 FSM은 Moore machine, Mealy machine으로 나뉜다.

<Moore/Mearly FSM>

Moore/Mearly FSM은 FSM의 종류로 출력값을 어떻게 결정하느냐에 따라 차이점이 있다. Moore FSM은 현재상태에 따라 출력값이 정해지는 반면, Mealy FSM은 현재상태와 입력 두가지의 영향을 받아 출력값을 결정한다.

Moore FSM은 출력값을 결정할 때 현재 상태만을 고려하면 되기 때문에 더욱 직관적으로 출력값을 산출해낼 수 있다. Mealy FSM은 현재상태에 입력값을 고려하여 결과값을 산출해내야 하기 때문에 더 복잡하지만 state에 입력이 반영되어 Moore보다 더 적은 state를 가지게 된다. 따라서 단계를 축소할 수 있어 이러한 점에서는 효율적이라고 할 수 있다.

<Combinational Logic>

조합 논리(Combinational Logic)란 입력에 의해서만 출력이 결정되기에 이전상태를 기억하는 메모리가 존재하지 않는다.

<Sequential Logic>

순차 논리(Sequential Logic)란 입력과 이전상태에 대해서 출력이 결정되기 때문에 이전상태를 기억할 수 있는 메모리 소자가 있다.

<Counter>

카운터는 수를 세는 회로이다. Clock에 따라 정해진 순서로 state의 변화가 진행되는 Register라고 할 수 있다. Sequential Logic의 대표적인 예시로 사용된다.

<Ring Counter>

데이터가 회전하는 shift register를 말한다. 마지막의 flip flop의 값이 첫번째 flip flop의 입력값으로 전달되며 clock에 따라 연속적인 계산을 하는 회로이다.

3. 설계 세부사항

8-bit Loadable Shifter

LSL: Logical shift left는 register를 shift amount만큼 왼쪽으로 이동시킨 후 남는 빈공간에는 0이 삽입된다.

LSR: Logical shift right는 register를 shift amount 만큼 오른쪽으로 이동시킨 후 남는 빈공간에는 0이 삽입된다.

ASR -> Arithmetic Shift Right는 register를 shift amount만큼 오른쪽으로 이동시킨다. 오른쪽으로 이동시킨 후 빈 공간에는 이전의 값을 넣어준다.

설계된 FSM은 각 상태에 맞게 다음과 같은 동작을 수행할 것이다

NOP은 no operation의 약자로 아무 행동도 취하지 않는다.(현재의 register값을 그대로 출력)

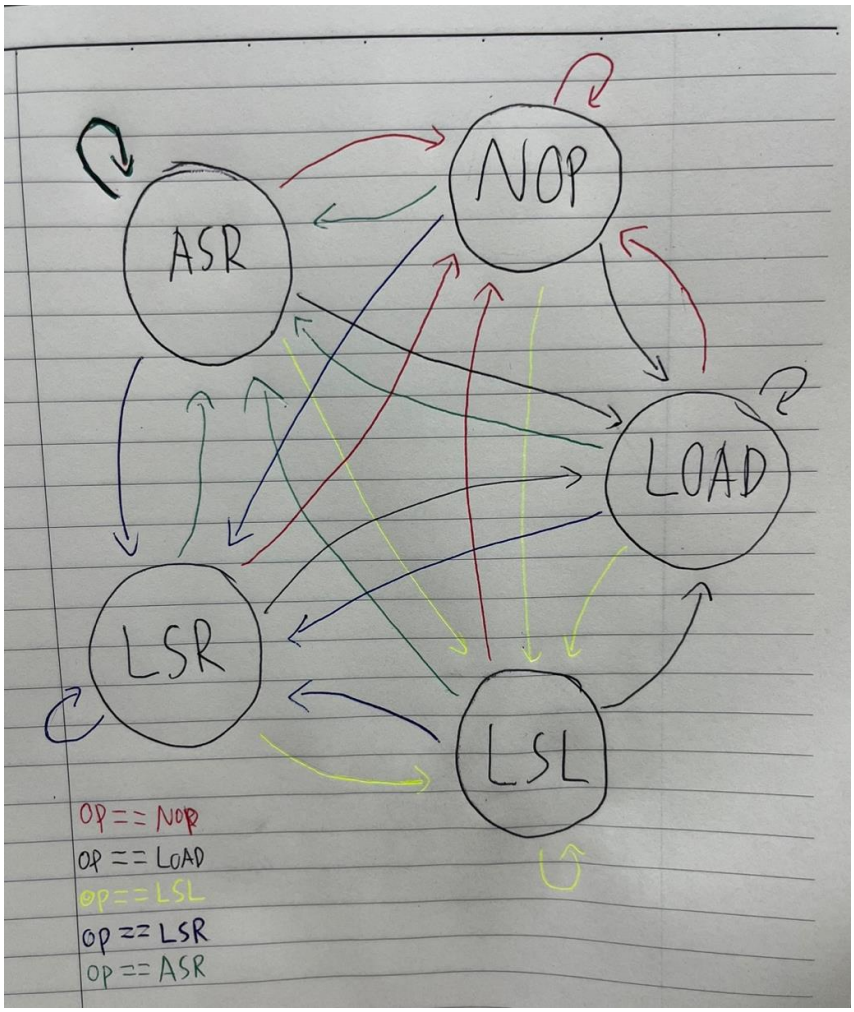
LOAD는 입력값을 그대로 가져와 출력한다.

LSL은 Logical shift left를 수행하고 출력한다.

LSR은 Logical shift right를 수행하고 출력한다.

ASR은 Arithmetic shift right를 수행하고 출력한다.

1) State diagram



2) Encoding state

opcode	Binary
NOP	3'b000
LOAD	3'b001
LSL	3'b010
LSR	3'b011
ASR	3'b100

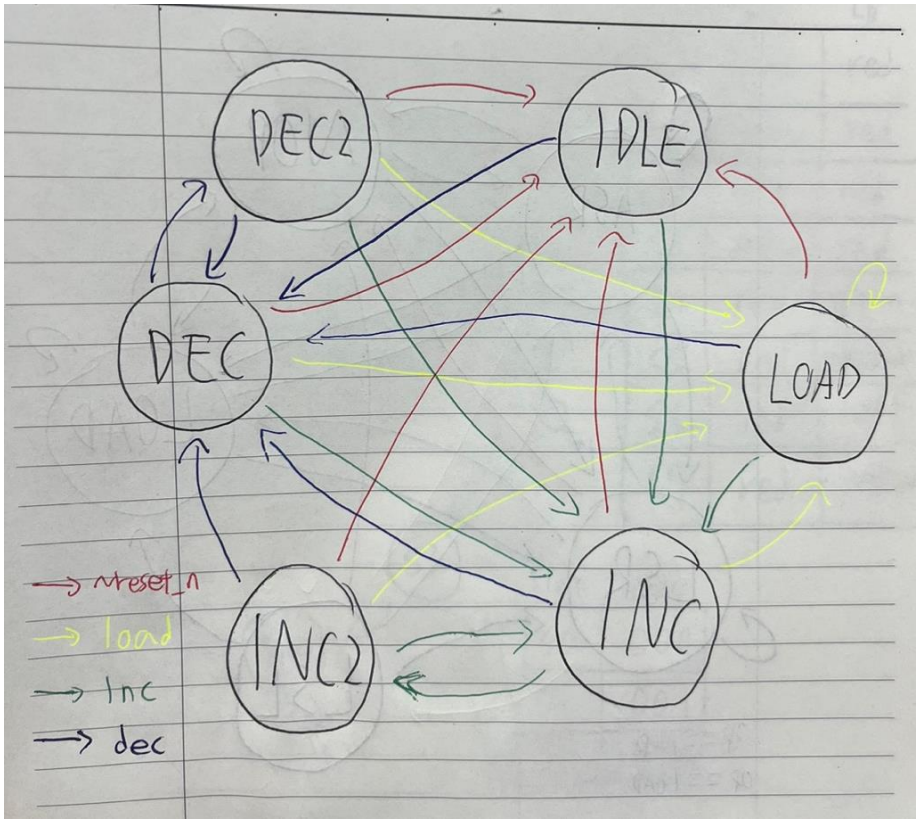
위와 같은 state diagram을 가지고 각 state들은 표와 같이 encoding된다.

Input같은 경우는 op(명령어)와 shamt(이동비트수), d_in(다음상태) output은 d_out(현재상태)로 구성된다. 명령어에 따라 5가지 상태 모두에 접근할 수 있는 프로그램이므로 verilog내에서는 parameter를 통해 state를 encoding 하였다. Op가 encoding된 state를 가리키면 위에서 설명한 각 state들의 동작방식에 맞게 프로그램이 진행된다.

<Counter>

우선 cntr의 state가 어떤 것이 있는지 또 state에 따라 어떻게 동작하는지 설명한다.
IDLE, LOAD, INC, INC2, DEC2, DEC로 총 6가지의 State가 존재하며
IDLE=0, LOAD=1, INC=2,INC2=3, DEC=4, DEC2=5의 값을 가지도록 설계하였다.
IDLE 명령어는 d_out을 0으로 만들어 d_in이 0이 되게끔 설정하고
LOAD 명령어는 d_out에 그대로 d_in을 인가하는 역할을 수행한다.
INC과 INC2는 값을 1증가시키는 같은 동작을 취하고 같은 명령어가 지속될 때 서로 상태를 바꾸며 동작한다.
DEC과 DEC2 또한 INC과 INC2와 같이 1을 감소시키는 동작을 취하고 같은 명령어가 지속될 때 서로 상태를 바꾸며 동작한다. 자세한 프로그램의 흐름도는 아래의 state diagram을 참조한다.

<State Diagram>



<state encoding>

opcode	Binary
IDLE	3'b000
LOAD	3'b001
INC	3'b010
INC2	3'b011
DEC	3'b100
DEC2	3'b101

위와 같은 state와 opcode를 가진다.

cntr8은 clock에 의해 수가 증가 혹은 감소하며 reset_n, load, inc, d_in을 input값을 가진다.

reset_n: register값을 0으로 초기화하며 active low에서 작동한다.

load: 데이터가 입력되면 d_in값을 count값에 할당한다.

inc: Counter의 증가, 감소를 제어하는 신호로, 1일 경우에는 가산, 0일 경우에는 감산을 수행한다.

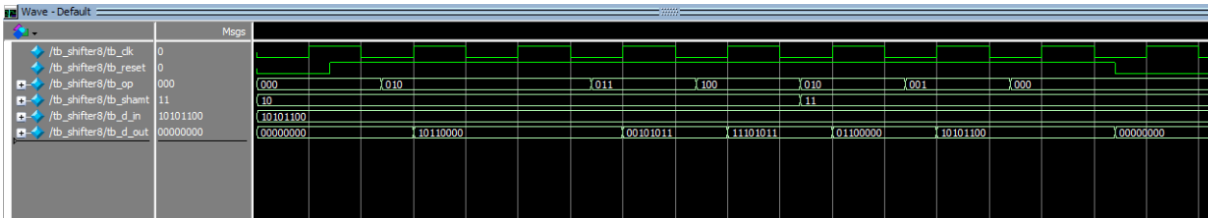
d_in: register의 output값으로 다시 logic에 인가되는 값이고 load가 1일 때 count로 할당되는 값이다.

output값은 d_out와 o_state으로 d_out은 d_in과 현재 상태 즉 명령어에 따라서 결정된다. o_state는 현재 어떤 명령어 상태인지를 확인할 수 있다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

<Shifter8>(top module)



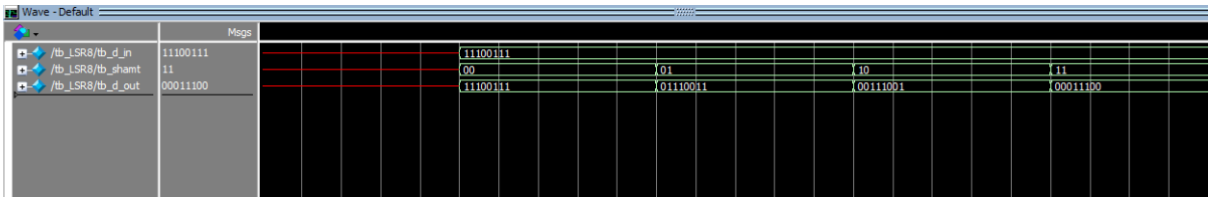
우선 앞에서부터 설명하면 reset은 active low이므로 0일 때 값들이 인가되지 않고 0을 유지하고 있는 것을 볼 수 있다. Reset이 1로 변하고 첫 rising edge clock을

만났을 때 op 010(LSL)이 shamt(10)과 input으로 들어가며 d_in(10101100)을 왼쪽을 2비트 이동시키는 동작을 수행한다. 따라서 d_out이 10110000으로 잘 나오는 것을 확인할 수 있다. 그 이후 만나게 되는 rising edge에서 op 011(LSR)이 인가되면서 10101100인 d_in을 오른쪽으로 2비트 이동한 00101011을 d_out에서 나타남을 확인할 수 있다. 그다음 op 100(ASR)은 arithmetic shift right연산이므로 현재 값이 10101100 MSB가 1인 음수값을 오른쪽으로 이동하는 연산을 하여 오른쪽으로 2비트 이동하고 sign extension이 일어난 것을 확인할 수 있다. 그 이후의 값들도 정상적인 처리 결과를 보여준다. 여러가지 state에 대한 결과를 확인할 수 있기에 testbench의 값을 위와 같이 설정했다.

<LSL8>(sub module)

비트가 잘 이동하는지 확인하기 위해 값은 11100111로 주었고 이동할 비트 수를 결정하는 shamt값은 00 01 10 11 값을 주었다. 빈공간은 0으로 채워지는 것으로 보아 올바른 값을 얻었다.

<LSR8>(sub module)



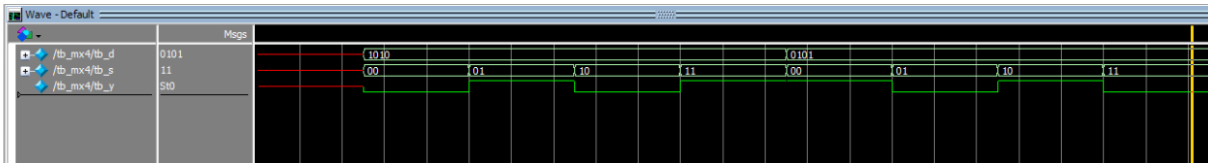
LSR8 또한 LSL8과 같이 비트가 잘 이동하는지 확인하기 위해 값은 11100111로 주었고 이동할 비트 수를 결정하는 shamt값은 00 01 10 11 값을 주었다.
빈공간은 0으로 채워지는 것으로 보아 올바른 값을 얻었다.

<ASR8>(sub module)

ASR8은 위의 두 비트 이동 연산과 다르게 MSB를 제대로 복사하여 빈공간에 채워넣는지가 핵심이므로 11100111과 01100111로 케이스를 나누어

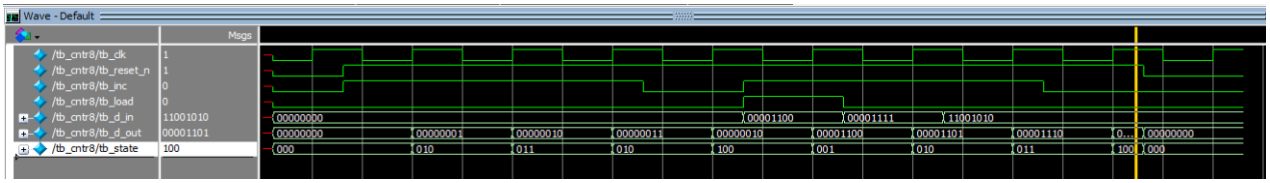
testbench값으로 넣어주었다. MSB를 잘 채워넣는것으로 보아 올바른 값을 얻었다.

<mx4>(sub module)



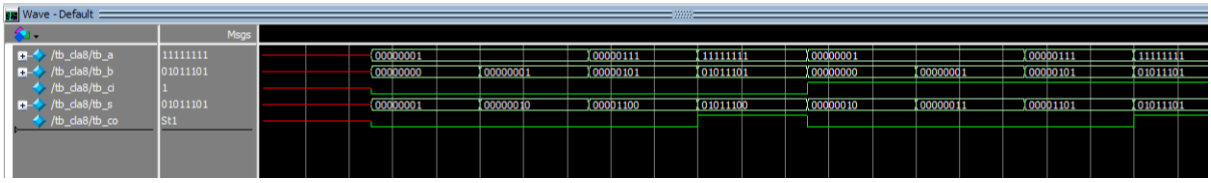
기존에 구현한 mx2를 3개 instance하여 만든 mx4 모듈이다. 각 자릿수를 select에 맞게 뽑아내는 것을 확인하기 위해 1010과 0101을 testbench의 케이스로 넣어주었다. 올바르게 값을 뽑아내는 것을 확인할 수 있다.

<cntr8>(top module)



reset이 0일 때 결과값들은 모두 0임을 확인할 수 있다. Reset이 1이되고 inc이 1이 되어 d_out값이 1증가한 것을 볼 수 있고 state가 010(INC)으로 변화함을 알 수 있다. 한번 더 inc이 rising edge에 의해 반영되자 state가 011(INC2)로 변화됨을 알 수 있고 inc이 0이 되기까지 delay를 고의적으로 크게 준 결과 inc이 1로 유지되면 010->011->010을 반복하며 count가 늘어나는 것을 확인 할 수 있다.
Inc이 0이되었을 땐 이전의 값에서 1을 감산하였고 state 또한 100(DEC)로 바뀐 것을 확인할 수 있다. 다시 load를 1로 올린 결과 inc의 값과 관계없이 00001100이라는 값이 잘 인가됨을 확인할 수 있으며 최종적으로 reset이 0이되어 초기화하는 것까지 확인하였다. 여러가지 state에 대한 결과를 확인할 수 있기에 testbench의 값을 위와 같이 설정했다.

<cla8>(sub module)



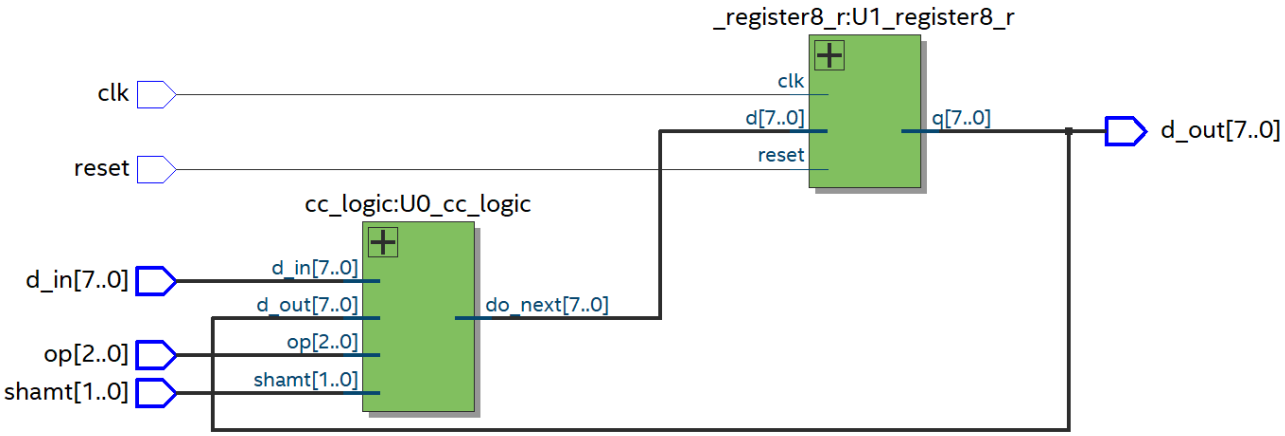
기존에 구현한 cla4 모듈 2개를 instance하여 구현한 cla8 모듈이다. Cla 모듈은 값의 연산을 제대로 해내는 지를 테스트하는 것이 중요하므로 ci가 0과 1일 때

두가지 케이스로 나누었고 carry가 발생하지 않는 경우,발생하는 경우, 두가지가 모두 각 자리에서 잘 이루어지는지, 8bit단위를 벗어나는 연산에서 co값을 잘 뽑아내는지를 확인하기 위해 위와 같은 testbench값을 구성하였다. 올바른 값을 얻어내는 것을 확인할 수 있다.

B. 합성(synthesis) 결과

<Shifter8>

<RTL Viewer>



설계한 shifter8은 combinational logic이 state을 register의 output 값을 받아 input에 맞게 변경해주고 다시 register의 input으로 인가해주는 구성으로 제작되었다. 위의 RTL viewer를 확인한 결과 회로가 올바르게 구성되었음을 확인할 수 있다.

<Flow Summary>

Flow Summary

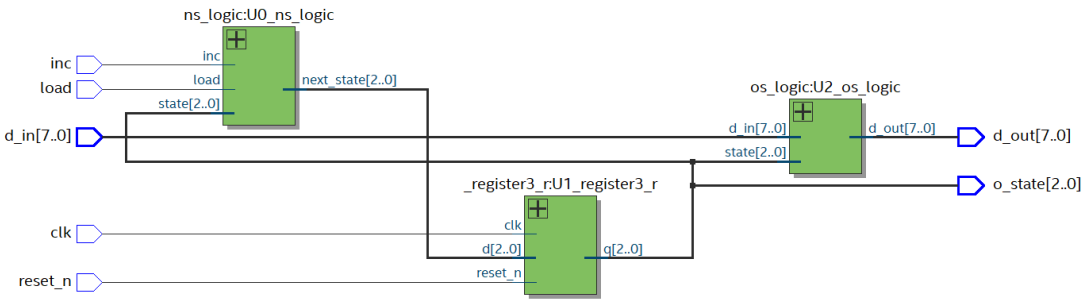
<<Filter>>

Flow Status	Successful - Tue Oct 18 03:48:38 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	shifter8
Top-level Entity Name	shifter8
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	8
Total pins	23
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

shifter8을 구현하는데 사용된 pin은 23개가 사용된 것을 볼 수있다.

<Cntr8>

<RTL Viewer>



설계한 counter는 next state가 input과 register의 output에 의해 결정되며 현재 state와 input의 의해 output이 결정되는 Mealy FSM의 설계 회로의 구성을 띄는 것으로 보아 회로가 잘 설계되었음을 확인할 수 있다.

<Flow Summary>

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Oct 17 23:44:59 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cntr8
Top-level Entity Name	cntr8
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	3
Total pins	23
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

cntr8을 구현하는데 사용된 pin은 23개가 사용된 것을 볼 수 있다.

5. 고찰 및 결론

A. 고찰

Barrel Shifter란 한번의 연산으로 데이터 워드내에 있는 다수의 비트를 이동하거나 회전시킬 수 있는 하드웨어를 의미한다. n-bit의 길이를 가지는 register를 n-bit 만큼 shift 시키고자 할 때 필요한 multiplexer의 개수는 n*log2n개이고 n개의 bandwidth를 필요로 한다.

Ring Counter는 환형 계수기이므로 마지막 flip flop을 지나 초기상태로 순회할 수 있다는 장점이 있고 따라서 FSM에서 사용할 때 계속 순회가 가능하기에 장점이 될 수 있을 것 같다. 그에 반해 순회가능이라는 특성 때문에 동일 값의 참조가 발생한다라는 단점이 존재한다. 또한 loadable counter는 초기 값을 설정할 수 있는 counter로 이번 실습의 측면에서 설명하면 임의의 새로운 값을 로드하여 다음 연산때는 그 값을 기준으로 증감연산을 할 수 있는 counter이다.
이러한 특성에 따라 MCU(micro controller unit)의 응용 시스템에서 많이 사용되며 클럭에 따라 데이터버스에 로딩되는 값을 입력으로 하고 다음 클럭부터는 해당 값부터 증감이 이루어지는 형식으로 사용 가능하다.

B. 결론

이번 실습에서는 이전 실습에서 배운 FSM을 기반으로 counter와 shifter를 구현해보았다. 이전 실습과 verilog 측면에서 가장 다른 점은 논리식을 간소화해서 ns_logic과 os_logic을 구현했던 지난 실습과 달리 parameter를 이용하여 코드내에서 state를 encoding하고 이를 case 구문과 조건문을 통해 값을 할당할 수 있다는 것을 새로이 배웠다. Traffic controller를 만드는 방법보다 좀 더 눈으로 코드를 이해하기 쉽고 코드를 구현하는 것도 문법만 제대로 이해한다면 더 쉬웠다. 고찰을 통해 ring counter나 loadable counter등 다양한 카운터에 대한 공부를 할 수 있었고 기존에 구현했던 mux를 통해 shift를 구현하는 방법 또한 다시금 배운내용을 상기할 수 있어 개념 공부에 효과적이었다.

6. 참고문헌

공영호 교수님/컴퓨터공학기초실험2/2022

KR20030078126A - 로드가능 업/다운 카운터 회로/

https://patents.google.com/patent/KR20030078126A/ko