컴퓨터 공학 기초 실험2 보고서

실험제목: Shifter & Counter

실험일자: 2022년 10월 19일 (수)

제출일자: 2022년 10월 25일 (화)

학 과:컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 수요일 0, 1, 2

학 번: 2019202021

성 명: 정 성 엽

1. 제목 및 목적

A. 제목

Shifter & Counter

B. 목적

flip flop과 combinational logic을 사용하여 sequential logic인 shifter와 counter를 설계하고 구성을 이해한다. 그리고 설계한 logic을 testbench를 통하여 shift와 counter가 정상적으로 작동했는지 확인한다.

2. 원리(배경지식)

A. shifter

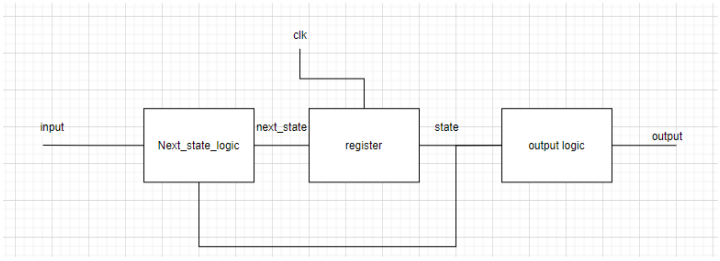
shifter는 register에 저장되어 있는 정보를 단방향 또는 양방향으로 이동 시킬 수 있는 하드웨어로 설계할 shifter는 8bit로 명령은 opcode를 통해 받는다. 중요하게 봐야하는 것은 LSL, LSR, ASR opcode로 실제로 Assembly Language에서 shift 연산을 사용할 때 사용하는 opcode이다. 이를 하드웨어적으로 구현하는 것이 이번 실험이라고 볼 수 있다.

예) LSL opcode인 경우

1'b0110_1001 -> 1'b1101_0010

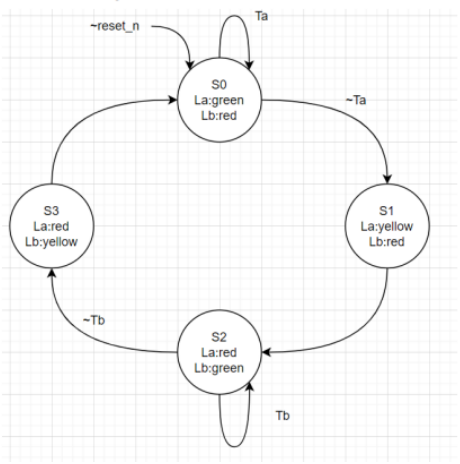
B. Moore FSM & Mealy FSM

i. Moore FSM



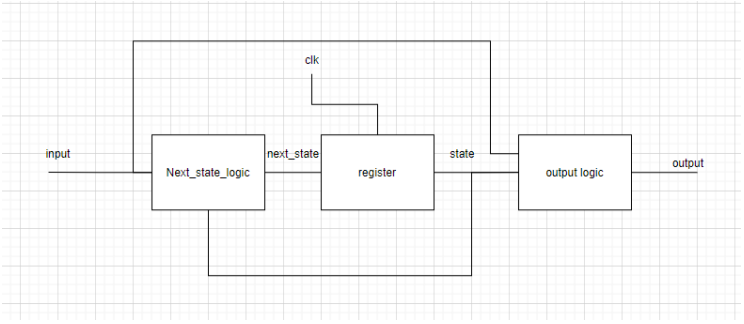
Moore FSM은 결과값이 input에 관계 없이 현재 상태에 따라서 output이 달라지는 finite state machine이다. 장점으로는 상태 변화하는 과정을 모두 states로 두어 설명하기 때문에 이해하기 더 쉽고 조합논리가 감소하는 특징이 있그며 단점으로는 state수가 많아서 output을 추출하는 시간이 mealy FSM보다 더 많이 걸린다. 그리고 state의 증가 때문에 flipflop의 개수가 증가하는 단점이 있다.

예) Traffic light Controller



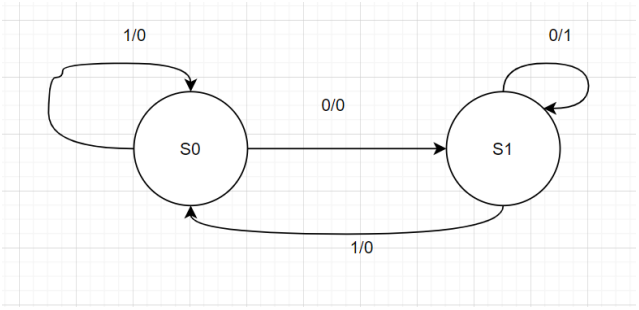
이전에 만든 Traffic light Controller를 보면 Ta의 이동 Tb 의 이동에 따라 state의 변화가 생기고 현재 상태에 따라서도 다음 state가 달라짐을 확인할 수 있다.

ii. Mealy FSM



Mealy FSM은 결과 값이 input과 현재 상태에 따라 output이 달라지는 finite state machine이다. Mealy FSM의 장점은 state의 개수를 줄일 수 있어 Moore FSM보다 output이 더 빨리 추출이 됩니다. 이유는 state가 줄어든 만큼 input값과 상태값만 입력이 된다면 바로 결과가 출력되기 때문이고 같은 이유로 Flipflop의 개수가 줄어든다. 단점으로는 Moore FSM에 비해 한눈에 파악하기 어렵다. 그리고 input 뿐만 아니라 상태도 같이 따지면서 output을 판단하기 때문에 조금 더 복잡하다.

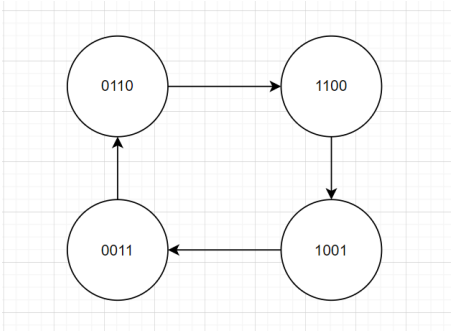
예) mealy FSM 예시



각 상태만 state로 나타내고 input에 따라 state가 변화하는 예시이다.

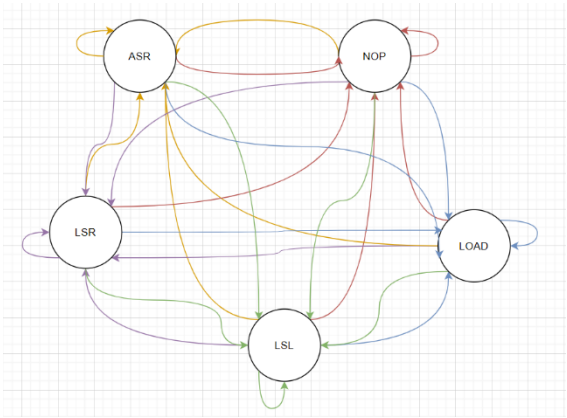
C. Ring counter

링 카운터는 어떠한 패턴으로 계속 회전하는 형태의 카운터 모듈로 예를 들어 4비트의 데이터를 loa의 값이 1일 때 d를 통해 입력시킨 후에 clock이 rising edge일 때 밑의 그림과 같이 shift를 하여 회전한다. 이 예제는 shift register의 역할도 하면서 카운트르 해준다



3. 설계 세부사항

A. 8-bit Loadable shifter



red line: op==NOP

blue line: op==LOAD

green line op==LSL

purple line op==LSR

orange line op==ASR

구분	이름	설명
top module	shifter8	top module
sub module	LSL8	8bit logical shift left
	LSR8	8bit logical shift right
	ASR8	8bit arithmetic shift right
	mx4	1 bit 4to1 MUX

이름	구분	이름	비트	설명
shifter8	input	clk	1bit	clock
		reset_n	1bit	active low reset signal
		op	3bit	operation code
		shamt	2bit	shift amount
		d_in	8bit	op에서 load 일때 d_in 통해 register 저장
	output	d_out	8bit	Register 값 출력

LSL8	input	d_in	8bit	data in
LSR8		shamt	2bit	shift amount
ASR8	output	d_out	8bit	data out
mx4	input	d0	1bit	first data
		d1	1bit	second data
		d2	1bit	third data
		d3	1bit	fourth data
		s	2bit	selection signal
	output	y	1bit	output

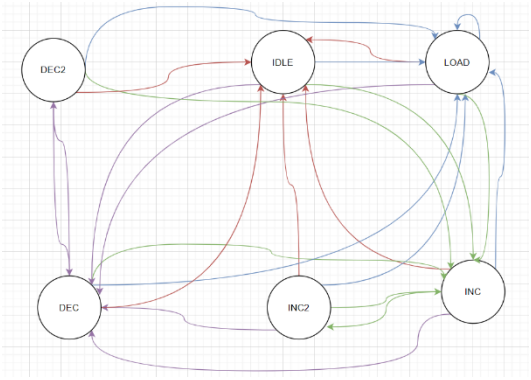
B. 8bit loadable up/down counter

red line: ~reset_n

blue line: load

green line: inc

purple line: dec



구분	이름	설명
top module	cntr8	top module
sub module	cla8	8bit carry look ahead adder top module에서 두개를 instance하여 inc가 0일 때와 1일 때 결과 값 계산

이름	구분	이름	비트	설명
cntr8	input	clk	1bit	clock
		reset_n	1bit	active low reset signal
		load	1bit	입력된 data count 값에 할당
		inc	1bit	count 값을1일 경우에 증가 0이면 감소
		d_in	8bit	load가 인가되면 해당 값 할당위한 data 입력
	output	d_out	8bit	count된 output
		o_state	3bit	현재 state 값 출력(검증용)
cla8	input	a	8bit	cla의 입력 A
		b	8bit	cla의 입력 B
		ci	1bit	cla의 carry in
	output	s	8bit	cla 출력 s
		co	1bit	cla의 출력 carryout

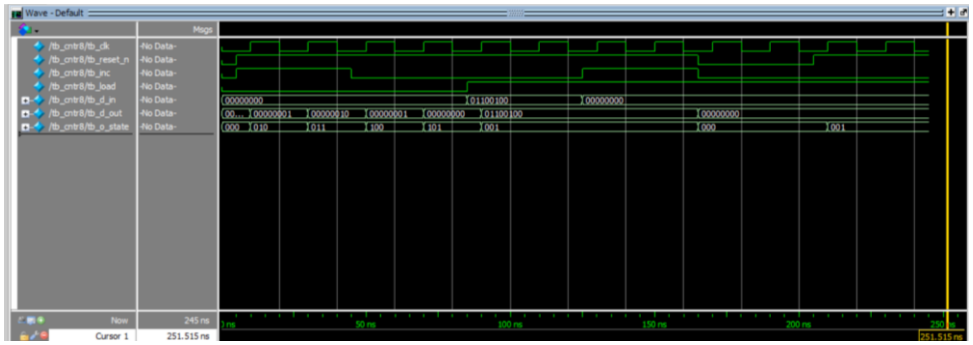
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

i. shifter8

처음에는 0으로 모두 초기화 하고 그 다음에 001이라는 opcode를 작동 시켜 값을 load 받고 010으로 LSL 연산하여 2배가 된다. LSR에서 shamt가 01이므로 오른쪽으로 1씩 shift 한다. ASR은 shamt의 양만큼 오른 쪽으로 shift하여 덮은 것을 확인하였다.

ii. cntr8

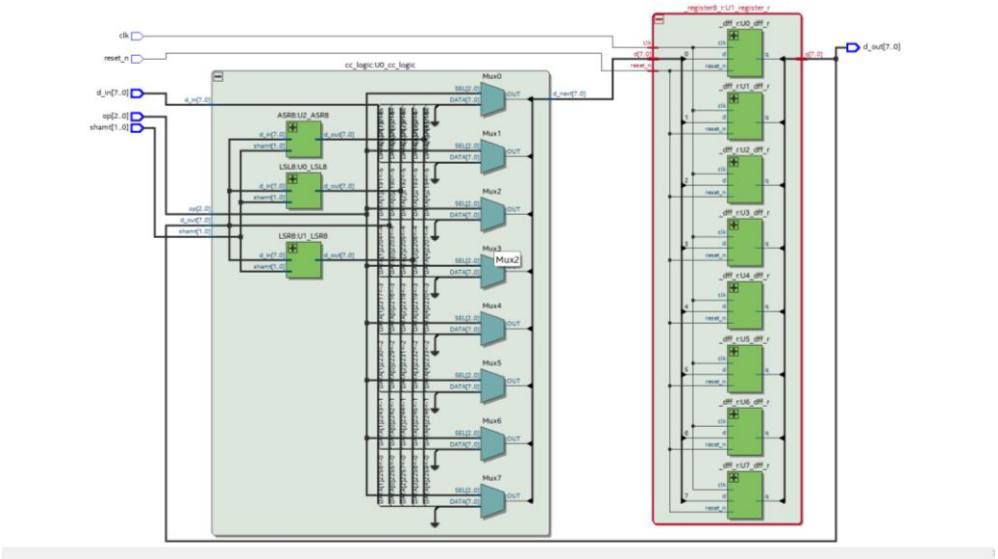


위의 testbench를 보면 001로 o_state가 reset이므로 d_out이 0이고 그 다음에 reset_n이 1로 set 되면서 clock rising edge 부터 load가 1이 될 때까지 clock의 모든 positive edge에서 count를 한다. inc는 1일 때 증가 0일 때 감소 하는 것을 확인하고 새로운 값을 넣어 다시 확인해보았다.

B. 합성(synthesis) 결과

i. shifter8

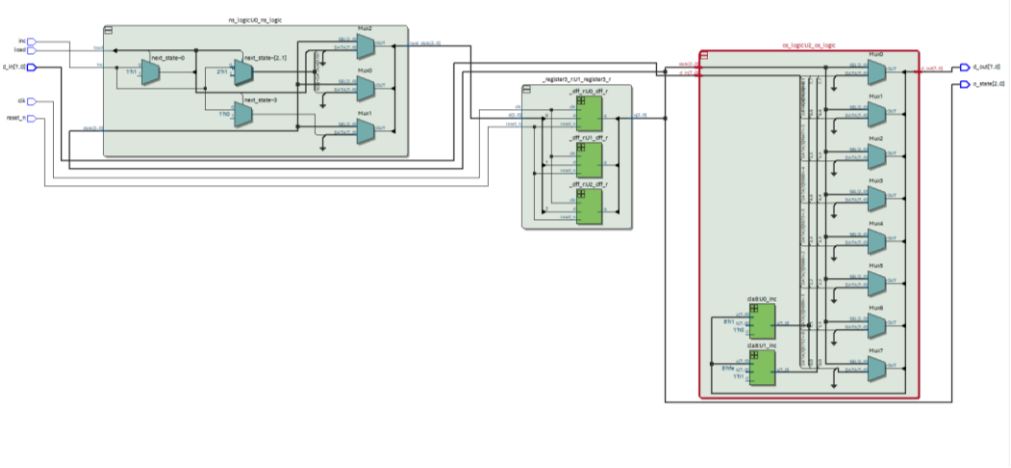
Flow Summary	
Flow Status	Successful - Tue Oct 25 23:17:16 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	shifter8
Top-level Entity Name	shifter8
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	20 / 41,910 (< 1 %)
Total registers	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 %)



cc_logic 그리고 register8을 instance 하여 8bit loadable shifter을 만들었고 d_in으로 쓰인 input 값은 Load할 때만 사용하고 register에서 나온 결과값을 cc_logic 그리고 d_out으로 대입하여 작동한다.

ii. cntr8

Flow Summary	
Flow Status	Successful - Tue Oct 25 23:36:16 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)	18 / 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 %)



ns_logic 서브모듈과 op_logic 서브 모듈과 register을 만들어 instance 하였고 cla8은 top 서브 모듈이기 때문에 d_out 값이 연결이 되어야 output logic으로 값이 전달 됨을 확인할 수 있다.

5. 고찰 및 결론

A. 고찰

이번 실험을 통해서 alwyas문과 mux,shifter을 이용하여 loadable shifter그리고 cntr8을 구현하였다. 어셈블리프로그래밍설계및실습 수업에서 사용했던 shift 연산은 이해하기 편하여 오히려 반가웠다. 다만 sub module에 대한 이해점이 필요했고 실습 자료에 주어진 코드를 참고하여 verilog를 완성하고 이를 testbench로 돌려보며 이 과제를 이해하였다.

다만 cntr8은 너무나 생소하였고 FSM도 주어진 것을 이해하기 어려웠다. 처음에는 각 state 조차 이게 무엇을 위해 존재하는 것인지 헷갈렸지만 이 또한 자료를 참고하여 과제를 완성하고 testbench를 돌려 확인함으로써 counter에 대한 이해가 수월할 수 있었다.

B. 결론

이번 실험을 통해 shifter와 counter의 구현 방법과 동작 방법 등에 대해 자세히 배울 수 있었고 어떤 방법으로 값이 저자오디고 읽히는지 대해서도 알게 되었다. 그리고 지난 TLC와 같이 FSM을 직접 짜보면서 counter를 만드는 것은 어려웠지만 그만큼 많은 것을 배울 수 있었다.

Loadable counter와 ring counter의 장단점에서 전자는 우선 Load를 하고 어떤 특정 값을 넣고 count 할 수 있다는 장점이 있고 값이 한정 되어있지 않고 원하는 만큼 비트 자리수만 맞춘다면 큰 값까지도 count 가능하다. 하지만 회로도가 복잡하고 사용되는 모듈 수가 ring counter에 비해 많다. 후자의 경우 case 문에서도 그냥 바로 다음 state나 이전 state에 연결만 하면 되기 때문에 좀더 구현이 쉽고 회로도 간단하다. 대신 표현되

는 숫자 범위가 전자에 비해 매우 작다.

Barrel shift는 combination logic을 이용하여 shift를 할 수 있는 shifter로 오직 mux만을 이용하여 구현할 수 있다. n-bit만큼 shift를 하고자 할 때에는 $nlog_2n$ 개의 1 bandwidth multiplexer가 필요하다. 왜냐하면 select의 경우의 수를 구할 때 n만큼 경우의 수가 나와야하고 그러므로 select의 경우 y라고 두면 2^y=n이 되어야 한다. 이를 y에 관한 식으로 나타내면 아래와 같다.

$$y = log_2 n$$

그리고 select는 1비트당 8개 mux를 사용해야 하므로 8을 곱한다.

6. 참고문헌

공진흥 교수님/컴퓨터공학기초실험2/광운대학교 컴퓨터정보공학부/2022

공영호 교수님/디지털논리회로2/광운대학교 컴퓨터정보공학부/2022