컴퓨터 공학 기초 실험2 보고서

실험제목: Counter & Shifter & Register file

실험일자: 2018년 10월 03일 (수)

제출일자: 2018년 10월 09일 (화)

학　　과: 컴퓨터공학과

담당교수: 공진흥 교수님

실습분반: 수요일 0, 1, 2

학　　번: 2015722025

성　　명: 정 용 훈

1. 제목 및 목적

A. 제목

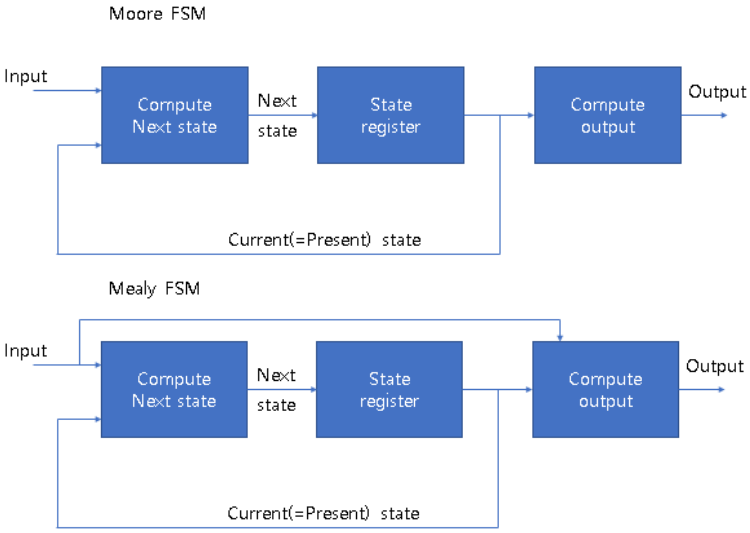
Counter & Shifter & Register file

B. 목적

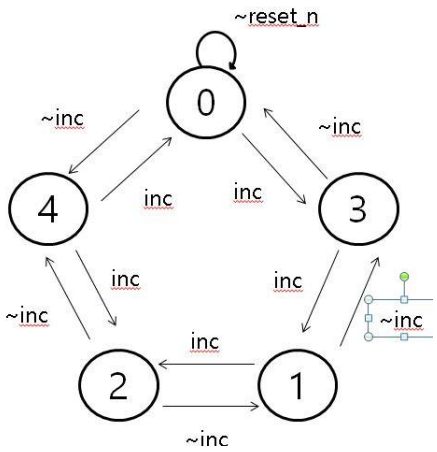
Counter, Shifter 와 Register file의 하위 module부터 구현하고 Top module이 구성되는 module이 쓰이는 용도를 이해하며, Test bench를 통하여 구현한 Top module이 정상적으로 작동하는지 알아본다. 또한 Test bench를 통한 값들이 어떻게 도출되는지 이해한다.

2. 원리(배경지식)

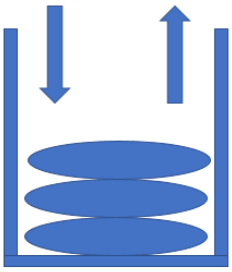
가장 먼저 Counter와 Shifter를 구현하기 위해서는 FSM에 대한 개념과 이해가 필요하다. FSM의 종류로는 Moore FSM과 Meal FSM으로 나뉜다. 두 FSM의 다른 점은 Output이 결정되는 방법에 차이가 있다. Moore FSM은 현 상태가 Output에 영향을 미치는 유일한 조건이지만 Mealy FSM은 현 상태만 영향을 주는 것이 아니라 Input의 영향도 같이 받는다.



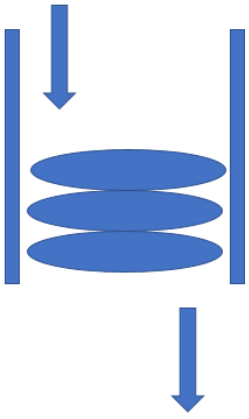
위 그림은 두 종류의 FSM을 그림으로 표현한 이미지이다. 여기서 의문이 생길 수 있다. 같은 FSM인데 왜 나눠 놓았을까? 두 기계의 장담점이 있기 때문이다. 우선 Moore FSM의 장점을 이야기하면 Output값의 결정이 state에만 있기 때문에 설계하는 입장에서는 편하고 이해하기 쉽다는 장점이 있다. 단점으로는 같은 설계로 Mealy FSM에 비해 Output을 표현하기 위한 State의 수가 더 많아야 한다는 단점이 있다. Mealy FSM의 장점은 앞서 Moore FSM의 단점을 보완할 수 있다는 점이다. Output의 결정이 Input과 State로 결정되기 때문에 Output을 결정하는데 있어 Moore FSM 보다 좀 더 적은 State로 같은 Output을 결정할 수 있다. 단점으로는 설계의 어려움이 있을 수 있다. State만으로 Output이 결정되는 Moore FSM에 비해 Input까지 고려를 하기 때문에 설계하는 입장에서는 이해하기 어려울 수 있다. 두 모델의 장단점이 쌍대성을 띄고 있기 때문에 어떤 FSM이 더 좋다고 판단하기 어려울 수 있다. 가장 좋은 방법은 Moore FSM을 기준으로 Mealy FSM을 응용하면 FSM을 좀더 최적의 모델로 구현할 수 있다고 생각된다. 다음은 우리가 구현하게 될 Counter의 대한 개념이다. 말 그대로 숫자를 세는 module이다. Input인 inc의 값에 영향을 받으며 inc의 값이 순차적으로 증가하며 inc의 값이 0이면 순차적으로 감소하게 된다. 우리가 구현하게 될 Counter의 형태는 Ring Counter로 마지막 플립플롭의 출력이 첫 번째 입력에 공급되는 형태를 하고 있다.



다음 이미지는 Ring Counter의 모습을 도식화한 것이며 inc의 값에 따라 Counter의 이동 방향이 다르며 마지막 값에서 다시 처음 값으로 넘어가는 것을 확인 할 수 있다. 다음으로는 Stack와 Queue에 대한 설명이다. Stack과 Queue는 자료구조를 배우는데 있어 중요한 개념이다. 우선 Stack은 LIFO의 입출력 방식을 따른다. LIFO란 Last In First Out으로 가장 나중에 들어간 정보가 가장 먼저 나오는 방식의 자료구조이다. 그림으로 쉽게 설명하면 다음과 같이 볼 수 있다.



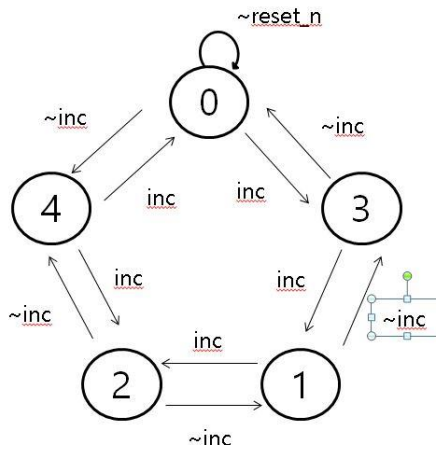
다음과 같은 방식의 입출력 방식이 Stack의 구조이다. 다음으로는 Queue에 관련된 설명이다. Queue는 FIFO의 입출력 방식을 따르고 있다. FIFO란 First In First Out으로 가장 먼저 들어간 자료는 가장 먼저 나와야 한다는 규칙을 따른다. 이미지를 보면 다음과 같이 나타낼 수 있다.



그림은 FIFO방식의 입출력 방식을 쉽게 나타낸 것이다. 이런 Stack과 Queue의 방식은 컴퓨터 상에서 많이 응용되고 있는 자료구조이다.

3. 설계 세부사항

ㄱ. Cnt5

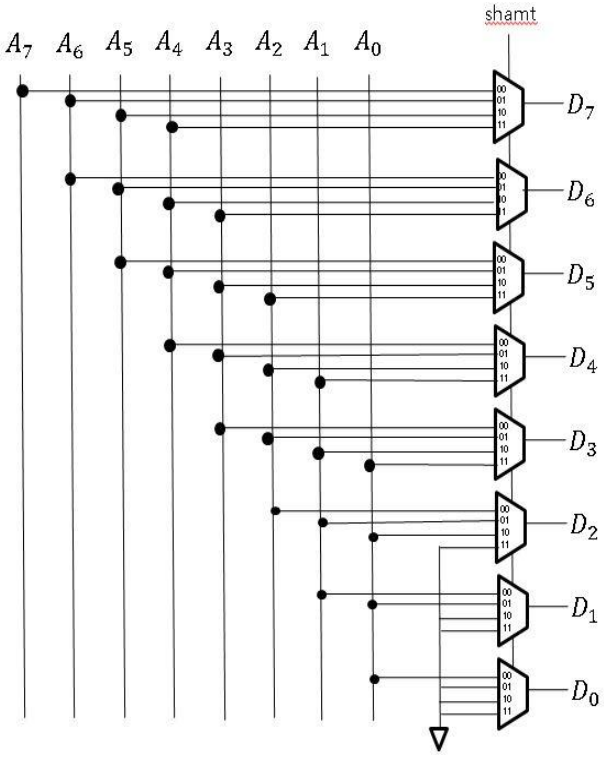


먼저 구현한 Top module은 Cnt5에 관련한 내용입니다. Cnt5의 회로도는 위 원리에서 언급한 회로도와 동일합니다. Inc의 값에 따라 가산이 될지 감산이 될지 결정됩니다. 해당 Top module을 구현할 때 다른 module을 instance하지 않았으며, parameter를 통해서 값을 넣는 방식을 이용했습니다. 해당 회로는 Clock이 rising edge일 때만, 동작하게 설정하였으며 모든 케이스에 대해서 설정을 해줘야 합니다.

ㄴ. Shifter

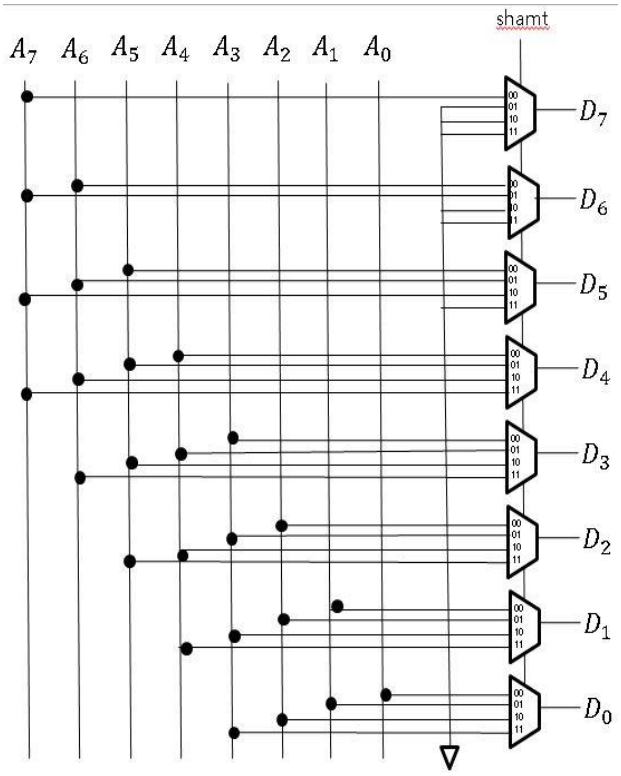
다음은 Shifter입니다. Shifter는 register에 저장된 값을 바탕으로 op code를 통해 동작할 연산을 결정해줍니다. 동작하는 연산에는 NOP, LOAD, LSL, LSR, ASR이 있습니다. 해당 연산의 세부동작은 LSL, LSR, ASR의 세부 설계와 설명다음 진행하겠습니다.

Logical Shift Left



다음 이미지는 LSL의 동작을 가능하게 해주는 회로입니다. Shamt은 옮기는 비트의 수를 결정해는 input이며 00일때는 원래의 자리의 수를 선택하며 옮기는 비트가 없게 됩니다. 01일때는 한 칸을 옮기는 것으로 D7은 다음 칸의 정보인 A6를 채택하게 됩니다. 또한 하위 bit에서는 채택할 숫자가 없는 경우가 있는데 그런 경우에는 0으로 bit를 채우게 됩니다.

Logical Shift Right

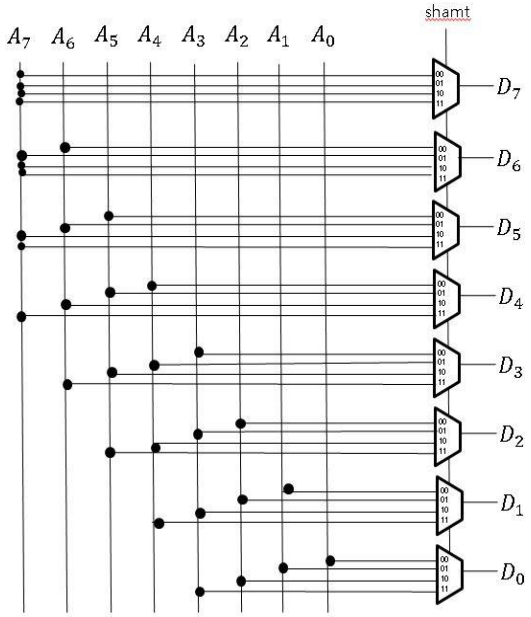


다음 이미지는 LSR의 동작을 하기 위한 회로입니다. LSL과 큰 차이는 없는 회로입니다. 다만 bit의 정보가 0으로 채워지는 경우의 수가 하위 비트에서 상위 비트로 바뀌었기 때문에 GND가 D7부터 걸려있는 것을 확인 할 수 있습니다. 동작의 이해는 LSL과 다르지 않습니다.

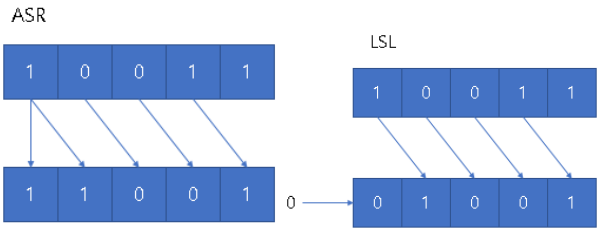
Original		LSR		LSL	
2진수	10진수	2진수	10진수	2진수	10진수
01010110	86	00101011	43	10101100	172
10101011	171	01010101	85	01010110	86

다음 진리표는 LSL과 LSR에 관련된 동작을 간단하게 나타낸 표입니다. Original의 값이 처음 받게 되는 Input에 값이며 op code에 따라 동작을 하면 다음과 같은 값이 나타나게 됩니다. Shamat의 값에 따라 이동하는 amount가 다르지만 해당 표는 shamat의 값을 01로 놓고 동작했을 때 값입니다.

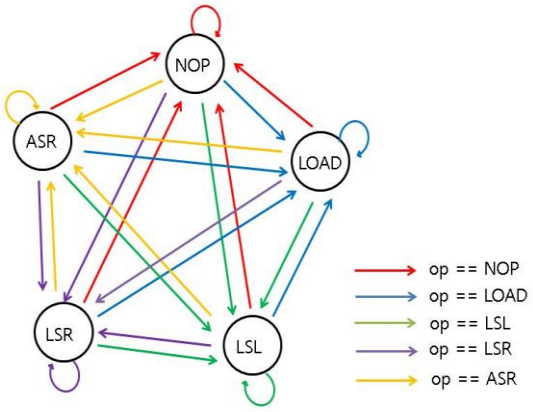
Arithmetic Shift Right



산술 Shift로 LSR과 거의 동일하지만 MSB의 값이 0이 아닌 Original MSB의 값을 따른다. ASL은 기존에 LSL과 같은 동작을 하기 때문에 따로 구현하지 않는다. 아래 사진은 ASR의 동작을 간단하게 나타낸 것이다.



위에서 설명한 동작들이 Shifter의 기본적인 동작들이 된다. 그럼 이러한 동작들은 어떻게 구분되어 Shifter가 연산을 하는 것일까? 바로 op code를 통하여 연산하게 될 동작을 결정해준다. 다음 그림은 Op code의 관련된 FSM을 나타낸 것이다. 해당 그림은 복잡해 보이지만 정말 단순한 그림이다. 해당 명령에서 다른 명령을 받으면 다른 절차 없이 받은 명령으로 바로 이동하는 것이 보인다.



다음으로는 Shifter를 구현하기 위한 과정과 필요한 module들을 정리한 표이다.

Signal	Description
Reset_n	low에서 작동하는 reset signal이다. regeister 내부의 값을 0으로 초기화 시킨다.
op	Opcode를 의미하는 데, 총 5가지의 명령어를 가지게 된다. 1. NOP : Not operation 현재 register에 저장되어 있는 값을 그대로 출력한다. 2. LOAD: 입력된 data 출력 3. LSL : Logical Shift Left를 실행 4. LSR : Logical Shift Right을 실행 5. ASR : Arithmetic Shift Right을 실행.
Shamt	Shift Amount의 약자로 몇 칸을 움직일지 설정. (2-bit)

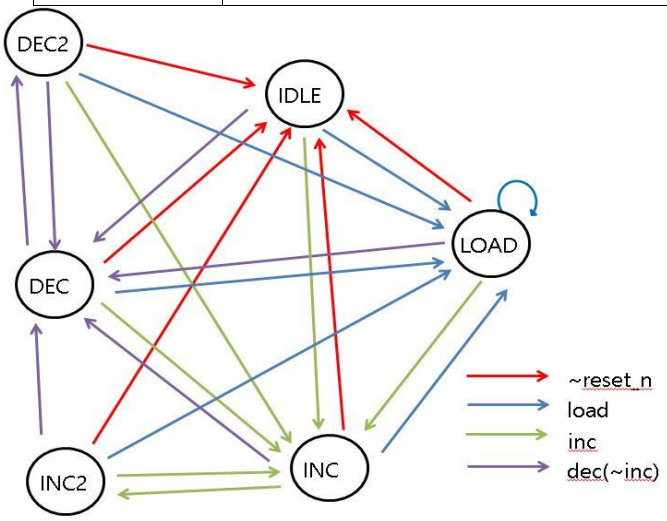
구분	이름	설명
Top module	shifter8	8-bit loader shiftsr's top module
Sub module1	LSL8	8-bit logical shift left
	LSR8	8-bit logical shift right
	ASR8	8-bit arithmetic shift right
	mx4	4-to-1 multiplexer.

Module Name	구분	이름	비트수	설명
shifter8	input	clk	1	clock
		reset_n	1	Reset that operate at the active low. It make the register to 0.
		op	3	Opcode. NOP, LOAD, LSL, LSR, ASR 중 1개를 고르게 함.
		shamt	2	shift amount
		d_in	8	op에서 LOAD 작동, 값 register에 대입
	output	d_out	8	Register 값 출력.
LSL8LSR8 ASR8	input	d_in	8	Data in
		shamt	2	shift amount
	output	d_out	8	Data out
mx4	input	d0	1	first data of multiplexer
		d1	1	second data of multiplexer
		d2	1	third data of multiplexer
		d3	1	fourth data of multiplexer
		s	1	selection code for multiplexer
	output	y	1	selection result

ㄷ. Cnt8

다음으로 설계할 module은 Cnt8이다. 위에서 설명한 Counter보다는 조금 더 복잡하지만 결론적으로는 동일한 동작을 한다. Clock에 의해 수가 증가하거나 감소하며 구현하게 될 Counter는 reset_n과 load, inc의 신호를 받으며 output을 결정한다.

Signal	Description
reset_n	Active low에 동작하는 reset. register값 0으로 초기화
load	입력된 data를 regiser값으로 load
inc	Counter의 증가, 감소를 제어하는 신호, 1일 경우 가산, 0일 경우 감산



해당 Counter는 원리에서 설명한 Moore FSM에 기반을 두고 있다. 해당 Clock이 edge할 때 마다 다른 Input(reset_n, load, inc)에 의해 상태가 변하며 그 상태에 맞는 값을 output으로 출력하게 된다. 상태에 대한 Diagram은 다음과 같이 표현할 수 있다.

해당 하드웨어 동작 중 reset의 영향을 받게 되면 IDLE상태로 가게 되며 output의 값은 0으로 초기화 된다. 또한 Load의 값을 받게 되면 미리 레지스터에 입력해 놨던 값이 불리면서 output의 값이 변하게 된다. Inc의 값에 의해 순차적으로 감산이 될지 가산이 될지는 미리 설계한 Cnt5와 동일한 역할을 한다.

아래 표는 해당 Top module을 구성하기 위한 sub module과 설명이다.

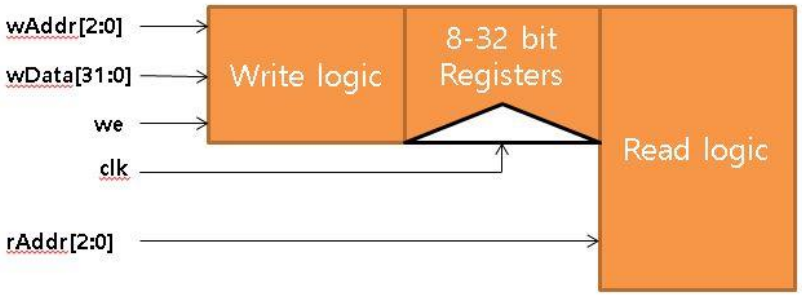
구분	이름	설명
Top module	cntr8	8-bit loadable up/down counter's top module
Sub module	cla8	8-bit carry look-ahead adder

Module Name	구분	이름	비트수	설명
cntr8	input	clk	1	clock
		reset_n	1	Reset that operate at the active low. It make the register to 0.
		load	1	입력으로 들어와 있는 data를 count
		inc	1	1일 경우 증가, 0일 때 감소
		d_in	8	load일 시, data입력
	output	d_out	8	count된 결과.
		o_state	3	현재 state값 출력
cla8	input	a	8	cla's input a
		b	8	cla's input b
		ci	8	cla's carry in
	output	s	8	cla's output s
		co	1	cla's output carry out

cla8	input	a	8	cla's input a
		b	8	cla's input b
		ci	8	cla's carry in
	output	s	8	cla's output s
		co	1	cla's output carry out

ㄹ. Register File

마지막으로 설계하는 module은 Register File입니다.



그림은 Register File의 기본적인 동작을 설명하기 위한 이미지입니다. 주소와 데이터를 write logic에 넘겨주고, register에 clock을 넣어 준 후, 읽을 주소를 넘겨주면 값이 Read logic을 통하여 반환되는 것을 확인 할 수 있습니다. 해당 모듈의 실질적인 동작은 다음 항목인 실험결과에서 조금 더 자세하게 다루겠습니다. 아래표는 해당 module을 구현하기 위한 Sub module과 설명입니다.

구분	이름	설명
Top module	register_file	top module
Sub module	register32_8	8개의 32bit register의 module
	write_operation	write address decode module
	read_operation	Select register using read address

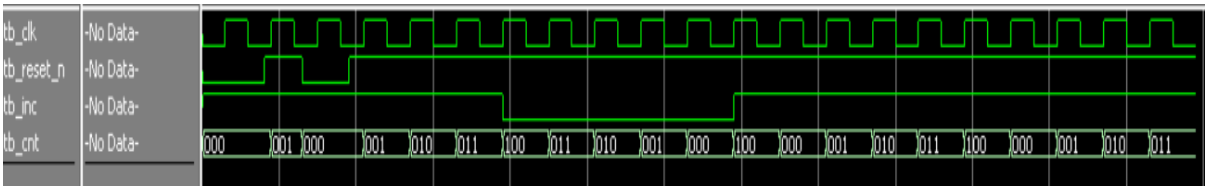
Module 이름	구분	이름	비트 수	설명
register_file	input	clk	1	clock
		reset_n	1	reset which operating at active low register goes to 0
		we	1	write enable
		wAddr	3	write address
		rAddr	.3	read address
		wData	32	write data
	output	rData	32	read data
register32_8	input	clk	1	clock
		reset_n	1	reset which operating at active low

				register goes to 0
		en	1	register enable
		d_in	32	Data input
	output	d_out0~7	32	Register Data output
write _operation	input	we	1	Write enable
		Addr	3	Write Address
	output	to_reg	8	Selected register enable signal
read _operation	input	from_reg0~7	32	8 registers` data
		addr	3	Read address
	output	Data	32	Data output

4. 설계 검증 및 실험 결과

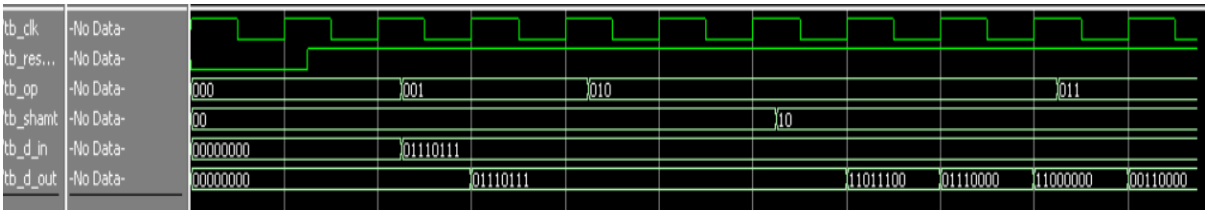
A. 시뮬레이션 결과

(1) Cnt5



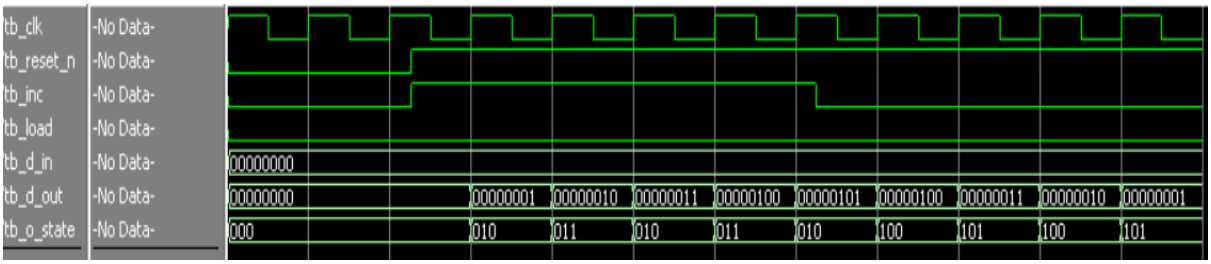
Reset의 동작을 검증하기 위하여 앞에서 reset의 값을 두 번 변경한 모습을 볼 수 있습니다. 또한 inc의 값이 1이면 가산이 되고 0이면 감산이 되는 모습을 확인 할 수 있습니다. 숫자의 표현이 4까지 있고 Ring Counter의 방식이므로 4다음 값은 0이 나오는 것도 확인 할 수 있습니다.

(2) Shifter8

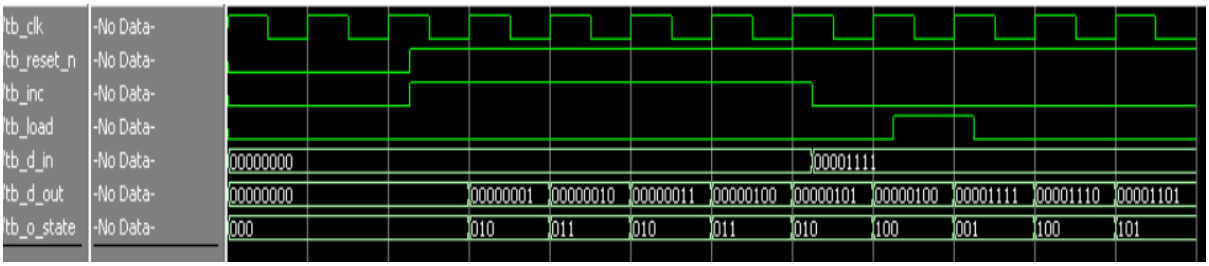


해당 wave form은 pdf의 첫번째 wave form을 응용하여 코드로 작성하여 사용한 것입니다. Op code 001은 Load명령으로 data in된 값을 가져와 연산하기 시작합니다. 010은 LSL의 명령으로 값을 왼쪽으로 shift해주는 명령입니다. Wave form을 보면 amount가 0이므로 동작을 하지 않고 있다가 10으로 바뀌어 2 bit를 이동시키는 연산을 하는 것을 확인 할 수 있습니다. 또한 op code가 011로 바뀌면 LSR동작을 합니다. 결과 값을 보면 정상적으로 동작하는 것을 확인할 수 있습니다.

(3) Cntr8

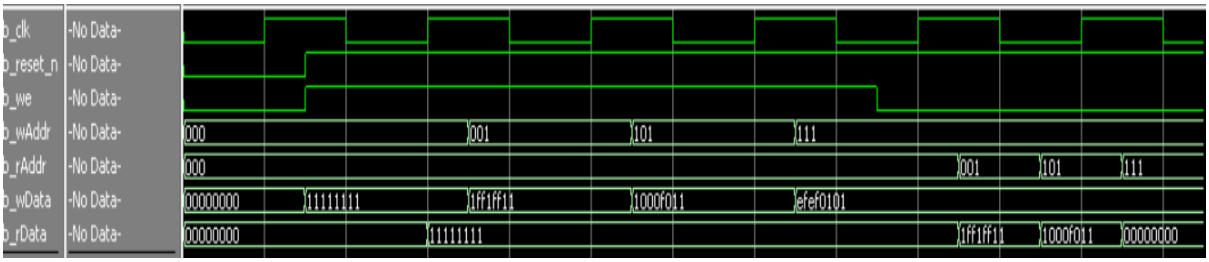


다음 그림은 Load가 되지 않은 Counter에 대한 설명입니다. 처음에는 reset의 동작으로 count가 되지 않지만 reset의 값이 1로 rising하고 순차적으로 count되는 모습을 확인할 수 있습니다. 또한 inc의 값에 따라 감산과 가산이 결정되기 때문에 정상적으로 동작하고 있는 것을 확인 할 수 있습니다.



같은 module로 load가 활성화되었을 때의 연산입니다. Data in의 값이 바뀌어도 load 명령이 없으면 값은 load되지 않습니다. sate또한 load가 되는 순간 001로 바뀌는 것을 확인할 수 있으며 load가 0이되어도 load된 값은 inc의 값에 따라 순차적으로 값이 변하는 것을 확인 할 수 있습니다.

(4) Register file

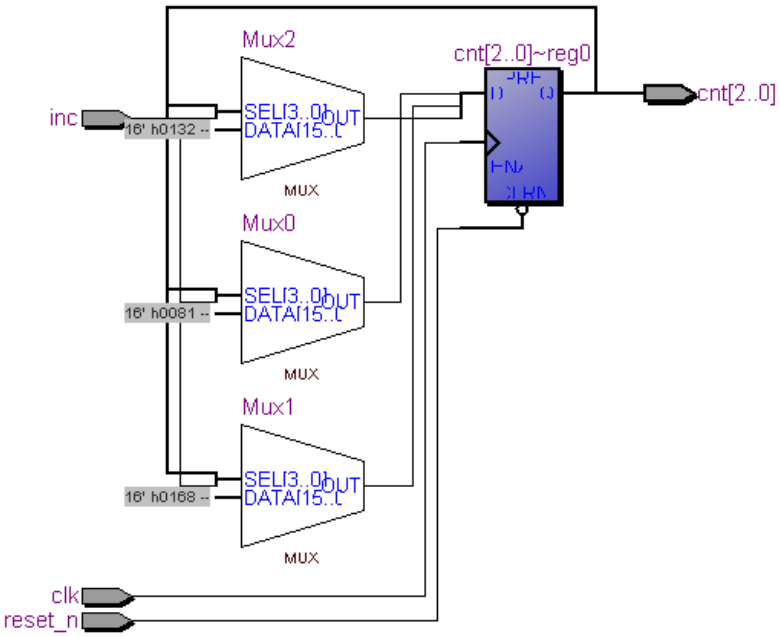


해당 wave form은 원하는 값을 원하는 주소에 저장하고 필요에 따라 해당 주소를 읽어 값을 불러오는 것을 잘 나타내고 있다. We는 Write Enable로 해당 input이 1이 되어야 write를 실행할 수 있다. 만약 도중에 we가 0이 된다면 다음 저장될 값은 저장되지 않아 값을 불러오지 못한다. Write address와 write data를 통하여 앞에서 저장하고 뒤에서는 read address를 통하여 저장된 값을 불러오는 것을 확인 할 수 있다. 111주소의 값은 we가 clk의 rising edge전에 0으로 down되었기 때문에 efef0101의 값을 저장하지 못하여 111의 주소에서 값을 불러올 때 해당 결과와 같은 값이 반환된 것이다.

B. 합성(synthesis) 결과

(1) Cnt5

RTL Viewer

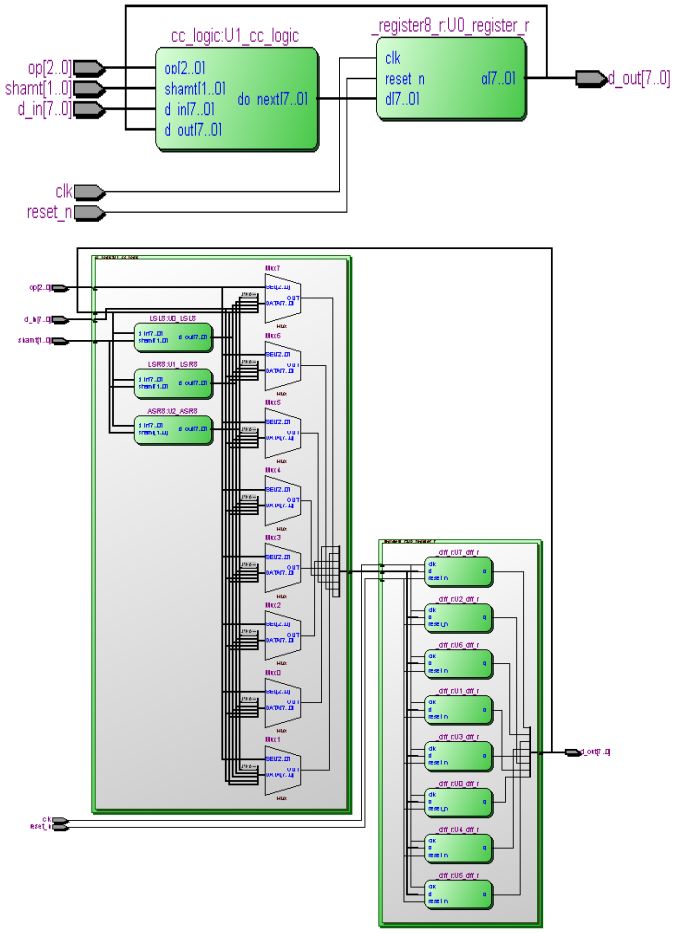


Flow summary

Flow Summary	
Flow Status	Successful - Sun Oct 07 18:27:29 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	cnt5
Top-level Entity Name	cnt5
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	3 / 68,416 (< 1 %)
Total combinational functions	3 / 68,416 (< 1 %)
Dedicated logic registers	3 / 68,416 (< 1 %)
Total registers	3
Total pins	6 / 622 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

(2) Shifter8

RTL Viewer

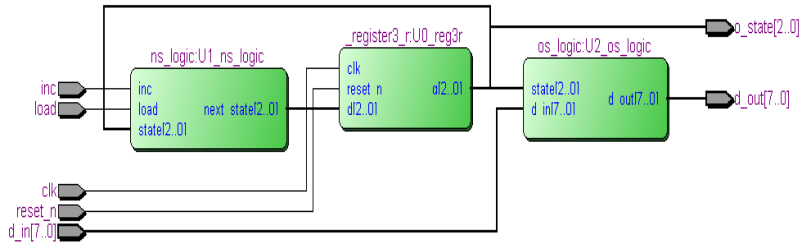


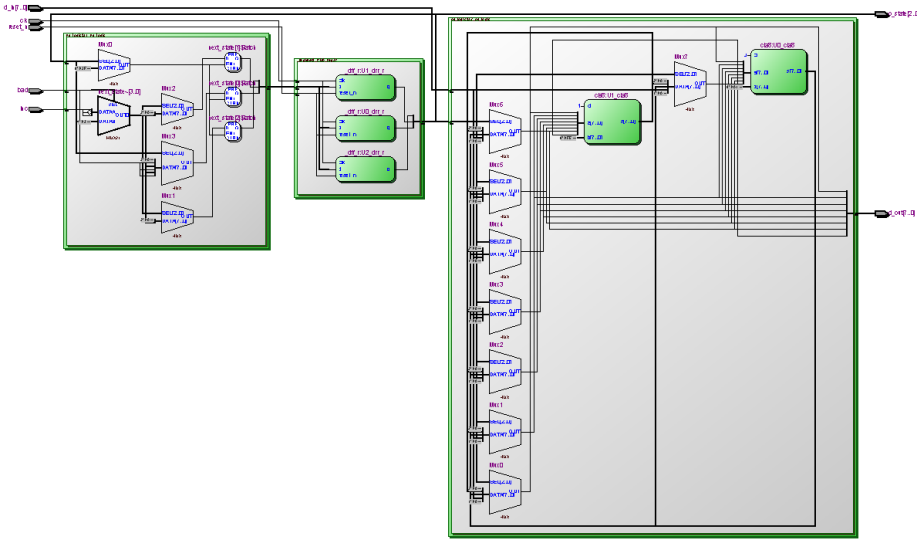
Flow summary

Flow Summary	
Flow Status	Successful - Sun Oct 07 18:29:14 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	shifter8
Top-level Entity Name	shifter8
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	52 / 68,416 (< 1 %)
Total combinational functions	52 / 68,416 (< 1 %)
Dedicated logic registers	8 / 68,416 (< 1 %)
Total registers	8
Total pins	23 / 622 (4 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

(3) Cntr8

RTL Viewer



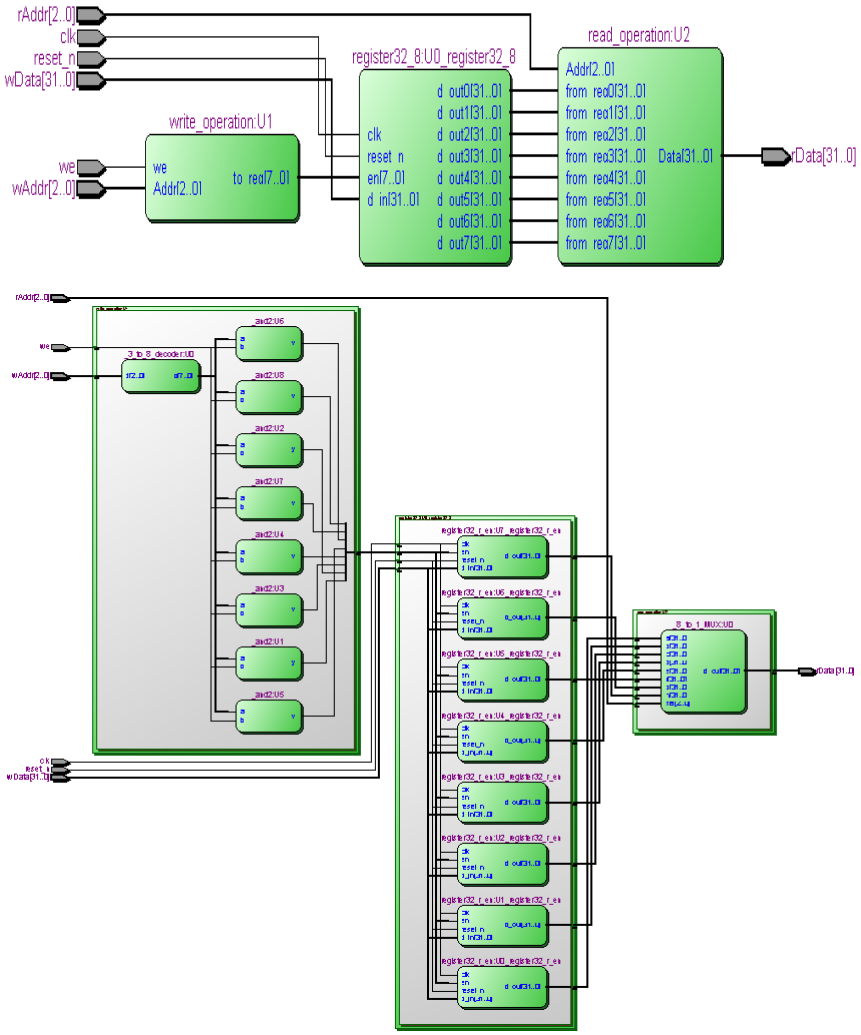


Flow summary

Flow Summary	
Flow Status	Successful - Sun Oct 07 18:30:56 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	cntr8
Top-level Entity Name	cntr8
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	33 / 68,416 (< 1 %)
Total combinational functions	33 / 68,416 (< 1 %)
Dedicated logic registers	3 / 68,416 (< 1 %)
Total registers	3
Total pins	23 / 622 (4 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

(4) Register File

RTL Viewer



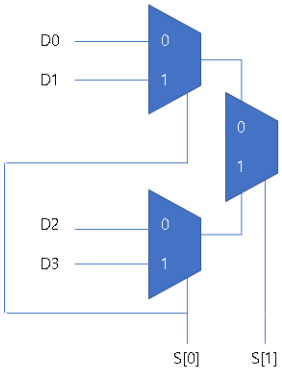
Flow summary

Flow Summary	
Flow Status	Successful - Sun Oct 07 18:33:17 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	Register_file
Top-level Entity Name	Register_file
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	296 / 68,416 (< 1 %)
Total combinational functions	168 / 68,416 (< 1 %)
Dedicated logic registers	256 / 68,416 (< 1 %)
Total registers	256
Total pins	73 / 622 (12 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

이번 실습은 Counter, Shifter, Register file을 설계하는데 목적을 두었습니다. 설계 중 문제점으로는 Test bench를 통해 Shift는 하고자 하는 amount 즉, Shmat의 값을 01로 설정하여 이동시키고자 하는 bit의 수를 1개로 설정했습니다. 하지만 wave form을 확인한 결과 2개의 bit가 이동하는 문제가 발생하였고 Shmat에 문제가 있다고 판단한 저는 Shift의 양을 결정하는 Mux에서 문제점을 발견하게 되었습니다.



다음 그림과 같이 S의 값이 설정이 되어 있어야 정상지만 저의 착각으로 값의 순서를 반대로 넣게 되었습니다. 그 결과 10일때는 1개의 비트가 움직였고 01일때는 두개의 비트가 움직이는 잘못된 결과가 나오게 되었습니다. 그리하여 코드를 수정한 후 wave form을 다시 확인 후 알맞은 결과를 도출할 수 있었습니다.

B. 결론

해당 실습은 Counter와 Shifter, Register file module을 직접 구현하여 테스트하는 실습이었습니다. 구현을 하는 중 Counter의 종류로 loadable counter와 ring counter가 있다는 것을 알게 되었다. Loadable counter는 말 그대로 load가 가능한 counter를 말한다. 구현하였던 cnt5는 inc의 값에 따라 값이 0부터 증가하고 감소하는 동작을 한다. 하지만 load가 가능한 cntr8의 동작을 보면 load의 값이 없으면 기존 counter와 같은 동작을 하지만 load의 값이 들어오면 입력한 값부터 연산이 시작된다. 이런 loadable counter는 우리가 데이터를 고르는 과정이 필요할 때 사용하는 Multiplexer가 없어도 사용할 수 있다는 장점을 가진 회로이다. 그와 동시에 데이터의 증감을 위한 조합회로의 최적화가 가능해서, 회로의 면적을 줄일 수 있다는 장점을 가지고 있다. 이런 장점은 현 시대에서 회로가 복잡해지고 면적이 넓어지는 것을 막을 수 있다고 생각된다. 다음은 Ring Counter이다. Ring Counter는 FSM을 만들기 위해 종종 하드웨어 설계에 사용된다. Binary Counter는 Ring Counter보다 더 복잡하고 bit의 수가 증가함에 따라 시간이 더 걸리는 가산기 회로를 필요로 하지만 Ring Counter의 지연은 코드의 bit수에 관계없이 거의 일정하다는 장점이 있다. 단점으로는 Binary code보다는 낮은 밀도의 Code라는 것이 단점이다. Ring

Counter를 사용할 수 있는 분야로는 지속적으로 순차적인 신호를 필요로 하는(ex. 신호등의 남은 시간 표시)장비에는 필수적이라고 생각됩니다. 다음으로는 Shifter에 대하여 배우게 되었습니다. 특히 새로 알아볼 Shifter는 Barrel Shifter입니다. Barrel Shifter는 순차 논리를 사용하지 않고 지정된 비트 수만큼 데이터 워드를 시프트 할 수 있는 디지털 회로이며 순수 조합 논리만 사용합니다. 이를 구현하는 한가지 방법은 하나의 Mux의 출력이 시프트 거리에 의존하는 방식으로 다음 Mux의 입력에 연결되는 Mux의 시퀀스입니다. 동작으로는 단일 클럭 사이클 내에 n bit가 입력되었을 때 최대 n bit를 이동 및 회전시키는데 사용됩니다. Barrel Shifter를 구현하는데 필요한 mux의 개수로는 다음 조건을 만족합니다. n bit의 정보를 n만큼 이동시키기 위해서는 n $\log_2(n)$만큼의 mux가 필요합니다. 예를 들어 8-bit라고 했을 때 $8 \times \log_2(8) = 8 \times 3 = 24$ 이라는 결과가 나오게 됩니다. 다음은 Barrel Shifter에 관련한 Bandwidth의 대한 설명입니다. Bandwidth란 대역폭이라는 뜻으로 컴퓨팅 언어에서는 "초당 여러 비트로 표현되는, 사용 가능하거나 소비된 정보의 비트레이트"라고 정의가 되어있습니다. 일정 시간을 초당이 아닌 Clock으로 두었을 때 한 Clock에 사용가능한 Bits의 수는 N으로 그 수는 사용되는 bits수에 따라 수시로 바뀔 것이라고 예상됩니다.

6. 참고문헌

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

Barrel Shifter / https://en.wikipedia.org/wiki/Barrel_shifter

Ring Counter / https://en.wikipedia.org/wiki/Ring_counter

Ring Counter / https://terms.naver.com/entry.nhn?docId=589893&cid=42340&categoryId=42340s

Loadable Counter / https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/45-misc/80-dcf77/LoadCount4.html

Bandwidth / https://en.wikipedia.org/wiki/Bandwidth

Bandwidth(computing) / https://en.wikipedia.org/wiki/Bandwidth_(computing)

김영민 / Finite State Machine / 광운대학교 / 2018

김영민 / FIFO/ 광운대학교 / 2018

김영민 / Register Files / 광운대학교 / 2018