지도교수 : 이준환

디지털 논리회로 2

Term Project

Conclusion

COUNTER (FIFO & BUS & TIMER)

광운대학교

컴퓨터공학과

2007720080

백병화

2011년 12월 6일

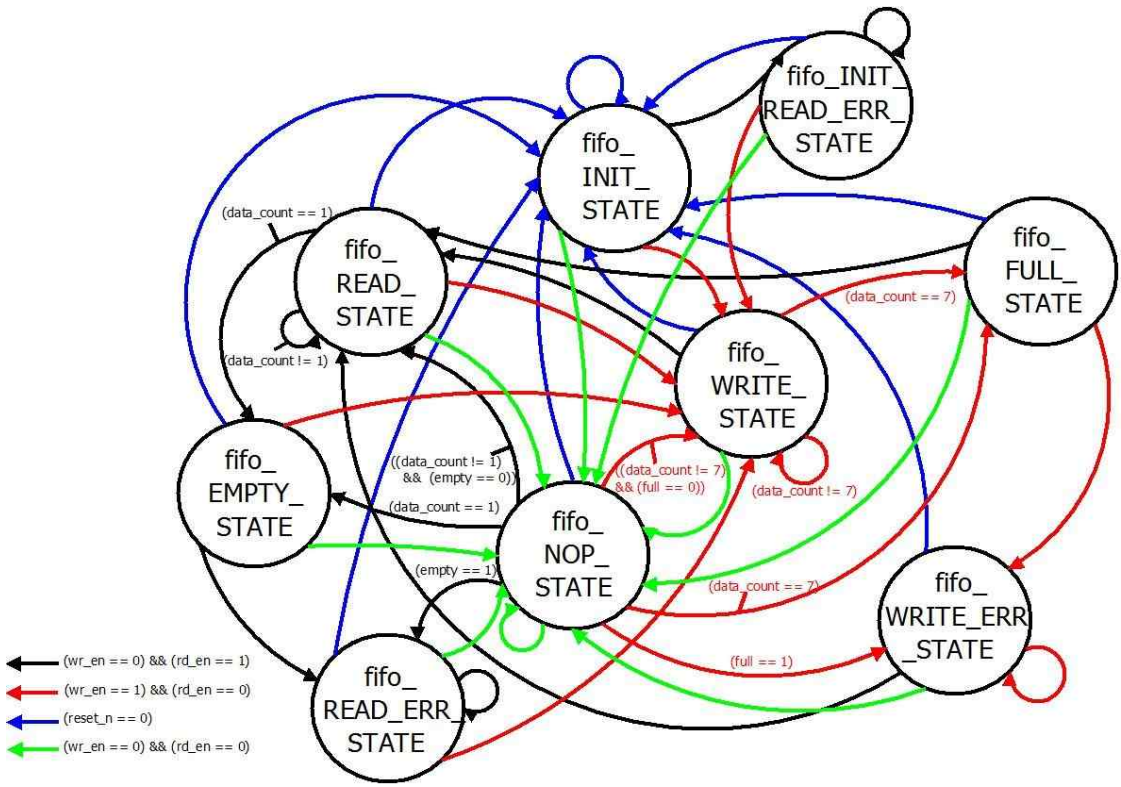
광운대학교 KWANGWOON UNIVERSITY

차 례.

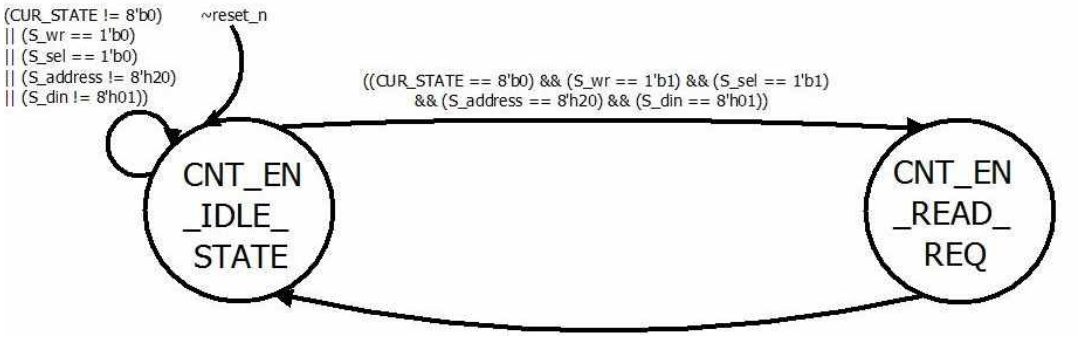
1. 그림 목차.

1-1. Schedule. - (4-1. Project 구현 일정 및 계획 참고 그림)

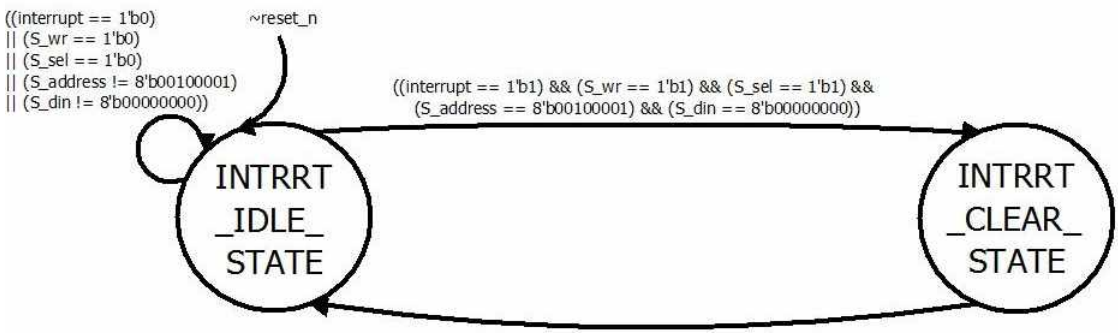
1-2. State Transition Diagram - Synchronous FIFO



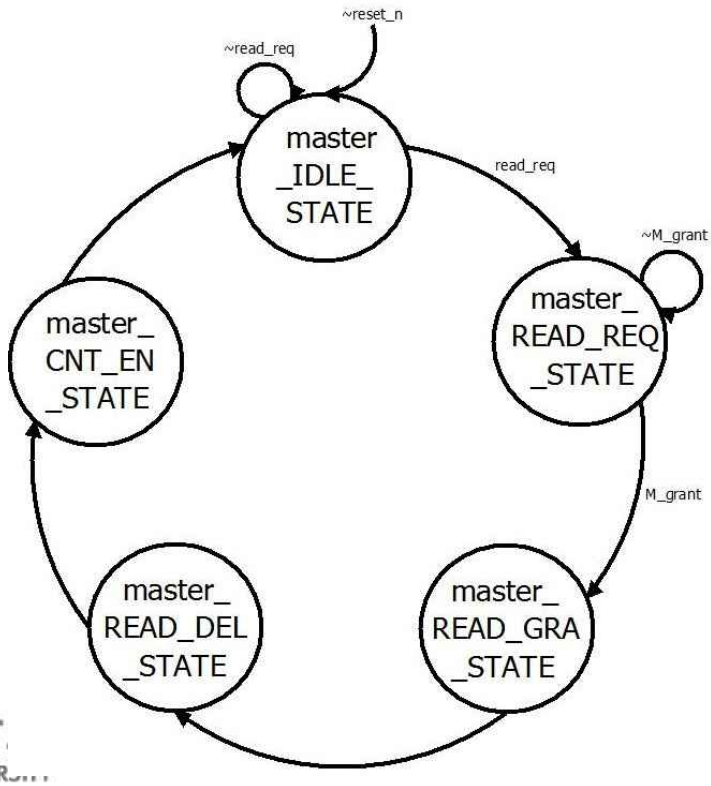
1-3. State Transition Diagram - TIMER - module이름 : timer_reg - CNT_EN_STATE



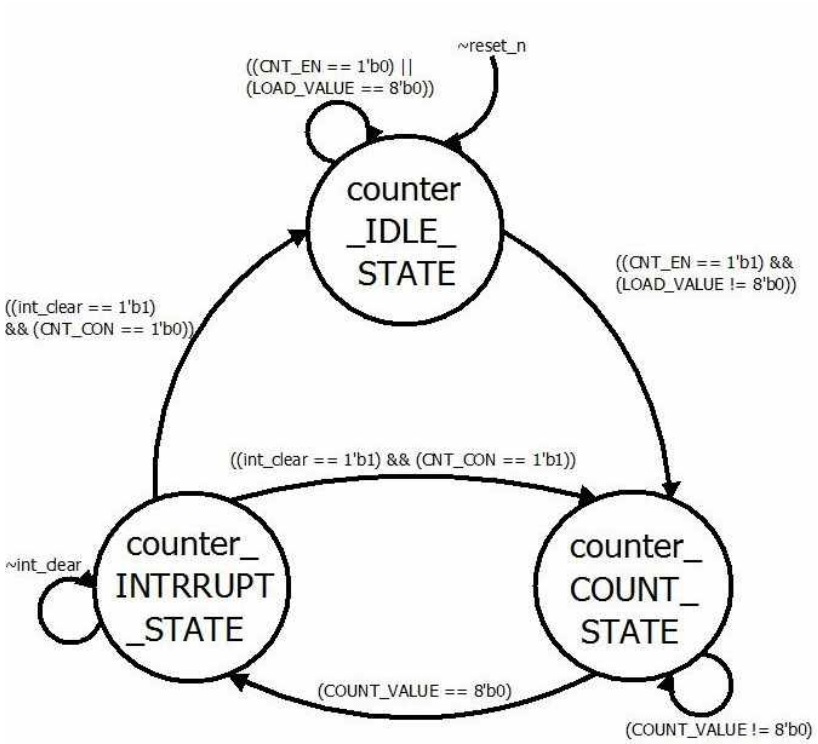
1-4. State Transition Diagram - TIMER - module이름 : timer_reg - INTRRT_STATE



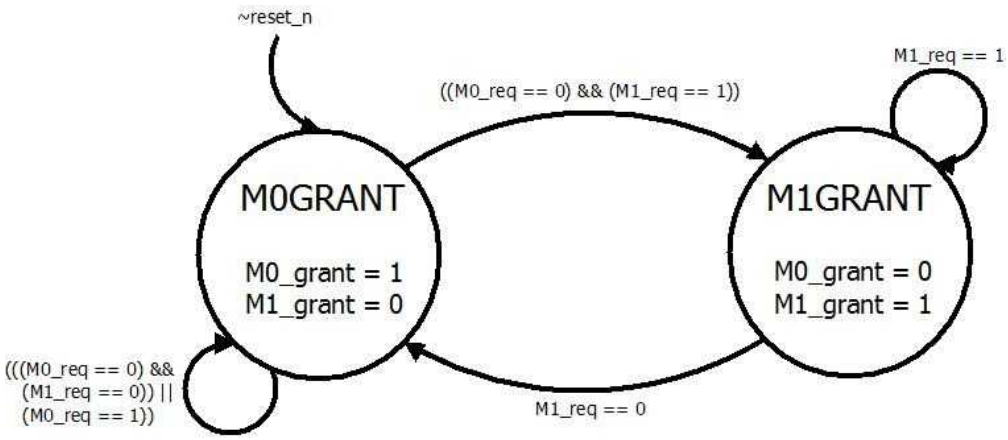
1-5. State Transition Diagram - TIMER - module이름 : timer_master - master_state



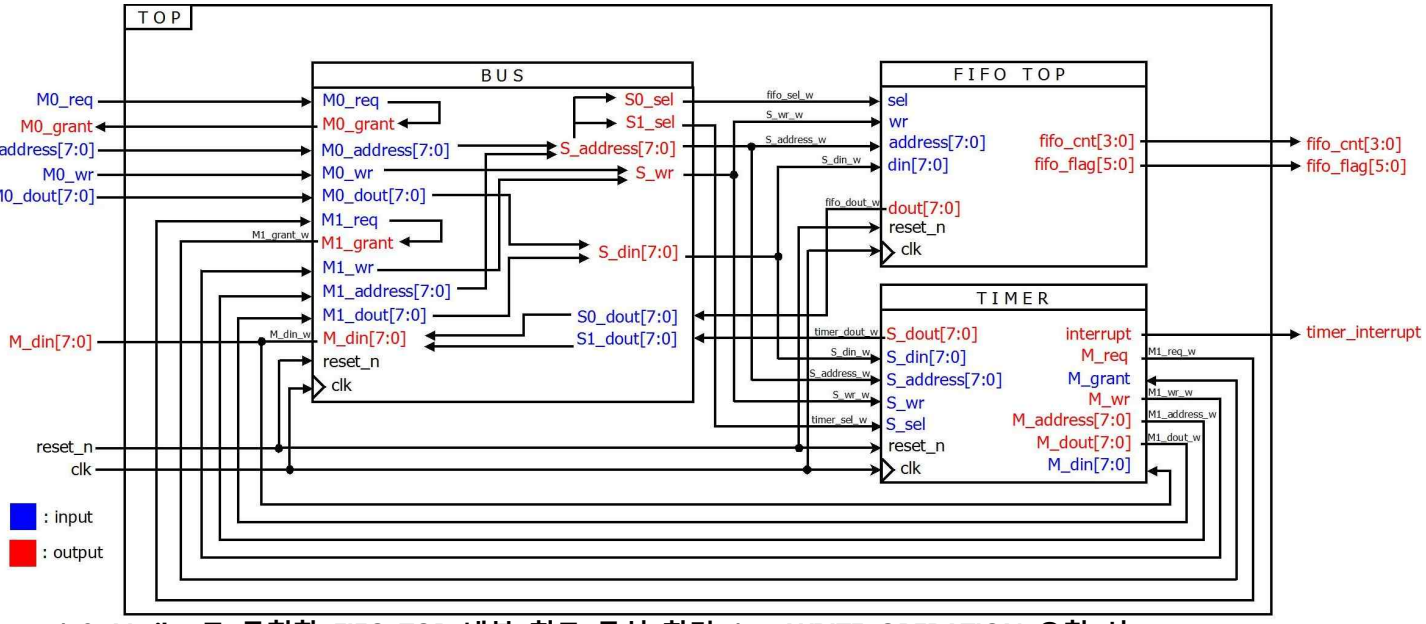
1-6. State Transition Diagram - TIMER - module이름 : timer_counter - counter_state



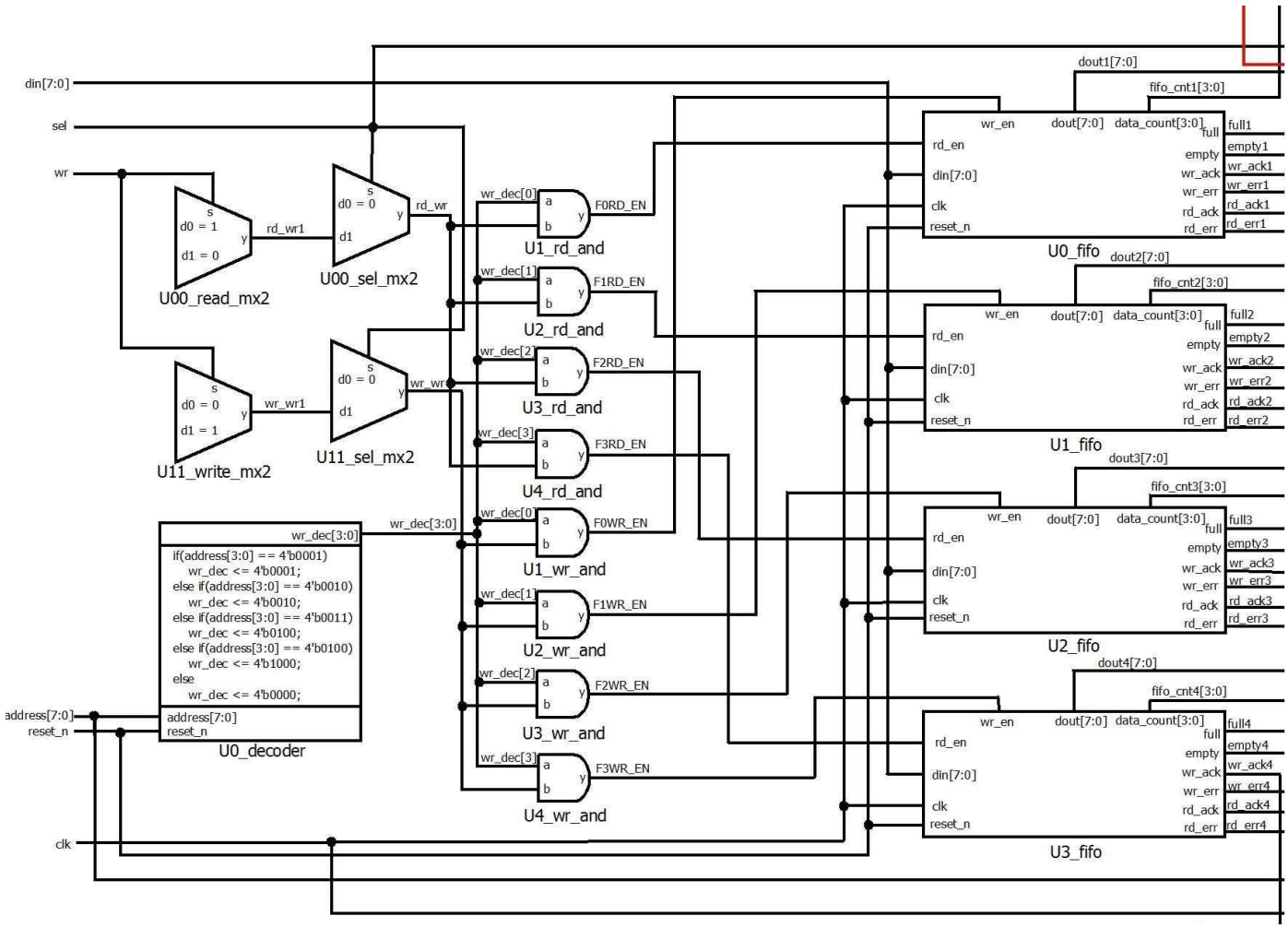
1-7. State Transition Diagram - BUS



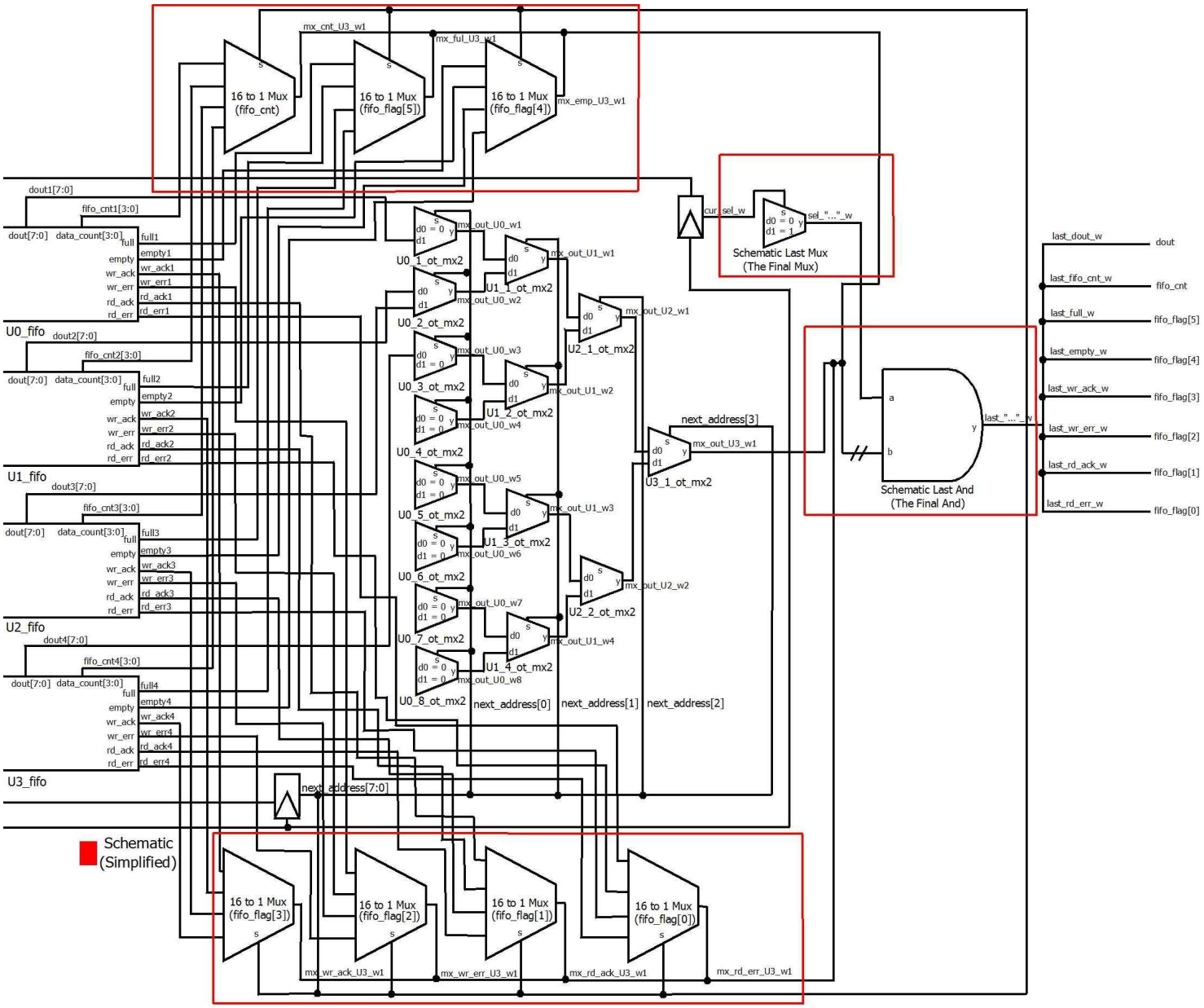
1-8. Schematic symbol of TOP & bus의 간략한 내부 system 구성 화면.



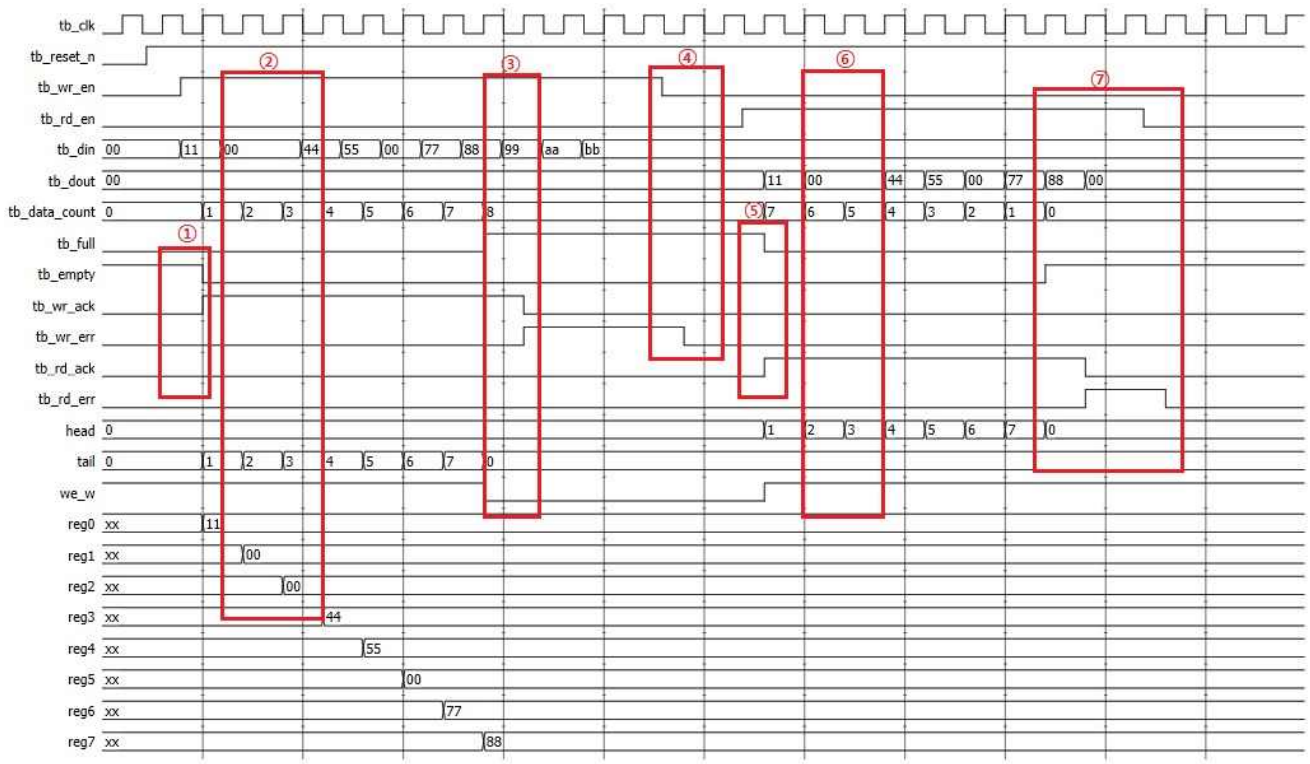
1-9. Verilog로 구현한 FIFO_TOP 내부 회로 구성 화면 1. - WRITE OPERATION 요청 시
(회로가 커서 WRITE와 READ로 나눠서 삽입하였음.)



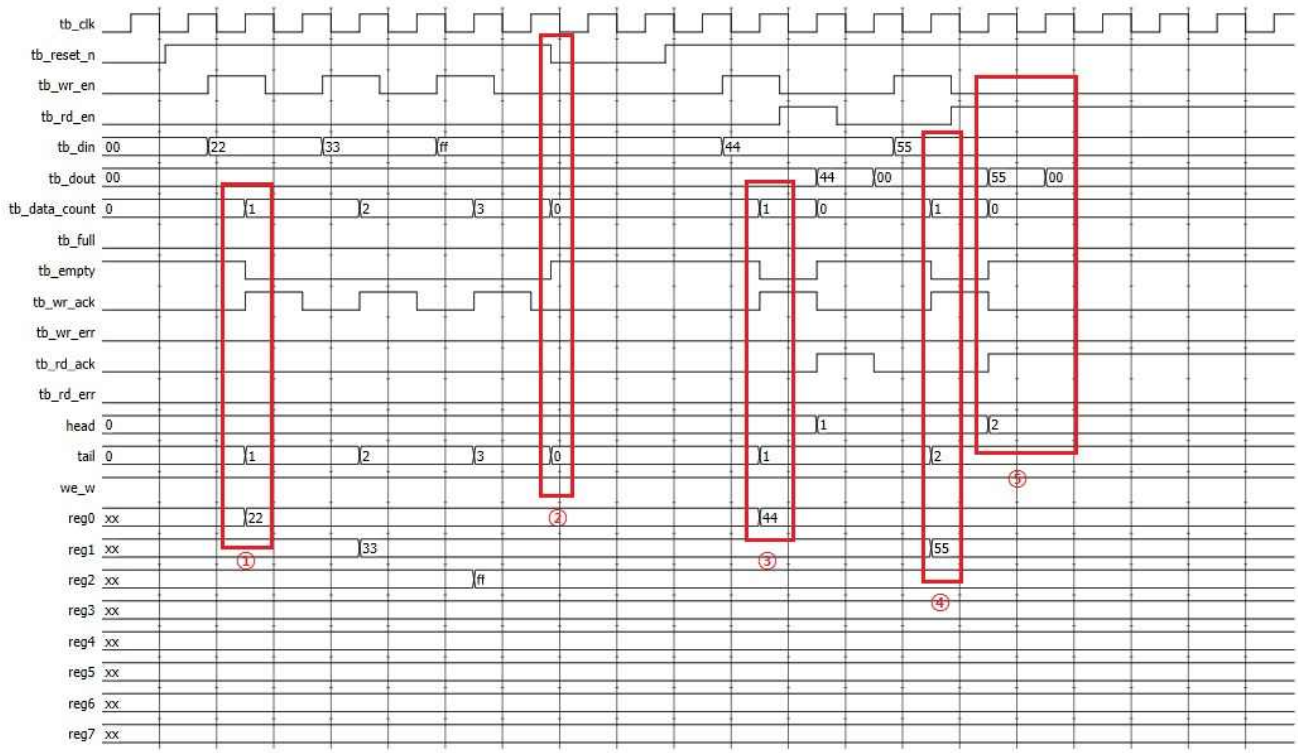
1-10. Verilog로 구현한 FIFO_TOP 내부 회로 구성 화면 2. - READ OPERATION 요청 시
(회로가 커서 WRITE와 READ로 나눠서 삽입하였음.)



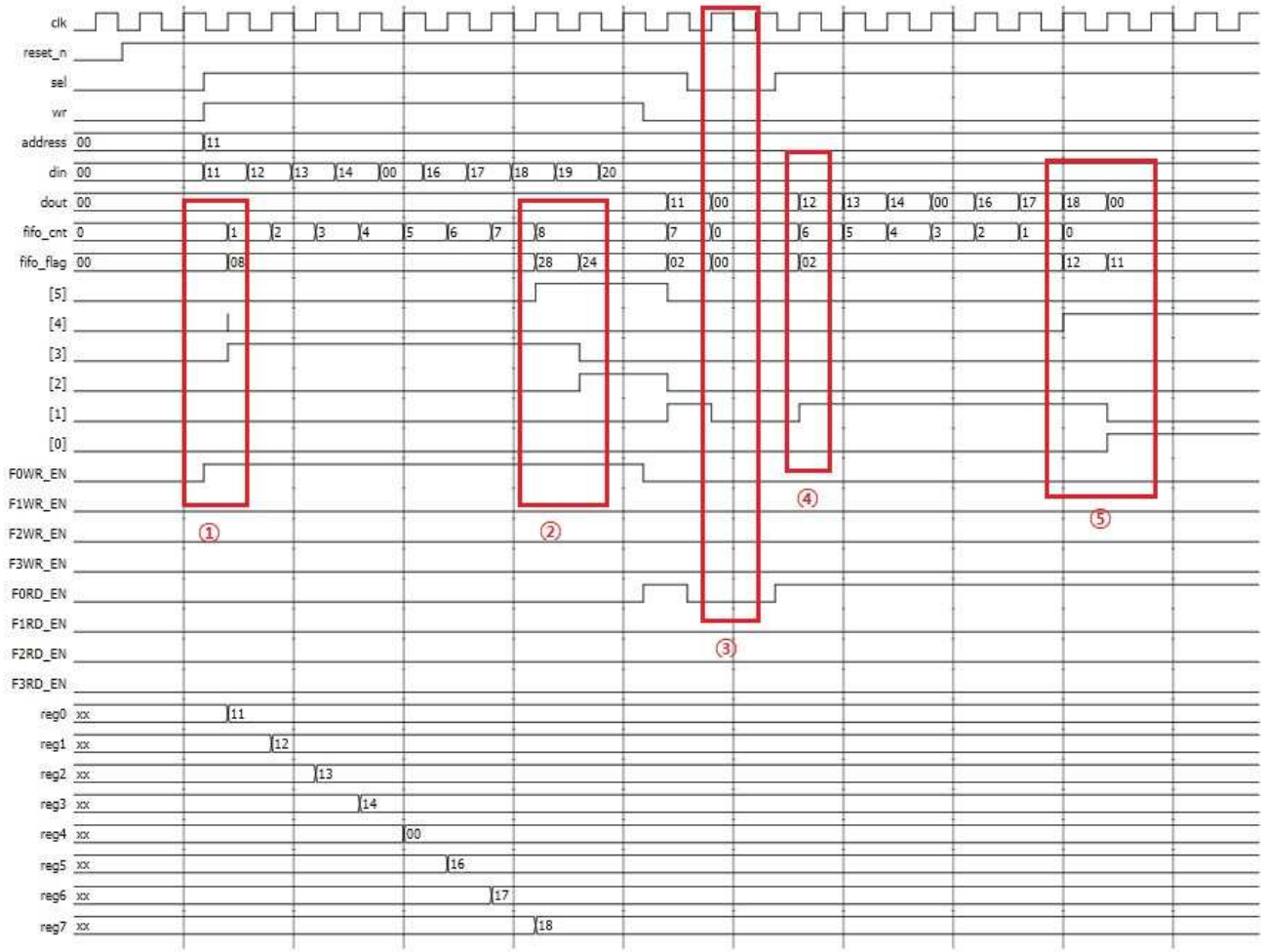
1-11. Design Verification (fifo 1)



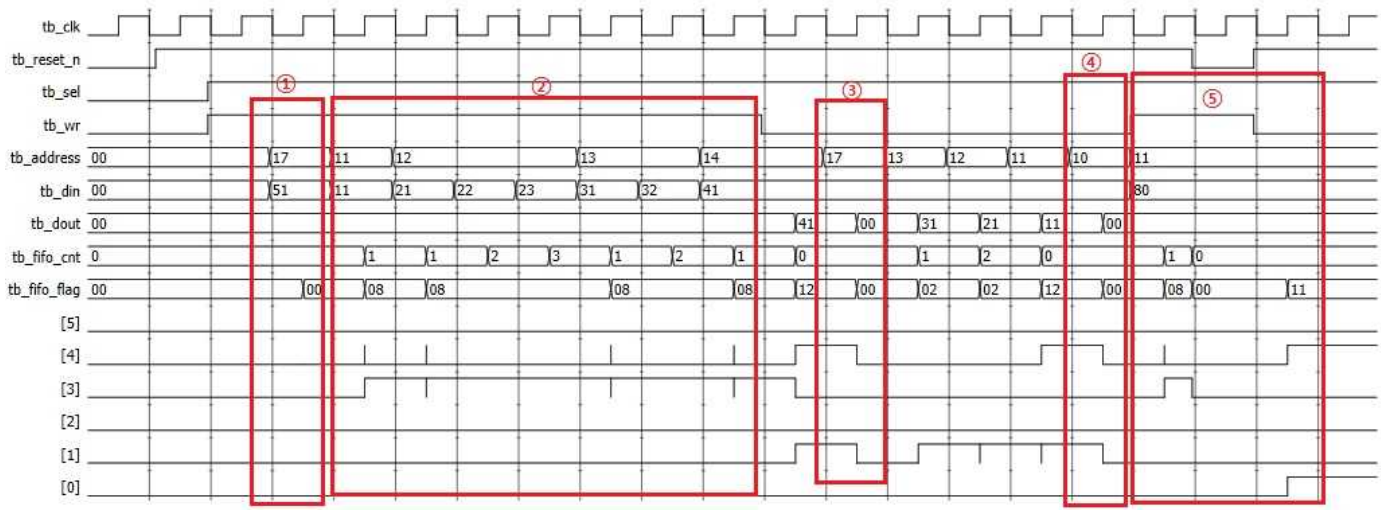
1-12. Design Verification (fifo 2)



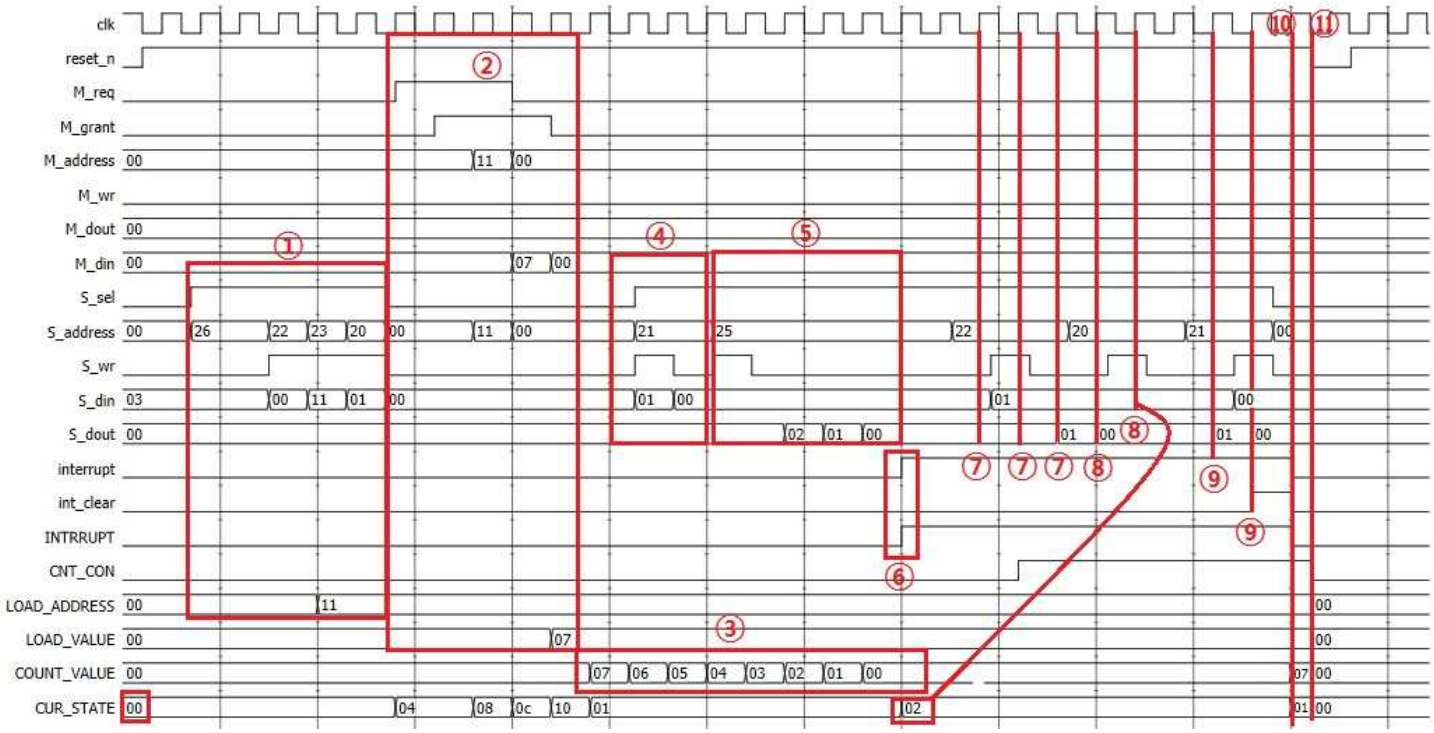
1-13. Design Verification (fifo_top 1)



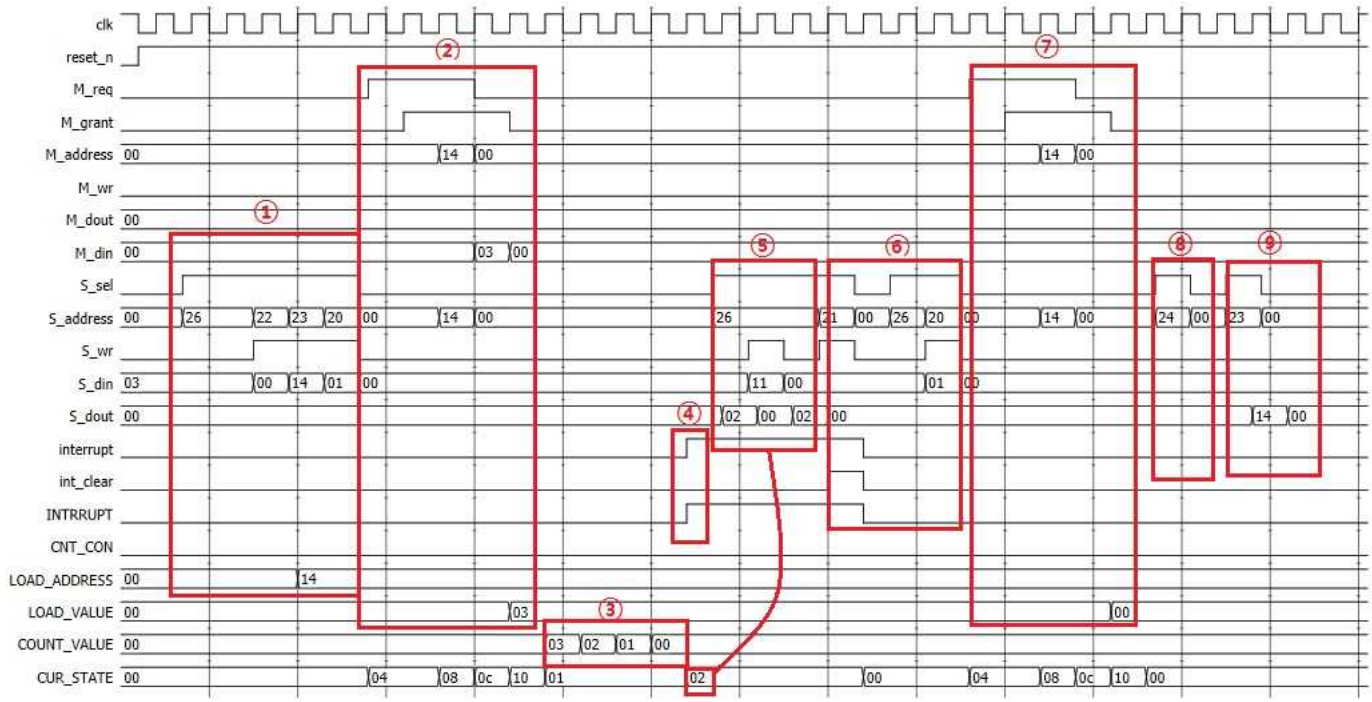
1-14. Design Verification (fifo_top 2)



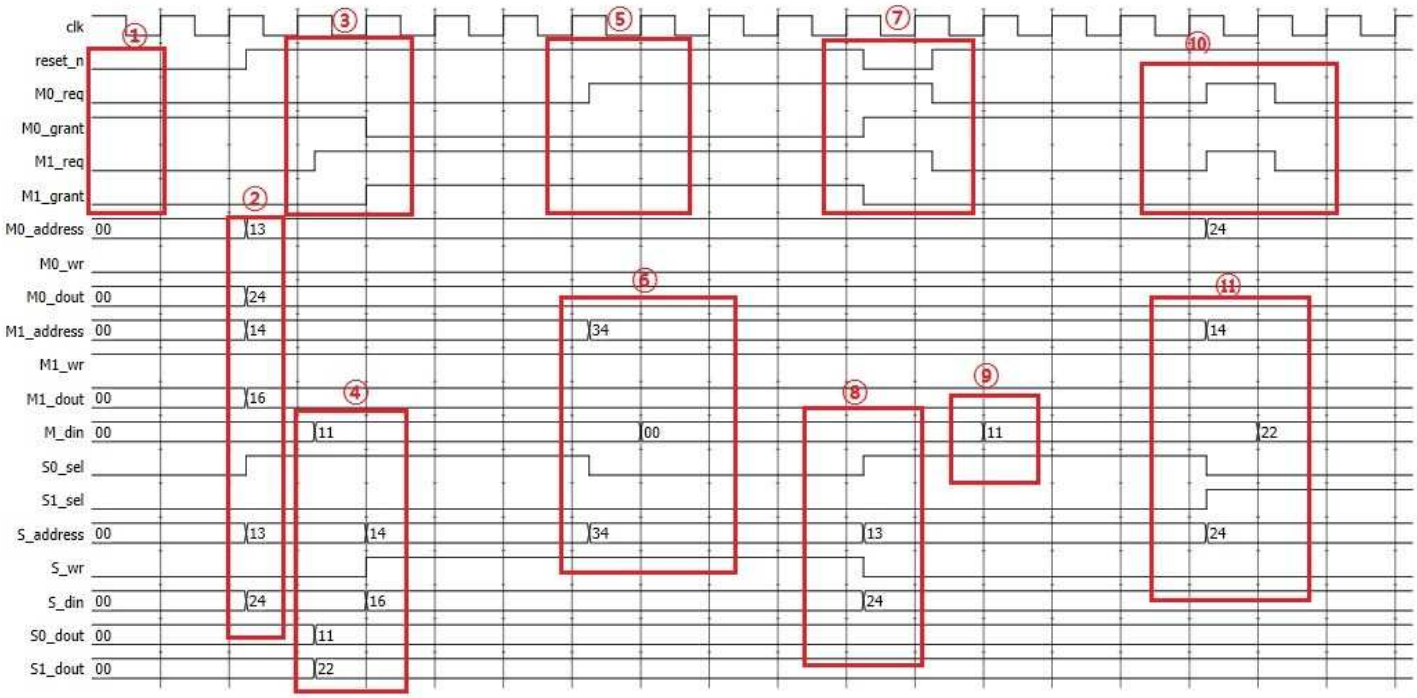
1-15. Design Verification (timer 1)



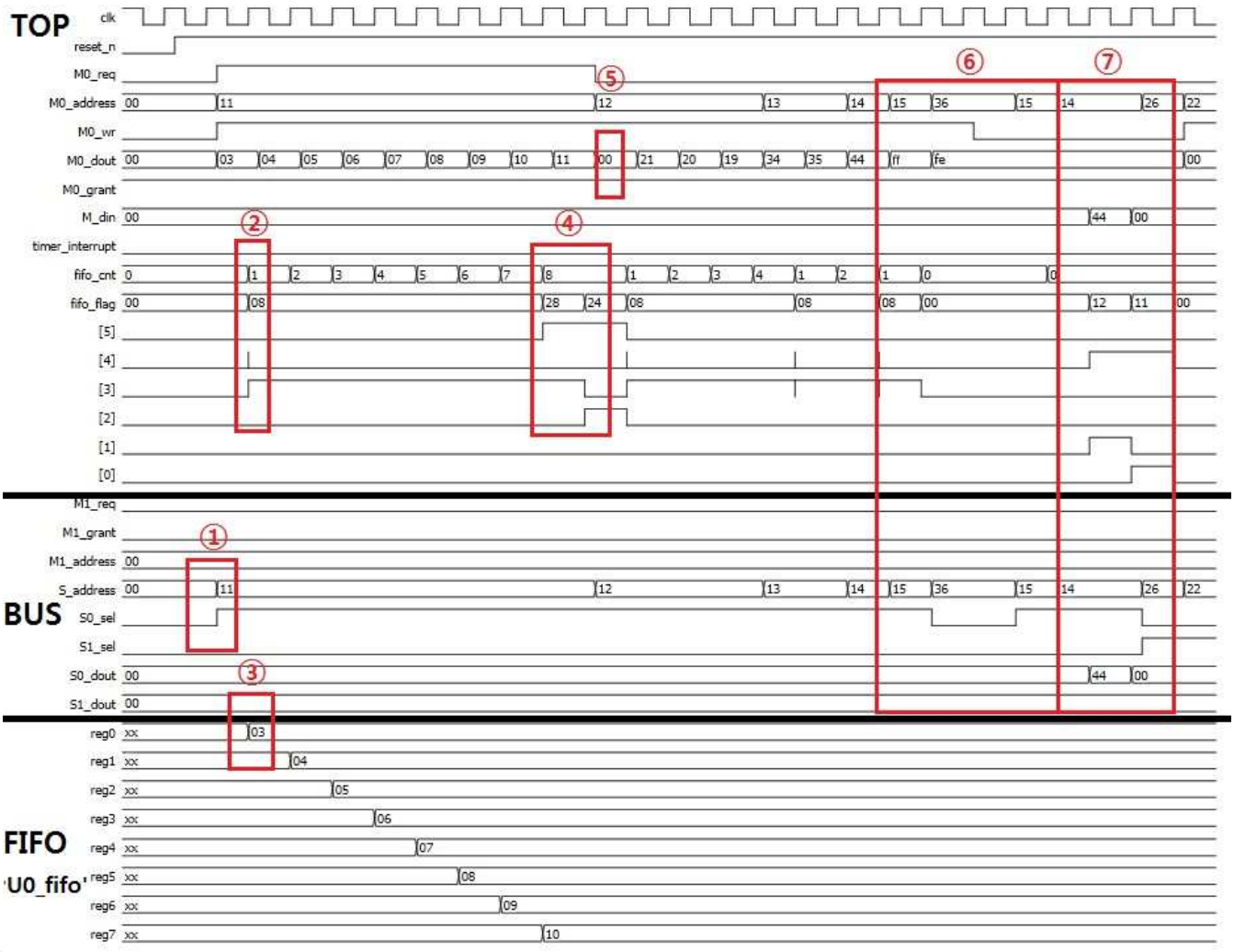
1-16. Design Verification (timer 2)



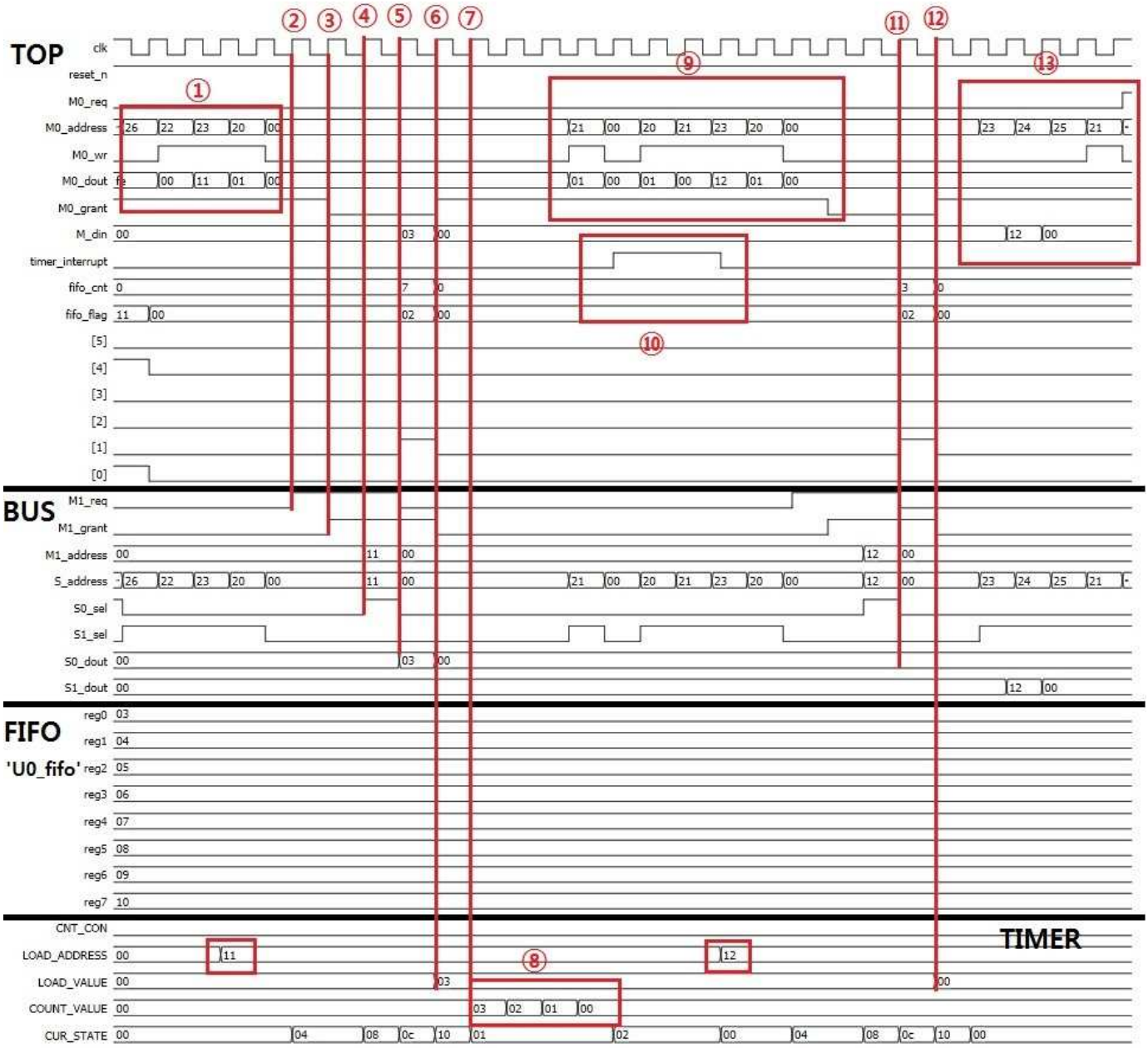
1-17. Design Verification (bus)



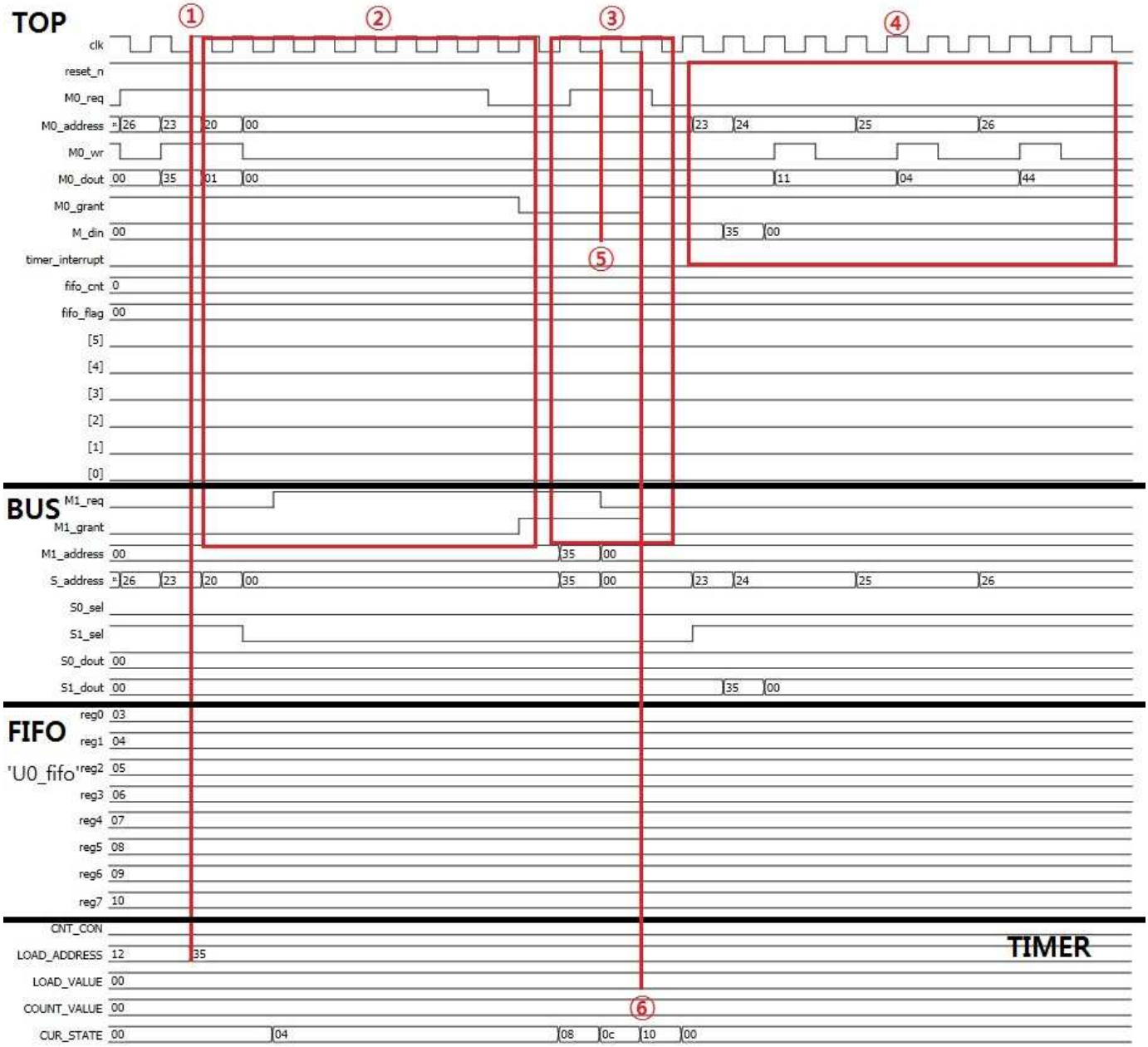
1-18. Design Verification (top1) (top2, top3까지 연결된다.)



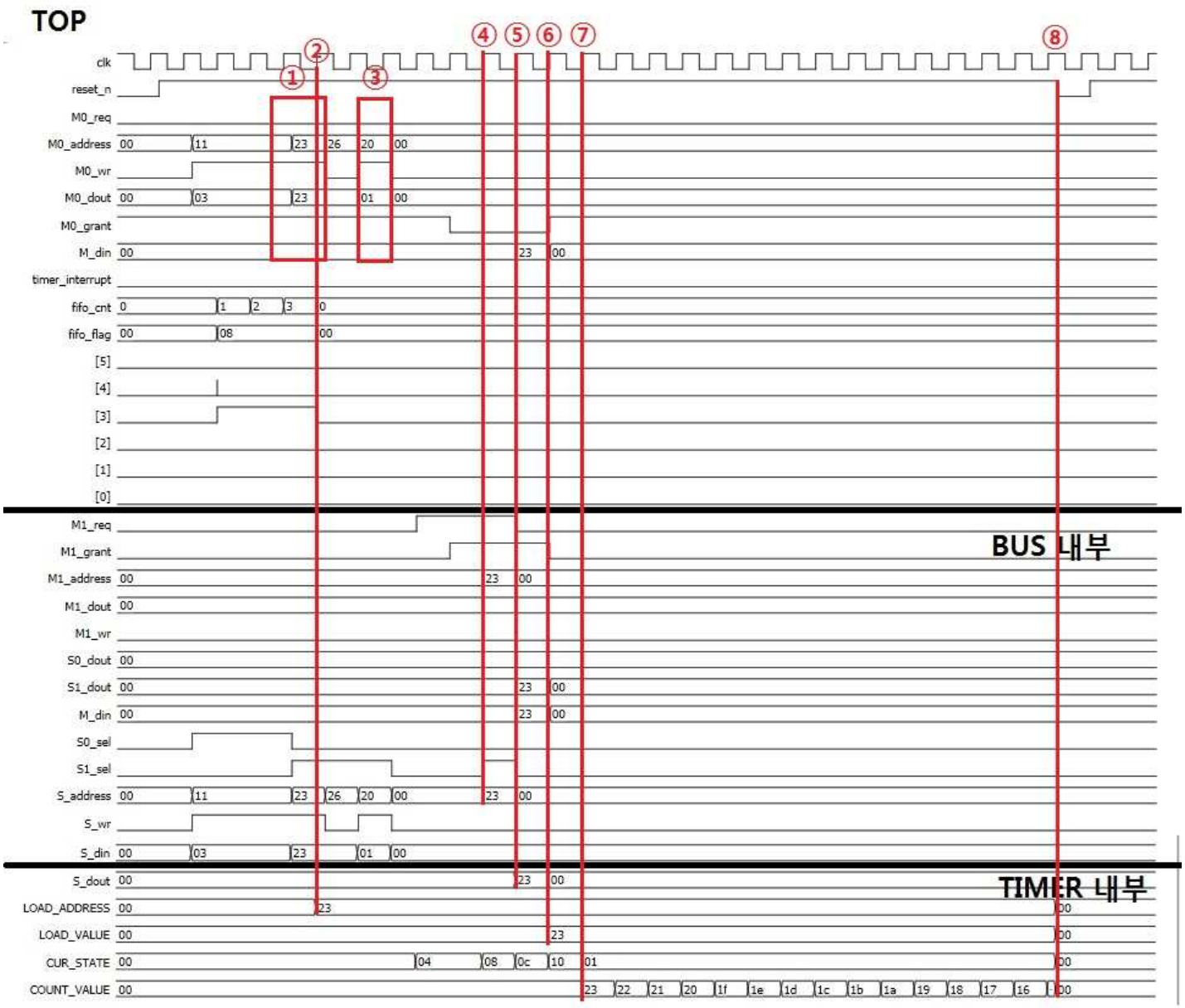
1-19. Design Verification (top2) (top1, top3과 연결된다.)



1-20. Design Verification (top3) (top1, top2와 연결된다.)



1-21. Design Verification (top4) (※ INVALID한 입력 - LOAD_ADDRESS 값이 8'h23일 때)



2. 표 목차.

2-1. Input/Output Description (Synchronous FIFO) - (5-1. Synchronous fifo.)

Direction	Port Name	Description
Input	clk	Clock (rising edge에서 read/write operation 수행)
	reset_n	Active low reset (flags와 pointer를 초기화)
	wr_en	Write enable (request)
	rd_en	Read enable (request)
	din[7:0]	Data input
Output	dout[7:0]	Data output
	data_count[3:0]	Data count (현재 FIFO에 있는 data의 수를 vector(unsigned binary)로 표현)
	full	Full flag (추가적인 write operation이 불가능)
	empty	Empty flag (추가적인 read operation이 불가능)
	wr_ack	Write acknowledge flag (Handshake signal로 wr_en이 활성화 되었을 때, 입력된 data가 FIFO에 쓰여졌음을 나타낸다.)
	wr_err	Write error flag (Handshake signal로 wr_en이 활성화 되었을 때, 입력된 data가 FIFO에 쓰여지지 않았음을 나타낸다.)
	rd_ack	Read acknowledge flag (Handshake signal로 rd_en이 활성화 되었을 때, FIFO로부터 data를 읽을 수 있음을 나타낸다.)
	rd_err	Read error flag (Handshake signal로 rd_en이 활성화 되었을 때, FIFO로부터 data를 읽을 수 없음을 나타낸다.)

2-2. Input/Output Description (FIFO-TOP) - (5-2. fifo-top)

Direction	Port Name	Description
Input	clk	Clock
	reset_n	Active low reset
	sel	Select signal (Bus를 통해 해당 component가 선택됨을 의미)
	wr	Write/read signal (1이면 write, 0이면 read operation 수행)
	address[7:0]	Address (입력된 address 중 하위 4bits만을 사용하여 FIFO 중에 하나를 선택)
	din[7:0]	Data input
Output	dout[7:0]	Data output
	fifo_cnt[3:0]	Number of data in FIFO (현재 선택된 FIFO의 data 수를 vector (unsigned binary)로 표현)
	fifo_flag[5:0]	FIFO flags (현재 선택된 FIFO의 flag를 출력한다. 각각의 bit는 다음과 같이 할당된다. fifo_flag[5] = full fifo_flag[4] = empty fifo_flag[3] = wr_ack fifo_flag[2] = wr_err fifo_flag[1] = rd_ack fifo_flag[0] = rd_err

2-3. Input/Output Description (TIMER) - (5-3. timer)

Direction	Port Name	Description
Input	clk	Clock
	reset_n	Active low reset
	M_grant	(Master interface) Grant (Bus에서 timer가 data를 주고 받는 것을 허락해주는 signal로, 해당 signal이 1인 동안 bus를 통해 다른 slave component를 제어할 수 있다.)
	M_din[7:0]	(Master interface) Data input
	S_sel	(Slave interface) Select
	S_wr	(Slave interface) Write/read
	S_address[7:0]	(Slave interface) Address (Timer에선 해당 address 8bits 중 하위 4-bits만을 offset으로 사용한다.
	S_din[7:0]	(Slave interface) Data input
Output	M_req	(Master interface) Request (Timer가 bus를 통해 data를 주고 받을 수 있도록 허락해달라고 요청하는 signal)
	M_wr	(Master interface) Write/read
	M_address[7:0]	(Master interface) Address

2-4. Input/Output Description (Register Description (in Timer)) - (5-3. timer)

Offset	Type	Name	Description	Default value
0x0	W	CNT_EN	Count를 enable 시킨다. 해당 register에서 [7:1] bits는 reserved이고, [0]bit가 해당 timer의 count를 enable하는 데 사용된다.	0x00
0x1	R/W	INTRRUPT	Interrupt가 발생했을 때 0을 써줌으로 interrupt를 clear 시킨다. 해당 register에서 [7:1]bits는 reserved 이고, [0]bit가 해당 timer의 interrupt를 clear하는 데 사용된다.	0x00
0x2	R/W	CNT_CON	Interrupt clear 후 계속 count down할 지 여부를 결정한다. [7:1] bits는 reserved 이고, [0]bit가 해당 timer의 interrupt를 clear한 후에 1이면 카운트를 다시 수행하고, 0이라면 카운트를 하지 않는다.	0x00
0x3	R/W	LOAD_ADDRESS	Data를 read할 곳의 address이다. [7:0] bits 모두 address에 사용된다.	0x00
0x4	R	LOAD_VALUE	Load address register를 통하여 load된 값으로 해당 값부터 count를 수행한다.	0x00
0x5	R	COUNT_VALUE	현재 count 되고 있는 값을 나타낸다. [7:0]bits 모두 값을 카운트하는 데 사용된다.	0x00
0x6	R	CUR_STATE	Timer의 현재 state를 나타낸다.	0x00

2-5. Input/Output Description (BUS) - (5-4. bus)

Direction	Port Name	Description
Input	clk	Clock
	reset_n	Active low reset
	M0_req	Master 0 request
	M0_wr	Master 0 write/read
	M0_address[7:0]	Master 0 address
	M0_dout[7:0]	Master 0 data output
	M1_req	Master 1 request
	M1_wr	Master 1 write/read
	M1_address[7:0]	Master 1 address
	M1_dout[7:0]	Master 1 data out
	S0_dout[7:0]	Slave 0 data out
	S1_dout[7:0]	Slave 1 data out
Output	M0_grant	Master 0 grant
	M1_grant	Master 1 grant
	M_din[7:0]	Master data input
	S0_sel	Slave 0 select
	S1_sel	Slave 1 select
	S_address[7:0]	Slave address
	S_wr	Slave write/read
	S_din[7:0]	Slave data input

2-6. Input/Output Description (TOP) - (5-5. top)

Direction	Port Name	Description
Input	clk	Clock
	reset_n	Active low reset
	M0_req	Master 0 request
	M0_wr	Master 0 write/read
	M0_address[7:0]	Master 0 address
	M0_dout[7:0]	Master 0 data output
Output	M0_grant	Master 0 grant
	M_din[7:0]	Master data input
	timer_interrupt	Timer interrupt
	fifo_cnt[3:0]	Number of data in FIFO (현재 선택된 FIFO의 data의 수를 vector (unsigned binary)로 표현)
	fifo_flag[5:0]	FIFO flags (현재 선택된 FIFO의 flag를 출력한다. 각각의 bit는 다음과 같이 할당된다. fifo_flag[5] = full fifo_flag[4] = empty fifo_flag[3] = wr_ack fifo_flag[2] = wr_err fifo_flag[1] = rd_ack fifo_flag[0] = rd_err

3. Term Project 소개.

3-1. Project 구현.

FIFO, TIMER, BUS를 연결하여 동작하는 COUNTER를 설계한다.

3-2. 목적.

- 과제에서 주어진 specification을 충분히 이해하고 과제를 수행한다.
- State Transition을 구성하고 FSM 설계 능력을 향상한다.
- testbench에서 Invalid한 입력이 주어졌을 때 어떻게 대처해야 효율적으로 작동할 수 있을지 여러 대처방안을 고안해보고 예외처리에 대한 생각을 넓힘으로써 회로 구현 능력을 향상시킨다.

4. 일정 및 계획.

4-1. Project 구현 일정 및 계획.

- ※pg3의 그림 (1. Schedule.) 참고.
- Project 제안서에서 계획했던 일정과 조금의 변화가 있어 다시 수정하였다. 2011년 11월 2일부터 2011년 11월 12일까지 주어진 specification을 제대로 이해하고 계획을 세웠다.
- 2011년 11월 13일부터 2011년 11월 16일까지 FIFO, FIFO-TOP, BUS, TIMER에 관한 State Transition Diagram을 작성하였고 각 module에 관한 상세 spec을 설계하였다.
- 2011년 11월 17일부터 2011년 11월 22일까지 각각의 module을 Verilog code로 구현하였다.
- 2011년 11월 23일부터 2011년 12월 3일까지 구현한 Verilog code를 검증하였다. 2011년 11월 29일에 첫 번째 예비 검증을 수행하였고 2011년 12월 3일 두 번째 예비 검증을 수행하였다.
- 2011년 12월 4일부터 2011년 12월 4일부터 2011년 12월 6일까지 최종 결과보고서를 작성하였다.

5. Project Specification.

5-1. Synchronous fifo.

- ※pg8의 표 2-1. Input/Output Description (Synchronous FIFO) 참고.

5-1-1. Introduction.

- Synchronous FIFO 는 First-In-First-Out memory queue로 내부에 read와 write에 대한 pointer를 관리하는 control logic이 있어 status flags를 생성하고 user logic과 interface하기 위한 handshake signal을 제공한다.

5-1-2. Features.

- Data widths 가 8bits인 8개의 data를 저장할 수 있는 memory depth를 가진다.
- Status flag로 full과 empty를 제공한다.
- Handshake signal인 wr_ack, wr_err, rd_ack, rd_err를 통해 write와 read 요청에 관한 feedback을 제공한다.
- data_count를 통해 fifo안에 있는 현재 data의 수를 확인할 수 있다.

5-1-3. Functional Description.

- write-enable input(wr-en)이 1이면, clock의 rising edge에서 다음에 이용할 수 있는 memory의 빈 공간에 쓰여지며 Memory full status output(full)은 module 내부 memory에 더 이상 빈 공간이 없음을 나타낸다.
- Synchronous fifo에 저장된 data는 clock의 rising edge에서 read-enable input(rd_en)이 1이면 쓰여진 순서대로 output port인 dout을 통해 빠져나간다. Memory empty status output(empty)의 값이 1이면 내부 memory에 더 이상 data가 존재하지 않음을 나타낸다.
- fifo는 invalid request에 의해 손상되어서는 안된다. empty flag의 상태가 활성화 되어있거나 full flag가 활성화 되었을 때 각각 read operation과 write operation을 요청할 때 fifo에 어떠한 영향도 미쳐서는 안된다. Handshake signal인 read error output(rd_err)과 write error output(wr_err)은 이러한 invalid request의 error가 발생했음을 알려준다.

5-1-4. Behaviors of status signals.

- Active low에 동작하는 reset_n은 내부 포인터를 재설정한다. (empty status output은 1로 하고, full status output은 0으로 초기화한다. reset_n을 통해 fifo에 저장되어 있지만 read되지 않은 data를 버림으로써 효과적으로 fifo 내부를 비울 수 있게 해준다.)
- Write acknowledge output(wr_ack)는 요청된 write 신호에 대해 승인함을 알려준다. (fifo 상태가 full 이 아닐 때.)
- Write error output(wr_err)는 요청된 write 신호에 대해 거부함을 알려준다. (fifo의 상태가 full일 때.)
- Read acknowledge output(rd_ack)는 요청된 read 신호에 대해 승인함을 알려준다. (fifo의 상태가 empty가 아닐 때.)
- Read error output(rd_err)는 요청된 read 신호에 대해 거부함을 알려준다. (fifo의 상태가 empty일 때.)
- write가 요청되면 read 요청은 될 수 없고, read가 요청되면 write가 요청될 수 없다.
- write와 read가 둘 다 요청 되지 않을 경우 비활성화 된다.

5-1-5. Additional Information.

- FIFO에서 read enable signal(rd_en)이 1이 되는 경우, output port인 dout으로 0을 보낸다.

5-2. fifo-top.

- ※pg8의 표 2-2. Input/Output Description (FIFO-TOP) 참고.

5-2-1. Introduction.

- 내부에 synchronous FIFO 4개를 instance한 component 이다. 외부로부터 해당 component가 선택되었을 경우 입력된 address signal 과 write/read signal을 해독하여 4개 중 하나의 FIFO를 선택하여 read/write를 수행한다.

5-2-2. Features.

- Select signal(sel)이 1일 때 작동한다.
- Write/Read signal(wr)이 1이면 write operation을 처리하고 0이면 read operation을 처리한다.
- 각각의 Synchronous FIFO의 주소는 8bits이며 Address signal은 이 fifo를 선택하기 위해 사용된다. 이중 상위 4bits는 사용하지 않고 하위 4bits를 offset으로 사용하여 4개의 fifo 중 하나를 선택한다.
- Data in signal(din) 은 bus를 통해 입력된 data로 4개 fifo의 data in과 연결되고, Data out signal(dout) 은 4개의 FIFO 중 선택된 FIFO의 dout을 결과로 연결한다.
- 4개의 FIFO 중 선택된 FIFO의 handshake signal, status flag, count vector를 연결하여 출력한다.

5-2-3. Functional Description.

- 4개의 FIFO는 offset이 각각 0x1, 0x2, 0x3, 0x4이며 그 외의 값일 경우에는 아무 일도 수행하지 않는다. (Synchronous FIFO 각각의 offset은 8'h11, 8'h12, 8'h13, 8'h14이다.)
- Select signal(sel)이 1일 때, wr_en이 1이면 data_in input(din)이 FIFO에 쓰여지도록 요청된 것이고 read enable signal(rd_en)이 1이면 FIFO에 저장되어 있는 값을 읽도록 요청된 것이다. 4개의 FIFO중 하나가 선택되었을 때 해당 FIFO에서 나오는 status flag, handshake signal, count vector 값을 FIFO TOP의 FIFO flag output(fifo_flag)와 FIFO count value(fifo_cnt)를 통해 출력된다. FIFO flag output(fifo_flag)은 6bits 값을 가지는데 각각의 비트는 최상위부터 순서대로 full, empty, write acknowledge(wr_ack), write error(wr_err), read acknowledge(rd_ack), read error(rd_err)를 나타낸다. 4개 중 아무것도 선택되지 않았을 경우 FIFO flag output(fifo_flag)과 FIFO count value(fifo_cnt)는 모두 0을 출력한다.

5-2-4. Additional Information.

- Fifo top에서 instance하는 4개의 Synchronous FIFO의 instance 이름은 각각 'U0_fifo', 'U1_fifo', 'U2_fifo', 'U3_fifo'로 정의한다.
- FIFO TOP module 이 선택되었지만 (sel == 1) 입력된 address가 4개의 FIFO중 아무것도 선택되지 않을 경우 아무 일도 하지 않는다. (하위 4bits offset이 0x1, 0x2, 0x3, 0x4가 아닐 경우)

5-3. timer.

- ※pg9의 표 2-3. Input/Output Description (TIMER) & 표 2-4. Input/Output Description (Register Description (in Timer)) - 참고.

5-3-1. Introduction.

- 입력으로 들어오는 펄스(CLOCK)를 특정 수만큼 count 하는 장치로 count 명령을 받으면 내부 load address register(해당 FIFO)에 저장된 값을 address로 하여 값을 읽어와 count한다. 클록이 상승에지일 때마다 count가 1씩 감소하며 count 값이 0으로 끝나면 interrupt를 발생시킨다.

5-3-2. Features.

- bus에 연결되어 동작한다.
- Master interface와 Slave interface를 가진다.
- Address input(M_address, S_address)는 모두 8bits이다.
- Data input/output(M_dout, M_din, S_dout, S_din)은 모두 8bits다.
- bus를 통해 data transfer를 요청하고자 할 때 Master request output(M_req)는 1이 된다.
- M_req를 요청하고 bus에서 허락할 때 M_grant는 1이 된다. M_grant가 1이 되면 timer는 master address output(M_address), master write/read output(M_wr), master data-out output(M_dout)를 통하여 해당 주소의 레지스터에 접근하며 그 레지스터에 저장되어있는 data를 master data-in input(M_din)을 통해 가져온다.
- slave interface를 통해 timer의 내부 register를 외부에서 제어할 수 있다. 제어가 필요한 register들은 각각 offset을 가지고 있다.
- 해당 FIFO에서 읽어온 data 값만큼 count down을 수행한다.
- Timer에서 count할 수 있는 값은 1 ~ 255 (0x1 ~ 0xFF)이며, count 값이 0일 경우에는 count를 수행하지 않는다. (※ FIFO에서 load하여 받은 값이 0일 경우 count를 수행하지 않는다.)
- count가 끝나면 interrupt를 발생시킴으로써 testbench에게 count가 끝났음을 통지한다.

5-3-3. Functional Description.

- TIMER는 외부로부터 제어를 받고 master interface를 통해 외부를 제어할 수 있다.
- Slave interface는 pin이름이 'S_' 로 시작하고 Master interface는 'M_' 으로 시작한다.
- 외부에서 timer를 동작시키기 위해서는 timer의 내부 상태가 IDLE 상태일 때, count enable(CNT_EN) register에 1이 쓰여져야 한다. CNT_EN이 1이 되면 timer는 내부 load address(LOAD_ADDRESS) register에 저장되어 있는 값을 address로 하여 해당 위치의 값을 읽어오고 읽어온 값을 count한다. count가 끝나고 interrupt가 발생하는데 내부 interrupt(INTRRUPT) register에 0을 써서 clear할 수 있다. interrupt가 clear된 후 continuous count(CNT_CON) register의 값이 1이면 해당 LOAD_VALUE register에 저장되어 있는 값을 COUNT_VALUE register에 복사하여 다시 count를 수행하고 continuous count(CNT_CON) register의 값이 0이면 더 이상 count를 수행하지 않으며 IDLE STATE으로 돌아가 대기한다.

5-3-4. Register Description.

- Timer는 8bits register를 가지고 있다.
- 표 2-4. Input/Output Description (Register Description (in Timer))을 참고하면 알 수 있듯이 type이 'W'인 경우 해당 register에 write operation만 가능하고 type이 'R'인 경우 해당 register의 값을 read하는 것만 가능하며 'R/W'일 경우 해당 register에 값을 쓰거나 해당 register의 값을 읽을 수 있다.

5-3-5. Additional Information.

- CUR_STATE register는 해당 register의 값을 read했을 때 0x00이면 IDLE 상태로 한다. (IDLE 상태는 어떠한 동작도 하지 않는 상태이다.)
- CNT_EN register는 IDLE 상태일 때 1이 쓰이면 외부로부터 값을 읽은 후 count를 시작하도록 구현하며 그 외의 경우에 CNT_register에 1이 쓰이면 무시한다.
- M_req port가 1이고 bus의 응답으로 M_grant port가 1이 되면 1 cycle 동안 LOAD ADDRESS register의 값을 요청 후 읽어진 data로 count한다.

- CNT_EN register의 값을 read 하려고 할 경우 slave data out(S_dout) port에 0x00을 출력한다.
- interrupt(INTRRUPT) register는 interrupt가 발생했을 때, 0을 write함으로써 interrupt를 clear 시키는 역할을 한다. 그 외의 경우에 register를 write 하는 것은 무시한다.
- interrupt(INTRRUPT) register에 read 요청이 들어오면 interrupt가 발생했을 때 slave data out(S_dout) portf를 통해 1을 출력하고 interrupt가 발생하지 않았을 때는 0을 출력한다.
- CNT_CON register의 값은 언제든 read하거나 write 할 수 있다.
- LOAD_ADDRESS register의 값은 언제든 read하거나 write 할 수 있다.
- 해당 레지스터의 값을 읽을 수 밖에 없게 설정된 LOAD_VALUE register와 COUNT_VALUE register, CUR_STATE register에 write operation이 요청되면 timer는 이를 무시한다.
- Master interface를 통해 read한 값을 LOAD VALUE register에 저장하며 이 값을 계속 유지한다. (reset_n이 활성화될 때와 다시 Master interface를 통해 data를 read할 경우 제외.)
- COUNT_VALUE register는 reset_n이 되었을 경우 default value(0x00)을 가지며 count를 하고 있지 않을 때 0을 가진다.

5-4. bus.

- ※pg10의 표 2-5. Input/Output Description (BUS) 참고.

5-4-1. Introduction.

- data를 transfer할 수 있도록 연결해주는 component로 새로운 device를 추가하기 쉽고 가격이 저렴한 특징이 있다.

5-4-2. Features.

- 2개의 master와 2개의 slave를 가진다.
- Address bandwidth는 8bits다.
- Data bandwidth는 8bits다.
- Master는 data를 transfer를 하기 위해 request 후에 grant 신호를 받아야 한다.
- 모든 Master의 req 요청이 없을 때 기본적으로 M0(testbench)가 grant를 받는다.
- Salve 0 은 0x10 ~ 0x1F 사이의 address를 memory map 으로 가진다. (Slave 0 =FIFO top)
- Slave 1 은 0x20 ~ 0x2F 사이의 address를 자신의 memory map으로 가진다. (Slave 1 = timer)

5-4-3. Functional Description.

- Master는 bus를 통해 data를 transfer하고자 할 때 자신에게 해당하는 request signal (M0_req or M1_Req)를 1로 하고 그에 대한 확인으로 grant signal(M0_grant or M1_grant)을 받은 후 data transfer를 할 수 있다. 여기서 M0는 testbench를 의미하고 M1은 timer를 의미한다.

5-4-4. Additional Information.

- 하나의 master가 request를 요청하여 grant를 받고 있으면, 다른 master에서 request를 요청하여도 request가 유지되는 동안은 grant를 뺏기지 않는다.
- 두 개의 master가 request를 요청하지 않다가 동시에 요청하면 grant를 받고 있던 Master 0가 grant를 유지한다.

5-5. top.

- ※pg10의 표 2-6. Input/Output Description (TOP) 참고.

5-5-1. Introduction.

- TOP은 FIFO TOP, TIMER, BUS를 instance 하여 연결한 component이다.

5-5-2. Features.

- 외부에서 master 0 interface를 사용하여 FIFO TOP과 TIMER를 제어할 수 있다.
- FIFO 를 제어할 때 fifo_cnt, fifo_flag를 통해 해당 FIFO의 출력을 확인할 수 있다.
- Timer의 count가 끝나면 interrupt output port(timer_interrupt)를 통해 interrupt를 받을 수 있다.

5-5-3. Additional Information.

- TOP에서 instance 하는 BUS, FIFO TOP, TIMER의 instance 이름은 각각 'U0_bus', 'U1_fifo_top', 'U2_timer' 다.

6. Design details.

6-1. Synchronous fifo.

- ※pg4의 (1-2. State Transition Diagram - Synchronous FIFO) 참고.

6-1-1. 구현한 내부 module 소개.

- 최상위 module 이름은 specification에서 주어진대로 'fifo'이다.
- 하위 module은 'register_file', 'fifo_mx2', 'fifo_and1'이 있다.
- 'fifo'의 구현은 State Transition Diagram을 통해 Moore FSM방식으로 설계하였다. Combinational Logic(Next_fifo_STATE와 Output Logic)과 Sequential Logic으로 나누어 설계하였고, Moore FSM 방식이므로 Next_fifo_STATE은 입력과 현재 STATE 상태에 의해 출력이 결정되며 Output Logic은 입력에 상관없이 오직 현재 STATE 상태에 의해 출력이 결정된다.
- 3bits의 head와 tail을 통해 Circular Queue 방식으로 data가 저장되고 출력된다. 총 8 bits bandwidth의 8개의 data가 저장될 수 있다. write operation이 요청되면 저장된 data의 수가 7개 이하일 때 input port인 din의 값이 레지스터에 저장된다. 이 때, tail은 1 증가한다. Circular Que 방식이기 때문에 만약 tail의 값이 7이라면 0이 된다. data_count는 1 증가한다.

  read operation이 요청되면 fifo 내부에 저장된 data가 존재할 때 (최소 data_count가 1일 때) output port인 dout을 통해 가장 처음에 저장된 data부터 차례대로 빠져나간다. 이 때, head는 1 증가한다. Circular Que 방식이기 때문에 만약 head의 값이 7이라면 0이 된다. data_count는 1 감소한다.
- module 'register file' 의 port는 input port로 clk, wAddr[2:0], wData[7:0], we, no_full, rAddr[2:0]이 있고 output port는 rData[7:0]가 있다. we_w의 값이 1일 때 wAddr의 주소에 따라 각각의 register에 wData의 값이 저장된다. module 'fifo'에서 status flag인 full의 값이 0이고 write operation을 요청하는 신호를 나타내는 wr_en의 값이 1일 때 we_w의 값이 1이 된다. we_w의 값이 1이면 wData의 값이 NEXT_register에 먼저 저장되고 클록이 상승에지일 때 NEXT_register에 저장된 data값이 해당 register에 저장된다. wAddr은 module 'fifo'의 NEXT_tail의 값이고, wData는 fifo에 입력으로 들어오는 din의 값이며, rAddr은 NEXT_head의 값이다. rAddr에 따라 각각 register의 값이 rData로 출력된다. rData의 값은 module 'fifo'에서 read operation 요청이 들어왔을 때 data를 읽을 수 있는 조건이 충족되면 module 'fifo'의 output port인 dout을 통해 출력된다.
- 해당 module 'fifo'의 fifo_STATE에 대해 살펴보겠다. (※ 참고 : 1-2. State Transition Diagram - Synchronous FIFO)

  fifo_STATE은 총 9개로 나누었는데 각각 fifo_INIT_STATE, fIfo_INIT_READ_ERR_STATE, fifo_WRITE_STATE, fifo_FULL_STATE, fifo_WRITE_ERR_STATE, fifo_READ_STATE, fifo_EMPTY_STATE, fifo_READ_ERR_STATE, fifo_NOP_STATE 이다.

6-1-2. fifo_STATE 소개.

- **6-1-2-1. fifo_INIT_STATE**
- fifo_INIT_STATE 은 가장 첫 번째 STATE이다. 만약 reset_n의 값이 0이면 어떠한 STATE에 있더라도 INIT_STATE으로 돌아온다.
  이 때, flags는 empty를 제외하고 모두 0이다.
  INIT_STATE은 fifo의 내부가 비어있는 상태이므로 circular queue와 같이 동작하는 synchronous의 head와 tail도 각각 0의 값을 가진다. 내부 포인터가 초기화됨으로 empty는 1로 활성화된다. read operation이 요청되면 fifo_INIT_READ_ERR_STATE 으로 분기한다.
  write operation이 요청되면 fifo_WRITE_STATE 으로 분기한다. 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.
- **6-1-2-2. fifo_INIT_READ_ERR_STATE**
- flags는 empty와 rd_err는 1이고 나머지는 0이다.
- read operation이 요청되면 fifo_INIT_READ_ERR_STATE 에 머문다. write operation이 요청되면 fifo_WRITE_STATE 으로 분기한다. 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-3. fifo_WRITE_STATE**
- flags는 empty = 0, wr_ack = 1, wr_err = 0, rd_ack = 0, rd_err = 0 이고 현재 data_count가 7개이면 full = 1로 출력하고 7개가 아니면 full = 0으로 출력한다.
- data_count가 7인데 write operation이 요청되면 fifo_FULL_STATE로 분기하고 7이 아니면 계속 fifo_WRITE_STATE에 머문다.
- read operation이 요청되면 fifo_READ_STATE으로 분기한다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-4. fifo_FULL_STATE**
- flags는 full과 wr_ack는 1을 출력하고 나머지는 0을 출력한다.
- write operation이 요청되면 fifo_WRITE_ERR_STATE으로 분기한다.
- read operation이 요청되면 fifo_READ_STATE으로 분기한다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-5. fifo_WRITE_ERR_STATE**
- flags는 full과 wr_err는 1을 출력하고 나머지는 0을 출력한다.
- write operation이 요청되면 fifo_WRITE_ERR_STATE으로 분기한다.
- read operation이 요청되면 fifo_READ_STATE으로 분기한다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-6. fifo_READ_STATE**
- data_count가 0이면 empty는 1을 출력하고 0이 아니면 empty는 0을 출력한다. 나머지 flags중 rd_ack만 1을 출력하고 나머지는 0을 출력한다.
- write operation이 요청되면 fifo_WRITE_STATE 으로 분기한다.
- read operation이 요청되면 data_count가 1일 경우, fifo_EMPTY_STATE 으로 분기하고 1이 아닐 경우, fifo_READ_STATE 에 머문다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-7. fifo_EMPTY_STATE**
- flags는 empty와 rd_ack는 1을 출력하고 나머지는 0을 출력한다.
- write operation이 요청되면 fifo_WRITE_STATE 으로 분기한다.
- read operation이 요청되면 fifo_READ_ERR_STATE 으로 분기한다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-8. fifo_READ_ERR_STATE**
- flags는 empty와 rd_err는 1을 출력하고 나머지는 0을 출력한다.
- write operation이 요청되면 fifo_WRITE_STATE 으로 분기한다.
- read operation이 요청되면 fifo_READ_ERR_STATE 에 머문다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

- **6-1-2-9. fifo_NOP_STATE**
- 만약 data_count가 8이면 full = 1, empty = 0이다. 만약 data_count가 0이면 full = 0, empty = 1이다. 그 외의 경우에 full 과 empty는 모두 0이다. 나머지 flags들은 모두 0을 출력한다.
- write operation이 요청되었을 때, data_count가 7이면 fifo_FULL_STATE 으로 분기하고 full == 1이면 fifo_WRITE_ERR_STATE 으로 분기하며 data_count가 7이 아니거나 full == 0이면 fifo_WRITE_STATE 으로 분기한다.
- read operation이 요청되었을 때, data_count가 1이면 fifo_EMPTY_STATE 으로 분기하고 empty == 1이면 fifo_READ_ERR_STATE 으로 분기하며 data_count가 1이 아니거나 empty == 0이면 fifo_READ_STATE 으로 분기한다.
- 그 외의 경우에는 fifo_NOP_STATE 으로 분기한다.

6-1-3. 'fifo' output port 소개.

- output port에는 dout, data_count와 status flags인 full 과 empty 그리고 Handshake signals인 wr_ack, wr_err, rd_ack, rd_err가 있다.
- dout은 read operation이 요청되고 fifo의 내부가 비어있지 않은 상태일 때 출력한다.

- data_count는 현재 fifo에 저장된 data의 수를 알려준다.
- status flag인 full 은 현재 fifo에 8개의 data가 저장되어 있을 때 1이 된다. (이번 프로젝트에서 한 개의 fifo가 저장할 수 있는 data의 수는 8개이다.
- status flag인 empty는 현재 fifo에 저장된 data의 수가 0일 때 1이 된다.
- handshake signal인 wr_ack는 write operation 요청이 들어와서 해당 fifo에 data가 제대로 쓰여지면 1이 된다.
- handshake signal인 wr_err는 더 이상 해당 fifo에 저장할 공간이 없음(data_count == 8, full == 1 로 내부가 꽉 찬 상태)에도 불구하고 write operation을 요청할 때만 1이 된다.
- handshake signal인 rd_ack는 read operation 요청이 들어와서 해당 fifo에 저장된 값이 output port인 dout으로 출력이 잘 되었을 때 1이 된다.
- handshake signal인 rd_err는 해당 fifo에 저장된 data가 없어 비어있는 상태임에도 불구하고 read operation을 요청했을 때만 1이 된다.

6-2. fifo-top.

- ※pg 7의 (1-9. Verilog로 구현한 FIFO_TOP 내부 회로 구성 화면 1. - WRITE OPERATION 요청 시)와 pg 8의 (1-10. Verilog로 구현한 FIFO_TOP 내부 회로 구성 화면 2. - READ OPERATION 요청 시) 참고.

6-2-1. 구현한 내부 module 소개.

- 최상위 module 이름은 specification에서 주어진대로 'fifo_top'이다.
- 하위 module은 'decoder', '_andcn', '_andcn4', '_andcn8', 'cnt_mx2', 'ot_mx2', 'mx2'이다.
- 이번 프로젝트에서 회로 구성능력과 설계 능력을 향상 시키기위해 fifo_top을 structure한 형태로 구성해 보았다. (always문을 거의 사용하지 않고 구현하도록 노력해보았다.) and gate와 multiplexer를 사용하여 구현하였다.

6-2-2. 구현한 회로 설명.

- wr이 1이면 write operation을 수행하고 wr이 0이면 read operation을 수행한다. 여기서 중요한 점은 input port인 address[7:0]의 상위 4bits의 값이 1이면 fifo_top이 선택된 것이고 sel이 1이 되는데, sel의 값이 1일 때만 요청된 write operation과 read operation을 수행한다. 이와 같은 동작을 4개의 1bit크기의 Multiplexer를 통해 구현하였다. 일단 wr이 1이면 U00_read_mx2에서 output 으로 0이 출력되고 U00_sel_mx2에서 sel의 값이 1이더라도 read operation을 수행하지 않는다. 또한 wr이 1이면 U11_write_mx2에서 output으로 1이 출력되고 U11_sel_mx2에서 입력으로 들어오는 sel의 값이 1이면 write operation을 수행한다.
  wr이 0이면 U00_read_mx2에서 output으로 1이 출력되고 U00_sel_mx2에서 sel의 값이 1이면 read operation을 수행한다. 또한 wr이 0이면 U11_write_mx2에서 output으로 0이 출력되고 U11_sel_mx2에서 입력으로 들어오는 sel의 값이 1이더라도 write operation을 수행하지 않는다.
- 총 4개의 fifo가 fifo_top안에 존재한다. 각각의 fifo의 주소를 살펴보면 'U0_fifo' 는 8'h11, 'U1_fifo'는 8'h12, 'U2_fifo'는 8'h13, 'U3_fifo'는 8'h14 이다.
- Specification에서 주어진 조건과 같이 하위 4bits만을 해독하여 4개의 fifo 중 한 개를 선택하여 write나 read를 수행한다. 이 조건을 만족시키기 위해서 'U1_rd_and' ~ 'U4_rd_and' 와 'U1_wr_and' ~ 'U4_wr_and' 의 이름을 가진 8개의 and gate를 사용하였다. 입력으로 들어오는 주소(address)와 wr의 값에 따라 8 개의 and gate 중 한 개의 gate 출력 값이 1이 되어 요청된 operation을 수행한다. (그림 (1-9. Verilog로 구현한 FIFO_TOP 내부 회로 구성 화면 1. - WRITE OPERATION 요청 시) 참고.)
- read operation이 요청되었을 때에 대해 설명하겠다. 일단, next_address 레지스터는 클록이 상승에지일 때 입력으로 들어오는 주소를 값으로 갖는다. 이 next_address의 하위 4bits 값에 따라 4개의 fifo 중 한 개가 선택이 되거나 아무것도 선택이 되지 않도록 설계하였다. 만약 next_address 레지스터의 하위 4bits 값이 1이면 'U0_fifo' 를 선택하고 2이면 'U1_fifo' 를 선택하고 3이면 'U2_fifo' 를 선택하고 4이면 'U3_fifo' 를 선택하여 해

당 fifo의 dout과 data_count, status flags, handshake signals를 fifo_top의 output인 dout, fifo_cnt, fifo_flag[0]~fifo_flag[5]를 통해 출력된다.

여기서 중요한 점은 16-to-1 multiplexer를 사용하여 하위 4bits의 모든 경우의 수에 대해 처리했다는 점이다. 만약, address의 하위 4bits 값이 1~4 가 아닌 0 또는 5 ~ 15의 값을 가질 경우, output 인 dout, fifo_cnt, fifo_flag[0]~fifo_flag[5]의 값은 모두 0이다.

※ 주의할 점은 read operation이 요청되었을 때 입력으로 들어온 address의 하위 4bits를 해독하여 output을 결정하므로 만약 상위 4bits의 값이 1이 아닌 다른 값을 가질 때에도 마치 fifo가 선택된 것처럼 동작할 수 있다는 것이다. 이런 점을 방지하기 위해 address 하위 4bits에 의해 결정된 output값들('mx_out_U3_w1', mx_cnt_U3_w1, mx_ful_U3_w1, mx_emp_U3_w1, mx_wr_ack_U3_w1, mx_wr_err_U3_w1, mx_rd_ack_U3_w1, mx_rd_err_U3_w1) 을 현재 sel과 and 연산 시켜주어서 sel이 1이면 해당 값을 출력하고 sel이 0이면 모든 output의 값은 0으로 출력하도록 설계하였다.

6-3. timer.

- ※pg5 (1-3. State Transition Diagram - TIMER - module이름 : timer_reg - CNT_EN_STATE), (1-4. State Transition Diagram - TIMER - module이름 : timer_reg - INTRRT_STATE), (1-5. State Transition Diagram - TIMER - module이름 : timer_master - master_state)참고.

6-3-1. 구현한 내부 module 소개.

- 최상위 module 이름은 specification에서 주어진대로 'timer'이다.
- 하위 module은 'timer_reg', 'timer_master' 'timer_counter', 'timer_reg_mx2'가 있다.
  총 4개의 State Transition Diagram을 사용하여 구현하였고 module 'timer_reg' 에서 2개, 'timer_master'에서 1개, 'timer_counter'에서 1개의 moore FSM으로 설계하였다.

6-3-1-1. module 'timer_reg'.

- module 'timer_reg'에서 총 2개의 FSM을 사용하였다.
- CNT_EN_STATE과 INTRRT_STATE이 해당 2개의 FSM이다.
- CNT_EN_STATE의 초기 STATE은 CNT_EN_IDLE_STATE이다.
- CNT_EN_IDLE_STATE에서 'timer_master' 의 master_state이 IDLE 상태이고 'timer_counter' 의 counter_state이 IDLE 상태일 때 CNT_EN 레지스터에 0이 쓰이면 CNT_EN_READ_REQ 으로 분기한다.
- CNT_EN_READ_REQ에서는 read_req 를 1cycle 동안만 1로 유지하고 다시 CNT_EN_IDLE STATE으로 분기한다. read_req가 1이 됨으로써 module 'timer_master'에서 FSM에 해당하는 master_state이 동작할 수 있도록 한다.
- INTRRT_STATE의 초기 STATE은 INTRRT_IDLE_STATE 이다. INTRRT_IDLE_STATE에서 output port인 interrupt가 1일 때, INTRRUPT 레지스터에 0이 쓰이면 interrupt가 0으로 클리어되는 조건을 충족함으로써 INTRRT_CLEAR_STATE으로 분기한다.
- INTRRT_CLEAR_STATE에서 1cycle동안 output port인 int_clear의 값을 1로 유지한다. int_clear의 값이 1이 됨에 따라 module 'timer_counter'의 counter_state에서 CNT_CON 레지스터의 값에 의해 카운터를 계속할지 아니면 그만할지 결정한다.
- timer가 slave일 때 외부에서 timer의 register 값을 읽도록 명령하는 경우가 있다. 이 때는 S_sel이 1이고 S_wr이 1일 경우다. 만약 INTRRUPT 레지스터의 값을 read 하는 명령이면 interrupt가 1이면 S_dout으로 1을 출력하고 0이면 S_dout으로 0을 출력한다.
  만약, CNT_CON 레지스터 값을 read하는 명령이면 CNT_CON의 값이 1이면 S_dout으로 1을 출력하고 0이면 S_dout으로 0을 출력한다.
  만약, LOAD_ADDRESS 레지스터 값을 read하는 명령이면 현재 저장되어있는 LOAD_ADDRESS 레지스터 값을 S_dout으로 출력한다.

- 만약, LOAD_VALUE 또는 COUNT_VALUE 값을 read하는 명령이면 각각 현재 저장되어있는 LOAD_VALUE 레지스터 또는 COUNT_VALUE 레지스터에 저장된 값을 S_dout으로 출력한다.
- 만약, 현재 상태가 무엇인지 read하는 명령이면 현재 state상태가 저장된 CUR_STATE와 같은 값을 S_dout으로 출력한다. 이 때 CUR_STATE은 timer_master의 master_state과 counter_state의 상태를 동시에 갖는다. CUR_STATE의 상위 3bits는 항상 0으로써 사용하지 않고 0~1 bit는 counter_state을 나타내고 2~4 bit는 master_state을 나타낸다.
- 만약, CNT_EN 의 상태를 read하는 명령이면 S_dout으로 0을 출력한다. 그 외의 경우에는 무조건 S_dout은 0을 출력한다.
- module 'timer_counter'의 COUNT_VALUE가 0이면 그 다음 cycle에 interrupt는 1로 발생한다. 이와 같이 발생한 interrupt는 0으로 clear 해주기 전까지 유지된다.

6-3-1-2. module 'timer_master'.

- module 'timer_reg'에서 1개의 FSM을 사용하였다. (master_state)
- master_state의 초기 상태는 master_IDLE_STATE이다.
- LOAD_VALUE 레지스터가 존재한다.
- module 'timer_reg'에서 read_req의 값으로 1을 보내주면 master_READ_REQ_STATE 으로 분기한다.
- master_READ_REQ_STATE에서 M_req가 1이 됨으로써 bus에 Master request를 한다. 그에 대한 응답으로 bus에서 M_grant 1을 받으면 master_READ_GRA_STATE 로 분기한다. 그 외의 경우에는 그대로 master_READ_REQ_STATE에 머문다.
- master_READ_GRA_STATE에서는 M_address의 값으로 LOAD_ADDDRESS의 값, M_wr의 값, M_dout의 값을 내보낸다. 그리고 master_READ_DEL_STATE 으로 분기한다.
- master_READ_DEL_STATE에서 M_req의 값으로 0을 갖는다. master_READ_GRA_STATE에서 요청한 주소에 저장된 data를 input port인 M_din으로 받는다. 이 때 M_din을 통해 들어오는 data는 LOAD_VALUE 레지스터에 저장된다. (reset_n이 작동하거나 그 다음 data가 저장되기 전까지 계속 유지) 그리고 master_CNT_EN_STATE으로 분기한다.
- master_CNT_EN_STATE에서 내부 signal인 CNT_EN을 1로 내보냄으로써 module 'timer_counter'에서 카운터를 할 수 있는 조건을 만들어준다.

6-3-1-2. module 'timer_counter'.

- module 'timer_counter'에서 1개의 FSM을 사용하였다. (counter_state)
- counter_state의 초기 상태는 counter_IDLE_STATE이다.
- COUNT_VALUE 레지스터가 존재한다.
- timer_master에서 들어오는 CNT_EN이 1이고 LOAD_VALUE의 값이 0이 아닐 때 counter_state은 counter_IDLE_STATE에서 counter_COUNT_STATE으로 분기한다. 또한 counter_COUNT_STATE으로 분기할 때 COUNT_VALUE 레지스터에 LOAD_VALUE의 값이 저장된다.
- counter_COUNT_STATE에서는 COUNT_VALUE를 1씩 감소하며 카운트를 수행하고 COUNT_VALUE가 0이 되면 counter_INTRRUPT_STATE으로 분기한다.
- counter_INTRRUPT_STATE에서는 int_clear가 1이고 CNT_CON이 1이면 다시 counter_COUNT_STATE으로 분기하여 카운트를 재수행하고 만약 int_clear가 1이고 CNT_CON이 0이면 counter_IDLE_STATE으로 분기하여 대기한다. int_clear가 0일 경우에는 counter_INTRRUPT_STATE에 머문다.
- 카운트가 끝난 후 COUNT_VALUE는 0을 유지한다.
- interrupt를 clear한 후 CNT_CON의 값이 1로써 재 카운트를 수행한다면 COUNT_VALUE의 값은 이미 저장되어 있던 LOAD_VALUE의 값을 초기 값으로 받는다.

6-4. bus.

- ※pg6 1-6. State Transition Diagram - TIMER - module이름 : timer_counter - counter_state 참고.

6-4-1. 구현한 내부 module 소개.

- 최상위 module 이름은 specification에서 주어진대로 'bus'이다.

- 하위 module은 'Arbitrator', 'Address_decoder', 'mx2_addr', 'mx2_bus' 가 있다.
- module 'Arbitrator'에서는 1개의 State Transiton Diagram이 존재한다.
- Master 0는 testbench이고, Master 1은 timer이며, Slave 0는 fifo_top이고, Slave 1은 timer이다. 주소가 8'h10 ~ 8'h1f이면 fifo_top에 해당되고 8'h20 ~ 8'h2f이면 timer에 해당된다.
- 하위 module 에 대해 설명하겠다.

6-4-1-1. module 'Arbitrator'.

- module 'Arbitrator'에서 1개의 FSM을 사용하였다. (Arbit_STATE)
- 두 개의 STATE이 있는데 M0GRANT일 때 output으로 M0_grant = 1, M1_grant = 0으로 testbench가 grant를 갖고, 반대로 M1GRANT일 때 output으로 M0_grant = 0, M1_Grant = 1로 timer가 grant를 갖는다.
- Arbit_STATE의 초기 상태는 M0GRANT이다.
- M1_req와 M0_req 모두 0일 때 기본적으로 M0GRANT 상태이다.
- M1_req가 1이고 M0_req가 0일 때 M1GRANT로 분기한다.
- M1GRANT에서 M1_req가 1일 동안만 M1GRANT에 머문다. 만약 현재 M1GRANT 상태에 M1_req가 1이라면 M0_req가 1로 변하더라도 grant는 timer가 계속 받는다.(M1GRANT 상태유지) M1_req가 0으로 바뀌면 M0GRANT로 분기한다.

6-4-1-2. module 'Address_decoder'.

- M0_address와 M1_address 중 현재 grant를 받고 있는 곳의 address를 입력으로 받는다. (M0_grant가 1이면 testbench에서 입력된 address를 입력으로 받고 M1_grant가 1이면 timer에서 M_address로 출력되는 주소를 입력으로 받는다.)
- 입력으로 받는 address의 상위 4bits를 판별하여 상위 4bits의 값이 1이면 fifo_top을 선택하여 S0_sel = 1, S1_sel = 0으로 출력되고 address의 상위 4bits의 값이 2이면 timer를 선택하여 S0_sel = 0, S1_sel = 1로 출력한다.

6-4-2. bus의 전체 동작에 관한 설명.

- module 'Arbitrator'에서 testbench가 grant를 받을지 timer가 grant를 받을지 결정한다.
- 현재 grant를 받는 곳에서 요청하는 wr, address, din 값을 각각 output S_wr, S_address, S_din으로 출력한다.
- S_address의 상위 4bits를 해독하여 값이 1이면 fifo_top이 동작되고, 값이 2이면 timer가 동작되며 값이 1도 2도 아닐 경우에는 아무 동작도 하지 않는다.
- Address decoder에서 S0_sel과 S1_sel을 기반으로 S0_sel이 1이면 fifo_top이 선택된 것이므로 output M_din의 값으로 S0_dout의 값을 출력한다. 만약 S1_sel이 1이면 timer가 선택된 것이므로 output M_din의 값으로 S1_Dout의 값을 출력한다.

6-5. top.

- TOP은 fifo_top, bus, timer를 instance한 최종 module이다.

6-5-1. top 내부 port 소개.

- input port에는 M0_req, M1_address, M0_wr, M0_dout이 있다. M0_req는 testbench가 grant를 받고자 할 때 bus에 요청하기 위해 사용한다. M1_address는 fifo_top 내부에 존재하는 4개의 fifo중 하나를 선택하거나 timer의 내부 register를 선택할 때 사용하는 주소이다. 만약 M0_address 상위 4bits의 값이 1 혹은 2가 아닌 다른 숫자가 입력된다면 어떠한 동작도 하면 안 된다. (예외처리 하였다.) 또한 M0_wr이 1이면 write operation을 요청하는 것이고 0이면 read operation을 요청하는 것이다. M0_dout은 입력 data 값이다. fifo_top의 내부 fifo 값을 쓰거나 timer의 해당 register에 값을 쓸 때 사용한다.

7. Design verification strategy& results.

7-1. Synchronous fifo.

- ※pg9 1-11. Design Verification (fifo 1) 참고.

위의 testbench에 의해 알 수 있듯이 reset_n이 0일 때 fifo는 init 상태가 되므로 data_count = 0 , full = 0, wr_ack = 0, wr_err = 0, rd_ack = 0 , rd_err = 0, empty = 1이 출력된다.

reset_n = 1이 된 후, wr_en이 1이 되면 tb_din의 값이 클록이 상승에지일 때 해당 reg에 쓰인다. 빨간 선으로 나타낸 ①을 참고하면 알 수 있듯이 data가 해당 fifo의 register에 정상적으로 쓰였기 때문에 empty = 0, wr_ack는 1을 출력한다. 또한 data_count가 1로 증가함으로써 fifo안에 있는 현재 1개의 data가 존재함을 알 수 있고, fifo의 내부 register 주소에 해당하는 tail이 1 증가한다.

- 빨간 선으로 나타낸 ②를 참고하면 알 수 있듯이 입력 값 (tb_din)이 0이더라도 data_count가 1 증가, tail이 1 증가함으로써 register에 정상적으로 쓰이는 것을 알 수 있다. 이 때 status flags와 handshake signals 는 원래의 값을 유지한다. 8개의 data가 모두 fifo에 쓰여지면 status flag인 full은 1이 된다. circular queue이기 때문에 tail의 값은 7에서 0이 된다. full인 상태에서는 더 이상 write operation을 수행할 수 없다.
- 빨간 선으로 나타낸 ③을 참고하면 full인 상태이므로 더 이상 write operation을 수행할 수 없으므로 wr_ack는 0이 되고 wr_err가 1로 출력된다.
- 빨간 선으로 나타낸 ④을 참고하면 write operation을 의미하는 wr_en이 0이 되면 클록이 상승에지일 때 wr_err도 0이 된다. wr_en이 0이고 rd_en이 0일 때는 'fifo_NOP_STATE'이므로 status flags는 그대로 유지하고 handshake signals는 0을 출력한다.
- 빨간 선으로 나타낸 ⑤을 참고하면 read operation이 요청되었을 때 tb_dout을 통해 가장 처음에 저장된 data 11을 출력하고 data_count는 1 감소하여 7을 나타낸다. data_count가 8이 아닌 7이므로 status flag인 full은 1에서 0으로 바뀌고 data가 dout을 통해 정상적으로 출력되었으므로 rd_ack는 1을 출력하며, read operation일 때 레지스터의 주소에 해당하는 head는 1 증가한다.
- 빨간 선으로 나타낸 ⑥을 참고하면 data_count는 감소하고 head는 증가하며 handshake signal인 rd_ack가 1을 출력함으로써 레지스터에 저장되있던 data 0이 2cycle동안 연속적으로 출력되는 것을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑦ 을 참고하면 data_count가 0이고 head와 tail이 같은 값을 가지고 있으며 status signal인 empty가 1인 상태이다. 이로써 fifo 내부 레지스터에 더 이상 data가 존재하지 않고 비어있는 상태인 것은 확인할 수 있다. empty가 1이 되고 그 다음 cycle에서 여전히 read operation이 요청되고 있으므로 handshake signal인 rd_ack는 0이 되고 rd_err는 1이 출력된다.

- ※pg9 1-12. Design Verification (fifo 2) 참고.

- reset_n = 1이고 wr_en이 1이면 클록이 상승에지일 때 din의 값이 tail의 주소에 해당하는 레지스터에 쓰인다. 빨간 선으로 나타낸 ①을 참고하면 data_count는 0에서 1이 되었고 status flag인 empty는 1에서 0이 되었으며 wr_ack는 1이 되었다. 또한 tail은 0에서 1로 증가하였고 reg0에 din의 값 8'h22가 쓰였다.
- 빨간 선으로 나타낸 ②를 참고하면 알 수 있듯이 reset_n이 0으로 활성화 되었을 때 data_count는 0으로 초기화되고 head와 tail, handshake signals는 0의 값을 갖고 status signal인 full은 0, empty는 1의 값을 갖는다.
- 빨간 선으로 나타낸 ③을 참고하면 reset_n이 1이 되고 write operation이 요청되었을 때 din 값 8'h44가 reg0에 쓰이고 data_count와 tail은 1 증가하고 empty는 0, wr_ack는 1이 된다.
- 빨간 선으로 나타낸 ④을 참고하면 fifo가 비어있는 상태에서 write operation을 수행하는 장면이다. head = 1, tail = 1인 상태였는데 tail이 1 증가해서 tail = 2가 되며 din 값 8'h55는 해당 tail을 주소로 갖는 reg1에 저장된다. status flag인 empty는 0이 되고 wr_ack는 1이 된다.
- 빨간 선으로 나타낸 ⑤을 참고하면 read operation이 요청된 상태로 클록이 상승에지일 때 reg1에 저장되어있던 8'h55가 dout으로 출력된다. data_count는 1 감소하여 0이 되고 fifo가 비어있는 상태이므로 empty는 1이 되며 write operation이 아닌 read operation이 요청 되었고 정상적으로 작동하였으므로 wr_ack 는 0이 되고 rd_ack는 1이 출력되며 head는 1증가하여 tail과 같은 값인 2가 된다.

7-2. fifo-top.

- ※pg10 1-13. Design Verification (fifo_top 1) 참고.

- 일단, reset_n이 0이 되면 모든 값들이 0으로 초기화된다.
- 빨간 선으로 나타낸 ①을 참고하면 sel이 1로 fifo_top을 선택된 상태로 wr = 1이고 주어진 address는 8'h11이며 din 값은 8'h11이다. address의 하위 4bits의 값이 1이므로 'U0_fifo' 가 선택되어 해당 register에 din값 8'h11이 저장된다. fifo_cnt는 1로 증가하고 fifo_flag[4]는 empty signal을 나타내는데 write operation을 수행하기 전 잠시 1이었다가 write operation이 수행함과 동시에 0이 된다. 화면을 보면 알 수 있듯이 약간의 tick 현상처럼 보일 수도 있다. 또한, fifo_flag[3]은 해당 wr_ack signal을 나타내며 정상적으로 write operation이 동작했음을 알 수 있다. 또한 F0WR_EN은 'U0_fifo'의 wr_en이 활성화 됨을 보여준다.
- 빨간 선으로 나타낸 ②를 참고하면 write operation이 요청되고 있다. 'U0_fifo'의 내부에 8개의 data가 모두 꽉 찬 상태로 full flag를 나타내는 fifo_flag[5]의 값은 1이 되며 fifo_cnt는 1증가하여 8이 된다. full임에도 불구하고 다음 cycle에도 write operation이 요청되므로 wr_ack를 나타내는 fifo_flag[3]은 0이 되고 wr_err를 나타내는 fifo_flag[2]는 1이 된다.
- 빨간 선으로 나타낸 ③은 sel의 값이 0으로 인해 fifo_top이 선택되지 않은 상태이다. sel이 0일 때는 dout, fifo_cnt, fifo_flag 모두 0을 출력한다.
- 빨간 선으로 나타낸 ④는 sel의 값이 1이고 read operation이 요청되었을 때를 나타낸다. 두 번째로 reg1에 저장되었던 8'h12가 dout으로 출력되며, fifo_cnt는 1 감소하여 6을 나타내며 rd_ack를 나타내는 fifo_flag[1]은 1을 나타낸다.
- 빨간 선으로 나타낸 ⑤는 read operation 요청에 의해 'U0_fifo'에 마지막으로 저장되어 있던 data 8'h18이 dout을 통해 출력되고 fifo_cnt는 0, empty flag를 나타내는 fifo_flag[4]는 1이 되는 화면이다.

1 cycle 후, 'U0_fifo' 내부가 비어있는 상태임에도 불구하고 read operation을 요청하므로 rd_ack를 나타내는 fifo_flag[1]은 0이 되고 rd_err를 나타내는 fifo_flag[0]은 1이 된다.

- ※pg10 1-14. Design Verification (fifo_top 1) 참고.

- 빨간 선으로 나타낸 ①을 참고하면 sel이 1로 fifo_top이 선택되어 있고 wr = 1로 write operation을 요청하는 상태이다. 이 때 해당 fifo의 주소가 아닌 8'h17가 address 값으로 주어졌다. invalid한 입력이므로 fifo_top은 어떠한 동작도 하지 않는다. dout, fifo_cnt, fifo_flag는 해당 값 0을 유지한다.
- 빨간 선으로 나타낸 ②는 write operation이 정상적으로 동작하는 화면이다. address가 11일 때 'U0_fifo'가 선택되며 12일 때 'U1_fifo', 13일 때 'U2_fifo', 14일 때 'U3_fifo'가 선택된다. 각각의 synchronous fifo가 선택되어 해당 레지스터에 din값이 쓰일 때마다 fifo_cnt는 1 증가하며 full 상태가 아니므로 wr_ack를 나타내는 fifo_flag[3]은 1을 나타내고 empty flag를 나타내는 fifo_flag[4]는 0을 유지한다.
- 빨간 선으로 나타낸 ③과 ④는 read operation이 요청 되었을 때 invalid한 주소가 입력되었을 때 fifo_top이 동작하지 않는 화면이다. dout, fifo_cnt, fifo_flag의 모든 값은 0이다.
- 빨간 선으로 나타낸 ⑤는 'U0_fifo'에 8'h80이라는 data가 쓰였지만 reset_n이 0이 되고 reset_n이 다시 1이 된 후 read operation을 요청해본 화면이다. 이 결과, 'U0_fifo'는 fifo_flag[4]가 1임으로 empty상태임을 확인할 수 있고, fifo_flag[0]이 1임으로써 read error가 발생함을 확인할 수 있다. 이 결론으로 reset_n이 0이 되면 모든 fifo의 내부 포인터는 초기화 된다는 것이 증명된다.

7-3. timer

- ※pg11 1-15. Design Verification (timer 1) 참고.

- 빨간 선으로 나타낸 ①은 기본적인 timer 동작 방법이다. 일단 CUR_STATE 상태가 IDLE상태인지 확인한다. (S_address = 8'h26,

그리고 내부 register에 값을 조정해줌으로써 카운트할 수 있는 조건을 만들어준다. CNT_CON(S_address = 8'h22) 레지스터는 0, LOAD_ADDRESS(S_address = 8'h23) 레지스터에 8'h11이 저장되고 CNT_EN (S_address = 8'h20)에 1을 써주었다.

- 빨간 선으로 나타낸 ②는 module timer_master의 master_state이 활성화된 모습이다. 일단 CNT_EN의 값으로 1이 쓰이면 data를 transfer하기 위해 bus에 Master request를 요청한다. testbench의 Master0 request가 0일 경우 timer는 M_grant를 받는다. M_grant가 1이 되면 그 다음 클록에 LOAD_ADDRESS 레지스터에 저장되어 있는 값을 M_address을 통해 내보내고, M_wr과 M_dout의 값 0을 bus로 보낸다. 그 다음 cycle에 M_din로 통해 해당 주소에 저장되어 있는 data가 들어온다. 이 때 M_req는 0이 된다. 그 다음 cycle에 M_req가 0이므로 M_grant 또한 0이 되며, M_din으로 들어온 data는 LOAD_VALUE 레지스터에 저장된다.
- 빨간 선으로 나타낸 ③은 LOAD_VALUE에 저장된 data로부터 카운트를 수행하고 있는 모습이다. COUNT_VALUE 레지스터는 LOAD_VALUE의 값이 0이 아니며 CNT_EN 값이 1일 때 카운트를 수행한다. 카운트를 수행하는 동안에는 reset_n의 값이 0이 되지 않는 한은 카운트를 진행한다. (어떠한 값이 들어와도 무시하고 COUNT VALUE가 0이 될 때까지 카운트 진행)
- 빨간 선으로 나타낸 ④ 는 interrupt가 발생하지 않은 시점에 강제로 INTRRUPT 레지스터에 1을 써준 화면이다. 우리는 카운트가 끝난 후 interrupt가 발생했을 때 그 발생한 interrupt를 clear 하기 위해 INTRRUPT 레지스터를 사용하므로 interrupt가 발생하지 않은 시점에 강제로 1을 입력하더라도 동작하지 않는 것을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑤는 카운트가 수행 중일 때 COUNT_VALUE의 값에 강제로 0을 입력하는 화면이다. 하지만 COUNT_VALUE는 READ ONLY기 때문에 어떠한 입력이 들어오더라도 무시한다. S_wr이 0일 때 S_dout으로 현재 카운트 중인 data가 출력됨으로써 COUNT_VALUE 레지스터는 write operation을 수행할 수 없음이 증명된다.
- 빨간 선으로 나타낸 ⑥은 카운트가 끝나고 (COUNT_VALUE의 값이 0이 된 후) 그 다음 cycle에 interrupt가 발생하는 모습이다.
- 빨간 선으로 나타낸 ⑦은 interrupt를 clear하기 전 CNT_CON 레지스터 값을 설정해주는 화면이다. '첫 번째 ⑦'에서는 현재 CNT_CON 레지스터의 값이 0임을 S_dout을 통해 확인할 수 있다.
  '두 번째 ⑦'에서는 CNT_CON 레지스터에 1을 입력한다.
  '세 번째 ⑦'에서 CNT_CON 레지스터 값을 확인하였는데 '두 번째 ⑦'에서 입력한 1이 출력됨으로써 CNT_CON 레지스터의 값이 현재 1로 유지되는 것을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑧은 CNT_EN 레지스터에 관한 화면이다. 일단 CNT_EN 레지스터는 WRITE ONLY 이기 때문에 CNT_EN 레지스터의 값을 읽는 것은 불가능하다.
  specification에서 주어진 바와 같이 만약 CNT_EN 레지스터 값을 읽도록 요청할 경우 S_dout을 통해 0이 출력된다.('첫 번째 ⑧')
  '두 번째 ⑧' 은 현재 timer의 상태가 IDLE 상태가 아님(interrupt가 발생한 시점이므로 ..)에도 불구하고 CNT_EN에 1을 쓰도록 요청하는 화면이다. 하지만 역시 IDLE 상태가 아니므로 아무런 동작도 하지 않는다.
- 빨간 선으로 나타낸 ⑨는 현재 INTRRUPT 레지스터의 값을 확인하고 interrupt를 clear하는 화면이다. '첫 번째 ⑨'을 통해 현재 INTRRUPT 레지스터의 값이 1임을 확인하고 '두 번째 ⑨'에서 ,해당 INTRRUPT 레지스터에 0을 입력한다.
- 빨간 선으로 나타낸 ⑩은 interrupt가 clear된 후 CNT_CON 레지스터 값이 1이므로 카운트를 다시 하는 화면이다. 카운트를 재수행할 때 COUNT_VALUE는 LOAD_VALUE 레지스터의 값을 받아 1씩 감소하며 카운트한다.

- 빨간 선으로 나타낸 ⑪은 reset_n이 동작하여 timer의 모든 레지스터의 값이 초기화 되는 화면이다.

- ※pg11 1-16. Design Verification (timer 2) 참고.

- 빨간 선으로 나타낸 ①은 기본적인 timer 동작을 나타낸다. 일단 CUR_STATE 레지스터의 값이 IDLE 상태인지 확인한다. 그리고 CNT_CON 레지스터 값으로 0을 해줌으로써 카운트 수행 후 interrupt가 발생하여 클리어 하게 되면 timer는 IDLE 상태에서 그 다음 명령이 들어올 때까지 대기한다. LOAD_ADDRESS 레지스터에 8'h14를 써준 뒤 CNT_EN 값으로 1을 써준다.
- 빨간 선으로 나타낸 ②은 기본적인 timer의 동작으로 M_req가 1이 되고 bus에서 M_grant값으로 1을 받으면 다시 bus로 M_address, M_wr, M_dout의 값을 보내주고 그 다음 클록이 상승에지일 때 M_din을 통해 data를 받아 LOAD_VALUE에 저장한다. ②에서는 현재 M_din의 값 3이 그다음 클록이 상승에지일 때 LOAD_VALUE 레지스터에 저장된 화면이다.
- 빨간 선으로 나타낸 ③은 카운트를 수행하는 화면이다.
- 빨간 선으로 나타낸 ④는 카운트를 수행 후 interrupt가 발생한 화면이다.
- 빨간 선으로 나타낸 ⑤는 현재 CUR_STATE 레지스터의 값을 확인하고 강제로 CUR_STATE에 data를 쓰도록 요청한 후 다시 레지스터의 값을 확인한 결과 CUR_STATE 레지스터 상태에 변화가 없음이 확인되는 화면이다.
- 빨간 선으로 나타낸 ⑥은 INTRRUPT 레지스터에 0을 씀으로써 발생한 interrupt를 clear하고 현재 timer의 상태가 IDLE 상태인지 확인한 후 CNT_EN 값으로 1을 입력한 화면이다.
- 빨간 선으로 나타낸 ⑦은 ②과 같이 bus에 Master request를 요청하고 grant를 받은 후 data를 transfer하는 화면인데 여기서 주목할 점은 입력으로 들어오는 M_din의 값이 0이라는 점이다. M_grant가 0이 되는 시점에 M_din의 값 0은 LOAD_VALUE 레지스터에 저장되는 것을 확인할 수 있다. LOAD_VALUE에 저장된 값이 0일 때 카운트를 수행하지 않는다. 그러므로 COUNT_VALUE 레지스터에는 값이 전달되지 않는다.
- 빨간 선으로 나타낸 ⑧은 LOAD_VALUE 레지스터의 값이 무엇인지 확인하는 화면이다. 원래 S_dout 값은 0이었으므로 화면에는 출력이 되지 않은 것처럼 보이지만 현재 LOAD_VALUE 레지스터의 값은 0이므로 S_dout을 통해 출력된 것이다.
- 빨간 선으로 나타낸 ⑨는 LOAD_ADDRESS 레지스터에 저장된 값이 무엇인지 확인하는 화면이다. 현재 LOAD_ADDRESS 레지스터에 저장된 값은 8'h14이므로 S_dout을 통해 8'h14가 출력된다.

7-4. bus.

- ※pg12 1-17. Design Verification (bus)참고.

- 빨간 선으로 나타낸 ①은 reset_n이 0일 때 기본적으로 grant는 M0이 받는 것을 증명한다. bus에서 M0는 testbench를 의미하고, M1은 timer를 의미하며 S0은 fifo_top을 의미하고 S1은 timer를 의미한다.
- 빨간 선으로 나타낸 ②는 reset_n이 1일 때 M0_address, M0_dout, M1_address, M1_dout이 bus의 input port로 입력된 상태이다. 현재 grant는 M0가 받고 있으므로 S_address는 M0_address의 값 13을 출력하고 S_din은 M0_dout의 값 24를 출력한다. 또한, S_address의 상위 4bits의 값이 1이므로 S0_sel은 1이 된다. M0_wr 은 0이므로 S_wr은 0이 된다.
- 빨간 선으로 나타낸 ③은 M0_req는 0일 때 M1_req가 1로 요청되어 M1이 grant를 받는 모습이다.
- 빨간 선으로 나타낸 ④는 input S0_dout과 S1_dout 의 값으로 각각 8'h11와 8'h22가 입력된 화면이다. 현재 S0_sel이 1이므로 S0_dout의 값 8'h11이 M_din으로 출력된다. 또한 클록이 상승에지일 때 M1가 grant를 받기 때문에 S_address는 M1_address의 값인 8'h14가 되고 S_din은 M1_dout의 값 8'h16이 된다.

또한, S_wr은 M1_wr이 1이므로 1이 된다.

- 빨간 선으로 나타낸 ⑤는 M0_req가 1이 되지만 이미 M1_req가 1인 상태로 M1이 grant를 받고 있으므로 grant에 있어 변화는 없다.
- 빨간 선으로 나타낸 ⑥은 M1_address의 값이 바뀐 화면이다. 현재 M1이 grant를 받고 있으므로 S_address는 M1_address의 값을 갖는다. S_address의 값은 8'h34로써 fifo_top과 timer의 주소가 아닌 invalid한 주소이므로 S0_sel과 S1_sel은 둘 다 0을 출력한다. 또한 S0_sel과 S1_sel의 값이 둘 다 0이므로 M_din은 S0_dout과 S1_dout 값에 상관없이 무조건 0을 출력한다. (클록이 상승에지일 때 출력)
- 빨간 선으로 나타낸 ⑦은 reset_n이 0일 때 M0가 grant를 받는다는 것을 알 수 있다.
- 빨간 선으로 나타낸 ⑧은 ⑦에 의해 M0가 grant를 가지므로 S_address의 값은 M0_address의 값인 8'h13이 되고 S_wr은 M0_wr과 같은 0이 되며, S_din은 M0_dout과 같은 8'h24를 갖는 화면이다. 또한 S_address가 8'h13으로 상위 4bits가 1이므로 S0_sel은 1이 된다.
- 빨간 선으로 나타낸 ⑨는 ⑧에서 S0_sel이 1로 활성화 됨에 따라 S0_dout의 값이 M_din으로 출력되는 화면이다.
- 빨간 선으로 나타낸 ⑩은 M0_req와 M1_req가 둘 다 0인 상태에서 동시에 1이 될 경우 M0가 grant를 갖는 화면이다.
- 빨간 선으로 나타낸 ⑪은 M0_address와 M1_address의 값이 각각 8'h24와 8'h14로 입력됨에 따라 S_address의 값이 M0_address의 값인 8'h24가 되고 S_address의 상위 4bits의 값이 2이므로 S0_sel은 0이되고 S1_sel은 1이 됨으로써 클록이 상승에지일 때 S1_dout의 값인 8'h22가 M_din으로 출력되는 화면이다.

7-5. top.

- ※pg12. 1-18. Design Verification (top1)참고.

- 빨간 선으로 나타낸 ①은 현재 testbench (M0) 가 grant를 받고 있을 때 M0_address의 값으로 8'h11이 입력되어 bus의 output port인 S_address가 8'h11이 되고 S_address의 상위 4bits의 값이 1이므로 S0_sel이 1이 된 화면이다.
- 빨간 선으로 나타낸 ②는 'U0_fifo'에 첫 번째 data 값 8'h03이 쓰임으로써 empty flag를 나타내는 fifo_flag[4]는 0을 유지하고 wr_ack를 나타내는 fifo_flag[3]은 1을 나타내며, fifo_cnt가 1 증가하는 화면이다.
- 빨간 선으로 나타낸 ③은 ②에서의 동작을 수행할 때 'U0_fifo' 의 첫 번째 register인 reg0에 03이라는 data가 저장된 화면이다.
- 빨간 선으로 나타낸 ④는 'U0_fifo'에 총 8개의 data가 참으로 인해 full flag를 나타내는 fifo_flag[5]는 1이 되고 꽉 차있는 상태임에도 불구하고 M0_wr은 1로써 write operation을 요구하므로 그 다음 cycle에 wr_ack를 나타내는 fifo_flag[3]은 0이 되고, wr_err를 나타내는 fifo_flag[2]는 1이 되는 화면이다.
- 빨간 선으로 나타낸 ⑤는 8'h12의 주소를 갖는 'U1_fifo'에 data 값 0을 저장하는 화면이다. 클록이 상승에지에 fifo_cnt가 1로 증가하고 wr_ack를 나타내는 fifo_flag[3]이 1이 되므로 data 값 0이 정상적으로 저장된 것을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑥은 M0_address의 값으로 a synchronous fifo의 주소나 timer register의 주소가 아닌 invalid한 주소를 입력하였을 때를 나타낸다.
  module 'bus' 관점에서 invalid한 주소가 들어왔을 때를 분석해보겠다. 일단 처음에 입력한 주소 8'h15는 상위 4bits의 값이 1이므로 bus에서는 fifo_top을 나타내는 S0_sel의 값을 1로 유지한다. 그리고 두 번째 invalid한 주소 8'h36은 상위 4bits의 값이 3이므로 Slave 1인 timer나 Slave 0인 fifo_top의 해당 주소가 아니므로 S0_sel과 S1_sel모두 0의 값을 갖게 된다.

- 빨간 선으로 나타낸 ⑦은 M0_address의 값이 8'h14일 때 'U3_fifo'의 저장된 data를 출력하는 화면이다. bus와 fifo, top 관점에서 전체적인 흐름에 대해 알아보겠다. 현재 전체적인 grant는 testbench(M0_grant)가 받고 있으므로 입력한 값 M0_address값이 S0_address로 보내진다. 이 때 S0_address의 상위 4bits의 값이 1이므로 S0_sel은 값 1을 유지하고 그에 의해 'U3_fifo'에 저장된 data 8'h44는 bus의 input port인 S0_dout으로 들어온다. 그리고 S0_dout 값은 bus의 output port인 M_din을 통해 top의 output port M_din으로 출력된다. data가 읽힌 후, 'U3_fifo'에 더 이상 저장된 data가 존재하지 않으므로 empty flag를 나타내는 fifo_flag [4]는 1을 출력하고 rd_ack를 나타내는 fifo_ flag[1]은 1을 출력한다. fifo 내부가 비어있음에도 불구하고 계속 read operation을 요청하므로 fifo_flag[4]는 1로 유지되고 rd_ err를 나타내는 fifo_flag[0]은 1을 출력한다.

- ※pg13. 1-19. Design Verification (top2)참고.

- 빨간 선으로 나타낸 ①은 CUR_STATE 레지스터의 값을 읽어 현재 상태가 IDLE상태인 것을 확인하고 CNT_CON 레지스터의 값은 0, LOAD_ADDRESS의 값은 8'h11로 'U0_fifo'의 주소를 입력한 후 CNT_EN에 1을 넣어줌으로써 카운트를 하기 위한 조건이 충족된 화면이다.
  이 때 bus의 관점에서 살펴보면 현재 testbench(M0)가 grant를 받고 있으므로 S_address의 값으로 현재 입력된 CUR_STATE 레지스터, CNT_CON 레지스터, LOAD_ADDR ESS레지스터, CNT_EN 레지스터의 주소를 가지며 이 S_address의 상위 4bits의 값은 2이므로 S0_sel은 0이되고 S1_sel은 1이 된다.
- 빨간 선으로 나타낸 ②는 timer에서 bus에 request를 요청한 화면이다.
- 빨간 선으로 나타낸 ③은 M0가 request를 요청하지 않는 상태에서 M1 request 요청에 의해 bus로 부터 grant를 허락받은 화면이다.
- 빨간 선으로 나타낸 ④는 timer가 bus로부터 grant를 받은 그 다음 클록에 bus로 M1_wr, M1_address, M1_dout을 내보내는 화면이다. 이 때 M1_address의 값은 8'h11이고 현재 timer가 grant를 받고 있으므로 S_address는 8'h11이 된다. S_address의 상위 4bits가 1이므로 S0_sel이 1이 되고 S1_sel은 0이 된다.
- 빨간 선으로 나타낸 ⑤는 top의 output port인 M_din으로 timer의 LOAD_ADDRE SS를 주소로 갖는 'U0_fifo'의 data 8'h03을 출력한 화면이다. 이 data는 timer의 input M_din으로 입력된다. 또한, 해당 fifo로부터 1개의 data가 읽힘으로써 fifo_cnt는 1 감소하여 7이 되고, 정상적으로 읽혔으므로 rd_ack를 나타내는 fifo_flag[1]은 1이 된다.
  이제 bus의 관점에서 살펴보겠다.
  일단, input S0_dout에 8'h03이 저장된다. (fifo_top으로부터 받음)
  bus에서는 M1_address의 값이 00으로 변환되는데 그 이유는 timer에서 한 cycle 동안만 bus로 LOAD_ADDRESS 레지스터 값을 M1_address로 내보내기 때문이다. 그 외의 경우에는 0을 내보낸다. 이에 의해 S_address의 값 또한 8'h00이 되고 S_address의 상위 4bits 값이 0이므로 S0_sel과 S1_sel은 모두 0의 값을 갖는다.
- 빨간 선으로 나타낸 ⑥은 현재 S0_sel과 S1_sel의 값이 0이므로 M_din값으로 0을 출력하는 화면이다. 또한 S0_sel이 0이므로 fifo_top이 선택되지 않은 상태이므로 fifo_cnt와 fifo_flag는 모두 0이 된다. 또한, S0_sel이 0이므로 (fifo_top이 선택되지 않았다는 의미) fifo_top의 output dout은 0을 출력하고 이 값은 S0_dout의 입력으로 들어온다.
- 빨간 선으로 나타낸 ⑦과 ⑧은 LOAD_VALUE로부터 data를 받아 count하는 화면이다.
- 빨간 선으로 나타낸 ⑩은 COUNT_VALUE의 값이 0이 되고 그 다음 cycle에 interrupt가 발생한 화면이다.
- 빨간 선으로 나타낸 ⑨는 timer의 레지스터에 강제로 입력을 하고 값이 변환되었는지 확인하는 장면이다.

일단, interrupt가 발생하지 않은 시점에 INTRRUPT 레지스터에 1을 씀으로써 강제로 interrupt가 발생하도록 해보았지만 여전히 interrupt는 0의 값을 유지하므로 영향을 미치지 않는 것을 확인할 수 있다.

카운트가 끝나고 interrupt가 발생한 후 'TIMER' CUR_STATE 을 확인하면 8'h02 의 값을 가지므로 IDLE상태가 아닌 것을 확인할 수 있다. 이 상황에서 CNT_EN레지스터에 1을 쓰면 M1_req를 요청할 지에 대해 실험해본 결과 아무런 영향도 미치지 않았다.

그리고 INTRRUPT 레지스터에 0의 값을 써줌으로써 interrupt를 clear하였다. (⑩에서 timer_interrupt의 값이 0이 됨)

LOAD_ADDRESS 레지스터에 'U1_fifo'의 주소인 8'h12 를 저장한 후 CNT_EN에 1을 씀으로써 그 다음 cycle에 M1_req를 요청한다.

- 빨간 선으로 나타낸 ⑪은 'U1_fifo'로부터 정상적으로 data를 읽어옴으로써 rd_ack를 나타내는 fifo_flag[1]은 1이 되는 화면이다. 여기서 중요한 점은 'U1_fifo'로부터 read해온 data의 값이 '0'이라는 점이다.
- 빨간 선으로 나타낸 ⑫에서 LOAD_VALUE 레지스터에 0이 저장되는 것을 확인할 수 있고 LOAD_VALUE의 값이 0일 때는 카운트를 수행하지 않는 것을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑬은 LOAD_ADDRESS 레지스터에 8'h12가 저장되어 있는 것을 알 수 있고 LOAD_VALUE, COUNT_VALUE의 값이 0인 것을 확인할 수 있다.

- ※pg14. 1-20. Design Verification (top3)참고.

- 빨간 선으로 나타낸 ①은 LOAD_ADDRESS 레지스터에 8'h35라는 invalid한 주소를 입력하는 화면이다. (주소 상위 4bits의 값이 1이나 2가 아닐 때 동작을 어떻게 할지에 대해 검증)
- 빨간 선으로 나타낸 ②는 현재 M0가 request를 요청하는 상태에서 grant를 받고 있을 때 timer에서 CNT_EN 레지스터에 1이 쓰여 M1_req가 요청된 화면이다. M0_req가 0이라면 M1_req가 요청되고 timer는 바로 다음 cycle에 grant를 받을 수 있겠지만 현재 상태는 M0_req가 1이므로 M0_req가 0이 되기 전까지는 timer가 grant를 가질 수 없다.
- 빨간 선으로 나타낸 ③은 timer가 grant를 받고 있는 상황에 M0_req가 0에서 1로 바뀌지만 여전히 grant는 M1이 가지고 있는 화면이다. (M1_req가 1인 상태이므로..)
- 빨간 선으로 나타낸 ④는 timer의 LOAD_ADDRESS 레지스터, LOAD_VALUE 레지스터, CUR_STATE 레지스터에 강제로 값을 입력해보지만 read only인 이 세 개의 레지스터의 값은 변하지 않음을 확인할 수 있다.
- 빨간 선으로 나타낸 ⑤는 LOAD_ADDRESS의 값 8'h35가 S_address에 쓰여 짐에 따라 그다음 클록에 timer의 input M_din으로 0이 입력되고 최종 module top의 output M_din으로 0이 출력되는 화면이다.
- 마지막 빨간 선으로 나타낸 ⑥은 M_din으로 들어온 0의 값이 LOAD_VALUE에 저장되는 화면이다. LOAD_VALUE의 값은 0이므로 카운트는 동작하지 않는다.

- ※pg15. 1-21. Design Verification (top4)참고.(※ **INVALID한 입력 - LOAD_ADDRESS 값이 8'h23일 때**)

- 빨간 선으로 나타낸 ①은 timer 내부 레지스터인 LOAD_ADDRESS 레지스터에 LOAD_ADDRESS 레지스터의 주소와 똑같은 값 8'h23이 입력되는 화면이다.
- 빨간 선으로 나타낸 ②는 LOAD_ADDRESS 레지스터에 입력 값 8'h23이 저장된 모습이다.
- 빨간 선으로 나타낸 ③은 해당 주소에 저장된 data를 read해오기 위해 CNT_EN 레지스터에 1을 입력하는 모습이다.

- 빨간 선으로 나타낸 ④는 timer의 output M_address를 통해 빠져나간 값 8'h23이 'bus 내부' 에서 input M1_address를 통해 들어오고 S_address에 8'h23이 저장되는 모습이다.
  ※ 여기서 주의할 점은 현재 S_address의 상위 4bits의 값이 2이므로 S1_sel이 1이 되고 이에 의해 bus관점에서 보았을 때 timer가 slave상태가 된다. 전체적인 시점에서 보았을 때 현재 M1이 grant를 받고 있으므로 bus의 output인 S_wr은 0, S_address 는 8'h23이 된다.
- 빨간 선을 나타낸 ⑤를 주목해보자. 위에서 S_wr의 값 0과 S_address의 값 8'h23는 timer input port인 S_wr, S_address값으로 들어온다. 이 때, timer의 S_sel은 1로 활성화 되고 S_wr는 0이 되며 S_address는 8'h23이 된다. Timer 내부에서 8'h23의 주소를 갖는 것은 LOAD_ADDRESS 레지스터이므로 LOAD_ADDRESS 레지스터 내부에 저장된 값 8'h23은 Timer의 output S_dout을 통해 빠져나와 bus의 S1_dout으로 전달된다. 전체적인 관점에서 보았을 때, M1이 grant를 받고 있으므로 S1_dout 값이 bus output port인 M_din으로 빠져나가고 이 값은 최종적으로 timer의 input port M_din으로 입력되는 것이다.
- 빨간 선을 나타낸 ⑥에서 M_din으로 입력받은 8'h23이 LOAD_VALUE 레지스터에 저장된다.
- 빨간 선을 나타낸 ⑦에서 COUNT_VALUE에 LOAD_VALUE의 값 8'h23이 저장되고 1씩 감소하며 카운트한다.
- 빨간 선을 나타낸 ⑧에서 reset_n이 0이 됨으로써 Timer 내부 레지스터가 초기화되는 것을 확인할 수 있다.
- 이와 같이 만약 timer LOAD_ADDRESS 레지스터에 LOAD_ADDRESS 레지스터 주소와 같은 값 8'h23을 저장하고 CNT_EN을 1로 활성화 할 경우 timer는 값을 자기 자신으로부터 읽어와 카운트를 수행함을 알 수 있다. testbench에서 값의 입력을 넣어줄 때 이와 같이 invalid한 경우를 고려하여야 겠다.

8. Conclusion.

- Synchronous fifo.

- Synchronous fifo의 동작은 위의 specification에서 주어진 바와 같이 write operation과 read operation와 status flags의 상태에 따라 값이 입력되거나 읽혀질 수 있다. 요청된 operation에 승낙했는지 거절했는지에 대해서는 handshake signals를 통해 알 수 있다.

- fifo_top

- fifo_top은 위의 specification에서 주어진 바와 같이 4개의 fifo가 instance 되어 있는 상태로 외부에서 선택한 fifo에 write operation을 수행하고 read operation을 수행한다. 각각의 instance된 module 이름은 'U0_fifo', 'U1_fifo', 'U2_fifo', 'U3_fifo'이고 해당 fifo의 주소는 각각 8'h11, 8'h12, 8'h13, 8'h14이다.
- bus에서 S_address의 상위 4bits 값이 1일 경우 fifo_top이 선택되며, fifo_top의 input인 sel은 1이 된다.

- Timer

- Timer는 LOAD_ADDRESS 레지스터에 저장된 값을 주소로 하여 data를 읽어 들이고 그 값으로부터 카운트를 수행하는 module이다. CNT_EN 값이 1이 되면 Timer는 bus에 Master를 달라고 요청하며 testbench의 M1_req가 0일 때에만 bus는 Timer에게 grant를 1로 보내준다. 이 grant는 bus를 통해 data transfer 하는 것을 허락하는 것이다. 이 때 timer는 bus에게 해당 주소를 보내주는데 해당 주소의 data를 읽어 와서 카운트를 수행하며 만약 bus에 보내는 M_address의 상위 4bits가 1 혹은 2가 아닌 다른 경우일 때 timer는 M_din값으로 0을 갖는다. 정상적으로 작동할 경우 timer는 fifo로부터 값을 읽은 후 카운트를 진행한다. 카운트된 값이 0이 되면 외부에 interrupt를 날림으로써 사용자에게 카운트가 끝났음을 알려준다.

Timer 내부 레지스터는 specification에서 주어진 바와 동작하며, 주의할 점은 testbench에서 내부 레지스터에 값을 write할 경우, LOAD_ADDRESS는 11~14 사이의 값을 기입하여 fifo_top 내부 fifo 중 한 개가 선택될 수 있도록 하고 Timer에서 bus로 master request를 요청하고 싶다면 Timer 전체 내부 상태가 ILDE 상태일 때에만 CNT_EN에 1을 write 해야 한다는 점이다. 또한, 해당 레지스터에 값을 쓸 수 없고 읽기만 지원하는 레지스터에는 웬만하면 write 을 시도하지 않는 것이 좋다.

- Bus

- Bus는 간단히 data를 transfer할 수 있는 중간 매개체로 기본적으로 testbench에게 grant를 주어 testbench의 명령을 따른다. 여기서 이해하고 넘어갈 점은 Bus에서 M0은 testbench를 뜻하며, M1은 timer를 의미하고 S0은 fifo_top을 의미하며 S1은 timer를 의미한다. Bus에 M1_req가 요청되는 경우는 timer의 내부 레지스터 CNT_EN에 1이 쓰였을 경우이며 이 때에도 testbench(M0)의 M0_req가 1일 경우에는 timer가 grant를 받을 수 없다.
- 만약, M0가 grant를 받고 있는 상황이라면 bus의 output인 S_wr, S_address, S_din은 testbench에서 입력된 M0_wr, M0_address, M0_dout값이 된다. 여기서 S_address의 상위 4bits를 해독하여 1일 경우 S0_sel값이 1이 되고 2일 경우 S1_sel값이 1이 되며 그 외의 값을 가질 경우 S0_sel과 S1_sel값은 0이 된다. 이 S0_sel과 S1_sel의 값에 따라 bus의 input으로 들어오는 S0_dout과 S1_dout중 어떠한 값이 output M_din으로 빠져나갈지 결정된다. S0_sel이 1이라면 S0_dout 값이 M_din으로 빠져나가고, S1_sel이 1이라면 S1_dout 값이 M_din으로 빠져나가며 그 외의 경우에 M_din은 0의 값을 갖는다.

- 최종결론.

- 이번 프로젝트에서 생각보다 invalid한 입력이 들어오는 경우가 많았다. 이번 프로젝트를 수행한 의도는 bus라는 중간 매개체를 이용하여 testbench에서 입력한 값을 주소를 갖는 4개의 fifo 중 한 개를 선택하여 data를 입력하고 그 data를 가지고 timer에서 카운트를 수행하며 카운트가 끝난 후 interrupt를 발생하게 하는 목적을 갖고 있다는 생각이 들었다. 하지만 마지막 verification에서 얻은 결과와 같이 만약 timer의 내부 레지스터인 LOAD_ADDRESS의 값으로 LOAD_ADDRESS 레지스터의 주소 값과 같은 8'h23일 경우, timer는 fifo로부터 data를 read해오는 것이 아니라 자기 자신 LOAD_ADDRESS 레지스터에 저장된 주소 값을 가지고 카운트를 할 수도 있으므로 사용자가 testbench에서 입력 값을 넣을 때 이러한 경우를 고려해야 된다는 결론을 내릴 수 있었다.