컴퓨터 공학 기초 실험2 보고서

실험제목: Memory & Bus

실험일자: 2023년 11월 20일 (월)

제출일자: 2023년 11월 22일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적

A. 제목

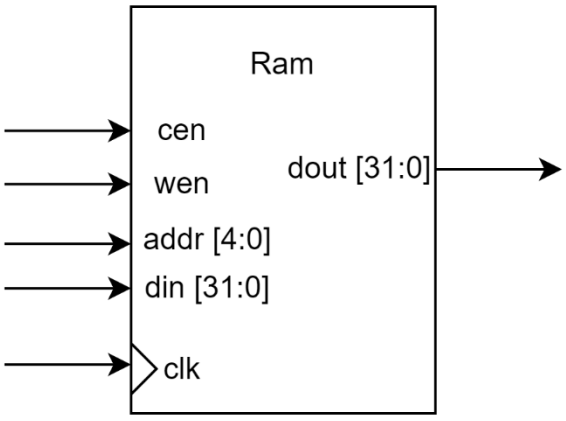
Memory & Bus

B. 목적

Memory에 대해 이해하고 이를 32bit Memory 32개로 설계하여 검증해본다. Bus의 구조에 대해 공부하고 이를 이용하여 Arbiter, Address decoder등의 각 module들을 설계하도록 한다. 이들을 instance하여 전체적인 Bus를 만들고 검증하는 것을 목적으로 한다.

2. 원리(배경지식)

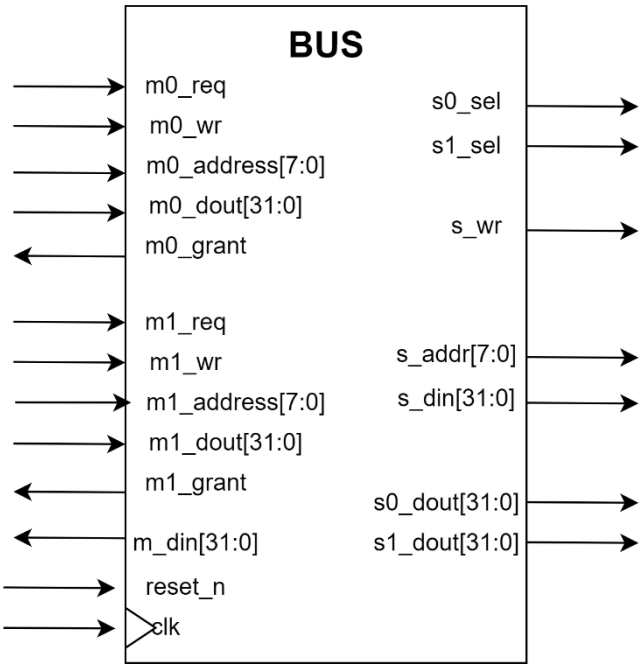
A. Memory



Address에 기반하여 각 데이터들을 해당 주소에 저장할 수 있는 하드웨어를 Memory라고 한다. 저장된 데이터들을 순차적이 아닌 임의의 순서로 접근할 수 있으며, 다른 유형의 메모리 장치들 같은 경우에는 미리 정해진 순서로만 데이터에 접근할 수 있다는 특징을 가지고 있다. RAM은 각 작업의 진행 상황을 기억하는 동시에 빠르게 전환할 수 있도록 해주기 때문에, 메모리 용량이 커질수록 더욱 빠르고 간편하게 정보에 접근할 수 있다.

일반적으로 Ram은 SRAM(static ram), DRAM(dynamic ram)으로 나뉜다. SRAM은 전원이 꺼지더라도 데이터를 보존할 수 있어 오래 유지하는 용으로 사용된다. 각 bit는 filp-flop으로 구성되어 있기 때문에 데이터를 저장할 수 있는 것이다. 따라서 빠른 속도를 가지지만 비용이 높다는 단점이 있다. DRAM은 일정 시간 동안만 저장하는 용으로 사용되며, 각 bit는 capacitor와 transistor로 구성되어 있다. 비용이 상대적으로 낮고 다량의 데이터를 저장할 수 있지만 속도가 느리다는 단점이 있다.

B. Bus



여러 개의 component들 간에 데이터를 전송할 수 있도록 연결해주는 component를 Bus라 한다. 컴퓨터의 구성요소들을 서로 연결하고 데이터 전달을 위한 경로가 되며, 주소, 데이터, 제어 버스로 구성된다. Address BUS는 메모리의 주소나 I/O의 포트들을 전달하고 주소 전달은 CPU에서 메모리로만 가능하다.

Data Bus는 각 구성요소들에서 양방향으로 데이터 전달이 가능한 버스를 사용한다. Control Bus는 Read, Write의 제어 신호가 전달되며 마찬가지로 양방향으로 데이터 전달이 가능한 Bus를 사용한다. CPU와 I/O unit간에는 input, output, inputerrupt가 있다. input은 Read신호를 전달하고 output은 Write신호를 전달한다. inputerrupt는 요청을 보낼 시 CPU에 입출력 작업을 요청할 수 있다. 요청을 하면 I/O unit이 CPU에 입출력 작업을 시작할 것을 요청하고, 확인을 하면 CPU가 입출력 동작을 수행할 것을 I/O unit에게 알리는 형식이다. 이러한 BUS는 새로운 component들을 추가하기가 쉽고 가격이 저렴하다는 장점을 가지고 있다.

3. 설계 세부사항

A. Memory

Address는 5bit으로, Data는 32bit으로 하여 32개의 Ram을 생성해 저장한다. 이때 초기의 Ram은 for문을 이용하여 0값으로 초기화를 시켜준다. 신호에 따른 상태 변화는 다음과 같이 설계한다.

cen && wen	write, dout=0
cen && !wen	read
!cen	dout=0

-I/O

input	clk	Clock
	cen	Chip enable
	wen	Write enable
	addr [4:0]	Address
	din [31:0]	Data in
output	dout [31:0]	Data out

B. Bus

master, slave가 각각 2개씩 있으며 address는 8bit이고 data는 32bit으로 한다. 이때 slave가 가지는 주소의 범위는 다음과 같이 설계한다.

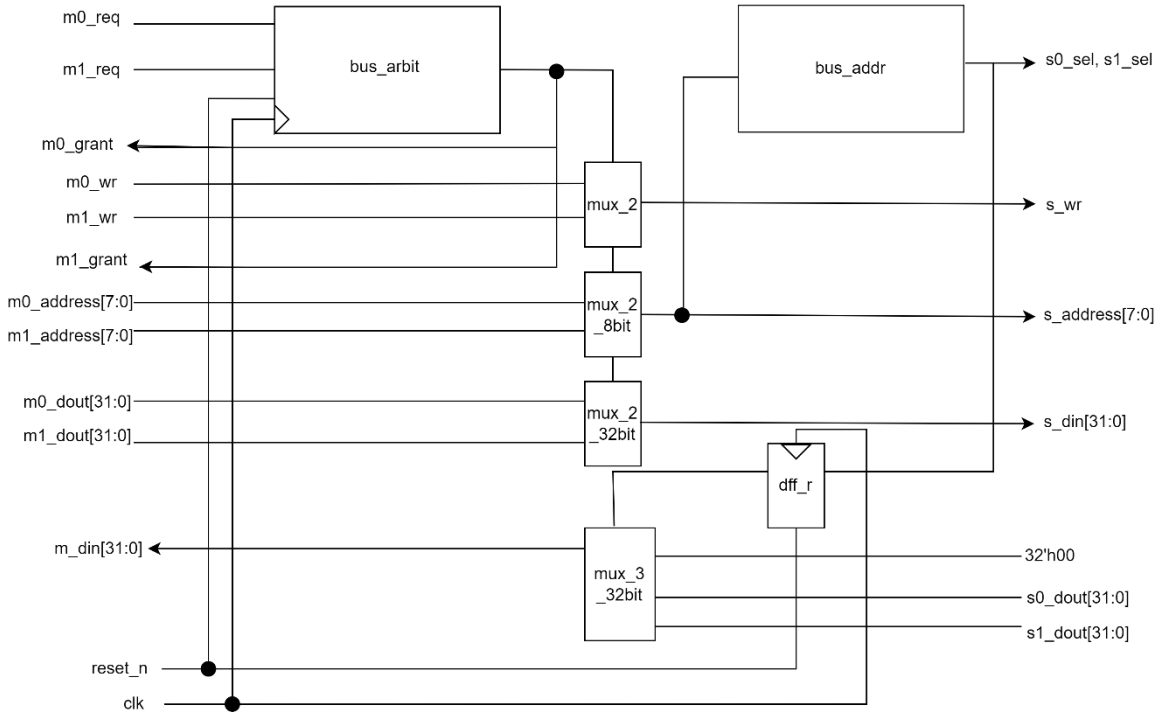
slave 0	0x00~0x1F
slave 1	0x20~0x3F

-I/O

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 [31:0]	Master 0 data output
	m1_req	Master 1 request
	m1_wr	Master 1 write/read
	m1_address [7:0]	Master 1 address

	m1_dout [31:0]	Master 1 data out
	s0_dout [31:0]	Slave 0 data out
	s1_dout [31:0]	Slave 1 data out
output	m0_grant	Master 0 grant
	m1_grant	Master 1 grant
	m_din [31:0]	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 [31:0]	Slave data input

-회로도

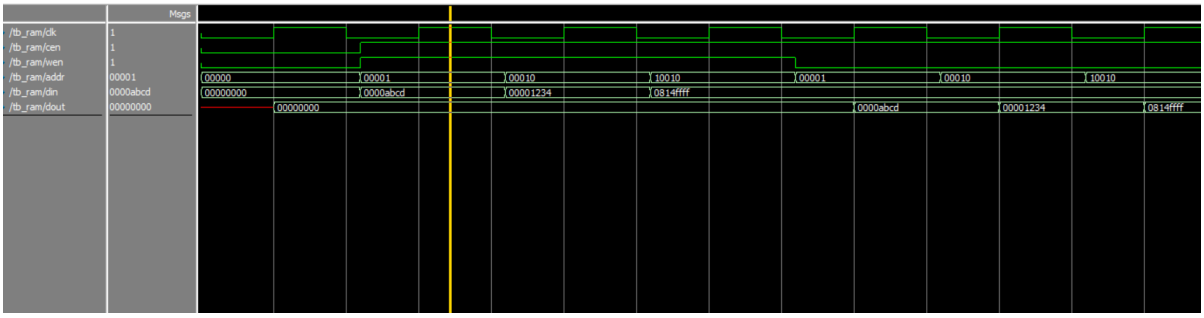


4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

1. Memory

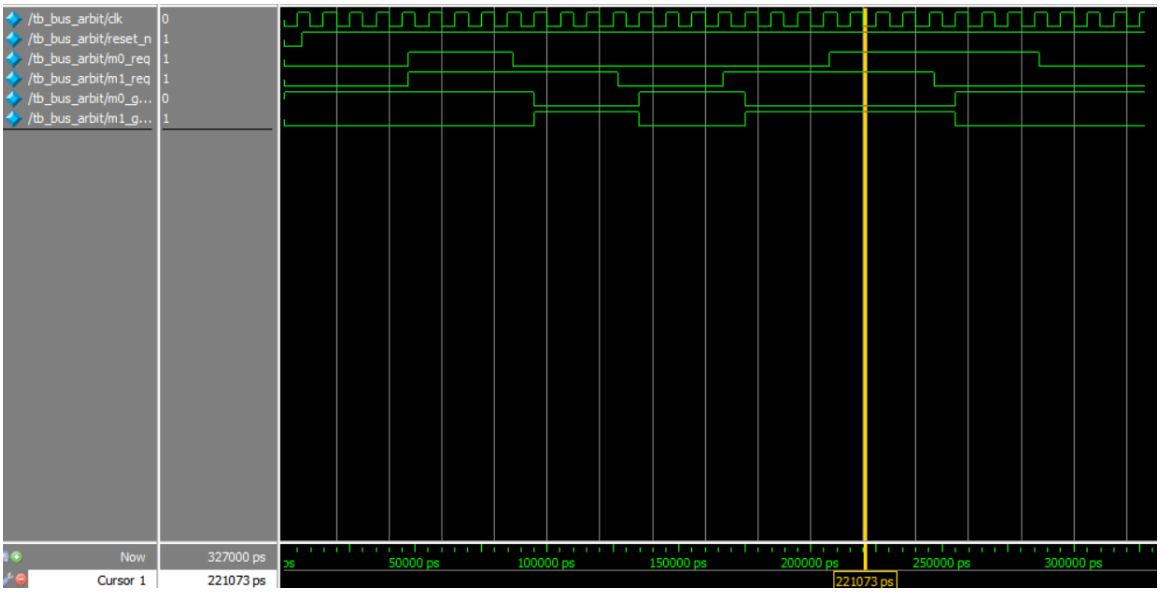
처음에 cen이 0인 상태에서는 dout이 0으로 출력된다. 이후 cen이 1이 되고 wen도 1이 되면 5bit의 address를 받아서 해당하는 주소에 din값을 write한다. 이때 동안에도 dout은 계속 0값을 유지한다. 그리고 나서 wen이 0이 되면 read 모드로 전환되어 해당하는 주소의 저장된 값을 dout으로 출력하게 된다.



2. Bus

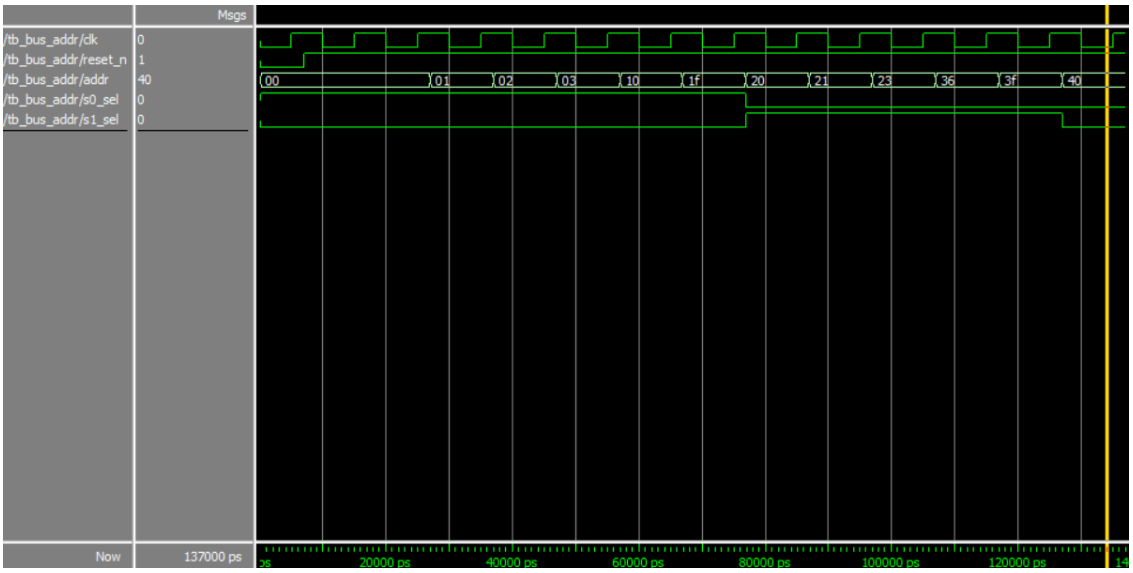
1) bus_arbit

Bus의 arbiter를 검증한 결과로, 처음에 reset_n이 0인 동안에는 M0 grant가 1이 된다. reset_n이 1이 된 후 상태를 유지하다가 m0_req, m1_req 둘 다 1이 되면 마찬가지로 상태를 유지한다. 그 후 m0_req가 0이 되면 다음 cycle때 m1_grant로 넘어가며, 둘이 바뀌게 되면 m0_grant로 넘어간다. 그리고 다시 m0_req가 0이 되고 m1_req가 들어오면 m1_grant로 바뀌는데, 이때 m1인 상태에서 m0_req가 켜지더라도 m1_req가 먼저 1인 동안에는 bus의 소유권을 빼앗기지 않고 계속 m1상태를 유지할 수 있게 되는 걸 알 수 있다.



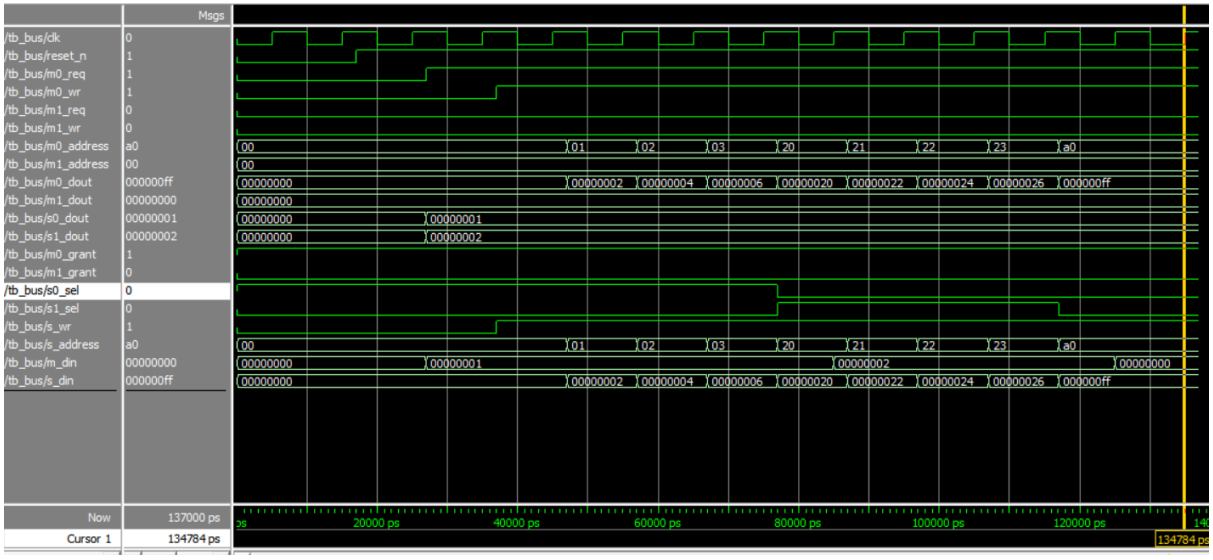
2) bus_addr

Bus의 Address Decoder를 검증한 결과로, address의 주소에 따라 slave0 또는 slave1이 선택되는 모습이다. 8bit Address가 0x00~0x1F의 범위일 때는 s0_sel이 선택되며, 0x20~0x3F 범위일 때는 s1_sel이 선택된다. 마지막처럼 범위를 벗어난 주소가 들어올 시에는 아무것도 선택되지 않게 한 것을 볼 수 있다.



3) bus

top module인 bus를 검증한 결과로, 각 input값에 따라 output(m0_grant, m1_grant, m_din, s0_sel, s1_sel, s_address, s_wr, s_din)들이 나온 모습이다. 선택된 grant의 address에 따라 slave가 선택된다. 여기서는 m1 grant가 계속 선택되어 해당 address에 따라서 00~03까지는 s0_sel이, 20~23까지는 s1_sel이 선택된 것을 볼 수 있다. 마지막에 a0와 같이 범위를 벗어난 주소 값이 들어올 때는 어떤 slave도 선택되지 않으므로, 둘 다 0으로 되며 m_din 또한 0값으로 나오게 된다.

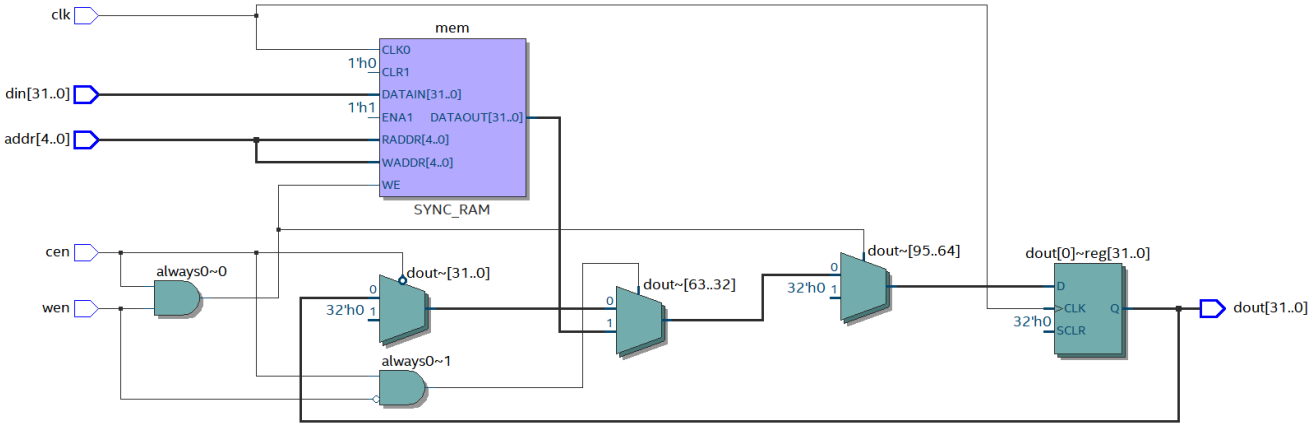


B. 합성(synthesis) 결과

1. Memory

-RTL viewer

처음에 memory를 초기화 해주고 그 다음 cen과 wen신호에 따라 dout값이 결정되는 형태이다.



-Flow summary

전체 회로의 크기와 register 수를 볼 때 상당히 큰 편이라는 것을 알 수 있다.

Flow Summary

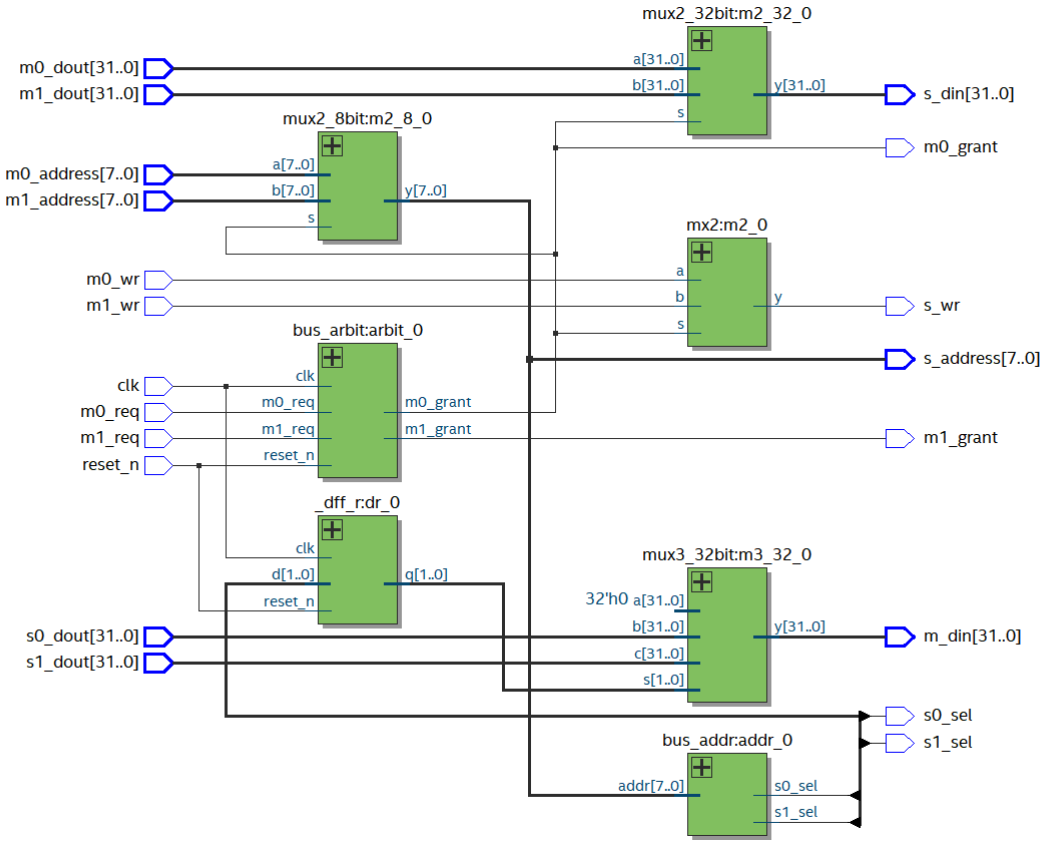
<<Filter>>

Flow Status	Successful - Sat Nov 18 22:27:20 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ram
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	527 / 41,910 (1 %)
Total registers	1056
Total pins	72 / 499 (14 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

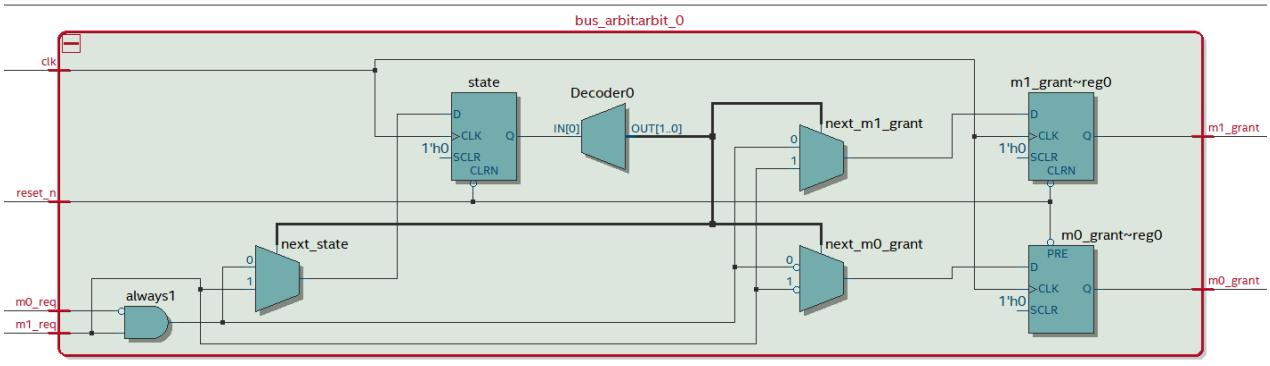
2. Bus

-RTL viewer

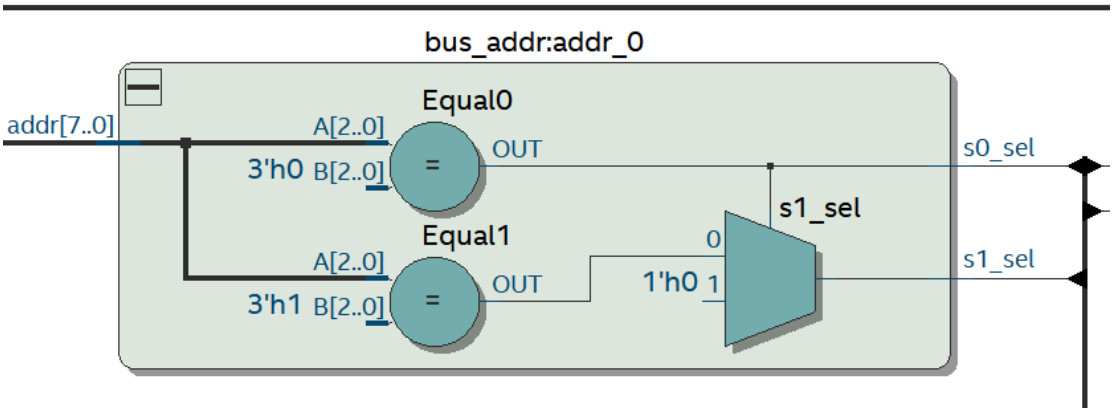
각 req들이 arbiter에 전달되어 선택된 grant의 정보를 select 신호로 하여 3개의 mux로 들어간다. 여기서는 m0_grant의 값이 0인지 1인지에 따라 구분하도록 설계하였다. 각 wr이 mux로 들어가 선택된 값이 address decoder로 들어가고, 이 값을 정보로 mux3으로 들어간다. 여기서는 {s0_sel, s1_sel}로 하나로 묶어서 register에 전달한 다음에 나온 q값을 전달하도록 설계하였다.



-Arbiter



-Address decoder



-Flow summary

전체 회로의 크기를 볼 때, 작은 편이라는 것을 알 수 있다.

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Nov 21 14:58:52 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	bus
Top-level Entity Name	bus
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	55 / 41,910 (< 1 %)
Total registers	5
Total pins	227 / 499 (45 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

처음에 Bus의 arbiter를 설계할 때 state가 전환되는 부분에서, 전환 후에 next grant값을 바꾸도록 설계하였다. 이런 식으로 하였더니 grant값이 바뀌는데 2cycle이 걸리는 문제가 생겼었다. 그래서 해당 state case문 안에서 조건을 확인한 뒤, state를 바꿈과 동시에 grant값을 정해주었더니 1cycle에 바로 전달될 수 있었다.

Bus의 Address decoder를 설계할 때는 주소를 가지고 바로 0x00식으로 비교하는 조건문을 작성하였다가 오류가 났었다. 이는 다시 방법을 바꾸어서 주소의 상위 3bit를 비교하는 식으로 하였더니 선택될 수 있었다.

전체적으로 Bus top module에서 회로도에 따라 각 module들을 instance하는 과정에서, grant나 sel 신호들이 1bit으로 할 지 2bit으로 할 지에 대해 시행착오가 있었다. 이에 grant는 따로 조건문이나 mux를 사용하지 않고 m0_grant신호를 기준으로 0이면 m1, 1이면 m0인걸로 생각하여 구현했다. 그래서 s_sel 신호들도 같은 방식으로 하였다가 3-to-1 MUX에서 문제가 발생하였다. 여기서는 2bit으로 구분해야 되기 때문에, 아예 register에 전달할 때 s0_sel, s1_sel을 묶어서 2bit으로 전달하여 출력된 2bit wire를 MUX에 연결해주는 식으로 하였더니 해결할 수 있었다.

B. 결론

이번 Memory 설계를 통해 test bench가 아니더라도 처음에 메모리 값들을 for문을 통해 초기화할 수 있다는 점을 깨달았다. 또한 Bus를 설계하면서 cycle을 단축할 수 있는 점을 고민해보게 되어서 앞으로 효율성 등을 더욱 따져볼 수 있을 것 같다는 생각이 들었다. 마찬가지로 Bus의 각 module들을 차례대로 설계한 뒤 검증하면서 Bus의 구조를 더 자세히 이해할 수 있었던 것 같다.

Bus의 장점 중 새로운 component들을 추가하기 쉽다는 점이 직접 설계하면서 이해가 갔다. 각 module들의 wire등을 조금 조정해주면 다른 형식의 Bus로도 응용할 수 있을 거란 생각이 들었다. 설계한 뒤의 Flow summary의 결과도 보면서, 가격이 저렴하단 장점도 이해가 갔다. 여러 module들을 설계하고 합친 것에 비해 크기가 작은 편으로 나왔기 때문이었다. 이러한 Bus를 응용하여 memory와 합치고 다른 module들과 추가로 연결하는 등의 작업으로 응용할 수 있을 것 같다.

6. 참고문헌

David Money Harris and Sarah L. Harris / Digital Design and Computer Architecture / Elsevier / 2007