컴퓨터 공학 기초 실험2 보고서

실험제목: Memory & Bus

실험일자: 2020년 11월 13일 (금)

제출일자: 2020년 11월 18일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 5,6,7

학 번: 2019202009

성 명: 서여지

1. 제목 및 목적

A. 제목

Simple Memory & Bus

B. 목적

입력된 address에 해당하는 위치에 정보를 읽거나 쓰는 memory와, 2개의 master와 2개의 slave 사이를 연결하는 bus의 동작을 이해한다. memory와 bus를 설계하고 Verilog를 이용하여 구현한다. 구현한 memory와 master에 대한 검증을 진행하고 결과를 분석한다.

2. 원리(배경지식)

(1) RAM

Random Access Memory는 data가 저장된 순서에 관계없이 빠른 시간에 사용할 수 있는 메모리이다. RAM은 구조에 따라 static RAM(SRAM)과 dynamic RAM(DRAM)으로 나눌 수 있다. SRAM의 경우 전원이 공급되는 동안 메모리를 구성하는 CELL이 정보를 계속 저장하고 있지만, DRAM의 경우 정보를 계속 유지하려면 전원이 꾸준히 refresh 되어야 한다.

(2) 메모리 선언과 초기화

reg를 이용하여 메모리를 선언할 수 있다. 이번 실험에서는 reg [31:0] mem [0:31]을 이용하여 32bit 정보를 32개 저장할 수 있는 메모리 mem을 선언하였다. 메모리는 mem[0]과 같은 방법으로 접근할 수 있다. 선언한 메모리는 초기 한 번 실행되는 initial구문의 내부에서 for 반복문을 이용하여 초기화 한다. integer i를 선언하고 i에 0부터 31까지의 값을 순서대로 저장하여 32개의 공간을 모두 초기화한다.

(3) BUS

bus는 여러 component 사이를 연결하여 data 이동을 가능하게 하는 component이다. bus가 연결하는 component는 master와 slave로 나눌 수 있다. master는 slave에 특정 주소에 어떤 값을 저장하거나, 특정 주소의 값을 읽어오도록 명령할 수 있다. slave는 master의 명령을 수행하고 결과를 반환한다. bus는 여러 개의 master와 slave를 가질 수 있으며 내부의 Arbiter와 Address Decoder에서 수행할 명령의 master와 slave를 결정한다. master를 결정하는 arbiter는 각 master의 request 여부에 따라 동작한다. 현재 명령을 지시하고 있는 master의 request 신호가 종료되기 전 까지는 다른 master로 이동하지 않는다. arbiter는 FSM을 이용하여 설계한다. Address Decoder는 slave의 memory map에 따라 slave를 선택한다.

3. 설계 세부사항

(1) Memory

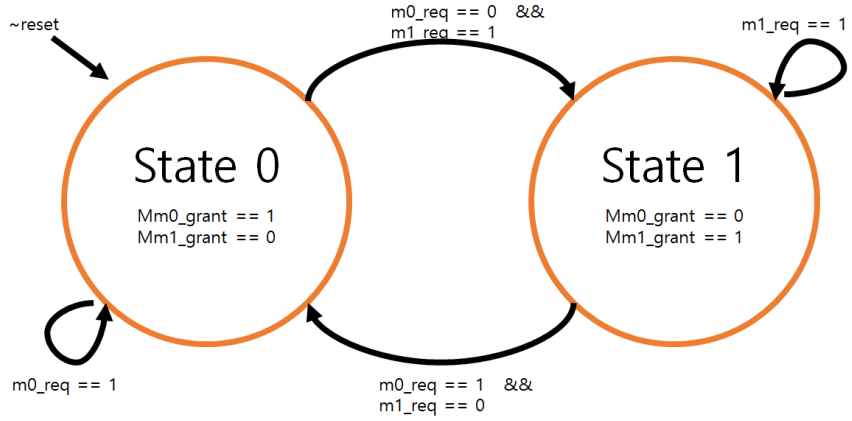
이번 실험에서 설계한 메모리는 address에 해당하는 register에 din으로 입력되는 32 bit 정보를 저장하거나, 해당 위치에 저장된 정보를 출력하는 동작을 한다. 사용하는 address는 5 bit 길이이므로 총 32가지의 정보를 각기 다른 register에 저장할 수 있다. 각 register의 크기는 32bit이다. 또한 cen과 wen입력에 따라 동작을 결정한다. cen과 wen이 모두 1이면 din을 addr에 해당하는 register에 저장하고, cen이 1이고 wen이 0이면 addr의 정보를 dout으로 출력한다. cen이 0이면 wen 값에 관계없이 항상 0을 dout으로 출력한다.

메모리를 이용하기 전, initial 문에서 for 반복문을 이용하여 mem을 0으로 초기화 한다. 이후 clk의 posedge를 이용한 always 문의 내부에서 cen과 wen의 조건을 확인하여 mem에 값을 쓰거나 읽어온다.

(2) Bus

이번 실험에서 설계한 bus 2개의 master와 2개의 slave를 갖는다. 8bit address중 5자리를 사용하고, 32bit 길이의 data를 읽거나 쓸 수 있다. slave1은 0x00부터 0x1F까지, slave2는 0x20부터 0x3F까지의 주소에 접근한다. req신호에 따라 arbiter에서 state를 변환시키며, req신호가 1인 동안 해당 state가 계속 유지된다. wr신호는 쓰기여부를 표시하며 0인경우 address에 해당하는 data를 읽어온다. m_dout 신호는 wr가 1일 때 slave에 전달하여 작성하는 내용이고, s_dout은 wr가 0일 때 slave가 메모리에서 읽어오는 data이다. arbiter에서 결정한 state에 따라 grant신호가 출력되고, m_in 출력은 wr가 0일 때 slave에게서 master가 전달받은 data이고(s_dout), s_in 출력은 wr가 1일 때 master에게서 slave가 전달받은 data로(w_dout), address에 해당하는 메모리에 써야하는 값이다.

arbiter의 state transition diagram



bus_addr의 동작

	시작주소		끝주소	
slave1	0000_0000	0x00	0001_1111	0x2F
slave2	0010_0000	0x20	0011_1111	0x3F

address의 첫 3자리를 읽어 비교한 뒤, 000이면 slave1을 선택하고, 001이면 slave2를 선택한다.

bus의 동작(reset이후)

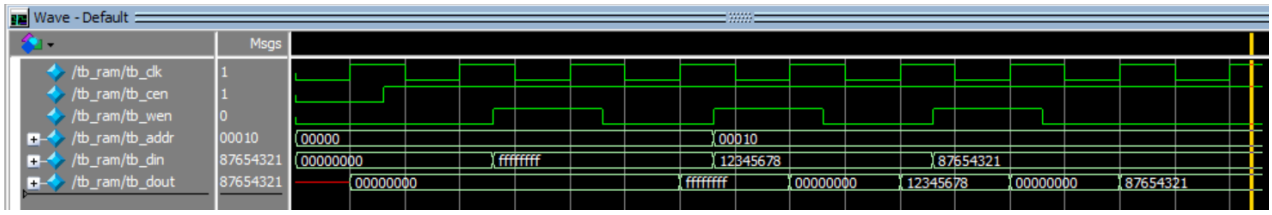
m0_req	m1_req	m0_wr	m1_wr	master	동작
0	0	x	x	0	대기
0	1	x	0	1	읽기
0	1	x	1	1	쓰기
1	x	0	0	0	읽기
1	x	1	1	0	쓰기

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

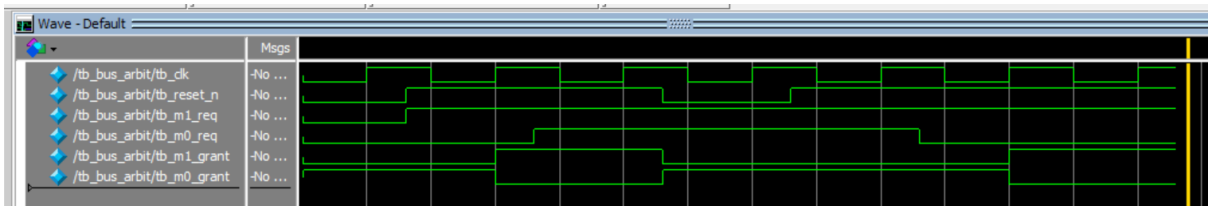
(1) ram

testbench는 각 주소에 임의의 값을 저장하고 읽어오는 것으로 작성하였다. 주소가 0인 위치에 ffffffff를 저장하고, 2인 위치에 12345678과 87654321를 순서대로 저장하고 읽는 동작을 정상적으로 수행하는 것을 확인할 수 있다.

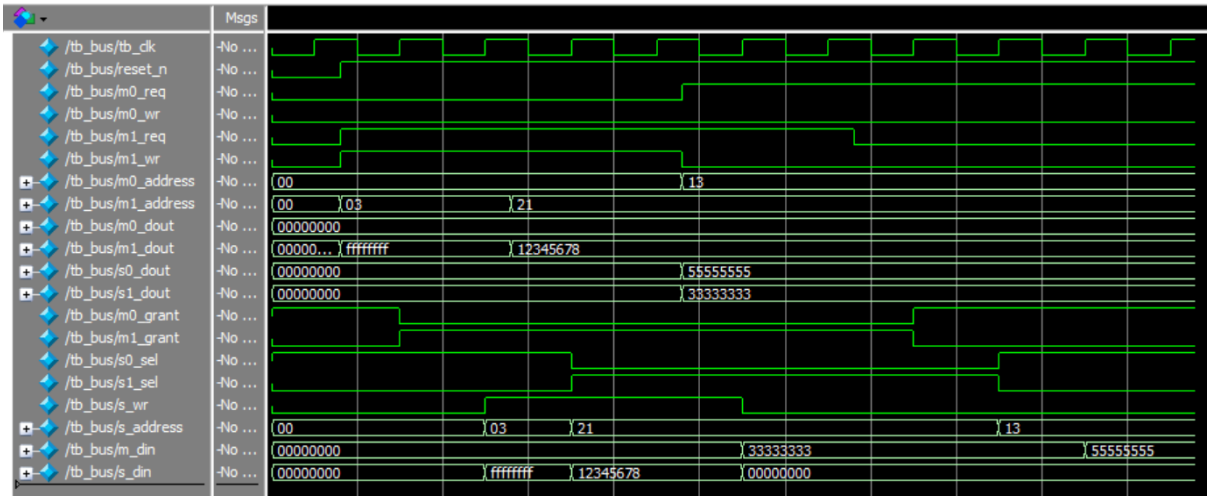


(2) bus

bus를 구성하는 module 중, bus_arbit module의 testbench를 작성하여 동작을 확인하였다. req 신호에 따라 적절한 grant 신호를 출력하는 것을 확인할 수 있다.



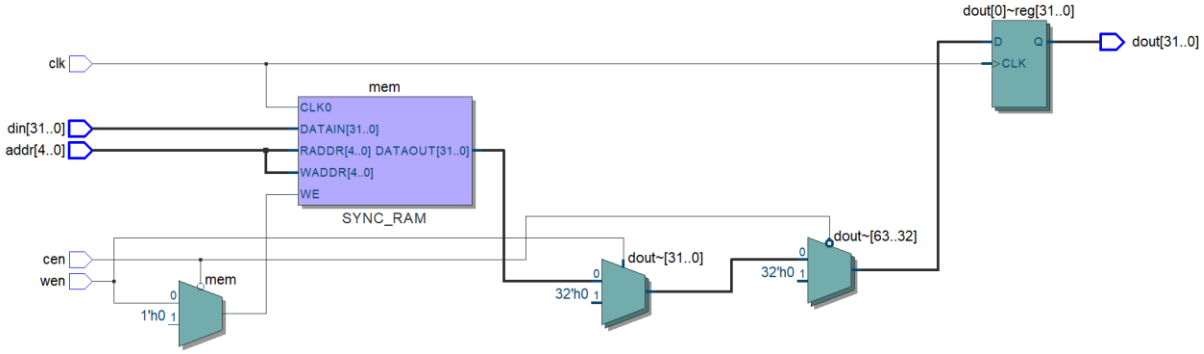
bus에 대한 testbench를 작성하여 동작을 확인하였다. 주소가 03인 위치에 ffffffff를 쓰고, 21인 위치에 12345678을 저장하는 동작을 수행한다. 이후 13인 위치의 값과 21인 위치의 값을 읽는다. 13과 21의 data는 임의의 dout을 입력하여 확인하였다.



B. 합성(synthesis) 결과

(1) ram

ram module의 RTL viewer 결과는 다음과 같이 나타났다. mem으로 선언된 메모리가 clk에 동기되어있고, cen과 wen값에 따라 출력되는 것을 확인할 수 있다.



flow summary는 다음과 같다. 사용된 pin은 모두 72개로, din입력과 dout출력이 각각 32개의 pin을 사용하고, addr입력이 5개, clk, cen, wen이 각각 하나의 pin을 이용하였다. register는 1056개가 이용되었다. 32bit의 정보를 32개 저장하여 1024개의 register를 mem에서 이용하고, dout을 출력할 때 32개의 register를 추가로 이용하였다.

Compilation Report - ram

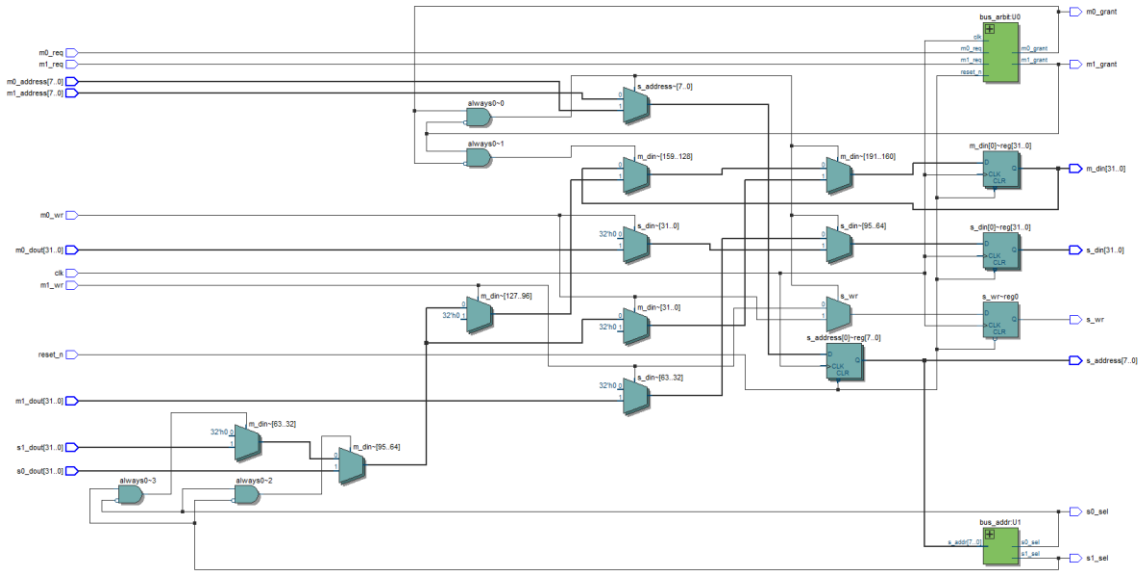
Table of Contents

Flow Summary

Flow Status	Successful - Tue Nov 17 12:27:44 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	ram
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	528 / 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

bus의 RTL viewer 결과는 다음과 같이 나타났다. req, wr등의 신호를 이용하여 사용할 master와 slave를 결정하는 것에 여러 개의 mux가 이용된 것을 확인할 수 있다.



flow summary는 다음과 같다. clk(1), reset_n(1), m0_req(1), m0_wr(1), m0_address(8), m0_dout(32), m1_req(1), m1_wr(1), m1_address(8), m1_dout(32), s0_dout(32), s1_dout(32) 로 입력에 총 150개의 pin이 이용되었고, m0_grant(1), m1_grant(1), m_din(32), s0_sel(1), s1_sel(1), s_address(8), s_wr(1), s_din(32) 로 출력에 77개의 핀이 이용되었다. 사용된 register는 모두 75개로, grant 신호를 제외한 출력에 이용되었다.

Flow Summary	
Flow Status	Successful - Tue Nov 17 18:40:02 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	bus
Top-level Entity Name	bus
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	43 / 41,910 (< 1 %)
Total registers	75
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

bus를 구현한 뒤, testbench를 작성하여 시험하는 과정에서 그 결과가 원하는 값과 다르게 0으로 출력되었다. write를 한 뒤, 같은 주소에서 read를 실행했으나 0이 출력된 것이다. 이것은 bus의 오류가 아닌 정상 동작으로, bus 자체는 data를 저장하지 않기 때문에 나타난 결과였다. 0으로 초기화된 s_dout값이 출력된 것이다. 자세한 비교를 위해 s1_dout과 s2_dout에 임의의 다른 값을 넣어 다시 확인한 결과 bus가 원하는 대로 동작함을 확인할 수 있었다.

B. 결론

첫 번째 과제는 memory를 구현하는 과제였다. verilog에서 메모리는 c언어와 유사하게 mem[0]과 같은 방법으로 쉽게 접근할 수 있었다. 메모리를 초기화하는 것에 사용한 integer i를 이용하는 방법뿐만 아니라 2진수 값인 addr을 그대로 입력하여 사용할 수도 있다는 점이 편리했다. 두 번째 과제는 2개의 master와 2개의 slave를 갖는 bus를 구현하는 과제였다. bus는 master나 slave를 추가하는 것으로 확장하기 쉽기 때문에 여러 개의 component를 이용하는 module을 구현할 때 이번에 구현한 bus를 활용할 수 있을 것이라 기대된다.

6. 참고문헌

RAM, https://www.britannica.com/technology/RAM-computing

메모리초기화,

https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProjects/hdl/vlog/vlog_pro_inferred_memories.htm
이준환교수님, 디지털논리회로2 강의자료, 광운대학교 컴퓨터정보공학과,2020
이준환교수님, 컴퓨터공학기초실험2 강의자료, 광운대학교 컴퓨터정보공학과,2020