컴퓨터 공학 기초 실험2 보고서

실험제목: Simple Memory and Bus

실험일자: 2022년 11월 23일 (수)

제출일자: 2022년 11월 29일 (화)

학 과:컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 수요일 0, 1, 2

학 번: 2019202021

성 명: 정 성 엽

1. 제목 및 목적

A. 제목

Simple Memory and Bus

B. 목적

데이터를 임시로 저장하는 Memory(RAM)를 원리를 이해하고 직접 Verilog로 설계한다. 그리고 데이터 전달 통로 역할을 하는 Bus의 원리를 이해하고 직접 Verilog로 설계한다. 설계한 Verilog를 Testbench를 통해 정상 작동하는지 확인한다.

2. 원리(배경지식)

A. Memory(Ram)

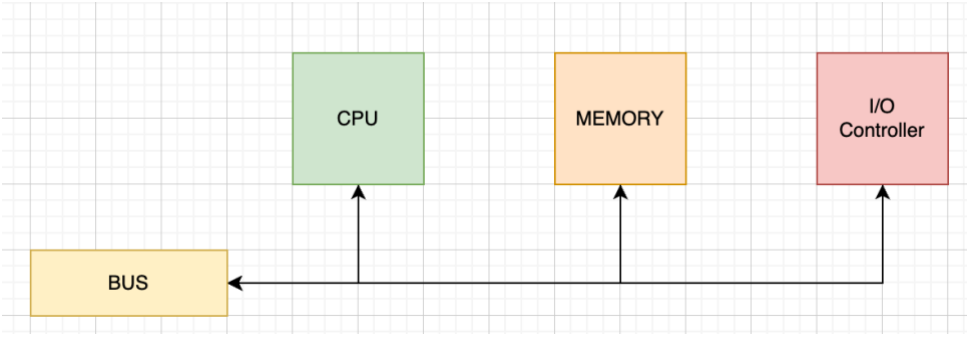
Memory(Ram)은 register와 함께 2진 정보를 저장한다. 이 Memory와 register의 차이점은 Register은 CPU 내부에 존재하지만 Memory는 외부에 존재하여 Register보다 더 많은 정보를 읽고 쓸 수 있다. 다만 외부에 존재하는 만큼 속도는 느리다.

다른 비교군으로 ROM을 비교할 수 있다. ROM은 Read Only Memory로 영구적으로 데이터를 저장하기만 가능하고 쓰기는 불가능하다. 이에 반해 RAM은 Random Access Memory로 전원이 나가면 데이터가 손실되지만 속도는 빠르고 읽고 쓰기 가능하다. 최근에 ROM은 NAND Flash Memory로 제작되어서 읽고 쓰기 모두 가능하다.

B. Bus

Bus는 하드웨어 간의 데이터 이동을 위해 연결해주는 연결 통로의 역할을 하는 것으로 이때 Master은 어떠한 기능을 수행할지 그리고 Slave는 명령 수행 역할을 하여 서로 연결하며 1개의 통로를 이용하여 여러 레지스터를 연결 가능하다. 또한 Master는 request에 따라 구분할 수 있고 Slave 또한 지정된 Address의 범위에 다라 각각 구분된다.

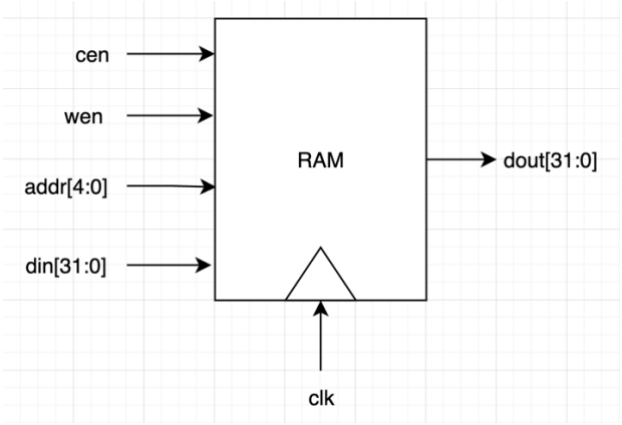
Bus는 레지스터가 포함된 시스템에서 효율적인데 이는 레지스터에서 다른 레지스터로 이동할 때 연결되는 경로를 무한정으로 만들기보다 Bus를 통해 각 레지스터와 연결되면 이 Bus를 공용 통로로 이용하는 것이 더 효율적이기 때문이다. 이를 통해 데이터의 이동을 할 때마다 여러 연결을 만들 필요없이 Bus만 설계하여 구현한 후 서로를 연결하는 효율적인 설계가 가능해진다.



3. 설계 세부사항

A. Ram

1) Block diagram



5bit의 address에 32bit din data를 저장하는 Memory(Ram)을 제작한다. Verilog로 메모리 변수를 직접 선언하여 RAM을 구현하고 이 때 주소가 5bit 이므로 2^5개의 데이터를 저장할 수 있다.

Cen은 Ram을 사용할 때 1로 set하며 사용하지 않으면 0으로 set한다. Wen은 0일 경우 지정된 Address에 저장된 값을 dout으로 출력하고 1로 set 된 경우 지정된 Address로 din의 값을 저장한다.

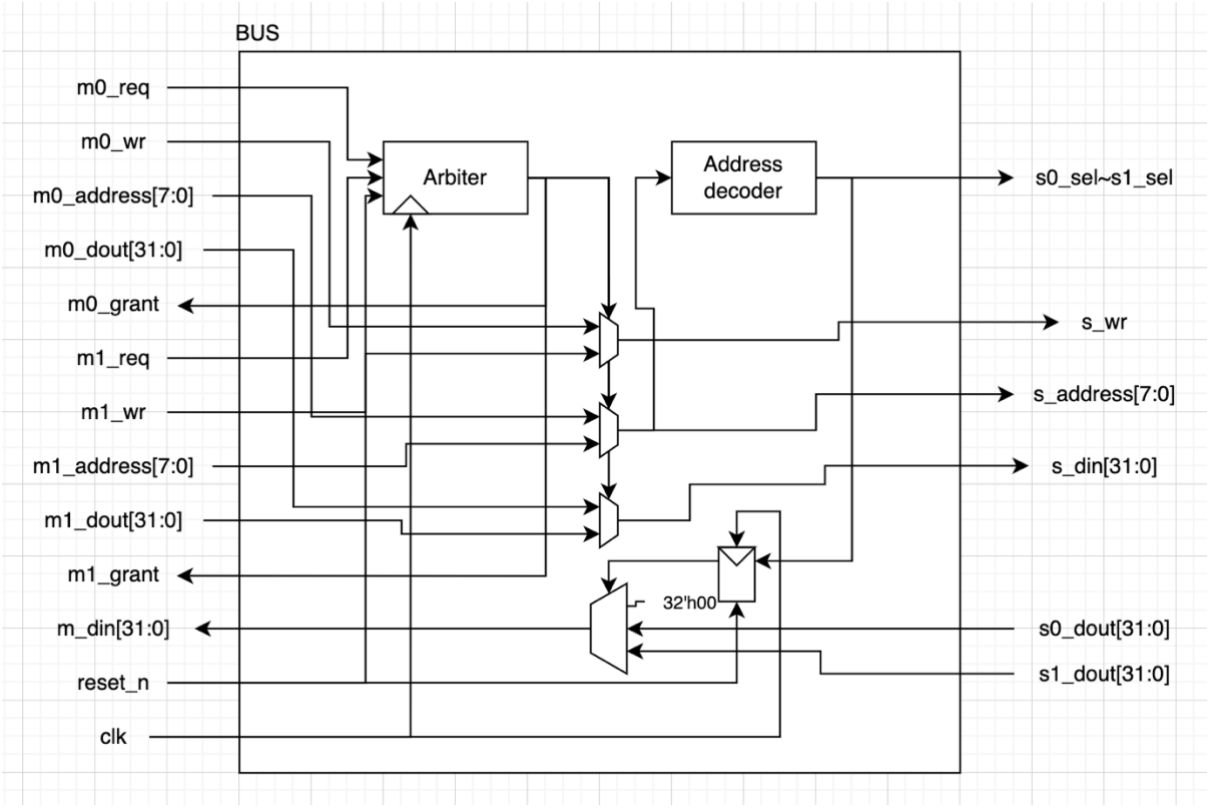
2) I/O configuration

RAM

구분	이름	비트 수	설명
input	Clk	1bit	Clock
	Cen	1bit	Chip enable
	Wen	1bit	Write enable
	Addr	5bit	Address
	Din	32bit	Data in
output	Dout	32bit	Data out

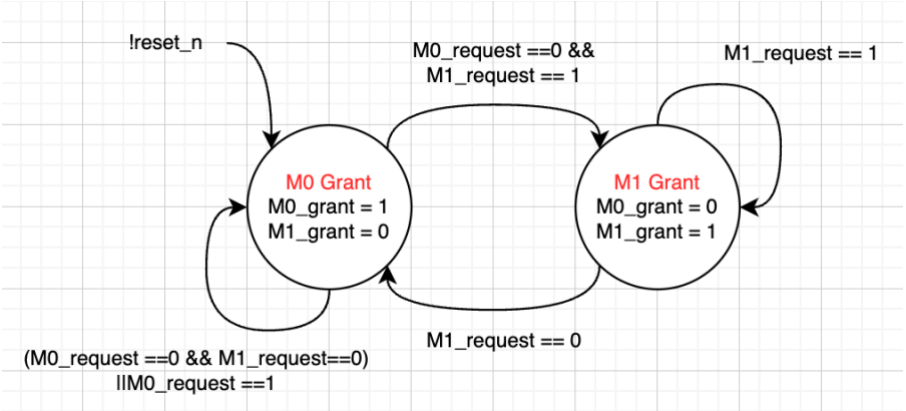
B. Bus

1) Block diagram



신호를 받아서 주소와 입력할 곳에 m0 또는 m1에 연결하고 arbiter에서는 m0_req, m0_wr, m1_req, m1_wr을 입력 받아서 FSM 구조에 따라 state를 m0_grant인지 m1_grant인지 다음 state를 결정한다. 이 arbiter의 output은 각 2-to-1 mux에 선택 신호로 연결하여 후에 처리한다. 이 때 2-to-1 MUX는 arbiter에서 나오는 각. M1 또는 m0의 신호를 받아 해당 master에 알맞는 값을 선택하여 출력하고 address decoder에서는 선택된 master의 address를 분석하였고 어느 slave로 전달한다. 그리고 이 전에 2-bit D-f/f 저장 후 3-to-1 mux로 값을 전달한다.

2) State diagram



Bus의 Arbiter에서의 FSM 구조를 그려낸 것으로 master의 req 값에 따라 해당 master의 state로 이동하며 해당 grant를 출력하는 FSM구조로 설계되었다.

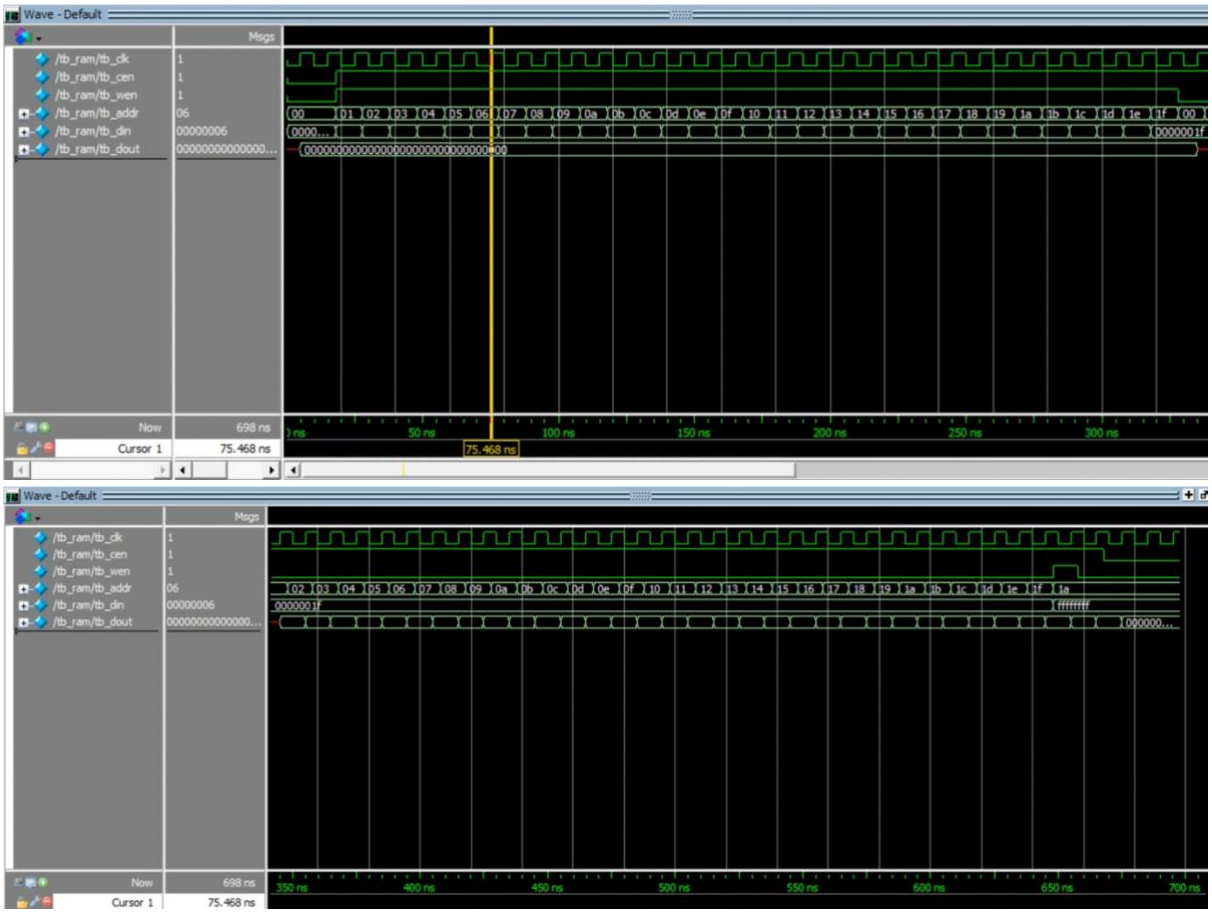
3) I/O configuration

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[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 output
	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	Slave data input

4. 설계 검증 및 실험 결과

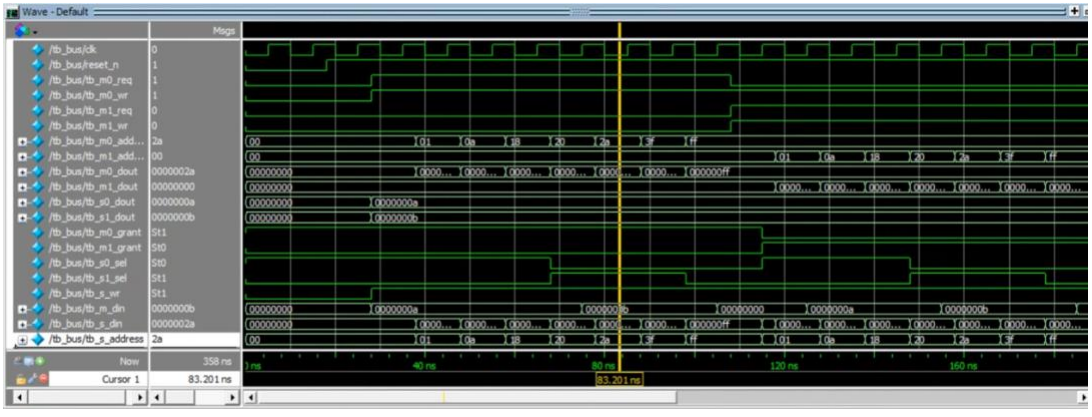
A. 시뮬레이션 결과

i. RAM

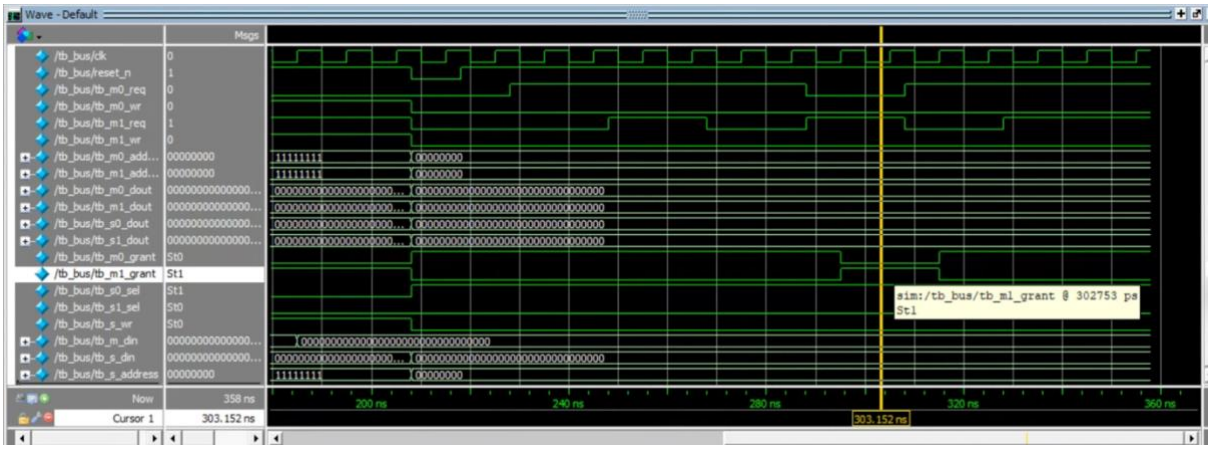


32개의 주소에 데이터를 쓰고 읽는 testbench를 설계하였으면 wen과 cen이 모두 set 되었을 때 쓰고 wen이 0으로 reset 된다면 읽는 것을 확인할 수 있다. 그리고 마지막에 cen을 0으로 reset 하였을 때 dout이 0으로 변한 것을 확인할 수 있다. 이는 ram의 작동을 정지시켰다고 볼 수 있다.

ii. BUS



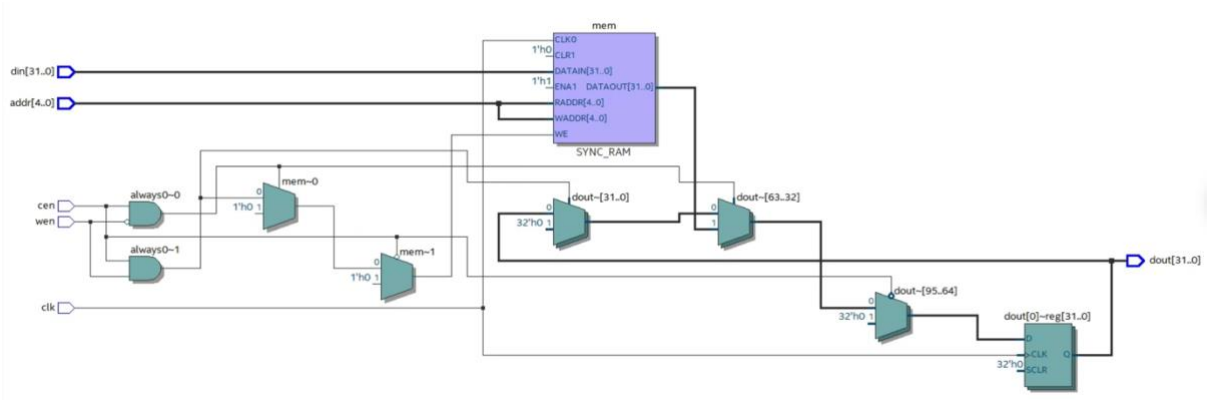
Req값에 따라 작동하는 waveform을 볼 수 있으며 처음이는 m0를 request하였고 slave가 m0의 입력이 되는 것을 확인할 수 있다. 그리고 slave는 address가 0이상 그리고 20미만이면 10으로 slave0이 set 되고 20이상 40미만이면 01으로 slave1이 되고 모두 아니라면 00으로 모두 set되지 않는다. 이는 m1을 불러올 때도 똑같이 작동한다.



Reset_n을 Reset하였다 다시 set하여 초기화하고 확인해보았다. M0_grant에서 m0_req를 set할 때 m0_grant로 유지되고 m1_req가 set 되어도 값이 같은 것을 확인할 수 있다. 그리고 m0_req을 0으로 Set하고 m1_req를 1로 set 했을 때 m1_grant로 바뀐 것을 확인할 수 있다. 마지막에 둘 다 1로 set 되었을 때는 m0_grant로 유지됨을 다시 확인할 수 있다.

B. 합성(synthesis) 결과

i. RAM

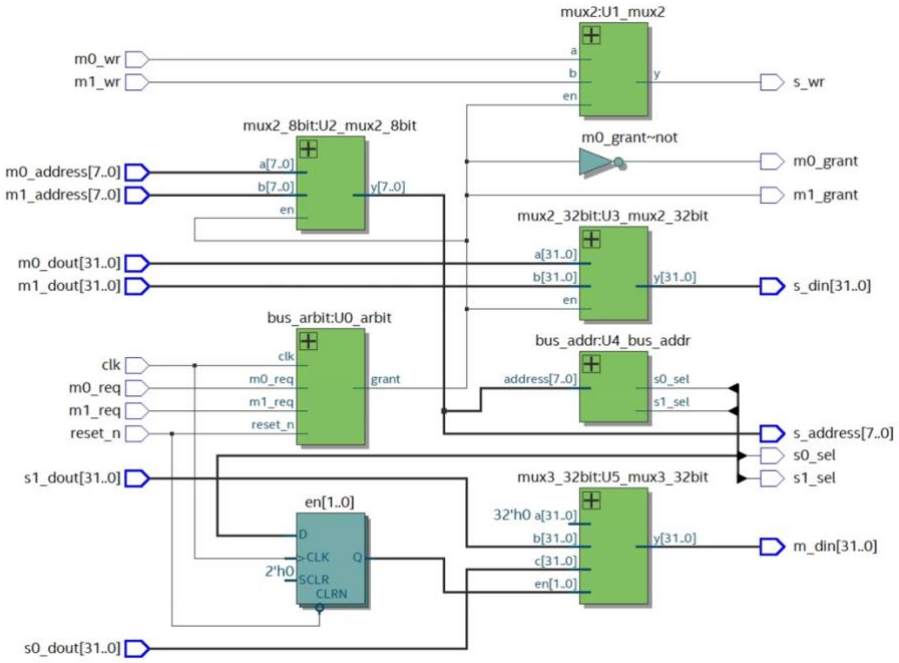


Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Nov 29 22:21:53 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ram
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	527 / 41,910 (1 %)
Total registers	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 %)

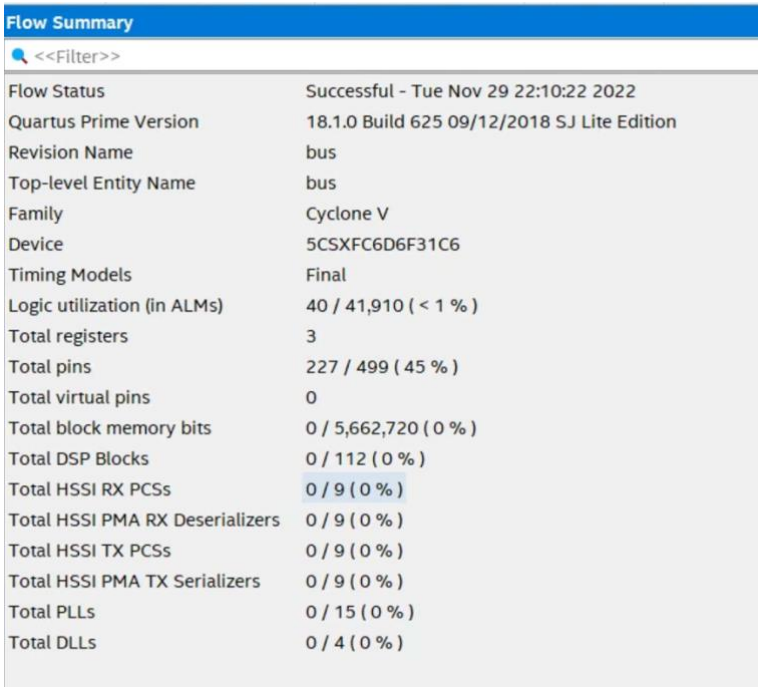
Verilog에서 기본적으로 제공하는 memory 변수를 사용하여 mem을 구현하여 module처럼 선언된 것을 확인 할 수 있다. Cen과 wen 에 따라 각 조건을 mux로 구현된 것 또한 확인 할 수 있다.

ii. BUS

1) RTL Viewer



2) Flow Summary



Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Nov 29 22:10:22 2022
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)	40 / 41,910 (< 1 %)
Total registers	3
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 %)

Arbiter와 address decoder가 모두 정상적으로 구현되어있음을 확인할 수 있고 또한 always@로 구현한 d-flipflop의 모듈이 정상적으로 선언된 것 또한 확인할 수 있다. 이 d-flipflop은 slave의 위치를 출력하여 mux3로 가져가 en으로써 활동한다.

5. 고찰 및 결론

A. 고찰

이번 실험은 simple memory인 ram과 각 2개의 master과 그에 해당하는 2개의 slave를 가지는 bus를 제작하였는데 이번 실험에서 새로웠던 점은 이전에 사용했던 Verilog 사용이 현저히 적다는 것이다. 특히 gate.v는 거의 필수적으로 가져오던 파일이었는데 이번 실험에서는 사용하지 않았다.

Memory에서 Verilog 자체적으로 제공하는 메모리 변수를 활용하였으며 for문으로 memory를 0으로 초기화하기 위해 integer를 이용하여 for문을 완성하여 초기화 하였다.

Bus는 단순하게 생각하면 데이터의 연결 통로라고 생각할 수 있지만 각 조건에 따라 데이터 이동 허용과 제한을 구현해야 했다. i/o configuration에서도 각 변수에 대한 설명이 있지만 단순히 그것만을 읽고 구현하기에는 떠오르지 않았다. 그래서 block diagram을 한 wire 마다 따라들어가고 각 wire 의 값에 따라 어떤 값이 나올 지 먼저 testbench를 서면으로 적어가면서 이해하였다. 또한 중간에 2bit d-flipflop이 등장하는데 이는 always문으로 대신 해결하였다.

B. 결론

이번 실험을 통해 simple memory인 ram과 bus를 설계하고 제작하였으며 bus는 여러

모듈을 많이 사용하여 회로를 구현할 때 이용하면 각 모듈마다 연결하는 것이 아닌 bus를 공용으로 사용함으로써 효율적으로 빠르게 이어줄 수 있다는 것을 알게 되었다. 또한 Memory는 register와 특징은 비슷하지만 원리에서 설명했듯이 여러 차이점이 존재하고 추가적으로 ROM과 비교하면 RAM의 특성을 더 잘 이해할 수 있었다.

Bus는 여러 개의 master과 slave로 구성되어 있고 설계자의 의도에 따라 각 개수를 정할 수 있다. 이번 실험 같은 경우 각 2개씩 존재하여 request 신호에 따라 arbiter에서 state를 정하고 2-to-1 mux에 의해 각 값이 전달 된다. 마지막으로 slave는 address decoder에서 나온 값을 통해 해당 범위를 보고 3-to-1 mux를 통해 slave에 출력한다.

6. 참고문헌

1) 공영호 교수님/디지털논리회로2/광운대학교 컴퓨터정보공학부/2022
2) 공진흥 교수님/컴퓨터공학기초실험2/광운대학교 컴퓨터정보공학부/2022