# 컴퓨터 공학 기초 실험2 보고서

실험제목: Simple Memory & Bus

실험일자: 2023년 11월 22일 (수)

제출일자: 2023년 11월 22일 (수)

학 과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 수요일 0, 1, 2

학 번: 2022202075

성 명: 우나륜

# 1. 제목 및 목적

## A. 제목

Simple Memory & bus

## B. 목적

간단한 memory와 bus의 작동 원리와 설계 방법에 대해 학습하고 이를 Verilog를 통해 구현하는 데 목적을 둔다. 또한, memory를 선언하고 memory 내의 값을 write하거나 read하는 방법을 익힌다.

# 2. 원리(배경지식)

### 1. Memory

컴퓨터에서 메모리는 주기억자치 또는 컴퓨터 메모리라고 불리며, 컴퓨터에서 수치/명령/자료 등을 기억하는 역학을 수행하는 컴퓨터 하드웨어 장치를 말한다. 그중에서 단기적 데이터를 저장하거나 액세스 공간을 제공하는 컴퓨터의 임시 저장장치인 RAM (Random Access Memory)는 하드 드라이브보다 실시간으로 더 많은 데이터에 접근하고 읽을 수 있다. 하지만 RAM은 전원이 꺼지면 저장된 데이터가 사라지게 된다.

VerilogHDL에서 Memory를 선언하고 Ram처럼 사용할 수 있다. Memory를 선언하는 방식은 reg [wordsize – 1] variable_name [0 : storagesize – 1] 이다. 메모리를 사용하기 전에 for문을 사용하여 메모리를 초기화한 후 사용하면 된다.

본 과제에서 memory를 선언하여 입력받은 address에 접근하여 입력받은 데이터를 write하거나 read하는 작업을 수행하는 간단한 ram을 구현하였다. 또한, cen과 wen 두 개의 signal을 사용하여 memory 내에 읽거나 쓰는 작업을 조작할 수 있다.

### 2. Bus

Bus는 master interface와 slave interface들 간의 data를 전송할 수 있도록 연결해주는 component이다. 이러한 bus는 새로운 component들을 추가하기 쉽고, 가격이 저렴하다는 특징을 가진다. 또한, bus는 컴퓨터 내의 부품들 간에 데이터와 주소, 제어 신호들을 전송하는 통로가 된다. 버스의 종류는 크게 system bus와 I/O bus가 존재한다. System bus는 CPU와 메모리를 연결하는 subsystem을 system bus라고 하고, I/O bus는 메모리와 다른 입출력 장치와 통신을 가능하게 하는 subsystem을 I/O bus라고 한다.

본 과제에서 bus는 2개의 master interface와 2개의 slave interface를 가진다. Master interface로부터 m0_req와 m1_req를 받아 그 값에 따라 bus에 대한 소유권

m0_grant와 m1_grant를 master interface에 할당한다

3. Verilog에서 Integer

Verilog에서 수를 다르기 위해 일반적인 범용 레지스터 데이터형인 integer를 사용할 수 있다. 흔히 사용하는 데이터형인 reg는 음수를 저장할 수 없는 반면에, integer는 음수를 저장할 수 있다. Integer의 기본 비트수는 최소 32bits이다.

4. For 구문

for문은 repeat, while, forever와 같이 Verilog 내의 loop command이며, 순차회로인 always문 또는 initial문 내에서만 사용이 가능한 구문이다. For문의 문법은 C/C++과 비슷하게 for (초기값; 조건식; 오퍼레이션) 으로 구성된다. 하지만 C/C++과는 다르게, 오퍼레이션에서 for문의 index를 증가시킬 때, i++과 같은 형식으로 증가시킬 수 없다. 만약, index를 1씩 증가시키고 싶다면 i = i + 1로 선언하면 된다. 또한, for문의 index는 integer로 선언하여 사용해야 한다. 본 과제에서 선언한 memory내의 모든 값을 초기화하기 위해 for구문을 사용하여 작업을 수행하였다.

## 3. 설계 세부사항

1. Simple memory

Simple memory인 ram module인 cen과 wen을 사용하여 메모리 내에 din값을 write하거나 read할 수 있다. 또한, addr를 사용하여 특정 위치의 메모리에 접근할 수 있고, 읽은 메모리의 값은 dout에 write하여 그 값을 확인할 수 있다.

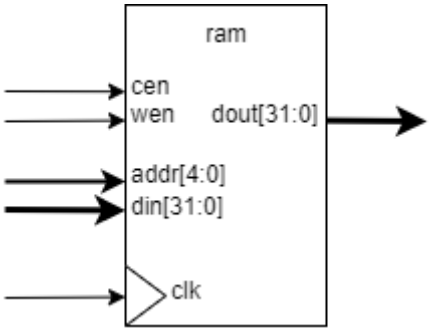



Figure 1. Schematic symbol of ram

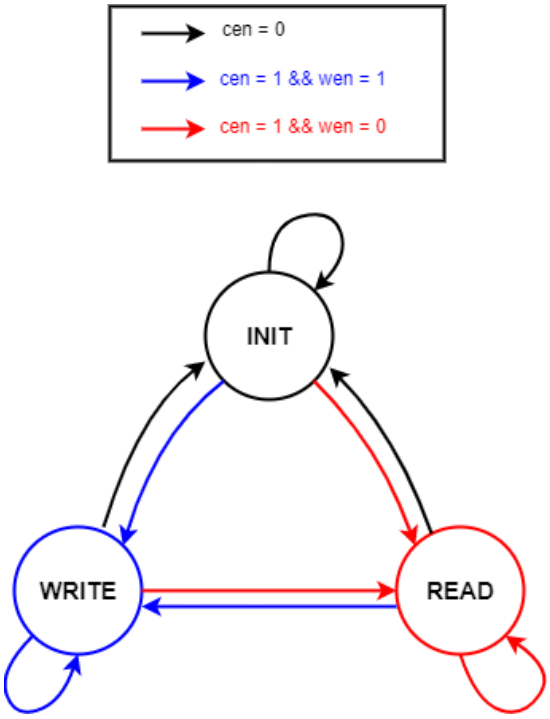



Figure 2. Memory FSM

위 Figure 2는 Memory의 FSM을 나타낸 것이다. Cen의 값이 0이면, INIT 상태로, 특정 주소의 메모리 값을 write하거나 read하지 않는다.

| Module name | Direction | Port name | Bit | Description |
|---|---|---|---|---|
| ram | Input | clk | 1-bit | Clock |
| | | cen | | Chip enable |
| | | wen | | Write enable |
| | | addr | 5-bit | Address |
| | | din | 32-bit | Data in |
| | Output | dout | | Data out |

Table 1. I/O configuration

위의 Table 1은 ram module의 input과 ouptut을 나타낸 표이다.

2. Bus

Bus는 2개의 master interface (master0, master1)를 가지고 2개의 slave interface (slave0, slave1)을 가진다. 입력값과 multiplexer를 사용하여 master0과 master1 중 어떠한 interface를 사용할 지 결정할 수 있고, 메모리의 값에 대해 slave0과 slave1를 선택한다.

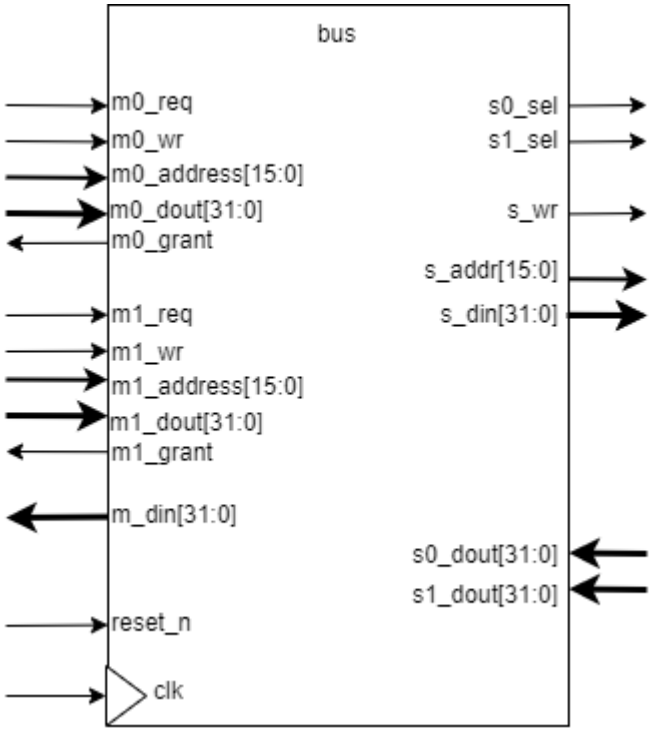



Figure 3. Schematic symbol of bus

아래의 Figure 4는 bus의 간단한 design을 나타낸다. Bus module의 주요 하위 module은 master0과 master1을 선택할 arbiter와 s0과 s1 중 어떠한 slave interface를 사용할 지 Address Decoder가 존재한다. 그 외에는 3개의 multiplexer를 사용하여 arbiter의 output 값에 따라 여러 개의 신호들 중 하나의 신호만을 선택한다.

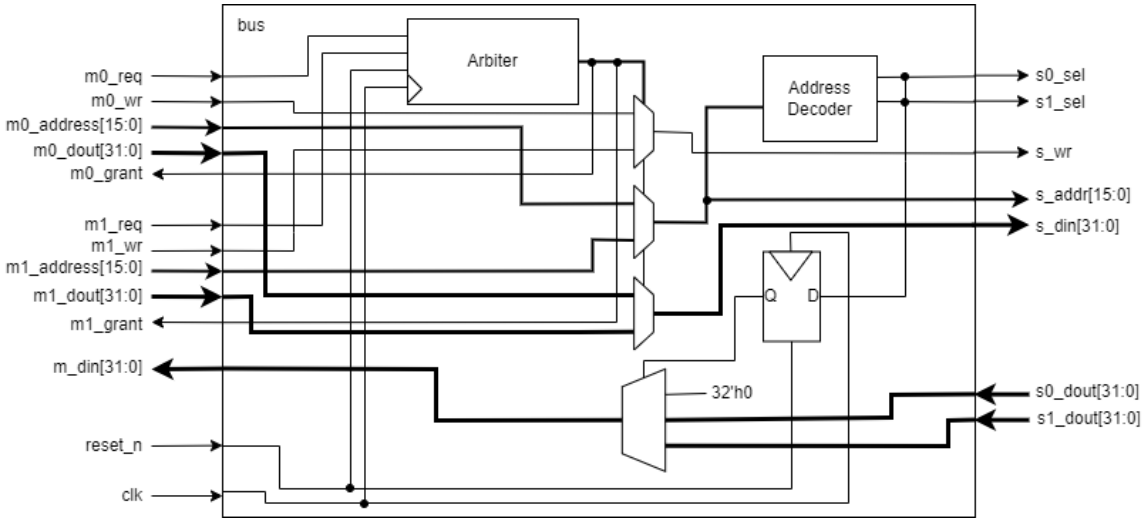



Figure 4. Bus Design

아래의 Figure 5는 arbiter의 FSM을 나타낸다. 입력값 reset_n, m0_req, m1_req의 값에 따라 m0_grant, m1_grant의 값을 결정한다. M0 Grant의 상태는 m0_grant = 1, m1_grant = 0으로 설정한다. M1 Grant의 상태는 m0_grant = 0, m1_grant = 1으로 설정한다. 또한, reset_n의 값이 0이거나 m0_req와 m1_req의 값이 모두 0이면, M0 Grant의 상태를 사진다.

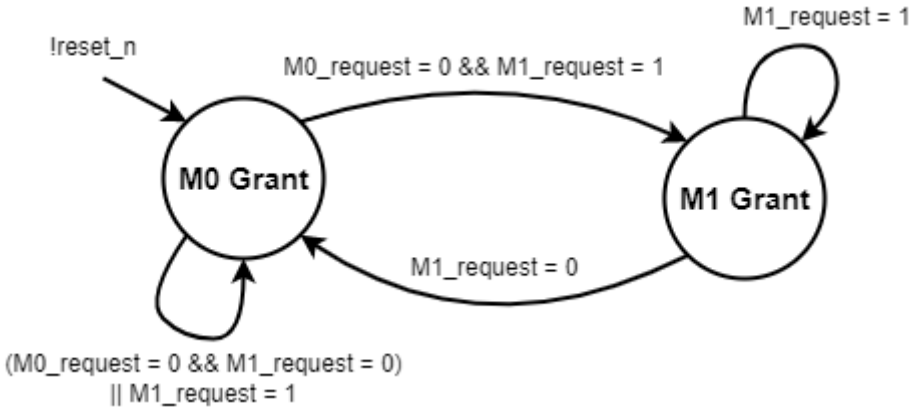



Figure 5. Arbiter FSM

아래의 Table 2은 bus module의 input과 ouptut을 나타낸 표이다. 아래의 bus module은 master0 또는 master1을 선택할 bus_arbit module과 slave0 또는 slave1을 선택할 bus_addr module을 instance화하여 사용한다. 또한, 8-bit 2-to-1 MUX를 사용하여 slave_address를 선택하고, 32-bit 2-to-1 MUX를 사용하여 slave data in을 선택하고, 1-bit 2-to-1 MUX를 사용하여 slave의 write/read signal을 선택하며, 32-bit 3-to-1 MUX를 사용하여 s0_dout, s1_dout, 32'h0들 중 하나의 값을 master data input으로 선택한다.

| **Module name** | **Direction** | **Port name** | **Bit** | **Description** |
|---|---|---|---|---|
| bus | Input | clk | 1-bit | Clock |
| | | reset_n | | Active-low reset |
| | | m0_req | | Master0 request |
| | | m0_wr | | Master0 write/read |
| | | m0_address | 8-bit | Master0 address |
| | | m0_dout | 32-bit | Master0 data output |
| | | m1_req | 1-bit | Master1 request |
| | | m1_wr | | Master1 write/read |
| | | m1_address | 8-bit | Master1 address |
| | | m1_dout | 32-bit | Master1 data output |
| | | s0_dout | | Slave0 data out |
| | | s1_dout | | Slave1 data out |

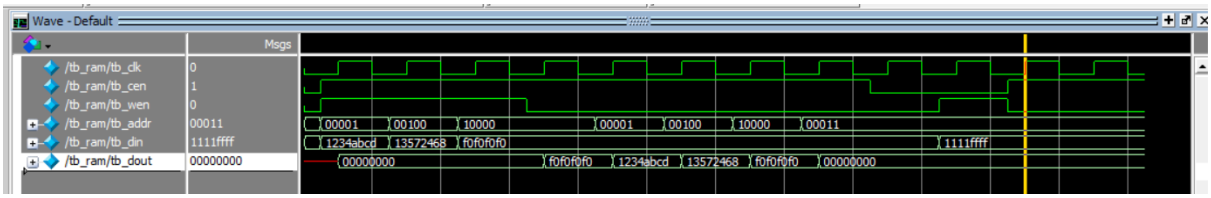
| | | | | |
|---|---|---|---|---|
| | Output | m0_grant | 1-bit | Master0 grant |
| | | m1_grant | | Master1 grant |
| | | m_din | 32-bit | Master data input |
| | | s0_sel | 1-bit | Slave0 select |
| | | s1_sel | | Slave1 select |
| | | s_address | 8-bit | Slave address |
| | | s_wr | 1-bit | Slave write/read |
| | | s_din | 32-bit | Slave data input |

Table 2. I/O configuration

# 4. 설계 검증 및 실험 결과

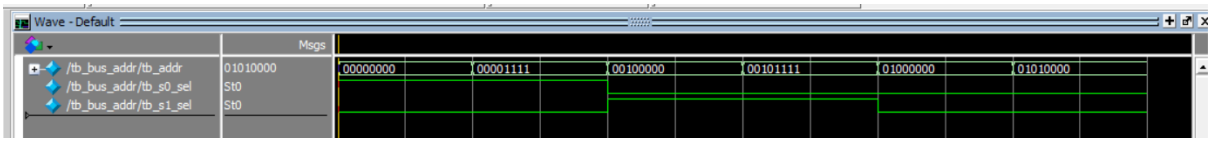
## A. 시뮬레이션 결과

### 1. Simple Memory



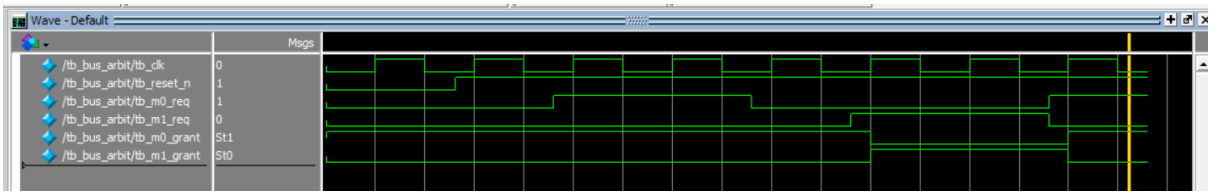

위의 waveform은 ram.v의 testbench를 실행한 결과화면이다. Signal인 cen = 1, wen = 1일 때, 입력된 addr의 주소에 din의 값을 write하고 이 동안 dout의 값은 0이다. cen = 1, wen = 0이면 read 동작을 수행한다. 따라서 이전에 저장한 address의 값을 입력하여 memory에 제대로 write되었는 지 확인하였다. 또한, 저장한 적 없는 메모리 주소 00011을 read하면 0을 출력하는 것을 확인할 수 있다. 이후 cen = 0, wen = 1로 하면 아무런 동작을 수행하지 않는데, 이는 cen이 다시 1이 되고 wen = 0일 때, 해당 주소에 저장된 값을 read할 때 0이 출력되는 것으로 확인할 수 있다.

### 2. Address decoder of bus



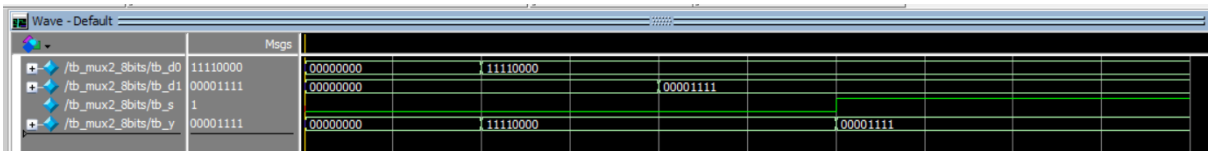

위의 waveform은 bus_addr.v의 testbench를 실행한 결과화면이다. Slave0의 메모리 주소내의 값이 입력되면, s0_sel = 1이 되고, s1_sel = 0이 된다. Slave1의 메모리 주소 내의 값이 입력되면, s0_sel = 0이 되고 s1_sel = 1이 된다. 만약 slave0과 slave1 밖의 메모리 주소 값이 입력되면, s0_sel = 0, s1_sel = 0이 된다.

3. Arbiter of bus



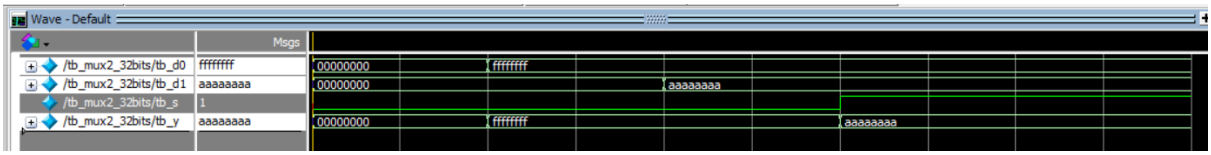

위의 waveform은 bus_arbit.v의 testbench를 실행한 결과화면이다. Reset_n, m0_req, m1_rew의 값에 따라 m0_grant, m1_grant의 값이 변화하는 것을 확인할 수 있다.

4. mux2_8bits



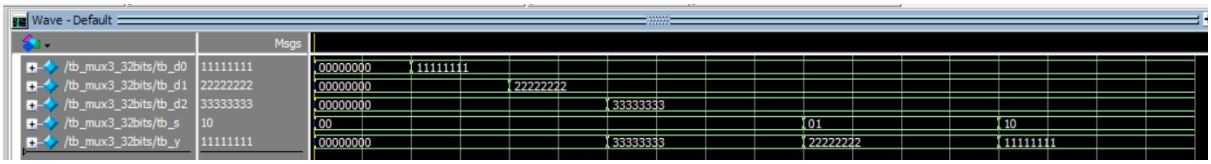

위의 waveform은 mux2_8bits.v의 testbench를 실행한 결과화면이다. S = 0이면 d0를 선택하고, s = 1이면 d1의 값을 선택하는 것을 확인할 수 있다.

5. mux2_32bits



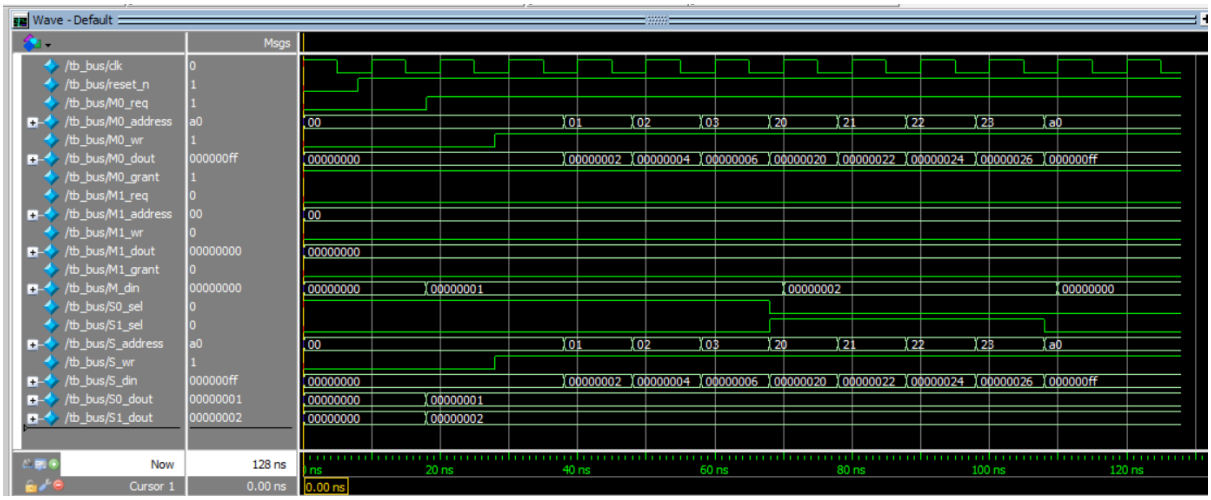

위의 waveform은 mux2_32bits.v의 testbench를 실행한 결과화면이다. S = 0이면 d0를 선택하고, s = 1이면 d1의 값을 선택하는 것을 확인할 수 있다.

6. mux3_32bits




위의 waveform은 mux3_32bits.v의 testbench를 실행한 결과화면이다. S = 00이면 d2를 선택하고, s = 01이면 d1의 값을 선택하며, s = 10이면 d0의 값을 선택하는 것을 확인할 수 있다.

7. Bus




위의 waveform은 bus.v의 testbench를 실행한 결과화면이다. Reset_n과 m0_req, m1_req의 값에 따라 m0_grant, m1_grant의 값이 변화하는 것을 확인할 수 있다. 그리고 m1_grant의 값을 selection 신호로 선택하여 slave write/read signal, slave address, slave data input 값을 선택하는 mux의 s값에 할당하고 그에 따른 결과값이 제대로 나오는 것을 확인할 수 있다.

## B. 합성(synthesis) 결과

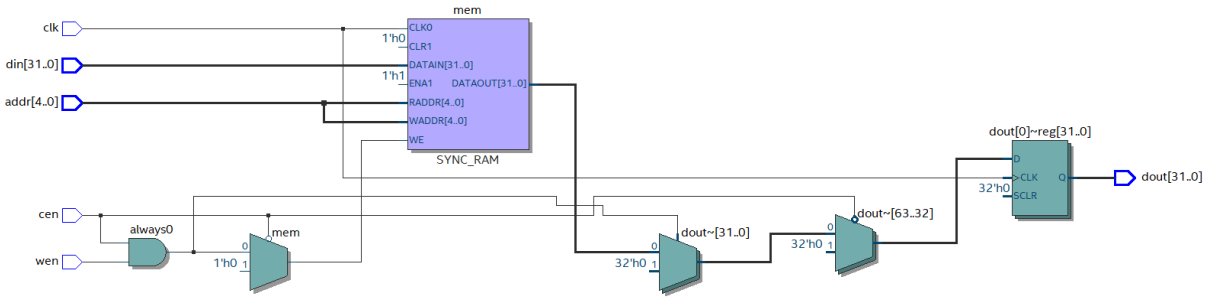
1. Ram

**Flow Summary**

<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Tue Nov 21 13:49:07 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) | 521 / 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 % ) |

Ram module은 logic utilization이 521 로 1%이고 total registers = 1056, total pins = 72 (14%)인 것을 확인할 수 있다.




위의 그림은 ram의 RTL map viewer이다. 해당 top module내에 선언한 memory가 포함되어 있는 것을 확인할 수 있다.

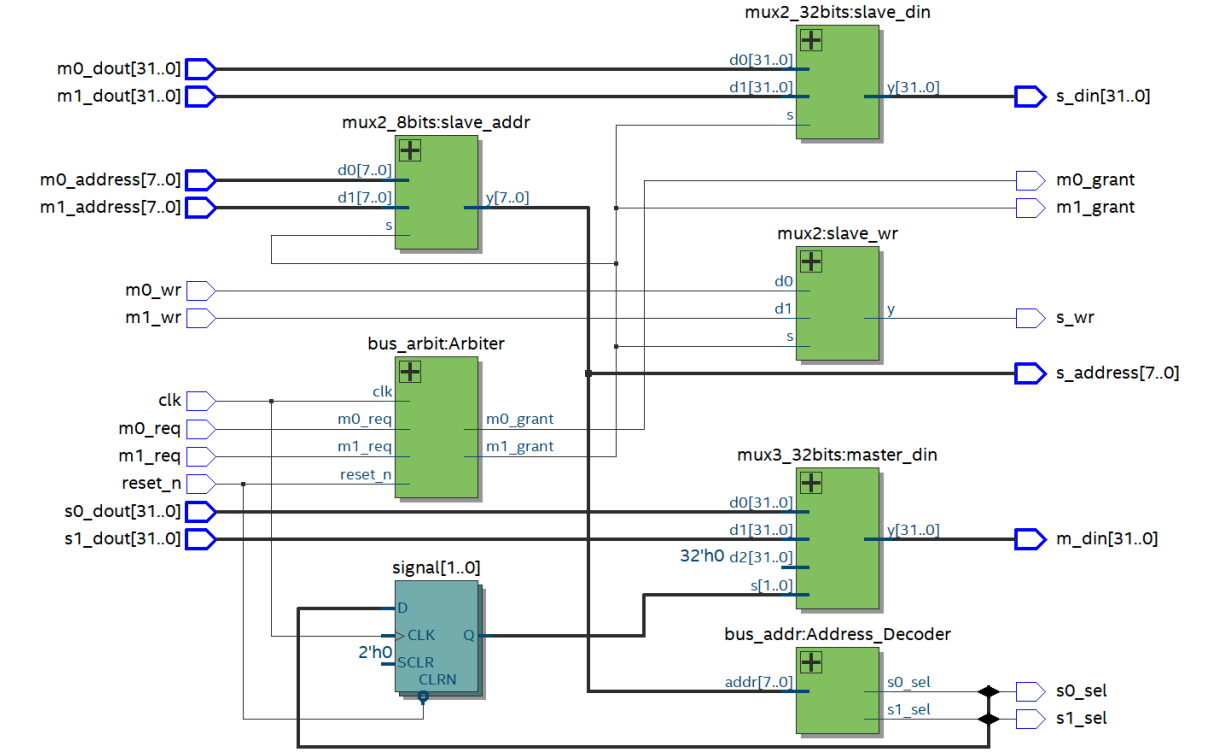
2. Bus

**Flow Summary**

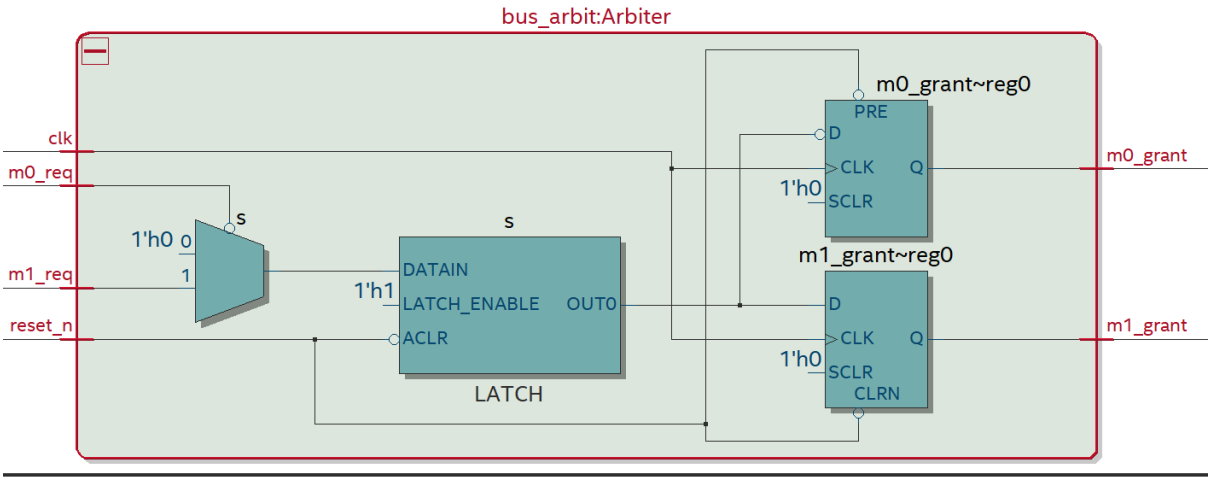
<<Filter>>

| | |
|---|---|
| Flow Status | Successful - Tue Nov 21 20:22:24 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) | 41 / 41,910 ( < 1 % ) |
| Total registers | 4 |
| 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 % ) |

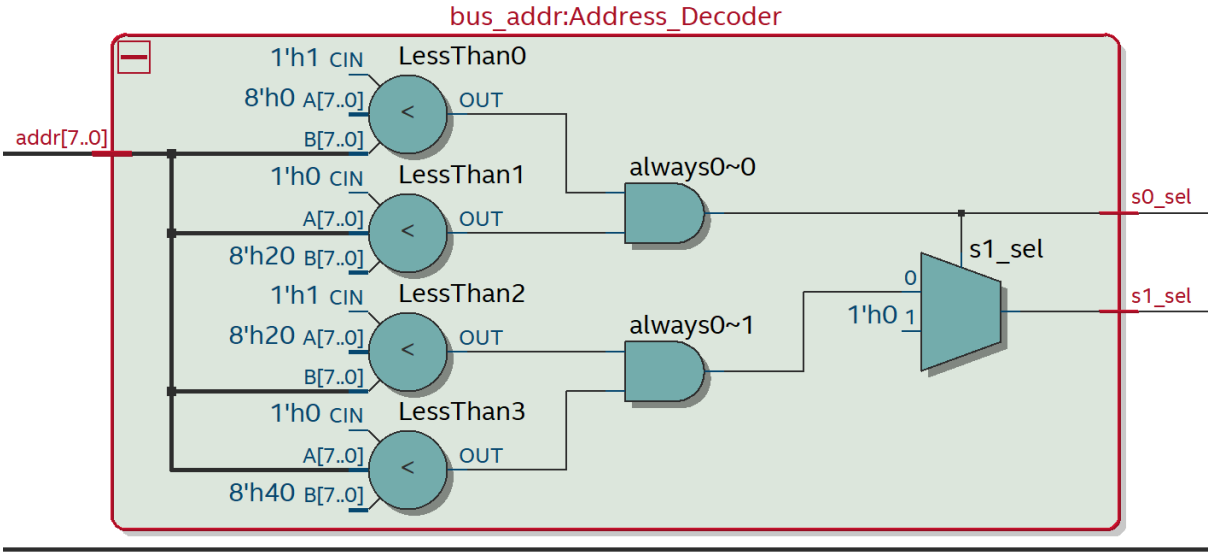
Bus module은 logic utilization이 41로 1%미만이고 total registers = 4, total pins = 227 (45%)인 것을 확인할 수 있다.



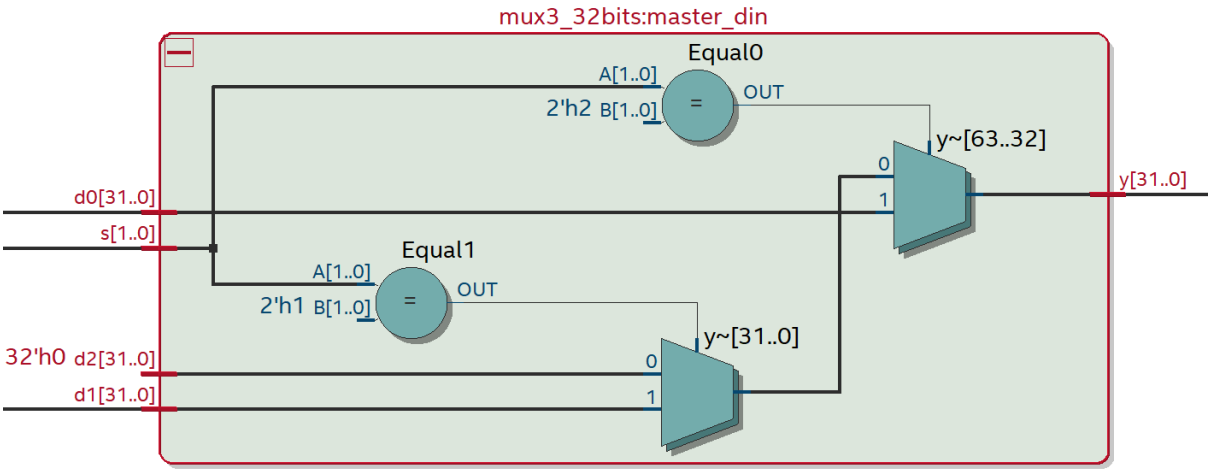

위의 그림은 bus의 RTL map viewer이다. 해당 top module내에 한 개의 1-bit 2-to-1 MUX, 8-bit 2-to-1 MUX, 32-bit 2-to-1 MUX, 32-bit 3-to-1 MUX가 instance화 되어있다. 또한 Arbiter와 Address Decoder의 instance도 포함되어 있다.




위 그림은 bus_arbit instance를 확대한 것이다.




위 그림은 bus_addr instance를 확대한 것이다.




위 그림은 mux3_32bits instance를 확대한 것이다.

## 5. 고찰 및 결론

### A. 고찰

1. Memory 내 특정 address의 값을 읽어오는데 문제가 발생하였다. Testbench waveform 결과로 dout의 값이 x로 나타났기 때문이다. 이는 ram module내에서 addr의 크기를 [5:0]인 6-bit로 잘못 작성해서 발생한 문제였다. 따라서 [4:0]인 5-bit로 수정하니 해결되었다.
2. Testbench의 waveform에서 s_addr의 값이 z가 되어 나타났다. 이는 top module인 bus module에서 address의 값을 8비트로 선언하여 생긴 문제였다.
3. Address Decoder testbench에서 illegal digit 오류가 발생하였다. 이는 hexadecimal 값으로 이루어진 값 앞에 h가 아닌 b를 사용하여 발생한 오류였다.

4. 32-bit 3-to-1 mux에서 처음에 할당된 값으로 M_din의 값이 바뀌지 않는 문제가 발생하였다. Always 구문과 if-else문으로 M_din의 값을 결정하였는데, 이러한 구문 대신에 3항 연산자를 2번 사용하여 d0, d1, d2의 값을 설정하니 제대로 M_din의 값이 할당되었다.

### B. 결론

본 과제에서 구현한 memory와 bus는 이후 factorial computation system에 사용될 주요한 module중 하나이다. Bus module을 사용하여 master interface와 slave interface 간의 상호작용을 수행할 수 있으며, memory를 사용하여 master interface로부터 받은 값을 write하거나 read하는데 사용할 수 있다.

## 6. 참고문헌


[1] 이혁준 / 컴퓨터공학기초실험2 Lab#10 Memory & Bus / 광운대학교 / 2023

[2] 주기억장치 / https://zrr.kr/wFhw

[3] RAM / https://zrr.kr/sOAq

[4] System bus / System Bus[버스란?] (tistory.com)

[5] integer / https://katteniiki.tistory.com/60