컴퓨터 공학 기초 실험2 보고서

실험제목: Simple Memory & Bus

실험일자: 2016년 10월 27일 (목)

제출일자: 2016년 11월 03일 (목)

학 과: 컴퓨터공학과

담당교수: 이혁준 교수님

실습분반: 목요일 0, 1, 2

학 번: 2013722095

성 명: 최재은

1. 제목 및 목적
   1. 제목

Simple Memory & Bus

* 1. 목적

RAM으로써 동작하는 간단한 Memory을 구현해보고 내부적으로 어떠한 신호를 통해서 어떤 값을 출력하도록 조정되는지를 이해함으로써 이것이 어떠한 방식으로 동작하는지 배운다. Bus가 무엇인지 알아보고, 이에 대한 개념을 적용하여 2개의 slave를 갖는 master 2개로 이루어진 간단한 형태의 bus를 구현해본다. 이를 통해 실제 컴퓨터가 어떤 식으로

Data를 처리하는지 공부한다.

1. 원리(배경지식)

RAM이란 본래 기억된 정보를 읽거나 다른 정보를 기억시킬 수 있는 메모리의 일종으로써 전원이 끊기면 기록된 정보가 날아가므로 휘발성을 성질로 갖는다. 이 때문에 데이터를 일시적으로 loading/saving 하는 데에 사용된다. 내부적으로 flip flop 방식의 메모리 셀을 가진 기억장치로, 소용량의 메모리나 캐시메모리에 주로 사용된다. 본래 pc에서는 임의의 주소에 데이터를 저장하는데, 본 실험에서는 입력 주소를 직접 주고 이에 32bit의

Data를 저장하고 있다. Bus란 컴퓨터상의 정보 전송 회로로써, 내부 기기 간의 데이터나 전원을 전송하는 용도로 사용된다. 또한 새로운 component를 추가하기 쉽고, 가격이 저렴한 특징을 갖는다. 본 실험에서는 2개의 slave와 master를 사용하여 구현하였으며, request와 grant signal을 통해서 master가 bus의 소유권을 갖는다. 이 소유권을 가진 master만이 bus를 통해서 data를 transfer할 수 있으며, 소유권을 갖고, request signal이 1인 동안에는 bus의 소유권을 상실하지 않는다. 두 개의 master가 모두 소유권을 갖지 않는 경우에는 master 0이 소유권을 갖는다.

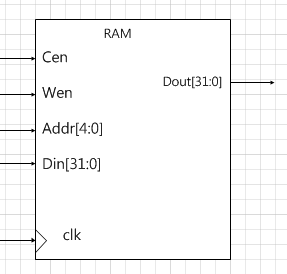
1. 설계 세부사항

1) Memory

- 32bit 길이의 data를 32개까지 address에 기반하여 저장합니다..

- module을 실행하면 32개의 register를 초기화합니다.

- 아래와 같은 간단한 형태로 표현할 수 있습니다.



- 아래와 같은 인자들을 갖습니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 이름 | 비트 수 | 설명 |
| Input | clk | 1 | Clock |
| cen | 1 | Chip enable |
| wen | 1 | Write enable |
| addr | 5 | Address |
| din | 32 | Data in |
| Output | dout | 32 | Data out |

- 다음과 같은 동작을 수행합니다.

|  |  |  |
| --- | --- | --- |
| Input | | Operation |
| cen | wen |
| 1 | 1 | Write data into memory which address’ pointing |
| 1 | 0 | Write memory data which address’ pointing into ‘dout’ |
| 0 | x | Set ‘dout’ as 0 |

2) Bus

- 여러 component 간에 data를 전송할 수 있도록 연결해주는 component.

- 2개의 slave와 master를 가지고 있습니다.

- slave는 아래와 같은 주소 범위를 갖게 되며, 이를 넘어가는 주소에 접근하게 되면

어떠한 slave도 선택하지 않습니다.

|  |  |
| --- | --- |
| Memory map | |
| Slave 0 | 0x00 ~ 0x1F |
| Slave 1 | 0x20 ~ 0x3F |

- master가 필요 시에 request signal을 high상태로 유지하고, 이에 대한 확인으로 grant

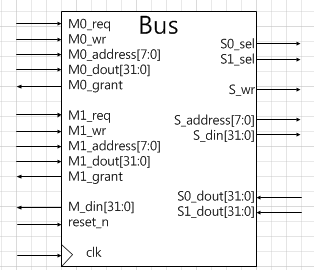
signal을 받은 후에 data를 전송할 수 있습니다.

- grant signal을 받은 master가 request signal을 high 상태로 유지하는 동안에는 bus의

소유권을 유지합니다.

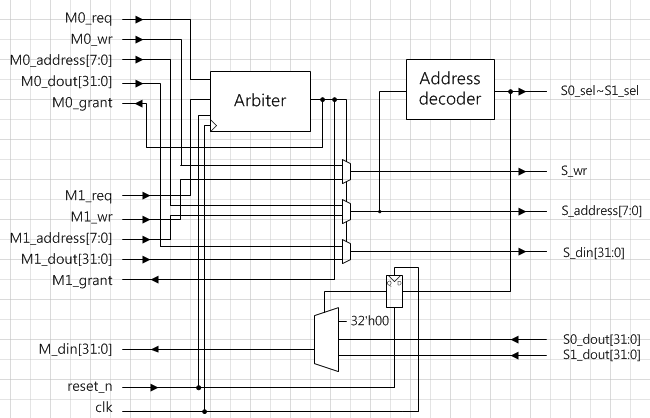
- 아래와 같은 인자들을 사용합니다.

|  |  |  |
| --- | --- | --- |
| **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 output |
| S1\_dout[31:0] | Slave 1 data output |
| Output | M0\_ | Master 0 grant |
| M1\_ | 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 |



- 오른쪽과 같은 형태로 간단하게 표현됩니다.

- 전체적인 구조는 다음과 같습니다.

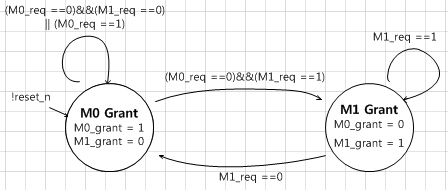


●bus\_arbit (arbiter)

- master의 grant에 따른 state를 결정해주는 module로써 각 master의 request와 현재

State에 따라 출력 값이 바뀝니다.

- 아래와 같은 state diagram으로 표현할 수 있습니다.



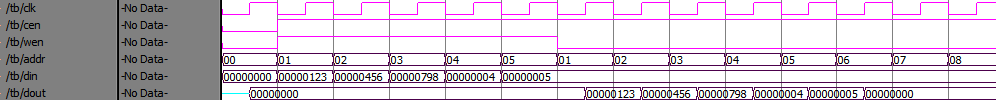
●bus\_addr (address decoder)

- 주소 값을 입력 받아 slave의 주소 범위에 따라서 selector 역할을 할 output을 출력합니다.

- 아래와 같은 조건하에 값을 출력합니다.

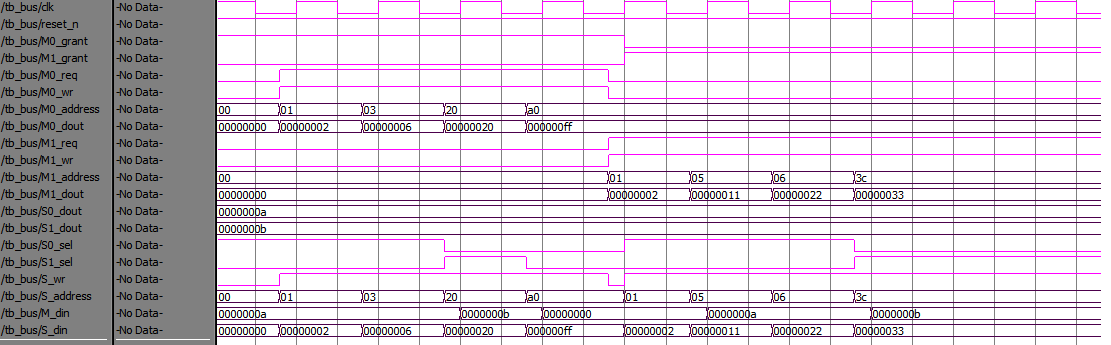
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | Condition | | | Output | |
| 0x00 | 0x20 | 0x40 | S0\_sel | S1\_sel |
| address | >= | < |  | 1 | 0 |
|  | >= | < | 0 | 1 |
| < |  | >= | 0 | 0 |

1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

 1) Memory

- 입력한 주소 값에 입력한 data 값이 저장 되고, read operation을 할 때 입력된 주소 값에

저장된 data가 출력됨을 알 수 있었습니다.

 2) Bus

- 위의 waveform에서 확인 할 수 있듯이, 초기 grant는 m0가 갖고 있으며, req와 wr가 1이 되고 입력한 address에 입력한 data (dout)이 입력됨을 확인할 수 있습니다.

- 두 master의 req가 변경되어 grant가 변경되는 것을 확인할 수 있습니다.

- 또한 입력 address가 0x20 이상이 되자 S0\_sel, S1\_sel이 바뀌는 것을 확인할 수 있습니다.

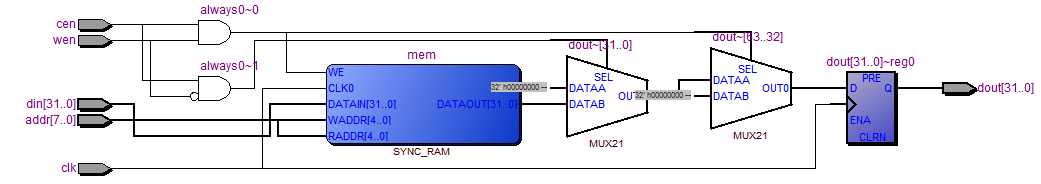
- address가 0x3f 이상인 경우에는 slave selecter가 모두 0이 되는 것을 확인할 수 있습니다.

- slave관련 변수들 또한 정상적으로 출력되었습니다.

* 1. 합성(synthesis) 결과

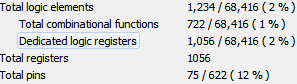
1) Memory

**<RTL Viewer>**

 - din이 memory로 들어가 cen, wen에 의해 조건적으로 flip flop으로 들어가며, 클럭에

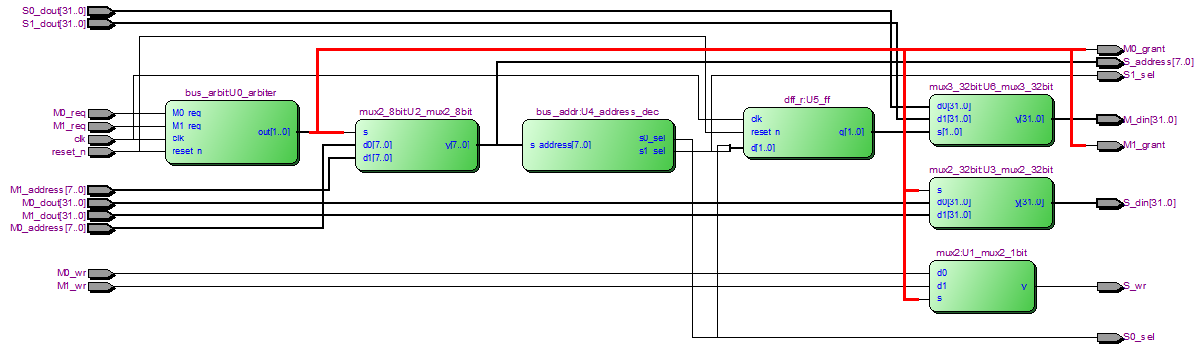
동기화되어 주기적으로 출력되는 것을 확인할 수 있었습니다.

**<Flow summary>**

 - 총 1234개의 logic이 사용되었고, 32bit 변수 가 33개(dout, mem [0:31]) 사용되었으므로 1056개의 register가 사용되었으며, integer i,

1bit input clk, cen, wen, 32bit input din, 32bit output dout해서 총 75개의 pin이 사용되었습니다.

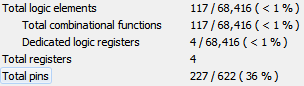
2) Bus

**<RTL Viewer>**

**-** 설계방법에서 설명한 것과 동일하게 입출력이 일어나는 것을 확인할 수 있었습니다.

- bus\_arbit module에서 출력되는 값의 2의 자리 비트 값을 selector로 하여 32bit, 1bit

mux2 module을 control합니다.

**<Flow summary>**

- 총 117개의 logic이 사용되었으며, Output

reg로 M0\_grant, M1\_grant, S\_address, dec\_out을 사 용하여 4개의 register가 사용되었고, 입출력 변수로 총 227개의 pin이 사용되었습니다.

.

1. 고찰 및 결론
   1. 고찰 & 결론

이번 실험을 진행하면서 특별하게 어려운 점은 없었습니다. Test bench를 진행하면서 slave output이 나오지 않아서 당황했었는데, S0\_dout, S1\_dout도 input 이라는 것을 잠시 망각하여 생긴 실수였습니다. 이번 실험을 통해서 register file을 이용하지 않고, d flip

flop만을 이용해 간단한 RAM을 만들 수 있다는 것과 arbiter로 master의 결정권을

조건적으로 설정해 주는 것을 통해 실제 pc에서처럼 bus를 우선권에 따라 사용하는 것의 원리를 배웠습니다. 도대체 이런 개념을 확립하고 설계했던 사람들은 어떤 정신세계를 갖고 있는지마저 궁금해졌습니다.

1. 참고문헌

네이버 지식백과