디지털논리회로설계 2 – Term Project

"꿈을 가지십시오. 그리고 정열적이고 명예롭게 이루십시오."

1. System Overview

이번 학기 설계 과제는 Testbench 가 BUS 를 통해 ALU with Multiplier 에 명령을 내리고, 그 결과를 memory 에 저장하는 디지털시스템을 설계하고 검증하는 것이다. 과제의 자세한 specification 을 충분히 이해하여 과제를 수행하도록 한다.

Figure 1 은 구현할 시스템의 블록 다이어그램이다. 본 시스템은 ALU with Multiplier, BUS, Memory 로 구성되고 testbench 를 이용하여 시스템의 동작을 제어한다. ALU with Multiplier 는 BUS 를 통해 접근할 수 있는 register 집합 (i.e., *operandA, operandB, opcode, …* (Figure 1 에 *Italic font* 로 표기))을 가지고 있다. 이 register 를 통해 외부 모듈과 데이터를 주고받을 수 있다. Multiplier 와 ALU 의 자세한 register 의 기능과 register map 은 뒤의 자세한 내용을 참고한다.

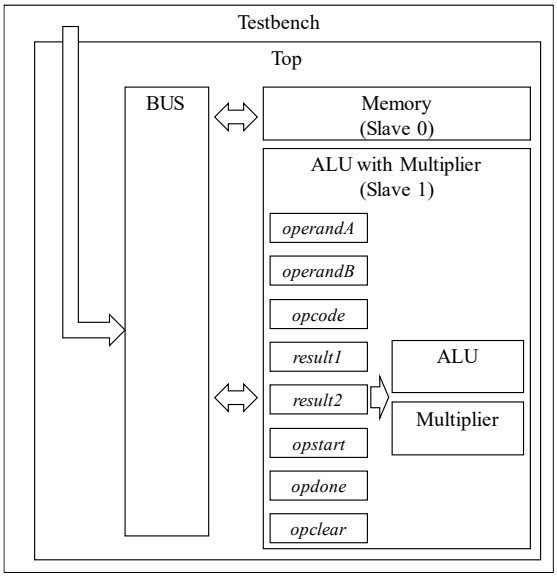

Figure 1. System overview

본 시스템은 testbench 로부터 20 개의 data 를 memory 1 에 각각 저장한다. Testbench 에 지정된 20 개의 execution data set 을 이용하여 ALU with Multiplier 를 이용해 결과를 연산한다. 여기서 execution data set 은 ALU with Multiplier 가 동작하기 위한 데이터와 control 신호 집합(operandA, operandB, opstart, opcode)을 의미한다. 아래의 내용은 시스템의 수행 순서를 자세히 보여준다. 모듈의 pin 은 3.2 장을 register 의 이름과 기능은 3.3 장을 참고한다.

1. Testbench 는 ALU with Multiplier 의 *operandA* 와 *operandB* register 에 연산자 값을 저장하며, *opcode[3:0]* register 에 opcode 값을 저장한다. 자세한 opcode 는 3.4 장을 참고한다.
2. Testbench 는 *opstart[0]*에 1 을 써 ALU with Multiplier 의 연산을 시작한다.

3. Testbench 가 *opdone* 를 읽어 연산 진행 상황을 파악한다. 연산이 종료될 때까지 매 사이클 *opdone* 을 읽는다. *opdone* 이외의 register 은 ALU with Multiplier 의 동작이 완료될 때까지 접근하지 않는다.
   - ✓ *opdone[1:0]* == 2'b00: 연산 대기
   - ✓ *opdone[1:0]* == 2'b10: 연산 시작
   - ✓ *opdone[1:0]* == 2'b11: 연산 완료
   - ✓ *opdone[31:2]*: don't care
4. ALU with Multiplier 는 *opstart[0]*가 1 이 되면 연산을 수행한다. 연산을 시작할 때 다음과 같은 동작이 동시에 수행된다.
   - ✓ *opcode[3:0]*에 저장된 opcode 에 맞춰 *operandA* 혹은 *operandA* 와 *operandB* register 에 저장된 값을 이용해 ALU with Multiplier 내의 ALU 혹은 multiplier 를 이용해 연산을 수행한다.
   - ✓ 연산을 진행하는 상태를 나타내기 위하여 *opdone[1]*에 1 을 쓴다.
   - ✓ *opstart* 를 0 으로 초기화 한다.
5. ALU with Multiplier 는 연산이 완료될 때 다음과 같은 동작을 동시에 수행한다.
   - ✓ 곱셈 이외의 연산 결과는 *result1* 에 저장하며 곱셈 연산 결과는 *result1*(하위 32bit)와 *result2*(상위 32bit) register 에 저장한다.
   - ✓ *opdone*[0]에 1 을 쓴다.
6. Testbench 는 연산이 종료될 때 (*opdone*[1:0]==2'b11)일 때 앞서 *opcode* 에 저장한 연산에 맞춰 ALU with Multiplier 의 *result1* 혹은 *result1* 와 *result2* 값을 읽는다.
   - ✓ Testbench 가 bus 를 통해 읽을 수 있는 값은 32 bits 이므로 *result1* 과 *result2* 를 읽을 경우 2 회에 걸쳐 register 를 접근하여 값을 읽어 온다.
7. Testbench 가 result register 의 값을 읽은 후 다음 연산을 수행할 수 있도록 ALU with Multiplier 를 초기화 한다. Testbench 는 아래 방법 중 한 방법을 선택해 초기화 동작을 수행한다.
   - ✓ Testbench 가 ALU with Multiplier 의 *opdone* register 를 0 으로 초기화 한다.
   - ✓ Testbench 가 ALU with Multiplier 의 *opclear[0]*에 1 을 써 ALU with Multiplier 내의 모든 register 값을 초기화 한다.
8. Testbench 는 6 번 과정을 통해 읽은 계산 결과를 Memory 에 값을 쓴다. 주소는 Testbench 가 임의의 주소를 지정한다.
9. 1 에서 8 번까지의 과정을 필요에 따라 반복한다.

➢ Testbench 는 ALU with Multiplier 초기화 동작(7 번)과 Memory 에 값 쓰기(8 번)동작의 순서를 바꿔서 동작 할 수 있다.

구체적인 설계 사양은 업데이트될 수 있습니다. 업데이트될 때마다 공지사항에 공지할 테니 확인하기 바랍니다.

참고사항: Port/module 이름은 "Courier New" font 를 사용하며 register 이름은 *Italic font* 를 사용한다.

2. Top

Top 모듈은 BUS, ALU with Multiplier 와 Memory 를 instance 하여 연결한 모듈이다. Top 모듈의 input port 를 이용해 Top 모듈 내에 있는 BUS 의 master port 에 접근이 가능하다. Top 은 memory mapped I/O 방식을 사용하여 외부(testbench)에서 주소를 통해 Top 모듈의 slave (ALU with Multiplier, Memory) components 에 접근이 가능하다.

2.1. Features

Figure 2.는 Top 모듈의 schematic symbol 이다. Bus 의 master interface 가 input/output port 를 통해 연결되어 있다. Top 모듈 내의 memory map 은 Table 1 과 같다.

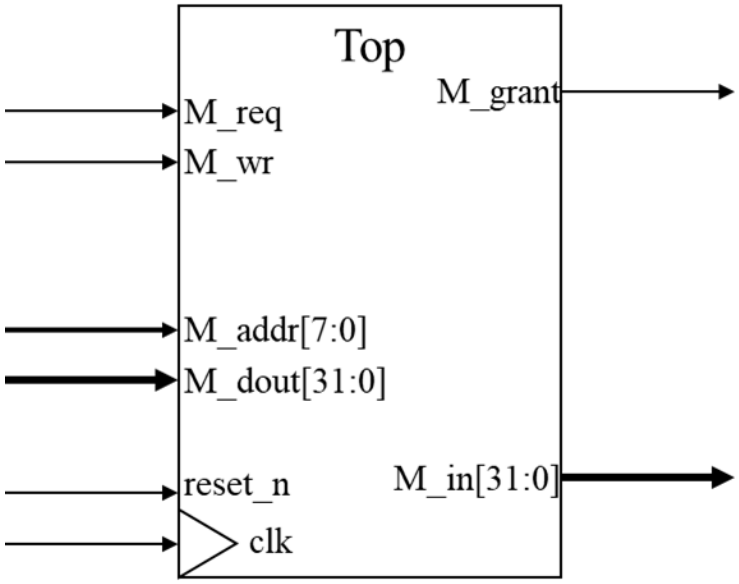

Figure 2. Schematic symbol of Top

Address region (in Hexadecimal)	Component
0x00 ~0x1F	Memory
0x30~0x3F	ALU with Multiplier

Table 1. Memory-mapped IO

2.2. Pin description

- Module 명은 `Top` 이다.
- Table 2 는 `Top` 의 port 의 이름과 방향을 설명한다.
- Port 의 이름과 bit width 는 반드시 동일해야 한다.

Table 2. Pin description of `Top`

Direction	Port name	Bit width	Description
Input	`clk`	1	Clock
	`reset_n`	1	Active low reset
	`M_req`	1	Master request
	`M_wr`	1	Master write/read
	`M_addr`	8	Master address
	`M_dout`	32	Master data output
Output	`M_grant`	1	Master grant
	`M_din`	32	Master data in

3. ALU with Multiplier

ALU with Multiplier 는 operand A 혹은 operand A 와 B 를 이용한 산술, 논리 연산 결과 값을 도출하는 하드웨어이다.

3.1. Features

- 곱셈 연산을 제외한 모든 연산은 32 bits operand 한 개 혹은 두 개를 이용해 연산하며, 32 bits 연산 결과를 *result1* 에 저장한다.
- 곱셈 연산의 경우 두 개의 32 bits operand 를 이용해 곱셈 연산을 수행한다. 곱셈의 결과는 64 bits 로 *result1* register 에 결과의 하위 32 bits 를 저장하며 *result2* register 에 결과의 상위 32 bits 결과를 저장한다.
- 산술 연산은 operator (예: 곱셈: *, 덧셈 +)로 구현하면 안 된다.
  - ✓ 덧셈 및 뺄셈은 CLA(carry look-ahead adder)를 이용한다.
  - ✓ 덧셈 및 뺄셈은 한 개의 CLA 를 사용한다.
  - ✓ 곱셈은 booth multiplier 를 이용한다.
- Shift 연산은 MUX 를 이용한 shifter module 을 instance 해 사용한다.
  - ✓ “<<” operator, “{}”concatenation operator 를 사용하지 않는다.
- *opstart[0]*가 1 일 경우 register 에 저장된 값을 이용해 연산을 시작한다.
- 연산을 시작할 때 *opdone[1]*에 1 을 쓰며 동시에 *opstart* 를 0 으로 초기화 한다.
- 연산이 끝날 경우 *opdone[0]*에 1 을 쓰고 대기한다.
- *opclear[0]* 가 1 일 경우 모든 register 의 값을 초기화 한다.
  - ✓ 자세한 register 의 동작과 기능은 3.3 을 참고
- Figure 3 은 ALU with Multiplier 의 schematic symbol 로 input/output port 의 이름과 bit width 를 나타낸다.

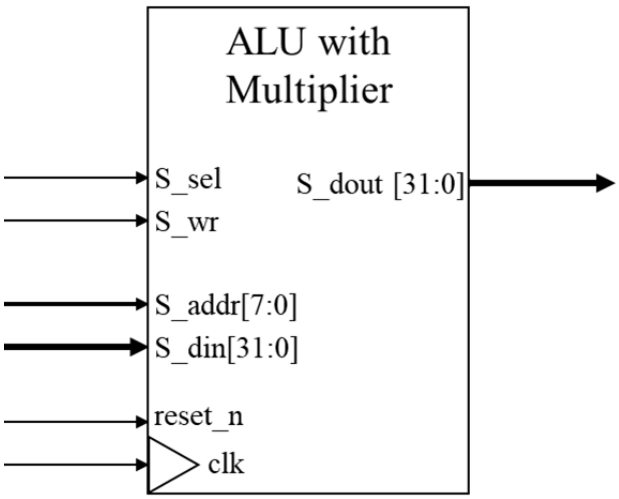

Figure 3. Schematic symbol of ALU with Multiplier

3.2. Pin description

- Module 명은 `ALUwMul` 이다.
- Table 3 는 `ALUwMul` 의 port 의 이름과 방향을 설명한다.
- Port 의 이름과 bit width 는 반드시 동일해야 한다.

Table 3. Pin description of `ALUwMul`

Direction	Port name	Bit width	Description
Input	`clk`	1	Clock
	`reset_n`	1	Active low reset
	`S_sel`	1	(Slave interface) Select
	`S_wr`	1	(Slave interface) Write/read
	`S_addr[7:0]`	8	(Slave interface) Address
	`S_din[31:0]`	32	(Slave interface) Data input
Output	`S_dout[31:0]`	32	(Slave interface) Data output

3.3. Register description

- Table 4 는 ALU with Multiplier 내부의 register 를 설명한 것이다. Register 의 bit width 는 기본적으로 32 bits 이다.

- ✓ Register type 은 W(write)와 R(read)로 구별된다.
  W(write)의 경우, slave interface 를 통해 외부에서 해당 register 에 값을 write 할 수 있다.
  R(read)의 경우, slave interface 를 통해 외부에서 해당 register 의 값을 read 할 수 있다.
  - Testbench 는 W type register 을 읽지 않으며 R type register 에 값을 쓰지 않는다.
- ✓ Register 에서 사용되지 않는 bit 는 reserved 이다.
  예를 들어 '*opstart[31:1]*가 reserved 이다'의 의미는 *opstart* register 의 [31:1] bits 에 쓰여진 값은 무시되며, read 된 값은 의미가 없음을 나타낸다.
- ✓ Default value 는 reset 이 되었을 때 초기 값을 의미한다.
- ✓ Table 4 에서 모든 register 의 default value 는 0x00000000 이다.

Table 4. Register description of `ALUwMul`

Offset	Type	Bit width	Name	Description	Default value
0x0	W	32 bits	*operandA*	피연산자 A 의 값을 저장하고 있는 register.	0x00
0x1	W	32 bits	*operandB*	피연산자 B 의 값을 저장하고 있는 register.	0x00
0x2	W	32 bits	*opcode*	연산자 정보를 가지고 있는 register. [31:4] bits 는 reserved 이다.	0x00
0x3	W	32 bits	*opstart*	[0]bit 에 1 이 써지면 *operandA, operandB, opcode* register 의 값을 이용해 연산을 시작한다. 연산 중 혹은 연산 완료 후 register 초기화 이전에 해당 register 의 [0]에 1 이 써지면 해당 값은 무시된다. [31:1] bits 는 reserved 이다.	0x00
0x4	R/W	32 bits	*opdone*	연산이 시작될 경우 [1] bit 에 1 을 써 동작을 수행하고 있음을 나타낸다. opcode 값에 해당되는 연산이 완료되었을 경우 [0] bit 에 1 을 써 연산이 끝났음을 기록한다. [31:2] bits 는 reserved 이다.	0x00
0x5	W	32 bits	*opclear*	해당 register 의 [0]bit 에 1 이 써지면 모든 register 의 값이 default value 가 된다. opclear 를 통한 reset 은 clock 과 synchronous 하다. [31:1] bits 는 reserved 이다.	0x00
0x6	R	32 bits	*result1*	연산의 결과 값을 저장할 때 사용되는 register.	0x00

0x7	R	32 bits	*result2*	곱셈 연산의 상위 32 bits 결과 값을 저장할 때 사용되는 register.	0x00

3.4. Opcode description

- Table 5 는 ALU with Multiplier 의 opcode 를 나타낸 표이다. ALU with Multiplier 는 총 14 가지의 연산을 수행할 수 있다.
  - ✓ Opcode 가 0xE, 0xF 가 되는 경우는 없다 가정한다.
- Opcode 는 수행할 연산을 나타내는 값으로 *opcode* register 에 저장되어 있다.
  - ✓ Opcode 의 bit width 는 4 bits 이다.
  - ✓ Set less than 명령과 set greater than 명령은 **부호가 없는** operand A 와 B 를 비교한 결과를 *result1* register 에 저장한다. 각 연산의 결과는 아래와 같다.
    - Set less than 명령은 operand A 와 B 를 비교하여 A 가 더 작을 경우 result 를 1 을 출력한다. Operand A 와 operand B 가 같을 경우 결과는 0 이다.
    - Set greater than 명령은 operand A 와 B 를 비교하여 A 가 더 클 경우 result 를 1 을 출력한다. Operand A 와 operand B 가 같을 경우 결과는 0 이다.
  - ✓ Shift 연산 (LSL, LSR, ASR)의 경우 shift amount 는 1 로 고정하며 *operandB* register 값을 이용해 연산한다.
    - 예를 들어 *operandB* register 값이 0x10 이며 LSL 연산을 수행할 경우 결과는 0x20 이다.

Table 5. Opcode of `ALUwMul`

Opcode	Operation
0x0	NOT A
0x1	NOT B
0x2	AND
0x3	OR
0x4	XOR
0x5	XNOR
0x6	Set less than
0x7	Set greater than
0x8	Shift left logical
0x9	Shift right logical
0XA	Shift right arithmetic
0xB	Addition
0xC	Subtraction
0xD	Multiply

3.5. Functional description

ALU with Multiplier 는 bus 와 연결된 slave component 이다. Bus 와 연결된 master (testbench)는 ALU with Multiplier 의 slave interface (`S_sel`, `S_wr`, `S_addr`, `S_din`)를 통하여 ALU with Multiplier 의 내부 동작을 제어한다. Master 는 slave interface 를 통하여 ALU with Multiplier 에 포함된 register 에 미리 정의된 값을 써 제어한다. 예를 들어, *opstart* 에 1 을 써 연산을 시작한다. 각각의 register 들은 offset 이 할당되어 있으며 register 의 bit 위치마다의 값에 따라서 모듈의 동작을 제어한다.

Offset 은 기준 주소(base address)와 목표 주소(target address)의 차이를 지칭할 때 쓰는 단어이다. 예를 들어 ALU with Multiplier 의 기준 주소가 0x30 이며 ALU with Multiplier 의 *result1* register 주소가 0x33 일 경우 offset 은 3 이 된다. ALU with Multiplier 의 register offset 을 넘어가는 주소가 입력될 경우 ALU with Multiplier 는 해당 입력을 무시한다.

ALU with Multiplier 의 slave interface 는 `S_addr` 를 이용해 offset 을 계산한다. Offset 과 `s_wr` 를 이용해 register 에 `S_din` 데이터를 write 하거나 register 값을 `S_dout` write (master 입장에선 read)한다. `s_wr` 가 0 일 경우 register read 를, `s_wr` 가 `1` 일 경우 register write 을 동작한다.

ALU with Multiplier 의 *opstart* register 에 값이 1 이 써질 경우 동작 순서는 다음과 같다

1. *opstart* 의 값을 0 으로 바꾸며 *opdone[1]*에 1 을 쓴다.
2. *opcode* 에 저장된 값을 읽어 연산에 맞는 *operand* 의 값을 읽는다.
   opcode 의 값이 13 를 넘어가는 경우(0xE 혹은 0xF 일 경우)는 없다 가정한다.
3. *opcode* 에 맞춰 연산을 수행한다.
   - *opcode* 의 값이 0xD 가 아닐 경우 (곱셈 연산이 아닐 경우) 모듈 내의 ALU 가 *operand* register 를 읽어 연산에 맞춰 연산을 수행한다. 모든 연산은 한 사이클 뒤 결과가 출력된다.
   - *opcode* 의 값이 0xD 일 경우 (곱셈 연산일 경우) 모듈 내의 multiplier 가 *operand* register 를 읽어 연산을 수행한다. 자신이 구현한 booth multiplier radix 에 따라 결과가 출력된다.
4. 앞선 과정을 통해 연산이 완료되었을 때 *opdone[0]*에 1 을 쓴다.
5. 아래의 방법으로 ALU with Multiplier 가 초기화 되기 전까지 register 값을 유지한다. 이 때 새로운 연산을 시작할 수 없다.
   - ✓ *Opdone* 을 0 으로 초기화한다.
   - ✓ *opclear*[0] 가 1 이 될 경우 *opclear* 를 포함한 모든 register 값을 default value 로 초기화한다.

4. BUS

Bus 는 여러 component 들 간에 data 를 전송(transfer)할 수 있도록 연결해주는 component 이다. Bus 는 새로운 component 들을 추가하기가 쉬우며, 가격이 저렴한 특징을 가지고 있다.

4.1. Features

- Bus 는 1 개의 master interface 와 2 개의 slave interface 를 가지고 있다.
- Address 는 8 bits 이다.
- Data 는 32 bits 이다.
- Slave 0 은 0x00 ~ 0x1F 사이의 address 를 memory map region 으로 가진다.
- Slave 1 은 0x30 ~ 0x3F 사이의 address 를 memory map region 으로 가진다.
- Slave 0 과 slave1 address area 외 주소에 접근할 경우 `S0_sel`, `S1_sel` 를 0 으로 바꾼다.
- Figure 4 은 Bus 의 schematic symbol 로 input/output port 의 이름과 bit width 를 나타낸다.
- 왼쪽은 master interface 를 나타내며, 오른쪽은 slave interface 를 나타낸다.

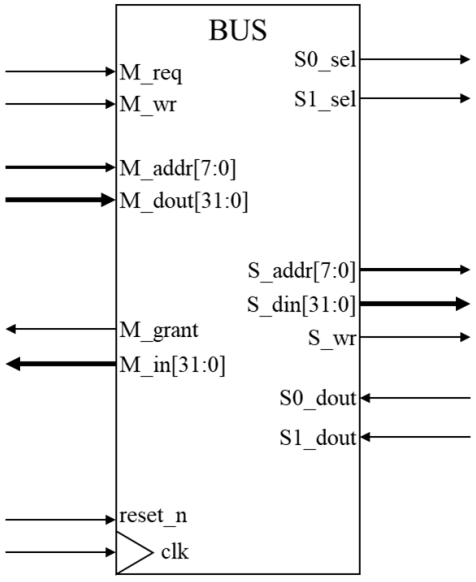

Figure 4. Schematic symbol of bus

4.2. Pin description

- Module 명은 `BUS` 이다.
- Table 6 은 `BUS` 의 pin 을 정리한 것이다.
- Port 의 이름과 bit width 는 반드시 동일해야 한다.
  - ✓ ***Notice***: Master 에서 data-out 이 output pin 이지만, BUS 에서는 해당 data-out 을 받아야 하기 때문에 data-out 이 input pin 이 된다.
  - ✓ ***Notice***: master 에서 data-in 이 input pin 이지만, BUS 에서는 data-in 이 output pin 이 된다. 이는 slave 에도 똑같이 적용된다.

Table 6. Pin description of `BUS`

Direction	Port name	Bit width	Description
Input	`clk`	1	Clock
	`reset_n`	1	Active low reset
	`M_req`	1	Master request
	`M_wr`	1	Master write/read
	`M_addr`	8	Master address
	`M_dout`	32	Master data output
	`S0_dout`	32	Slave 0 data out
	`S1_dout`	32	Slave 1 data out
Output	`M_grant`	1	Master grant
	`M_din`	32	Master data input
	`S0_sel`	1	Slave 0 select
	`S1_sel`	1	Slave 1 select
	`S_addr`	8	Slave address
	`S_wr`	1	Slave write/read
	`S_din`	32	Slave data input

4.3. Functional description

프로젝트에서 구현할 BUS 는 1 개의 master interface 와 2 개의 slave interface 로 구현되어 있다.

아래의 내용은 BUS 의 동작 순서이다. 이번 프로젝트는 master 가 1 개이므로 BUS 를 사용할 때 여러 개의 master 가 bus 를 요청하는 동작은 생략되어 있다.

1. Master 는 BUS 를 통해 slave 와 communication 하고자 할 때, request signal 인 M_req 신호를 1 로 바꿔 BUS 사용을 가능 여부를 확인한다.

2. BUS 는 master 가 bus 사용을 요청했을 때 master 에게 grant signal 인 M_grant 신호를 통해 BUS 사용을 허락한다. 다시 말해, M_req 는 master 가 BUS 에 대한 소유권을 요청하는 것이며, M_grant 신호는 BUS 가 master 에게 BUS 에 대한 소유권을 할당해주는 것이다.

3. Master 가 M_grant 신호를 받은 후에는 M_addr 신호를 통해 slave 의 memory map 에 따라 slave 와 communication 을 수행한다.

4. Master 의 communication 동작이 완료될 경우 M_req 신호를 0 으로 만들어 BUS 사용을 끝내며 BUS 는 M_grant 신호를 0 으로 만든다. 만약, Master 의 M_req 신호가 1 인 동안에는 M_grant 가 0 이 되지 않으며 communication 를 계속할 수 있다.

5. Memory

Memory 는 address 에 기반하여 data 를 저장하는 hardware 로, 해당 프로젝트에서는 random access memory(RAM)을 구현하도록 한다.

5.1. Features

- Address 는 5 bits 이다.
- Memory 의 address 는 `BUS` 의 LSB(least significant bit)부터 address 를 받는다.
  - ✓ Memory 의 `S_addr` 의 LSB 는 `BUS` 의 `S_addr` 의 LSB 와 일치한다. 다시 말해 Memory 의 `S_addr` 의 MSB 는 `BUS` 의 `S_addr` 의 5 번째 bit (`S_addr[4]`)와 일치한다.
- Data 는 32 bits 이다.
- Memory 는 내부에 32 개의 data 를 address 에 기반하여 저장한다.
- Figure 5 은 Memory 의 schematic symbol 로 input/output port 의 이름과 bit width 를 나타낸다.

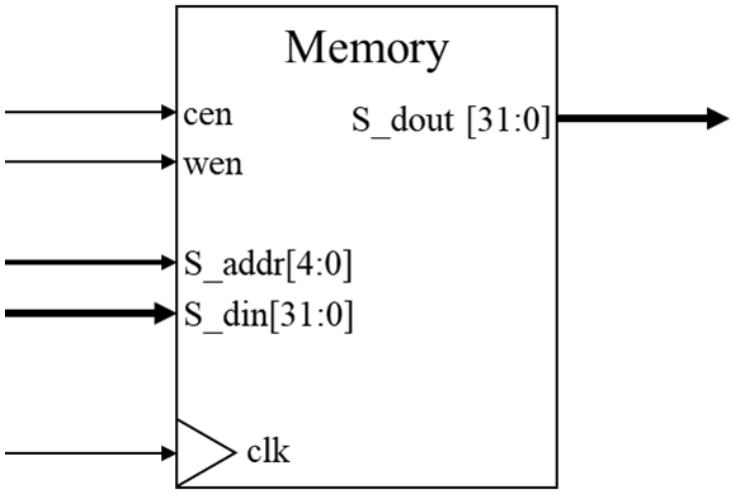

Figure 5. Schematic symbol of memory

5.2. Pin description

- Module 명은 `ram` 이다.
- Table 7 은 `BUS` 의 pin 을 정리한 것이다.
- Port 의 이름과 bit width 는 반드시 동일해야 한다.

Table 7. Pin description of `ram`

Direction	Port name	Bit width	Description
Input	`clk`	1	Clock
	`cen`	1	Chip enable
	`wen`	1	Write enable
	`S_addr`	5	Address
	`S_din`	32	Data in
Output	`S_dout`	32	Data out

5.3. Functional description

Memory 모듈의 동작은 아래와 같다

- `cen` 과 `wen` 이 모두 1 이면 address (`S_addr`) 가 가리키는 Memory 에 `S_din` 을 write 한다. 이때 `S_dout` 은 0 을 출력한다.
- `cen` 이 1 이고, wen 이 0 이면, address (`S_addr`) 가 가리키는 memory 의 값을 `S_dout` 에 write 한다. `cen` 이 0 이면, `S_dout` 은 0 이 된다.

6. Important dates

- Issue date: 11 월 11 일 금요일 (11 주차 금요일)
- Testbench 이용 결과 검증
  - ✓ 5 개의 testbench 를 이용하여 결과 검증
  - ✓ Testbench 내용은 미공개
- Project 결과 발표
  - ✓ 발표 희망자는 조교에게 메일 보낼 것(보너스 가점)
  - ✓ 12 월 2 일 (14 주차 금요일)
- 최종 결과와 보고서 제출
  - ✓ 12 월 3 일 23:59 까지 컴기설 2 분반 과제 제출란에 제출
    (실습 미수강은 디지털 논리회로 2 과제 제출란에 제출)
- 최종 결과 보고서의 경우, **늦은 제출은 받지 않습니다.**
- 최종 결과 보고서 제출시 source code 를 report 와 같이 압축하여 upload 한다.
  - ✓ Testbench 는 Top, Bus, ALU with Multiplier, memory 를 각각 검증한 파일을 포함한다.
- Source code 는 project 구현에 사용된 모든 설계 파일 및 이에 대한 testbench 를 포함한다.
- 파일 압축할 때 source code 는 db, incremental_db, simulation 폴더는 삭제하고, 프로젝트가 생성된 기본 폴더와 report 를 함께 압축한다.
- 파일이름은 'Project_(분반)_(학번).zip'
  - ✓ 중간 저장에 용이하게 Project 뒤에 버전을 기록해 KLAS 에 여러 번 제출해도 된다.
    - 예) Project_(분반)_(학번)_v2.zip'
  - ✓ 채점의 경우 최종 버전을 기준으로 한다.

7. Report outline

- 제안서
  - ✓ 보고서: 최소한 다음의 내용들이 포함되어야 하며 그 외의 것을 추가하는 것은 자유다. 과제제목, 과제목표, 일정, 각 모듈 별 구현 방법(state transition diagram 또는 내부 register map) 기술, 예상되는 문제점, 검증전략
- 결과보고
  - ✓ 발표: PowerPoint 로 발표 희망자에 한해 10 분 분량으로 작성
  - ✓ 보고서: 최소한 다음의 내용들이 포함되어야 하며 그 외의 것을 추가하는 것은 자유다.
    - Introduction
    - 일정 및 계획을 포함할 것
    - Project specification
    - Design details
    - Design verification strategy and results
    - Conclusion

      과제 완료 후 기대되는 학습효과를 포함할 것

      보고서의 ideal 한 양식은 논문의 형태입니다. (논문 형태는 공지사항에 업로드 예정)
- Score
  - ✓ 제안서: 20%
  - ✓ Testbench: 40%
    - Bus: 15%
    - Memory: 15%
    - ALU with Multiplier: 30%
    - TOP: 40%
  - ✓ 결과보고서: 40%