디지털 논리회로설계 2 – Term Project

Arithmetic & logical computing system

“꿈을 가지십시오. 그리고 정열적이고 명예롭게 이루십시오.”

1. System Overview

본 시스템은 arithmetic logical unit (ALU)을 이용하여 곱셈, 덧셈, 뺄셈 및 논리연산을 한다. 과제의 자세한 specification 을 충분히 이해해 수행한다.

Figure 1 은 구현할 시스템의 블록 다이어그램이다. 본 시스템은 ALU, direct memory access controller (DMAC), random access memory (RAM), bus 로 구성되고, testbench 를 이용하여 시스템의 동작을 제어한다.

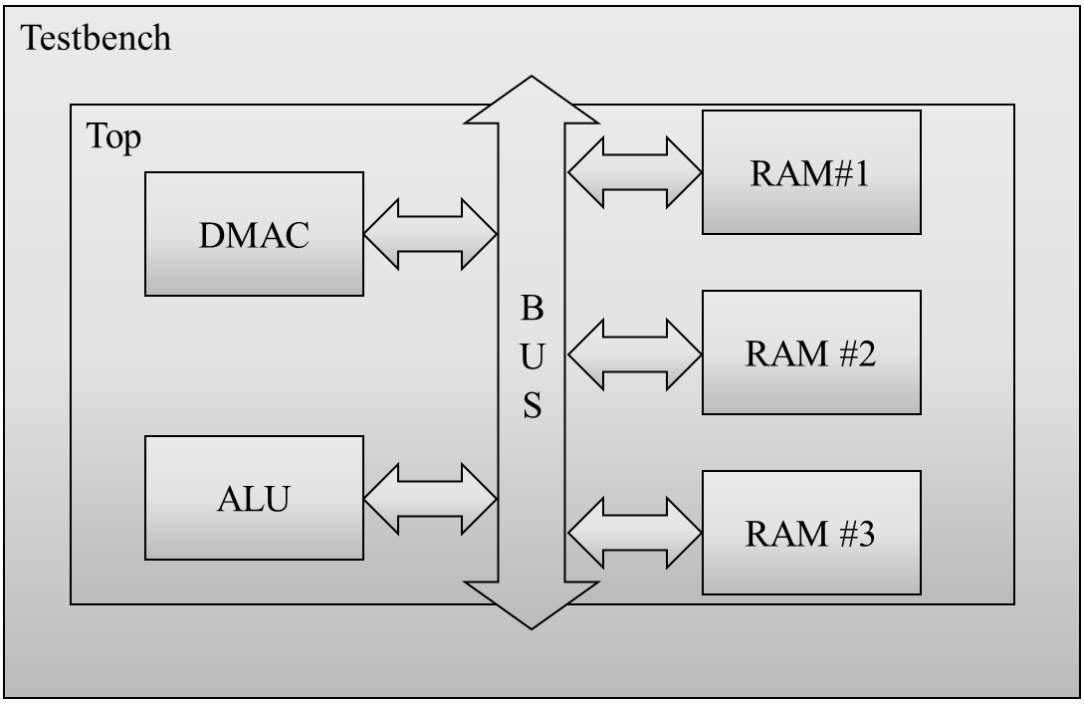

Figure 1. Architecture overview

본 시스템은 $2n$ 개 혹은 이보다 적은 수의 operand $OP = \{A_1,\ldots, A_i,\ldots, A_{2n}\}$ 에 대해 n 개의 instruction $INST = \{inst_1,\ldots, inst_i,\ldots, inst_n\}$을 ALU 에서 수행한 후, n 개의 결과 $RESULT = \{r_1,\ldots, r_i,\ldots, r_n\}$를 RAM#3 에 저장한다. 시스템의 동작순서는 다음과 같다.

본 시스템은 n ($1 \le n \le 8$) 개의 operand OP 에 대해 n 개의 instruction $INST$ 을 수행한 결과 $RESULT$ 를 생성한다. OP 는 $INST$ 에따라 32-bit 입력 1 개 혹은 32-bit 입력 2 개 일 수 있다. 다음은 수행순서를 자세히 보여준다.

1. 시스템이 시작되면, testbench 는 bus master 가 되어 OP 를 RAM#1 에 그리고 $INST$ 는 RAM#2 에 저장한다.

2. Testbench 는 DMAC 에게 source address 로부터 destination address 로 n 개의 operands OP 또는 n 개의 $INST$ 를 copy 하도록 OPERATION_START register 를 이용하여 (DMAC specification 참조) 시작 명령을 내린다. 자세한 사항은 DMAC specification 을 참조한다.

Source address 는 DMAC 가 copy 할 data 가 있는 주소로, OP 의 경우는 위 단계 1 에서 RAM#1 에 저장한 시작주소를 의미하고 $INST$ 의 경우는 마찬가지로 위 단계 1 에서 RAM#2 에 저장한 시작주소를 의미한다.

Destination address 는 DMAC 가 source address 로부터 읽어 온 data 를 저장할 주소로, OP 의 경우는 ALU register OPERAND00~OPERAND15 주소를 의미하고 $INST$ 의 경우는 ALU 의 INSTRUCTION register 주소를 의미한다. ALU register 주소는 아래 ALU specification 을 참고한다.

3. DMAC 는 bus master 가 되어 source address 로부터 한 번에 32bit data 씩 읽어서(read) destination address 에 copy (write)한다.

DMAC specification 에서 설명하듯 data 이동에 필요한 정보 REQ_i = {source address, destination address, data size}하나 마다 데이터 이동이 이루어진다. 본 과제에서는 DMAC 설계를 간단히 하기위해 data size 는 1 로 고정한다.

예를 들어, 16 개의 32-bit data 를 전송하려면, REQ_0 부터 REQ_{15} 까지 16 개의 REQ_i 를 DMAC 에 명령하여 전송한다. 본과제의 DMAC 는 총 16 개의 REQ_i 를 저장하는 descriptor 를 이용하여 data 이동을 수행한다. 따라서 16 개 이상의 32-bit data 를 전송하려면, DMAC 의 descriptor 를 수 차례 변경하여 명령을 수행해야 한다.

다만 DMAC specification 에 언급된 바와 같이 data size 를 1 보다 크게 하고 addressing mode 를 설계하여 시스템 성능이 개선된 경우, 보너스 점수가 있다. 일반적인 DMAC 는 이 동작을 DMAC 가 가지고 있는 DATA_SIZE register 에 쓰여진 횟수만큼 반복한다. 즉, 32 * DATA_SIZE bit 을 이동한다.

4. DMAC 가 data 전송을 마치면 testbench 에게 interrupt 신호를 1 로 보낸다.

5. Testbench 는 bus master 가 되어 DMAC 의 interrupt 를 clear (DMAC 의 INTERRUPT register 값을 1 에서 0 으로 변경)하고, 모든 OP 및 $INST$ 전송이 끝날 때까지, DMAC 에게 추가로 data 전송 명령을 하고 DMAC 의 interrupt 를 기다린다. 2 번~5 번 반복

6. 모든 OP 와 INST 전송이 완료된 후, Testbench 는 bus master 가 되어 ALU 에게 OPERATION_START register 를 이용하여 (ALU specification 참조) 시작 명령을 내린다.

ALU 내부에서 *OP* 는 register file 에 INST 는 FIFO 에 저장되어 있다

7. ALU 는 instruction FIFO 에 있는 $inst_i$ 를 하나씩 pop 하여 연산을 수행하고 그 결과 r_i 를 result FIFO 에 push 한다.

모든 operand A_i 는 32-bit 이지만, r_i 는 64-bit 이다. 이는 곱셈연산 때문인데, 모든 연산의 결과를 같은 크기로 하기 위해 r_i 는 64-bit 이다.

Result FIFO 는 한 번에 32-bit data 를 저장하고, r_i[63:32] 이후 r_i[31:0]를 2 cycle 이내에 걸쳐 저장한다.

8. Instruction FIFO 가 empty 되면, ALU 는 interrupt signal 을 1 로 만들어 testbench 에게 보낸다.

9. Testbench 는 ALU interrupt 를 clear 하고, DMAC 에게 Result FIFO 값을 RAM#3 로 이동하도록 OPERATION_START register 를 이용하여 명령을 내린다.

RESULT FIFO 의 값은 RAM#3 의 0 번지부터 pop 된 순서대로 이동한다

10. 전송완료 후, DMAC 는 interrupt 를 1 로 만들어 발생시킨다.

11. Testbench 는 DMAC interrupt 를 clear 한다.

시스템의 동작여부는 RAM#3 memory image 로 판별한다.

BUS 는 수업 중 설계한 simple bus 를 이용한다. BUS 를 통해 block 간 주고받을 수 있는 정보는 address 와 data 그리고 read/write 명령뿐이다. 따라서, hardware block 들 간에 다양한 명령을 주고받기 위해서는 미리 약속된 값들을 register (flip-flop array)라고 하는 특별한 공간에 쓰고 읽어야 한다. 예를 들어, DMAC block 에게 연산을 시작하도록 하기 위해서는 OPERATION START register 에 0x1 값을 써야한다. 이러한 약속들은 각 block 들의 register description 에 있다. 또한 이러한 register 들을 선택하기 위해서는 address 를 이용한다. 즉, memory-mapped IO 방식으로 register 들을 addressing 하게 된다. 따라서, 각 block 들의 register 들은 offset address 를 갖고 있고, 각 block 들을 선택하기 위한 base address 들은 Table 9. Memory Map~~Table 9. Memory Map~~ 에 정의되어 있다.

구체적인 설계 사양은 업데이트될 수 있습니다. 업데이트될 때마다 종합정보서비스의 해당 과목 공지사항에 공지하니 확인하시기 바랍니다.

2. ALU (Arithmetic Logic Unit)

2.1. Introduction

ALU 는 operational variable 두 개의 입력을 이용해 opcode(operation code)에 따른 연산을 이용해 결과값을 도출하는 hardware 이다. 이번 프로젝트에서 구현하는 ALU 는 곱셈을 제외한 연산에 대해서 64bit width 입력을 받으며 곱셈은 32bit width 이다. 모든 ALU 출력은 64 bit width 이다.

2.2. Features

변수 한 개 혹은 두 개를 이용해 산술, 논리 연산을 하여 결과값을 만든다.

ALU 의 모든 결과는 64bit 이다.

- ✓ 2 개의 32 bits 사용하는 곱셈의 결과의 bit length 는 64bits 이다.
- ✓ 곱셈 이외의 연산들은 sign extend 를 이용해 bit length 를 **64 bits 로 증가시켜 연산한다**.
  - ✓ 결과를 32bits 에서 64 bits 로 변환하는 것이 아닌 64bits 입력을 받는다.

OPERATION_START register 를 이용하여 계산을 시작하고 INTERRUPT register 를 이용하여 완료된 상태를 초기화한다.

ALU 는 실습 시간에 구현한 module 들을 기반으로 구현해도 된다.

- ✓ 단, behavioral RTL (예: 곱셈을 *기호를 써서 구현하는 것)로 구현하면 안된다.
- ✓ Shift 연산: LSL (left shift left), LSR (left shift right), ASR(arithmetic shift right)을 이용한다.
- ✓ 덧셈, 뺄셈: CLA (carry look-ahead adder)를 이용한다.
- ✓ 곱셈: booth multiplier 를 이용한다.

2.3. Functional description

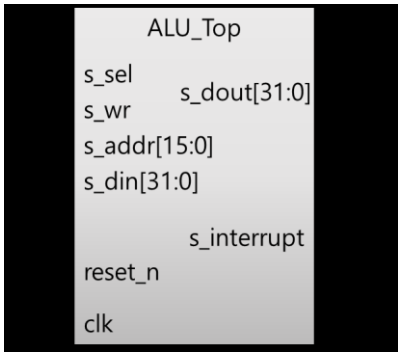

Figure 2. Schematic Symbol of ALU

Figure 2 는 ALU 의 schematic symbol 을 나타낸다. ALU 는 bus 와 연결되는 slave component 로, slave interface 를 통하여 외부에서는 내부 동작을 제어하고, ALU 를 제어하는 방법은 각각의

offset 이 할당된 register 를 slave interface 를 통하여 제어하는 것이다. ALU 의 register description 에 대한 상세한 설명은 뒤에서 다룬다. (Slave interface 는 pin 이름이 ‘s_’로 시작하는 signal 이다.)

2.4. Pin description

Table 1 은 ALU 의 pin 을 설명한 것이다. Module 의 이름은 ‘**ALU_Top**’이다.

Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	s_sel	(Slave interface) Select
	s_wr	(Slave interface) Write/read
	s_addr[15:0]	(Slave interface) Address (ALU 내부에선 해당 address 16 bits 중 하위 8-bits 만을 사용하여 register 를 구분하는 offset 으로 사용한다.)
	s_din[31:0]	(Slave interface) Data input
Output	s_dout[31:0]	(Slave interface) Data output
	s_interrupt	Interrupt (ALU 에서 연산이 완료되었을 때 내부 register 값에 따라 interrupt 를 발생한다.)

Table 1. Pin Description of ALU_Top

2.5. Register description

Table 2 는 ALU 의 register 를 설명한 것이다. Register 의 bandwidth 는 기본적으로 32bits 이다.

- Default value 는 reset 이 되었을 때, 초기 값을 의미한다.
- Register type 은 W(write)와 R(read)로 된다.
  - ✓ 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.
- Register 에서 현재 사용되지 않는 bit 는 reserved 이다.
  - ✓ (예) **‘OPERATION START’** register 의 [31:1] bits 가 reserved 이다’의 의미는 ‘**OPERATION START**’ register 의 [31:1] bits 에 write 는 무시되며, read 된 값은 의미가 없음을 나타낸다.

Table 2 에서 모든 register 의 default value 는 0x0000_0000 이다.

Offset	Type	Name	Default Value
0x00	R/W	OPERATION_START	0x0000_0000
0x01	R/W	INTERRUPT	0x0000_0000
0x02	R/W	INTERRUPT_ENABLE	0x0000_0000
0x03	R/W	INSTRUCTION	0x0000_0000
0x04	R	RESULT	0x0000_0000
0x05	R	ALU_STATUS	0x0000_0000
0x10~0x1F	R/W	OPERAND 00-15	0x0000_0000

Table 2. Register Description of ALU

아래의 내용은 Table 2 의 register 의 자세한 내용을 기술한 내용이다.

✓ OPERATION_START
- 해당 register 의 [0]bit 에 1 이 써지면 FIFO 에서 POP 하여 OPERAND register 와 INSTRUCTION register 의 값을 이용한 산술, 논리 연산이 시작된다.
- ALU 가 연산 중일 때, 해당 register 의 [0]에 0 혹은 1 이 써지는 경우는 없다.
  - (보충) 하드웨어는 모든 경우를 예외 처리할 할 수 없다. 동작 중 일 때, OPERATION_START 의 [0]에 다른 값이 써지는 경우는 testbench 시나리오 문제로 생각한다.
- ALU 동작을 마치면 testbench 가 INTERRUPT CLEAR 이전에 해당 register 를 클리어한다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ INTERRUPT
- 해당 register 는 ALU 가 모든 연산을 마쳤을 때 [0]bit 에 1 이 써진다.
- 해당 register 의 [0]bit 에 0 이 써지면 s_interrupt port 로 출력되는 값과 OPERATION_START register 를 clear 된다.
- ALU 가 polling method 일 때, testbench 는 이 register 값을 읽는다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ INTERRUPT_ENABLE
- 해당 register 는 모듈의 polling method 와 interrupt method 를 결정하는 register 이다.
- 해당 register 의 [0] bit 가 1 이며 INTERRUPT register 의 [0]bit 가 1 일 때 s_interrupt port 에서 1 이 출력된다.
- 해당 register 의 [0] bit 가 0 이며 INTERRUPT register 의 [0]bit 가 1 일 때 s_interrupt port 에서 0 이 출력된다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ INSTRUCTION
- DMAC 로부터 전달받은 *INST* 값을 저장하고 있는 register 이다.
- INSTURCTION 구조에 대한 자세한 내용은 아래에 기술되어 있다.
- **ALU 의 STATE 가 IDLE 상태인 경우에만 DMAC 로부터 받은 각각의 값들을 해당 register 에 저장한다.**
  - 그 외의 상태일 때 입력의 경우는 생각하지 않는다.
- INSTRUCTION register 의 정보를 저장할 때, 최대 8 개의 값을 저장할 수 있는 FIFO 를 이용/응용하여 구현한다.
  - 즉, 한 주소에 8 개까지 32-bit instruction 을 DMAC 가 write (즉, push)할 수 있다.
- INSTRUCTION register 에 write 하는 순간 (즉, write enable 이 1 이 되어 register 에 값이 쓰여진 후) FIFO 에 push 된다.
- INSTRUCTION register 를 read 한다면 FIFO 가 아닌 register 의 값을 읽는다.

✓ RESULT

- 결과 값을 DMAC 를 통해 2 cycle 에 걸쳐 RAM 에 저장할 때 사용되는 register 이다.
- RESULT register 의 정보를 저장할 때, 최대 16 개의 값을 저장할 수 있는 FIFO 를 이용/응용하여 구현한다
- 각 연산이 완료되면, result FIFO 에 순서대로 결과 값 r_i[63:0]을 저장한다.
- r_i [63:0]의 push 순서는 push r_i [63:32] 후 push r_i [31:0]이다.
- 따라서 pop 할 경우, r_i[63:32]이 먼저 나오고 r_i[31:0]가 나온다.

✓ ALU_STATUS

- 해당 register 의 값을 read 할 때 [1:0] bits 에 따라 ALU 의 state 를 나타낸다.
- 2'b00 = Waiting
  - ALU 가 IDLE 상태이며, bus 의 master 로부터 op_start 신호가 들어올 때까지 기다리는 대기 상태를 의미한다.
- 2'b01 = Executing
  - ALU 가 저장된 OPERAND register 와 INSTRUCTION 의 정보를 이용하여 산술, 논리 연산을 하는 상태를 의미한다.
- 2'b10 = Done
  - ALU 가 모든 INSTRUCTION 의 연산을 완료한 상태를 의미한다.
- 2'b11 = Fault.
  - ALU 가 정상적인 동작을 하지 못했을 경우의 상태를 의미한다.
  - 예를 들어, 동작을 시작할 때 INSTRUCTION FIFO 가 empty 면 ALU 의 동작을 완료하지 않고 fault 상태를 출력한다. (FIFO 가 full 일 때 추가로 push 명령이 들어오는 경우도 예외처리. 즉, FIFO 의 error signal 들이 발생할 때 예외처리)
  - Fault 상태가 되면 reset_n 을 이용해 reset 이 되기 전까지 상태를 유지한다.
- 해당 register 의 [31:2] bits 는 reserved 이다.

✓ OPERAND 0 – 15

- DMAC 로부터 전달받은 OP 를 저장하고 있는 register.
- Register file 을 이용해 구현한다.
- 모든 OP 는 32 bits 이며, ALU 에서 연산에 활용하기 위하여 r_i 로 변환할 때 sign extend 한다.

Register 의 경우, 위의 필수 register 를 포함해야 하며, 그 외의 offset 에는 사용자 정의 register 들(user-defined registers)을 포함하여 최대 32 개까지 포함할 수 있다.

✓ 위에서 정의된 register 를 제외하고 추가할 register 가 있다면 0x06 ~ 0x0F 사이에 자유롭게 추가하여도 무관하다.

2.6. Instruction Register

Instruction register 의 구조는 Table 3 에 기술되어 있다.

Bit Position	Name	Description
[31:14]	Reserved	Reserved
[13:10]	Opcode	ALU 의 연산을 결정하는 값을 나타낸다.
[9:6]	OpA_addr	OPERAND 의 RF address 값을 가지고 있다. (0x10 - 0x1F) ALU 의 입력 A 로 사용하는 값을 RF 에서 출력해 연산한다.
[5:2]	OpB_addr	OPERAND 의 RF address 값을 가지고 있다. (0x10 - 0x1F) ALU 의 입력 B 로 사용하는 값을 RF 에서 출력해 연산한다.
[1:0]	Shift_amount	Shift 연산을 할 때 사용하는 값을 기술, Shift 이외의 연산을 할 경우 이 값은 무시된다.

Table 3. Instruction Register Description

Opcode

- ALU 의 opcode 는 Table 4 를 이용해 ALU 의 동작을 구현한다.
  - ✓ NOP: 아무런 연산을 하지 않는다. 다음 clock cycle 일 때 instruction 을 pop 하여 연산을 진행한다.
  - ✓ (Operation) A/B: 입력 A 혹은 입력 B 를 이용해 연산 결과를 출력한다.
  - ✓ 그 이외의 연산은 A 와 B 를 이용해 연산 결과를 출력한다.

Opcode	Operation	Opcode	Operation
4’h0	NOP	4’h8	LSR A
4’h1	NOT A	4’h9	ASR A
4’h2	NOT B	4’hA	LSL B
4’h3	AND	4’hB	LSR B
4’h4	OR	4’hC	ASR B
4’h5	XOR	4’hD	ADD
4’h6	XNOR	4’hE	SUB
4’h7	LSL A	4’hF	MUL

Table 4. Operation Code Description

3. DMAC

3.1. Introduction

DMA 는 Direct Memory Access 의 약자로서, processor 이외의 device 가 memory 와 I/O device 간의 data 전송을 제어하는 것이다. (일반적으로 RISC 계열의 processor 에서는 load/store instruction 을 통하여 memory 와 I/O device 에 접근한다.) DMAC 는 이러한 DMA 동작에 특화된 hardware 를 의미한다. DMAC 가 system 에서 사용되면, 대량의 data 를 이동/복사할 때, processor 는 다른 작업을 수행할 수 있으므로 system 의 performance 를 향상시킬 수 있다.

3.2. Features

- 다른 master component 는 bus 의 slave interface 를 통해 DMAC 를 제어할 수 있다.
- DMAC 는 bus 의 master interface 를 통해 data 를 전송할 수 있다.
- DMAC 는 polling method 와 interrupt-driven method 를 모두 지원한다.
- Data 의 이동/복사를 위해 필요한 정보의 집합체 REQ_i = {source address src_i, destination address des_i, data size $size_i$} 하나 마다 data transfer 가 이루어진다. 본 과제에서는 $size_i$ = 1 로 고정이다.
  - ✓ (보충) 일반적인 DMAC 는 $size_i$ 가 1 보다 큰 경우, 별도의 addressing mode register 값에 따라 src_i 와 des_i 값의 변경 패턴을 정의한다. Addressing mode 는 srci 와 desi 마다 독립적으로 적용할 수 있다. 가장 보편적인 incremental addressing mode 는 src_i / des_i 가 1 씩 (data 1 개의 size 만큼) 증가하며 data 를 $size_i$ 만큼 이동한다. Fixed addressing mode 는 src_i / des_i 가 변하지 않고 고정된 값을 가지며 data 를 $size_i$ 만큼 이동한다. FIFO 에 data 를 전송할 때 사용하는 addressing mode 이다. 마지막으로 wrapping addressing mode 는 cache memory 의 cache line 을 이동할 때 사용하는데, 자세한 cache 관련 내용은 컴퓨터구조 등에서 다룬다.
  - ✓ (중요) DMAC 의 addressing mode 를 구현하여 시스템 성능이 개선된 경우, 보너스 점수 (프로젝트 총점의 5%)가 부여된다.
- 복수의 REQ_i 를 저장하고 있는 구조체를 descriptor 라 한다. 본 DMAC 는 16 개의 REQ_i 를 가지고 있는 descriptor 를 갖고 있다. 즉, 최대 16 개의 memory region 에 대한 이동/복사가 가능하다.
- DMAC 는 한 개의 memory region 에 대하여 최대 16 개의 데이터를 한번에 읽고 쓸 수 있다.

3.3. Functional description

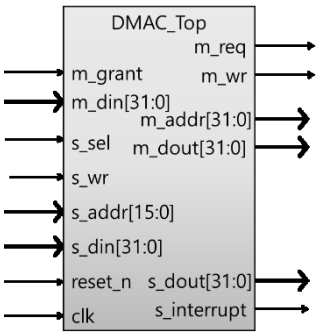

Figure 3 Schematic Symbol of DMAC

Figure 3 은 DMAC 의 schematic symbol 을 나타낸다. DMAC 는 bus 와 연결되는 master/slave component 로, DMAC 의 slave interface 를 통하여 외부에서 DMAC 의 내부 동작을 제어하고, master interface 를 통하여 DMAC 외부에 접근할 수 있는 component 이다.

외부에서 DMAC 를 제어하는 방법은 각각의 offset 이 할당된 register 를 slave interface 를 통하여 제어하는 것이다. DMAC 의 register description 에 대한 상세한 설명은 뒤에서 다룬다.

- 해당 schematic symbol 에서 왼쪽은 input pin 들을 나타내고, 오른쪽은 output pin 들을 나타낸다.
- Slave interface 는 pin 이름이 ‘s_’로 시작하는 signal 이며, master interface 는 pin 이름이 ‘m_’로 시작하는 signal 이다.

3.4. Pin description

Table 5 은 DMAC 의 pin 을 설명한 것이다. Module 의 이름은 ‘**DMAC_Top**’이다.

Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	m_grant	(Master interface) Grant (Bus 에서 DMAC 가 data 를 주고받는 것을 허락해주는 signal 로, 해당 signal 이 1 인 동안 bus 를 통해 다른 slave component 를 제어할 수 있다.)
	m_din[31:0]	(Master interface) Data input
	s_sel	(Slave interface) Select
	s_wr	(Slave interface) Write/read
	s_addr[15:0]	(Slave interface) Address (DMAC 내부에선 해당 address 16 bits 중 하위 8-bits 만을 사용하여 register 를 구분하는 offset 으로 사용한다.)
	s_din[31:0]	(Slave interface) Data input
Output	m_req	(Master interface) Request (DMAC 가 bus 를 통해 data 를 주고받을 수 있도록 허락해달라고 요청하는 signal 이다.)

	m_wr	(Master interface) Write/read
	m_addr[15:0]	(Master interface) Address
	m_dout[31:0]	(Master interface) Data output
	s_dout[31:0]	(Slave interface) Data output
	s_interrupt	Interrupt (DMAC 에서 data 의 이동/복사가 끝났을 때 내부 register 값에 따라 interrupt 를 발생한다.)

Table 5. Pin Description of DMAC_Top

3.5. Register description

Table 6 는 DMAC 의 register 를 설명한 것이다. Register 의 bandwidth 는 기본적으로 32bits 이다.

Default value 는 reset 이 되었을 때, 초기 값을 의미한다.

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.

Register 에서 현재 사용되지 않는 bit 는 reserved 이다.

- ✓ 예를 들어 '**OPERATION START** register 의 [31:1] bits 가 reserved 이다'의 의미는 **OPERATION START** register 의 [31:1] bits 에 write 는 무시되며, read 된 값은 의미가 없음을 나타낸다.

Offset	Type	Name	Default Value
0x00	R/W	OPERATION_START	0x0000_0000
0x01	R/W	INTERRUPT	0x0000_0000
0x02	R/W	INTERRUPT_ENABLE	0x0000_0000
0x03	R/W	SOURCE_ADDRESS	0x0000_0000
0x04	R/W	DESTINATION_ADDRESS	0x0000_0000
0x05	R/W	DATA_SIZE	0x0000_0000
0x06	R/W	DESCRIPTOR_PUSH	0x0000_0000
0x07	R/W	OPERATION_MODE	0x0000_0000
0x08	R	DMA_STATUS	0x0000_0000

Table 6. Register Description of DMAC

아래의 내용은 Table 6 의 register 의 자세한 내용을 기술한 내용이다.

- ✓ OPERATION_START
  - 해당 register 의 [0] bit 에 1 이 써지면, DMAC 가 data 의 이동/복사를 시작한다.
  - DMAC 가 연산 중일 때, 해당 register 의 [0]에 0 혹은 1 이 써지는 경우는 없다.
    - (보충) 하드웨어는 모든 경우를 예외 처리할 할 수 없다. 동작 중 일 때, OPERATION_START 의 [0]에 다른 값이 써지는 경우는 testbench 시나리오 문제로 생각한다.
  - 해당 register 의 [31:1] bits 는 reserved 이다.
- ✓ INTERRUPT
  - 해당 register 는 DMAC 가 모든 연산을 마쳤을 때 [0]bit 에 1 이 써진다.

- 해당 register 의 [0]bit 에 0 이 써지면 s_interrupt port 로 출력되는 값과 OPERATION_START register 를 clear 된다.
- DMAC 가 polling method 일 때, testbench 는 이 register 값을 읽는다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ INTERRUPT_ENABLE

- 해당 register 는 모듈의 polling method 와 interrupt method 를 결정하는 register 이다.
- 해당 register 의 [0] bit 가 1 이며 INTERRUPT register 의 [0]bit 가 1 일 때 s_interrupt port 에서 1 이 출력된다.
- 해당 register 의 [0] bit 가 0 이며 INTERRUPT register 의 [0]bit 가 1 일 때 s_interrupt port 에서 0 이 출력된다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ SOURCE_ADDRESS

- 해당 register 는 DMAC 가 data 의 이동/복사를 수행할 때, read 를 시작할 address 를 저장한다. 해당 register 의 [31:16] bits 는 reserved 이다.

✓ DESTINATION_ADDRESS

- 해당 register 는 DMAC 가 data 의 이동/복사를 수행할 때, write 를 시작할 address 를 저장한다. 해당 register 의 [31:16] bits 는 reserved 이다.

✓ DATA_SIZE

- 해당 register 는 DMAC 가 data 의 이동/복사를 수행할 때, 전송할 data 의 크기를 저장한다. 단위는 4 bytes 이다. 해당 register 의 [31:5] bits 는 reserved 이다.
- OPERATION MODE 를 구현하지 않을 경우, DATA_SIZE 는 1 로 고정이고, 이 값은 tsetbench 가 wirte 한다.

✓ DESCRIPTOR_PUSH

- Testbench 가 [0]번째 bit 에 1 을 쓰는 경우, 현재 SOURCE_ADDRESS, DESTINATION_ADDRESS, DATA_SIZE 값을 하나의 REQ 로 정의하여 DESCRIPTOR FIFO 에 push 한다.
- DESCRIPTOR FIFO 에 push 가 이루어 진 후, DMAC 는 바로 다음 cycle 에 [0]번째 bit 을 0 으로 바꾸어, 동일한 REQ 가 반복하여 DESCRIPTOR FIFO 에 push 되는 것을 방지해야 한다.
- 해당 register 의 [31:1] bits 는 reserved 이다.

✓ OPERATION MODE (추가구현)

- Source address 와 destination address 의 increment mode 가 있다.
- opmode[0]: source address increment mode (0-none, 1-linear)
- opmode[1]: destination address increment mode (0-none, 1- linear)
- 해당 register 의 [31:2] bits 는 reserved 이다.

✓ DMA STATUS

- 해당 register 의 값을 read 할 때 [1:0] bits 에 따라 DMA 의 state 를 나타낸다.
- 2'b00 = Waiting

  - DMAC 가 IDLE 상태이며, bus 의 master 로부터 OPERATION_START 의 [0]bit 에 1'b1 이 써질 때까지 기다리는 대기 상태이다.
- 2'b01 = Executing
  - DMAC 가 저장된 REQ_i 의 정보를 이용하여 데이터를 복사하는 동작 상태를 의미한다.
- 2'b10 = Done
  - DMAC 가 모든 REQ_i 의 동작을 완료한 상태를 의미한다.
- 2'b11 = Fault.
  - DMAC 가 정상적인 동작을 하지 못했을 경우의 상태를 의미한다.
  - 예를 들어, 동작을 시작할 때, Descriptor 에 해당하는 FIFO 가 empty 면 DMAC 의 동작을 완료하지 않고 fault 상태를 출력한다. (FIFO 가 full 일 때도 예외처리)
  - Fault 상태가 되면 reset_n 을 이용해 reset 이 되기 전까지 상태를 유지한다.

✓ 해당 register 의 [31:2] bits 는 reserved 이다.

➢ DMAC 의 동작이 완료되면 DMA STATUS 가 2'b10 이 된다. 이때 polling method 는 INTERRUPT register 의 값을 읽어서 DMAC 의 동작 완료를 확인하며, interrupt-driven method 일 때는 interrupt port 의 값이 1 인지 확인하여 DMAC 의 동작 완료를 확인한다.

➢ DMAC 의 동작 완료 후에는 testbench 가 INTERRUPT register 에 1'b0 을 write 하여 DMA STATUS register 의 값을 0x00 으로변경하고, DMAC 를 동작 대기 상태로 만든다.

➢ 위에서 정의된 register 를 제외하고 추가할 register 가 있다면 0x09 ~ 0x1F 사이에 자유롭게 추가하여도 무관하다.

➢ REQ_i 정보를 저장할 때, memory 또는 FIFO 를 이용/응용하여 구현한다.

3.6. Operation mode (추가구현)

➢ DMAC 는 source address 와 destination address 의 increment mode 가 존재한다. 각 모드는 병렬로 동작한다.

➢ Source address increment mode

✓ opmode 의 [0]bit 가 set 일 때 동작하고, data size 만큼 linear 하게 증가한다.

Destination address increment mode

✓ opmode 의 [1]bit 가 set 일 때 동작 하고, data size 만큼 linear 하게 증가한다.

4. RAM (Random Access Memory)

4.1. Introduction

RAM 는 address 에 기반하여 data 를 저장하는 hardware 로, 해당 프로젝트에서는 Random Access Memory 을 구현하도록 한다.

4.2. Features

Address 의 bandwidth 는 16 bits 이며 data 의 bandwidth 는 32 bits 이다.

RAM 은 내부에 64 개의 data 를 address 에 기반하여 저장한다.

- 이번 프로젝트에서 사용하는 3 개의 RAM 은 **non-delay** memory 를 구현한다.

4.3. Pin description

다음은 RAM 의 pin description 이다. Module 이름은 '**ram'**와 이다.

Direction	Port name	Description
Input	clk	Clock
	cen	Chip enable
	wen	Write enable
	addr[15:0]	Address
	din[31:0]	Data in
Output	dout[31:0]	Data out

Table 7. Pin Description of Memory

- cen 과 wen 이 모두 1 이면 address 가 가리키는 RAM 에 din 값을 write 한다. 이때 dout 은 0 을 출력한다.
- cen 이 1 이고, wen 이 0 이면, address 가 가리키는 RAM 의 값을 dout 에 write 한다.
- cen 이 0 이면, dout 은 0 이 된다.

5. BUS

5.1. Introduction

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.

5.2. Features

- 2 개의 master 와 5 개의 slave 를 가지고 있다.
- Address 의 bandwidth 는 16 bits 이다.
- Data 의 bandwidth 는 32 bits 이다.
- Slave 0 은 0x0000 ~ 0x001F 사이의 address 를 memory map region 을 가진다.
- Slave 1 은 0x0100 ~ 0x011F 사이의 address 를 memory map region 을 가진다.
- Slave 2 은 0x0200 ~ 0x023F 사이의 address 를 memory map region 을 가진다.
- Slave 3 은 0x0300 ~ 0x033F 사이의 address 를 memory map region 을 가진다.
- Slave 4 은 0x0400 ~ 0x043F 사이의 address 를 memory map region 을 가진다.
- Master 가 request 하고 bus 로부터 grant 를 받았을 때 (m_request = 1 && m_grant = 1)에만 읽기/쓰기 동작이 이루어지고, 그 외의 경우는 읽기/쓰기가 이루어지지 않는다
  - ✓ 본 시스템에서 두개의 master Testbench 와 DMAC 가 모두 bus 를 사용하지 않아 (즉, m0_request = 0 && m1_grant = 0), 위에 정의된 slave 이외의 주소를 접근할 경우에도 읽기/쓰기는 이루어지지 않게 된다.

5.3. Functional description

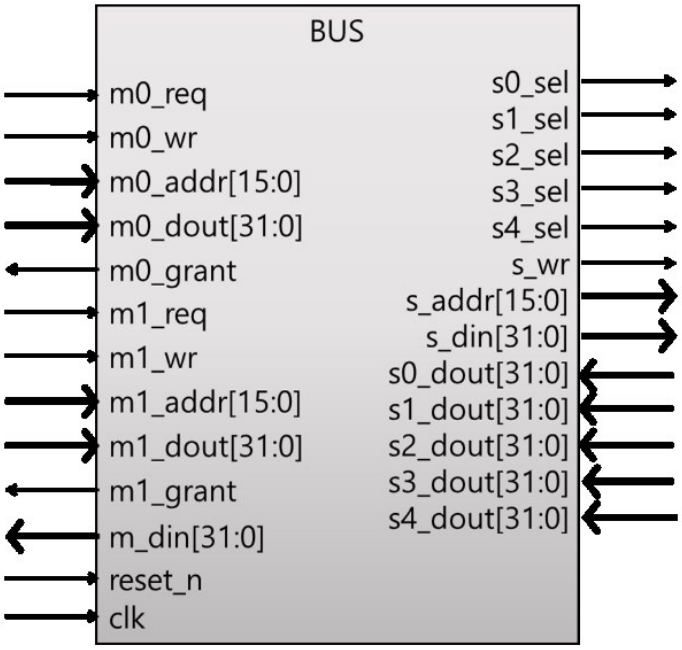

Figure 4. Schematic Symbol of Bus

프로젝트에서 구현할 BUS 는 2 개의 master 와 5 개의 slave 로 구현되어 있다. Figure 4 은 BUS 의 schematic symbol 이다.

- 왼쪽은 master interface 를 나타내며, 오른쪽은 slave interface 를 나타낸다.

Master 는 BUS 를 통해 data 를 transfer 하고자 할 때, 자신에게 해당하는 request signal 을 1'b1 로 BUS 에게 사용을 요청한다. Master 는 그에 대한 확인으로 grant signal 를 받은 후 data transfer 를 할 수 있다.

Master 가 grant signal 을 받은 후에는 communication 하고자 하는 slave 의 memory map 영역을 접근함으로써 communication 을 수행할 수 있으며, request signal 이 1 인 동안에는 BUS 의 소유권을 빼앗기지 않고 data transfer 를 계속할 수 있다. 예를 들어 $master_0$ 가 BUS 를 사용하고 있을 때, 다른 $master_1$ 가 request 를 요청하면 $master_1$ 는 grant 를 받을 수 없다.

모든 master 가 bus 에 req 신호를 보내지 않을 경우 m0_grant 가 1, m1_grant 가 0 이다. 이 때, master0 인 testbench 는 memory map 의 범위에 포함되지 않는 address 를 가지고 있다.

5.4. Pin description

다음 Table 8 은 BUS 의 pin 을 정리한 것이다. Module 이름은 'BUS'이다. 여기서 주의해야 할 점은 master 에서 data-out 이 output pin 이지만, BUS 에서는 master 의 data-out 을 받아야 하기 때문에 data-out 이 input pin 이 된다. 이는 slave 에도 똑같이 적용된다.

Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	m0_req	Master 0 request
	m0_wr	Master 0 write/read
	m0_addr[15:0]	Master 0 address
	m0_dout[31:0]	Master 0 data output
	m1_req	Master 1 request
	m1_wr	Master 1 write/read
	m1_addr[15:0]	Master 1 address
	m1_dout[31:0]	Master 1 data out
	s0_dout[31:0]	Slave 0 data out
	s1_dout[31:0]	Slave 1 data out
	s2_dout[31:0]	Slave 2 data out
	s3_dout[31:0]	Slave 3 data out
	s4_dout[31:0]	Slave 4 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
	s2_sel	Slave 2 select
	s3_sel	Slave 3 select
	s4_sel	Slave 4 select
	s_addr[15:0]	Slave address
	s_wr	Slave write/read
	s_din[31:0]	Slave data input

Table 8. Pin description of BUS

6. Top

TOP 은 DMAC, ALU, RAM 그리고 BUS 를 instance 하여 이들을 연결한 component 이다.

6.1. Features

해당 system 의 외부(testbench)에서 master 0 interface 를 사용하여 DMAC 와 ALU, RAM 에 접근할 수 있다.

- DMAC 와 ALU 의 interrupt signal 혹은 각 모듈의 INTERRUPT register 를 읽어 연산이 종료됨을 확인한다.
- Figure 5 는 Top 의 schematic symbol 이다.

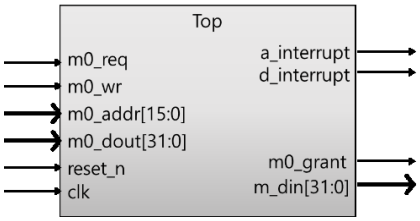

Figure 5. Schematic Symbol of Top

전체 시스템의 memory map 은 Table 9 와 같다

Component	Address region	
	Base Address	Offset
DMAC	0x0000	0x00 ~ 0x1F
ALU	0x0100	0x00 ~ 0x1F
RAM (Operand)	0x0200	0x00 ~ 0x3F
RAM (Instruction)	0x0300	0x00 ~ 0x3F
RAM (Result)	0x0400	0x00 ~ 0x3F

Table 9. Memory Map

6.2. Pin description

Table 10 은 top 의 pin 에 대한 설명이다. Module 의 이름은 '**Top**'이다.

Direction	Port name	Description
Input	clk	Clock
	reset_n	Active low reset
	m0_req	Master request
	m0_wr	Master write/read
	m0_addr[15:0]	Master address
	m0_dout[31:0]	Master data output
Output	m0_grant	Master 0 grant
	a_interrupt	ALU interrupt
	d_interrupt	DMAC interrupt
	m_din[31:0]	Master data input

Table 10. pin description of top

7. Important dates

Issue date: 10 월 24 일

제안서 발표: 11 월 5 일 화요일 디지털논리회로 2 수업시간

✓ 발표 희망자는 조교에게 메일 보낼 것(보너스 가점)

➢ 제안서 제출

➢

✓ Softcopy: **11 월 5 일 화요일 16 시 29 분까지 디지털논리회로 2 제출란에 제출한다.**

✓ **디지털논리회로 2 미수강시 컴퓨터공학기초설계및실험 분반 과제 제출란에 제출한다.**

➢

✓ 늦은 제출은 11 월 6 일 23 시 59 분까지

✓ U-Campus 에 제안서 upload 시 파일이름은 '(학번)_프로젝트제안서_이름.docx/hwp/pdf'

Testbench 이용 결과 검증

➢ ✓ 10 개의 testbench 를 이용하여 결과 검증

- Testbench 내용은 미공개
- 스펙 문서와 다른 pin name 혹은 module name 을 사용할 경우 0 점 처리

✓ 11/25~27 (컴퓨터공학기초설계실험 수업시간)때 예비검증 진행, 실습 미수강 학생의 경우 11/29 금요일 메인 조교를 방문하여 검증 진행

- Testbench 는 스펙 문서에 변화에 따른 예외처리에 대한 감점이 없으므로 이전과 동일하게 1 회만 수행 가능

➢

Project 결과 발표

✓ 12 월 3 일 화요일 디지털논리회로 2 기말고사 이후

➢ ✓ 발표희망자는 메일 도착 시간을 기준으로 선착순으로 결정

최종 결과 보고서 제출

✓ u-Campus(종합정보서비스)의 **12 월 1 일 일요일 23 시 59 분까지 디지털논리회로 2 제출란에 제출한다.**

✓ **디지털논리회로 2 미수강시 컴퓨터공학기초설계및실험 분반 과제 제출란에 제출한다.**

✓ 제출날짜: 12 월 1 일 23 시 59 분까지(모든 분반 통일)

✓ 최종 결과 보고서의 경우, 늦은 제출은 받지 않습니다.

✓ 최종 결과 보고서 제출시 source code 를 report 와 같이 압축하여 upload 한다.

✓ Source code 는 project 구현에 사용된 모든 설계 파일 및 이에 대한 testbench 를 포함

✓ 파일 압축할 때 source code 는 db, incremental_db, simulation 폴더와 *.bak 파일을 삭제하고, 프로젝트가 생성된 기본 폴더와 report 를 함께 압축한다.

✓ U-Campus 에 upload 시 파일이름은 '(학번)_(이름)_Project.zip'으로 한다.

8. Report Outline

제안서

- ✓ 발표: PowerPoint 로 발표 희망자에 한해 10 분 분량으로 작성
- ✓ 보고서: 최소한 다음의 내용들이 포함되어야 하며 그 외의 것을 추가하는 것은 자유다.
- ✓ 과제제목, 과제목표, 일정, 각 module 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.

-✓ Size 또는 clock 주기는 선택적으로 report 한다.

9. Score

제안서: 20%

Testbench: 40%

- ✓ Bus: 20%
- ✓ Memory: 10%
- ✓ ALU: 20%
- ✓ DMAC: 20%
- ✓ TOP: 30%

결과보고서: 40%