1. System Overview

본 시스템은 Bus와 간단한 processor역할을 하는 mini-processor block MP를 이용하여 곱셈, 덧셈, 뺄셈 및 논리연산을 한다. MP내에는 register들과 arithmetic logic unit (ALU)와 multiplier (MUL)이 있다. 과제의 자세한 specification을 충분히 이해해 수행한다.

Figure 1은 구현할 시스템의 블록 다이어그램이다. 본 시스템은 MP와 BUS 로 구성되고, testbench를 이용하여 시스템의 동작을 제어한다. MP는 Bus를 통해 접근할 수 있는 register들의 집합을 가지고 있다. 이 register들의 집합은 DATA\_REG, INST\_REG, CONT\_REG라 불리는 부분집합으로 이루어져 있다. 자세한 register map은 4장에 소개하는 MP block의 specification을 참고한다.

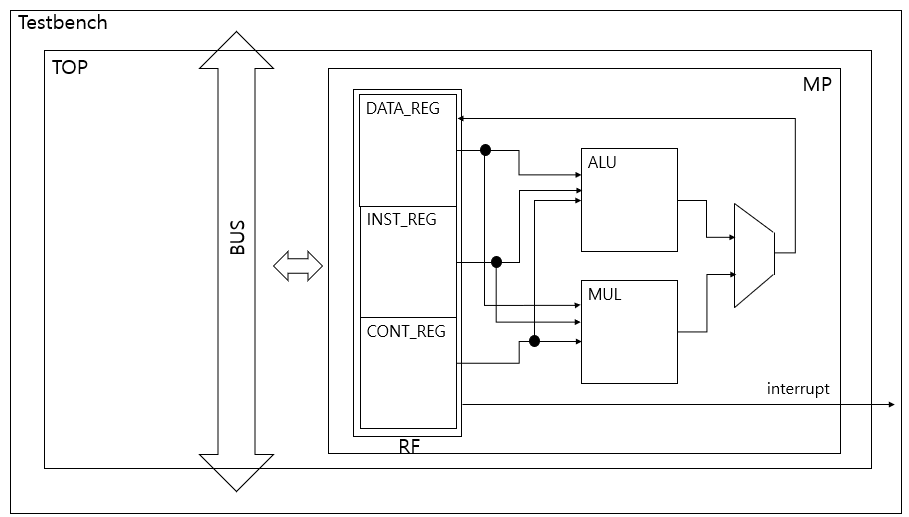


Figure 1. Architecture overview

본 시스템은 testbench로부터 10개의 data를 DATA\_REG에 초기화한 후 10개의 instruction을 INST\_REG에 저장한다. 그 후 CONT\_REG에 속한 op\_start register에 0x1을 testbench가 쓰면 INST\_REG에 있는 10개의 instruction들을 ALU에서 수행한 후, 결과 값들을 DATA\_REG에 저장한다. 다음은 수행 순서를 자세히 보여준다.

1. 시스템이 시작되면, testbench는 Bus master가 되어 10개의 data를 DATA\_REG에 그리고 10개의 instruction은 INST\_REG에 저장한다.
2. Testbench는 CONT\_REG에 속한 interrupt\_mask register에 0 또는 1을 쓴다. 0을 쓰는 경우, 5번에서 instruction 수행 완료 후, interrupt register에 1을 써도 MP block의 interrupt output port가 0값을 유지한다. 만약 1을 쓰는 경우, 5번에서 instruction 수행 완료 후, MP block이 interrupt register에 1을 쓰면, MP block의 interrupt output port가 1로 변경된다.
3. Testbench는 CONT\_REG에 속한 op\_start register에 0x1을 쓰고, MP block은 INST\_REG에 있는 첫번째 instruction부터 수행한다. 각 instruction은 opcode, rd (destination register address), ra, rb로 구성된다. 자세한 내용은 4장을 참고한다.
4. INST\_REG에서 나온 instruction을 통해 DATA\_REG로부터 데이터(ra, rb)를 받아와 ALU 또는 MUL에서 연산 후 결과를 DATA\_REG의 rd에 저장한다.
5. MP가 10개의 instruction을 수행 후 모든 결과를 저장하면 CONT\_REG에 속한 Interrupt register에 1의 값을 쓴다. 앞의 2번에서 interrupt\_mask register에 쓰인 값에 따라 interrupt output port값이 0 또는 1값으로 변경된다.
6. 앞의 2번에서 interrupt\_mask register가 0값을 가진 경우, testbench는 3번 동작 이후 일정한 clock cycle마다 (예를 들어 매 10 cycle마다) interrupt register 값을 읽어서 확인한 후, 1인 경우 DATA\_REG의 값을 모두 읽어서 예상한 결과값과 비교하여 pass 또는 fail 결과를 생성한다.
7. 앞의 2번에서 interrupt\_mask register가 1값을 가진 경우, testbench는 MP block의 interrupt output port값이 1이 될 때까지 기다린다. MP block의 interrupt port가 1이 되는 즉시 testbench는 DATA\_REG의 값을 모두 읽어서 예상한 결과값과 비교하여 pass 또는 fail 결과를 생성한다.
8. TOP

TOP은 BUS와 MP를 instance하여 이들을 연결한 component이다.

2.1. Features

해당 system의 외부(testbench)에서 master interface를 사용하여 MP와 BUS에 접근할 수 있다.

MP의 interrupt signal을 읽어 연산이 종료됨을 확인한다.

Figure 2는 Top의 schematic symbol이다.

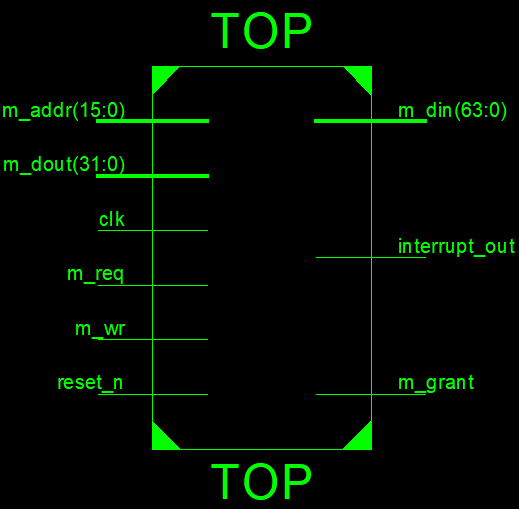


Figure 2. Schematic Symbol of Top

2.2. Pin description

Table 2는 top의 pin에 대한 설명이다. Module의 이름은 ‘Top’이다.

|  |  |  |
| --- | --- | --- |
| Direction | Port name | Description |
| INPUT | clk | Clock |
| reset\_n | Active low reset |
| m\_req | Master request |
| m\_wr | Master write/read |
| m\_addr[15:0] | Master address |
| m\_dout[31:0] | Mater data output |
| OUTPUT | m\_grant | Master grant |
| interrupt\_out | MP interrupt output |
| m\_din[63:0] | Master data input |

Table 1. pin description of top

1. BUS

3.1. Introduction

Bus는 여러 component들 간에 data를 전송(transfer)할 수 있도록 연결해주는 component이다. Bus는 새로운 component들을 추가하기가 쉽다.

3.2. Features

1개의 master와 1개의 slave를 가지고 있다.

Address의 bit-width는 16bits이다.

Data의 bit-width는 64bits이다

Master가 request하고 Bus로부터 grant를 받았을 때 (m\_request = 1)에만 읽기/쓰기 동작이 이루어지고, 그 외의 경우는 읽기/쓰기가 이루어지지 않는다

✓ 본 시스템에서 master Testbench가 Bus를 사용하지 않아 (즉, m\_request = 0), 위에 정의된 slave 이외의 주소를 접근할 경우에도 읽기/쓰기는 이루어지지 않게 된다.

✓ master와 slave가 각 1개이므로 grant와 req를 wire로 연결시켜 준다.

3.3. Functional description

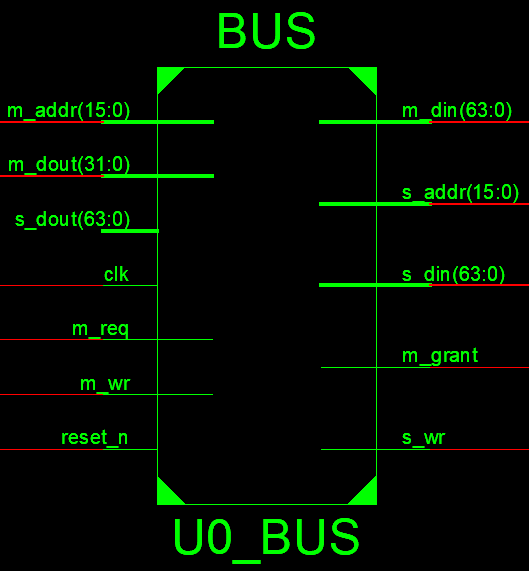


Figure 3. Schematic Symbol of Bus

프로젝트에서 구현할 BUS는 1 개의 master와 1 개의 slave로 구현되어 있다. Figure 3 은 Bus의 schematic symbol이다.

* 왼쪽은 master interface를 나타내며, 오른쪽은 slave interface를 나타낸다.

Master는 Bus를 통해 data를 transfer하고자 할 때, 자신에게 해당하는 request signal을 1’b1 로 Bus에게 사용을 요청한다. Master는 그에 대한 확인으로 grant signal를 받은 후 data transfer를 할 수 있다.

Master가 grant signal을 받은 후에는 communication하고자 하는 slave의 memory map영역을 접근함으로써 communication을 수행할 수 있으며, request signal이 1 인 동안에는 Bus의 소유권을 빼앗기지 않고 data transfer를 계속할 수 있다.

3.4. Pin description

다음 Table 8은 Bus의 pin을 정리한 것이다. Module이름은 ‘BUS’이다. 여기서 주의해야 할 점 은 master에서 data-out이 output pin이지만, Bus에서는 master의 data-out을 받아야 하기 때문에 data-out이 input pin이 된다. 이는 slave에도 똑같이 적용된다.

|  |  |  |
| --- | --- | --- |
| Direction | Port name | Description |
| Input | clk | Clock |
| reset\_n | Active low reset |
| m\_req | Master request |
| m\_wr | Master write/read |
| m\_addr[15:0] | Master address |
| m\_dout[31:0] | Master data output |
| s\_dout[63:0] | Slave data out |
| Output | m\_grant | Master grant |
| m\_din[63:0] | Master data input |
| s\_addr[15:0] | Slave address |
| s\_wr | Slave write/read |
| s\_din[63:0] | Slave data input |

Table 2. Pin description of BUS

1. MP (Mini Processor)

4.1. Introduction

MP는 operational variable 두 개의 입력을 이용해 opcode(operation code)에 따른 연산을 이용해 결과값을 도출하는 hardware이다. 이번 프로젝트에서 구현하는 ALU는 operational variable은 64bit width이다. 모든 ALU 출력은 64 bit width이다.

4.2. Features

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

✓ 곱셈의 경우, 아래 소개하는 data register R0~R9의 하위 32bit만을 operands로 이용하고 곱셈결과의 bit length는 64bits이다.

✓ 단, behavioral RTL (예: 곱셈을 \*기호를 써서 구현하는 것)로 구현하면 안된다.

✓ 덧셈, 뺄셈: CLA (carry look-ahead adder)를 이용한다.

✓ 곱셈: booth multiplier를 이용한다.

4.3. Functional description

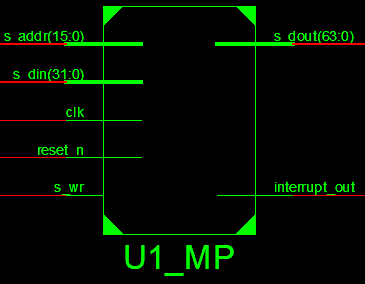


Figure 4. Schematic Symbol of MP

Figure 4는 MP의 schematic symbol을 나타낸다. MP는 Bus와 연결되는 slave component로, slave interface를 통하여 외부에서는 내부 동작을 제어하는 것이다. MP의 register description에 대한 상세한 설명은 뒤에서 다룬다.

4.4. Pin description

|  |  |  |
| --- | --- | --- |
| Direction | Port name | Description |
| Input | Clk | Clock |
| Reset\_n | Active low reset |
| S\_wr | Write/read |
| S\_addr[15:0] | Slave Address |
| S\_din[31:0] | Data input |
| Output | S\_dout[63:0] | Data output |
| Interrupt\_out | Interrupt(MP에서 연산이 완료되었을 때 내부 register값에 따라 interrupt를 발생한다.) |

Table 3. Pin Description of MP

4.5. Register description

MP는 아래 표에 보이는 바와 같은 register를 가지고 있다. 이해를 돕기 위해 register들을 크게 3개의 group으로 나누어 설명한다.

* registers - = {R0, R1,…, R9}
* registers - = {I0, I1,…, I9}
* controlregisters -



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Register name | Offset | Readable (R) or writable (W) | Bit width | Description |
| DATA\_REG | R0 | 0x0000 | R/W | 64 | Data register 0 |
| R1 | 0x0001 | R/W | 64 | Data register 1 |
| … | … | … | … | … |
| R9 | 0x0009 | R/W | 64 | Data register 9 |
| INST\_REG | I0 | 0x0010 | R/W | 64 | Instruction register 0 |
| I1 | 0x0011 | R/W | 64 | Instruction register 1 |
| … | … | … | … | … |
| I9 | 0x0019 | R/W | 64 | Instruction register 9 |
| CONT\_REG | OP\_START | 0x0020 | R/W | 64 | Operation start register로 MP동작의 시작 여부를 결정한다. OP\_START[0] = 1이면, I0부터 I9까지 10개의 instruction을 수행하여 결과를 data register들 R0~R9에 저장한다.  OP\_START[0] = 0이면 idle상태로 아무 일도 하지 않는다. |
| INT\_MASK | 0x0021 | R/W | 64 | Interrupt mask register로 output port Interrupt\_out의 값을 결정하는데 이용한다. Interrupt\_out = INT\_MASK[0] & INTERRUPT[0] 이다. |
| INTERRUPT | 0x0022 | R/W | 64 | Interrupt register로 MP가 10개의 instruction I0 ~ I9까지 모두 수행 완료하면 INTERRUPT[0]에 1을 쓴다. |

Table 4. Register Map

DATA\_REG

MP 연산이 사용할 data를 저장하는 Register들이다. 32비트 data 10개를 testbench로 입력 받아 사용한다. ALU 및 Multiplier에 Operand a,b가 wire로 연결되어 있어 연산을 시작하고 Destination address를 받아 결과 값을 저장할 수 있다.

INST\_REG

Instruction을 저장하는 Register들이다. 본 프로젝트에서는 10개의 Instruction을 저장한다. 각각의 instruction은 다음과 같다. Destination과 Operand a,b는 DATA\_REG의 저장될 공간의 주소이다.

* instruction은 16비트로 구성되어 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Destination (Rd) | Operand a (Ra) | Operand b (Rb) |
| 4bit | 4bit | 4bit | 4bit |

* Opcode은 다음과 같이 9개의 Function을 사용한다.

|  |  |  |
| --- | --- | --- |
| Opcode. | Mnemonic | Function |
| 0000 | NOT\_A | Rd = ~Ra |
| 0001 | NOT\_B | Rd = ~Rb |
| 0010 | AND | Rd = Ra & Rb |
| 0011 | OR | Rd = Ra | Rb |
| 0100 | XOR | Rd = Ra ^ Rb |
| 0101 | XNOR | Rd = ~(Ra ^ Rb) |
| 0110 | ADD | Rd = Ra + Rb |
| 0111 | SUB | Rd = Ra - Rb |
| 1000 | MUL | Rd = Ra \* Rb |

CONT\_REG

3개의 register는 OP\_START, INT\_MASK, INTERRUPT이다.

* OP\_START에 값이 1이 써지게 되면 연산을 시작한다.
* INTERRUPT는 모든 instruction이 수행되면 1이 저장되게 한다.
* testbench가 INT\_MASK에 1을 쓰면 interrupt가 외부로 출력이 되며 그렇지 않으면 interrupt출력이 나오지 않게 설계한다.