Microprocessors

SDSU BUS

Ana Maghradze

31st October 2021

Contents

1 Description of The Assignment **3**

2 Description of The Solution **3**

3 ModelSim Simulation **4**
3.1 Data Writing Process 4
3.2 Calculation Process 5
3.3 Multiplication Result 6

4 Appendix **7**

1 Description of The Assignment

For this assignment, we have to implement our "SDSU BUS" protocol. We should have master and slave modules and « single master to single slave » communication between them.

Requirements:

- We need to have 32 registers, each of 32-bit size.
- reg0 is for triggers, 0th bit of reg0 is for calculation trigger - if it's 1, multiplier starts calculation. We need this value to avoid 'multiple drivers' problem.

  reg1 is for first multiplicand (16 bits) and reg2 is for second multiplicand (16 bits).

  reg16 is for multiplication output.
- Multiplication should take 16-17 clock cycles and when calculation is done, it should generate ready signal to notify slave that calculation result is ready. Multiplier ready signal is connected to slave's ready signal.

We can either instantiate both master and slave modules in a single testbench or instantiate slave in the master module.

2 Description of The Solution

I decided to instantiate slave under master module and have a separate module for multiplication.

In Slave: I have REG array for 32 32bit registers. When `valid = 1`, slave receives write address and write data values from master and stores `WData` at `REG[WAddr]` - first multiplicand in `REG[1]` and second multiplicand in `REG[2]` and sets `ready = 0`.

When `valid = 0`, slave is in initial state or it has already received values and now it can start calculation. Now we need to check if 0th bit of `REG[0]` is 1, if so, `calculate = 1` and `REG[0][0] = 0` that means that, I tell multiplier to start calculation and then turn of calculation trigger. When `REG[0][0] = 0`, if multiplication is finished I set `calculate = 0`.

I instantiate multiplier in slave module and give it corresponding inputs from slave, when calculation is finished, multiplier returns result and generates ready signal that is connected to the slave's ready signal.

When multiplication is finished, `ready = 1` and I write `multiplication_result` into `RData` and `REG[RAddr]` - value of `RAddr` is provided by master and is 16.

In Multiplier: I have two 16-bit inputs A and B, 32-bit output result, 1-bit ready signal and also reset signal. If `reset = 0`, I initialize all the inputs and set outputs to 0. If `reset = 1`, that means that slave tells multiplier to start calculation. I additionally use `i` and `sum` variables. In `sum`, I store partial sums of calculation. while `i < 16`, if i-th bit of `B` is 0, `sum` remains the same, otherwise, `sum` becomes the previous `sum` plus the value of `A` shifted by `i` that is 2 to the power of `i`. If `i > 16`, multiplication is finished and ready signal is generated.

Multiplication takes exactly 16 clock cycles and at 17th clock cycle, multiplication result and ready signal is generated, at 18th clock cycle, result is written in slave's `RData and REG[RAddr]`. As shown on figure 3, for inputs A = 65535 and B = 65535, result is 4294836225.

In Master: I just instantiate slave module and provide values for inputs: `valid, WAddr, RAddr, WData` and connect slave outputs to `RData and ready`;

3 ModelSim Simulation

3.1 Data Writing Process

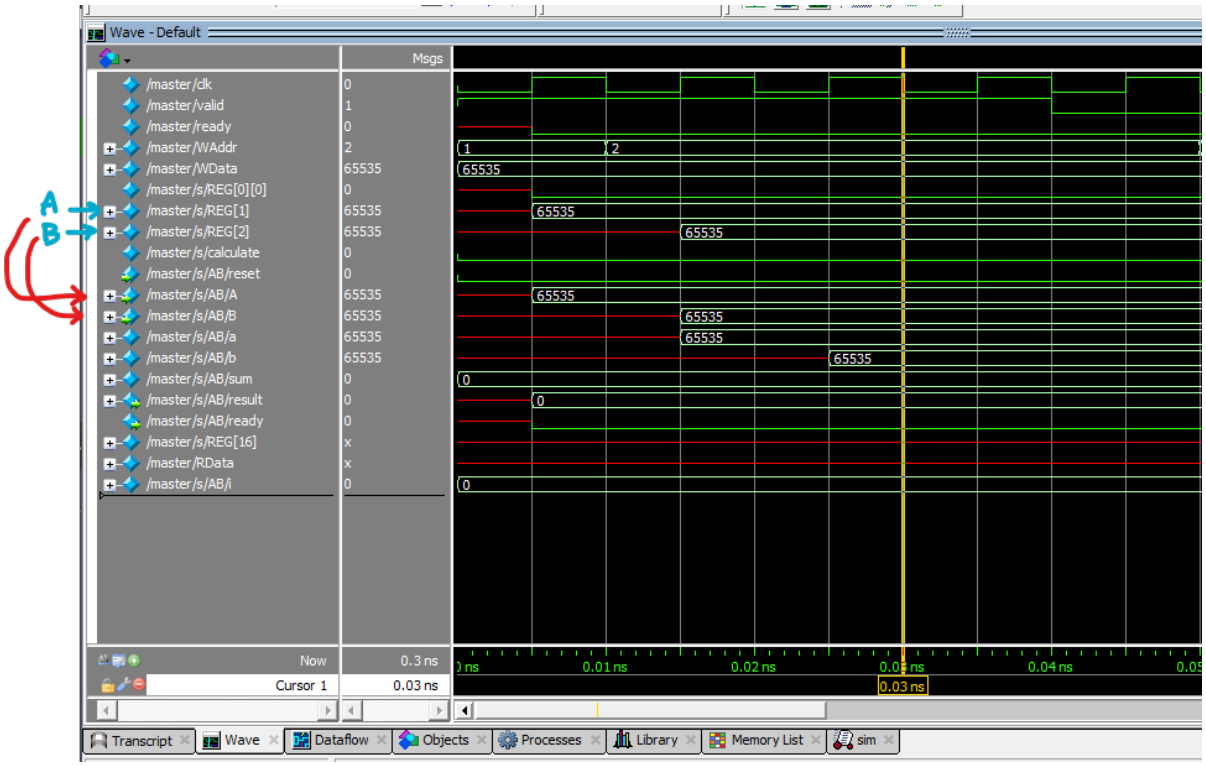

Figure 1: Data Writing Process

3.2 Calculation Process

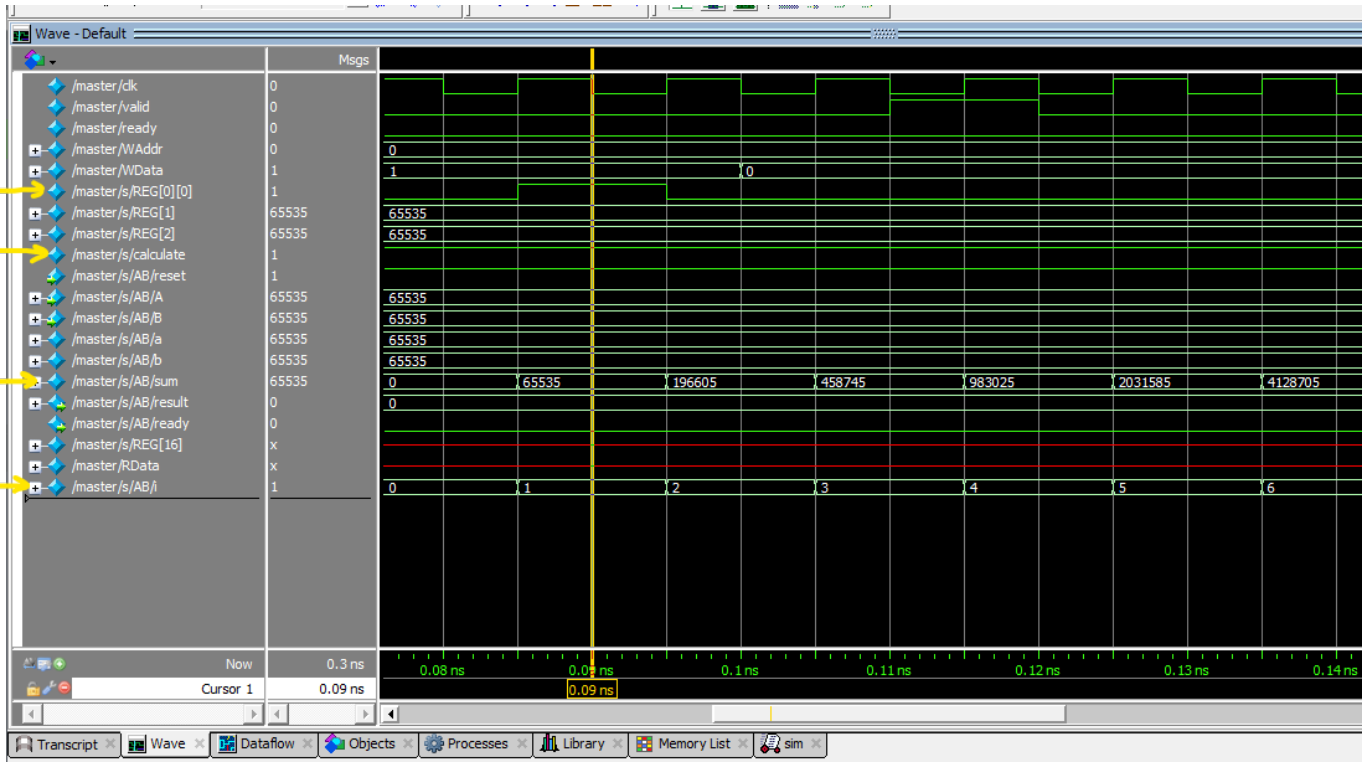

Figure 2: Calculation Process

3.3 Multiplication Result

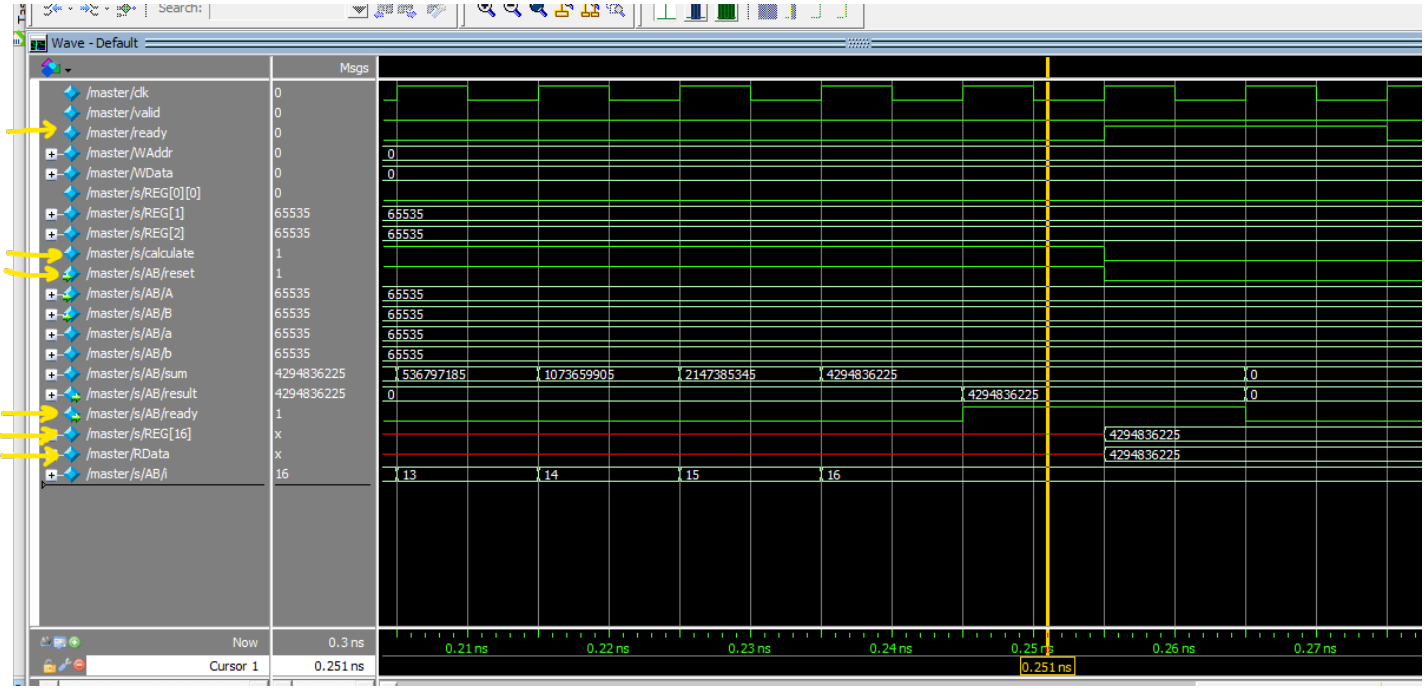

Figure 3: Multiplication Result

4 Appendix

```
module multiplier(
  input clk, reset,
  input [15:0] A, B,
  output reg [31:0] result,
  output reg ready
);

reg [15:0] a;
reg [15:0] b;
reg [5:0] i = 0;
reg [31:0] sum = 0;

always @(posedge clk)
begin
  if(reset == 0)
  begin
    a <= A;
    b <= B;
    sum <= 0;
    result <= 0;
    ready <= 0;
  end
  else
  begin
    if(i < 16)
    begin
      sum <= b[i] == 1 ? sum + (a << i) : sum;
      i <= i + 1;
    end
    else
    begin
      ready <= 1; // multiplication finished
      result <= sum;
    end
  end
end

endmodule
```

```
 module slave(
  input clk, valid,
  input [4:0] WAddr,
  input [4:0] RAddr,
  input [31:0] WData,
  output reg [31:0] RData,
  output reg ready
);

/*
REG: array of 32 registers, each of 32-bit length
- REG[0] -> triggers, REG[0][0] for calculation trigger
- REG[1] -> [15:0] for value of A,
- REG[2] -> [15:0] for value of B,
...
```

```
- REG[16] -> multiplication result
*/
reg [31:0] REG [31:0];

reg calculate = 0;

wire [31:0] multiplication_result;
wire multiplication_finished;


multiplier AB(clk, calculate, REG[1][15:0], REG[2][15:0], multiplication_result[31:0], multiplication_finished);


always @(posedge clk)
begin
  REG[0][0] <= WAddr == 0 && WData[31:0] == 1; // calculation trigger;
  ready <= multiplication_finished;

  if(multiplication_finished)
  begin
    REG[RAddr][31:0] <= multiplication_result;
    RData[31:0] <= multiplication_result;
  end

  if(valid == 0)
  begin
    if(REG[0][0] == 1)
    begin
       calculate <= 1; // when calc trigger is ON, start calculation
      REG[0][0] <= 0; // turn off calculation trigger
    end
    else if(REG[0][0] == 0 && multiplication_finished) calculate <= 0;
  end
  else
  begin
    REG[WAddr][31:0] <= WData[31:0];
    ready <= 0;
  end
end


endmodule
```

```
module master;

reg clk = 0;
reg valid = 0;
reg [4:0] WAddr = 0;
reg [31:0] WData = 0;
reg [4:0] RAddr = 0;
wire ready;
wire [31:0] RData;

slave s(clk, valid, WAddr, RAddr, WData, RData, ready);

always #5 clk <= ~clk;

initial
```

```
begin
  valid = 0;
  WAddr = 1; // address for A - reg1
  WData = 65535; // 16'b1111_1111_1111_1111; // A = 65535,
  valid = 1;
  #10 WAddr = 2; // address for B - reg2
  #10 WData = 65535;// 16'b1111_1111_1111_1111; // B = 65535
  #10 RAddr = 16; // address for multiplication output is reg16
  #10 valid = 0;

  #10 WAddr = 0; // address for calculation trigger
  #10 WData[31:0] = 1; // turn on calculation trigger to start multiplication
  #10 valid = 0;
  #10 valid = 0;

  #10 WAddr = 0;
  #10 WData[31:0] = 0;
  #10 valid = 1;
  #10 valid = 0;
end

endmodule
```