Team Project of CMPE 200: Implementation and Verification a memory transfer block

Chao Pi
Engineering *department*
San Jose State University
San Jose, CA
013785265

Zihao Gu
Engineering *department*
San Jose State University
San Jose, CA
012503543

***Abstract*— This project is targeted to implement a memory transfer block and test the timing diagram of it. We divide the whole block into several modules: MEM_counter_A (Counter A and Memory A), MEM_counter_B (Counter B and Memory B), Controller, Adder. We also write a test module to test if the time diagram shows progress of block memory transfer. We verify the simulation of each block and the whole system.**

Keywords—memory transfer, timming, simulation, bus tranfer

I. INTRODUCTION

This project is in order to implement a memory transfer block and verify it. Our data path is that download two packets of data from source memory A and pass them to destination memory B. If the first data packet is larger than the second one, their subtraction will be stored at AddrA in Memory A, otherwise their addition will be stored.

II. IMPLEMENTATION

A. MC_A

MC_A module includes a three-bit counter - Counter A, and a source memory block - Memory A.

In Counter A, Counter A generates AddrA[2:0] which is the address points to a row in memory A. When reset and IncA is low, the counter will reset and point to 0. When the reset signal is low and IncA is high, AddrA increases by one until reset.

In memory A, when WEA(write enable signal) is high, it begins to write that data from DataInA will be stored in the address AddrA[2:0] in Memory A. When WEA is low, it starts to read and data stored in the AddrA is provided to Dout1[7:0].

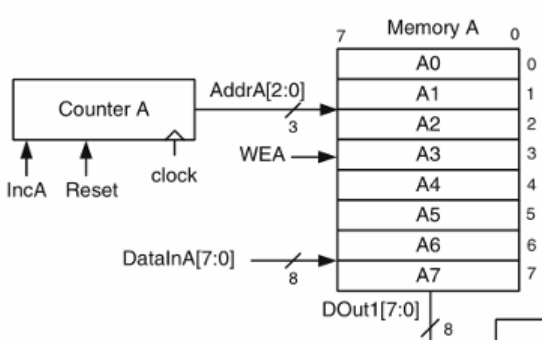

Fig. 1. Transmission between Counter A and Memory

The wave pattern of Memo_counter_A is showed as follow. When IncA and WEA both are high, memory A gets the data from DataInA and store them in AddrA. WEA make sure memory A will store data, not read data. IncA increase AddrA, to make sure new data from DataInA will be stored in 0 to 7. When finish write, counter will reset and WEA changes to low, memory starts to read data in the cycle 9. After a valid address is introduced, so Dout1[7:0] will be available in the cycle of 10. In store data period, we met a problem that memory A can't store A0. Because when IncA and WEA be 1, memory A must wait next positive edge to store value, in that moment, DataInA will already be A1. To solve this, we use negative edge to implement store value part to make sure memory A store data exactly when IncA and WEA change to high.

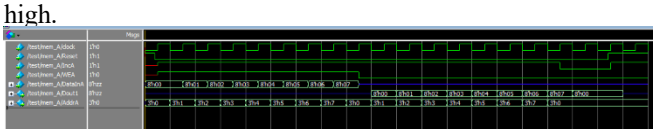

Fig. 2. Simulation of MC_A

B. MC_B

The MC _B module includes a two-bit counter - Counter B and a destination memory block Memory B.

In counter B, it generates an address AddrB that points to address in memory B. When the reset signal is high, AddrB is set to 0. When the reset signal is low and the enable signal IncB is high, AddrB increases by 1.

In memory B, we don't need to read from it. When WEB is high, DataInB is stored at AddrB in Memory B. When WEB is low, nothing is needed.

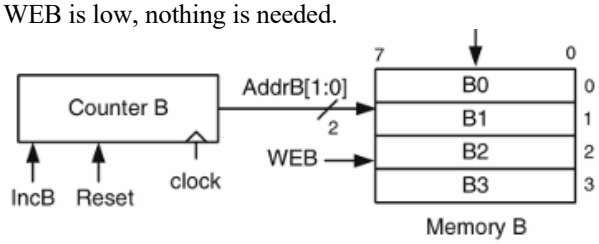

Fig. 3. Transmission between Counter B and Memory B

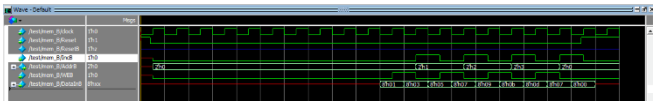

Fig. 4. Simulation of MC_B

C. Adder

Adder component has two 7-bit inputs: DOut1, Dout2. The output is ADDOut which is the sum of DOut1 and Dout2.

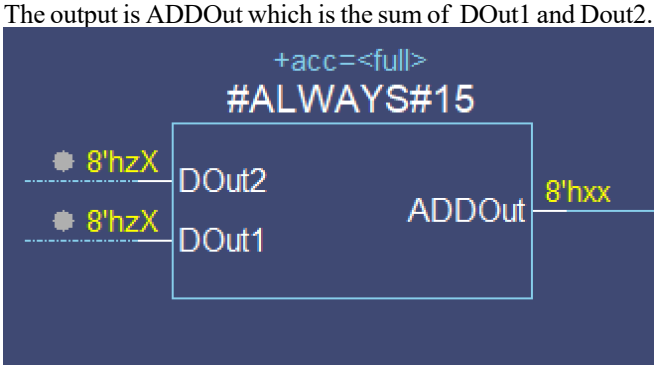

Fig. 5. component of Adder

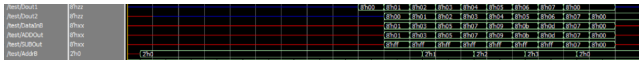

Fig. 6. Simulation of Adder

D. Comparator

D flip-flop generates a memory data output Dout2, which is the value of previous clock cycle Dout1. COMP compares Dout1 and Dout2 and provides a Sign signal to control. If Dout2 is less than Dout1, COMP points to 1, Dout1 and Dout2 are sent to an adder, ADDOUt will be DataInB. However, if Dout2 is greater than Dout1, COMP points to 0, SUBOUT will be DataInB. DataInB is one of input in MC_B.

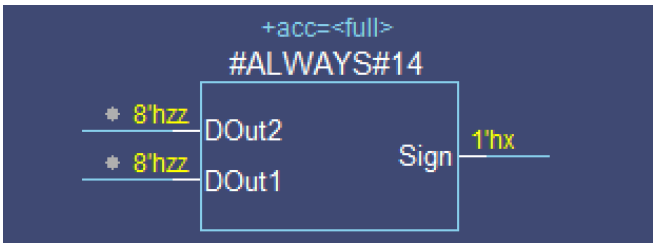

Fig. 7. Component design of Comparator

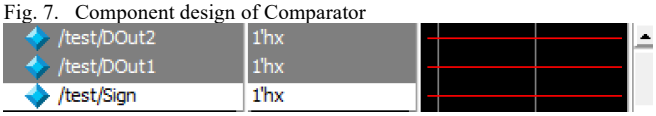

Fig. 8. Simulation of Comparator

E. D flip-flop

D flip-flop generates a memory data output Dout2, which is the value of previous clock cycle Dout1. COMP compares Dout1 and Dout2 and provides a Sign signal to control. If Dout2 is less than Dout1, COMP points to 1, Dout1 and Dout2 are sent to an adder, ADDOUt will be DataInB. However, if Dout2 is greater than Dout1, COMP points to 0, SUBOUT will be DataInB. DataInB is one of input in MC_B.

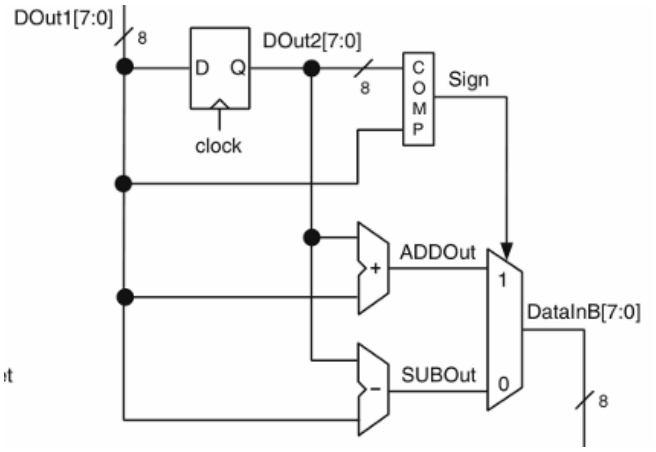

Fig. 9. Design of D flip_flop module

The wave of this module is below. In clock cycle 10, memory A has output data 8'h00. Because Dout2 is from Dout1 delay one cycle use a D flip-flop, so we can compare them at the same cycle. If Dout1>Dout2, mux will choose 1 and DataInB is ADDOut, otherwise mux will choose 0 and DataInB is SUBOut. In our test, Dout1 > Dout2 from is cycle11 to cycle 17, ADDOut data is DataInB, SUBOUT will shows ff in hex, which means 0-1,1-2,2-3 and so on.

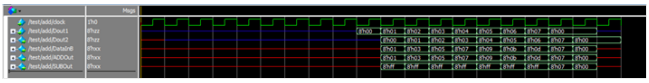

Fig. 10. Simulation of Add_Sub module

F. Controller

Controller has two inputs: Reset and clock, and four outputs: WEA, IncA, WEB, IncB.

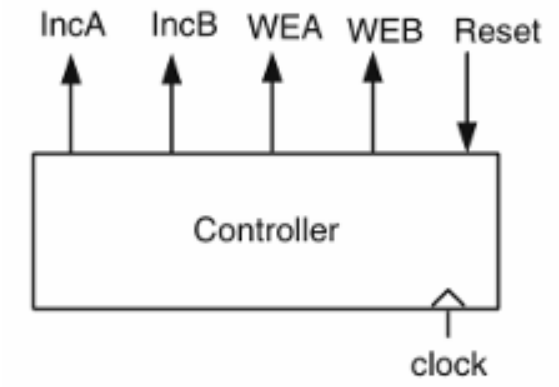

Fig. 11. Simulation of Controller

We use Moore type state machine to implement the controller. The following figure shows an implementation by Moore machine (Fig. 2.47 from Ref. [1]). The values of the present state outputs, WEA, IncA, WEB and IncB, The reset state, S0, is included in the Moore machine in case the data-path receives an external reset signal to interrupt an ongoing data transfer process. Whichever state the machine may be in, Reset = 1 always makes the state machine transition to the S0 state. Control module is to generate control signal. The pattern is showed as follow.

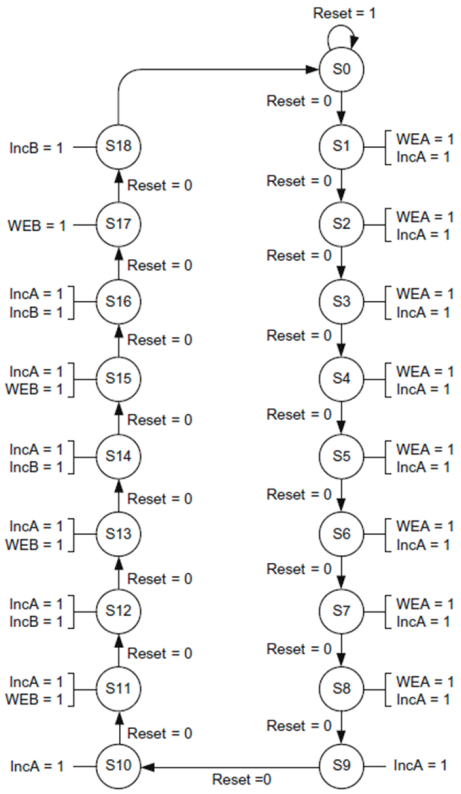

Fig. 12. Moore representation of the controller unit

Controller generates control signals to control modules. Wave is showed below.

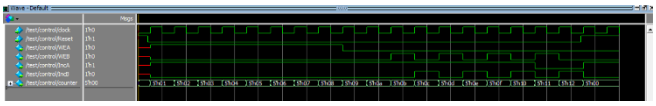

Fig. 13. Simulation of the Controller unit

G. *Memo_Trans_ts*

The wave of this module is below. In clock cycle 10, me The test module forms the four modules above, Reset, clock and DataInA. We select the clock as 200 ps. We use Reset 1 as an initial value, after 80 ps, make Reset 0, and also initialize DataInA's 8 bits all zeros. Every time wait 200ps (one cycle), DataInA increases by 1, until to fill in all rows in memory A (8 times).

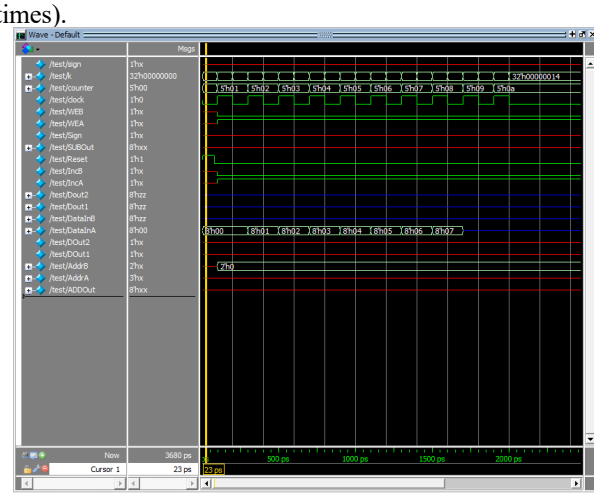

Fig. 14. Simulation of Memory Transfer Block

Learning Experience and Difficulties

Before this project, we need to spend a lot time to learn Verilog language grammar and get familiar with *ModelSim*. The difficulties is that how to do Simulation for several modules. We have learned how to verify block after homework in this class, but we never simulate several modules in one test. So the most challenge thing is how to simulation. The other challenge things is how to design controller. After read the textbook carefully, I found that controller can be design by Moore state machine and counter-decoder. Both are works well in general.

Conclusions

From this project, we know how to design a project using sequential logic and memory. The most important thing is that we need to know the logic blocks to govern the data path. If the block diagram is confirmed, we need to verify it. In this project, we verify each modules separately and then verify the whole project. The controller generates signal to control the whole system. In order to define the states of the controller, clock periods that generate different sets of controller outputs have to be confirmed at first.

References

[1] Ahmet Bindal, Fundamentals of Computer Architecture and Design, 2017

Source Code

Adder:

```
module Adder(
    DOut1,
    DOut2,
    ADDOut
);

input [7:0] DOut1;
input [7:0] DOut2;
output [7:0] ADDOut;

wire [7:0] DOut1;
wire [7:0] DOut2;
reg [7:0] ADDOut;

always @(*)
    ADDOut <= DOut1 + DOut2;

endmodule
```

Subber:

```
module Subber(
    DOut1,
    DOut2,
    SUBOut
);
```

```
input [7:0] DOut1;
input [7:0] DOut2;
output [7:0] SUBOut;

wire [7:0] DOut1;
wire [7:0] DOut2;
reg [7:0] SUBOut;

always @ (*)
SUBOut <= DOut1 - DOut2;

endmodule
```

Comparator:

```
module Comparator(
    DOut1,
    DOut2,
    Sign
);

input [7:0]DOut1,DOut2;
output Sign;

wire [7:0] DOut1;
wire [7:0] DOut2;
reg Sign;

always @(*)
if(DOut1 > DOut2)
    Sign <= 0;
else
    Sign <= 1;

endmodule
```

D-Flip-Flop:

```
module D_Flip_Flop(
    DOut1,
    clock,
    DOut2
);

input [7:0] DOut1;
input clock;
output [7:0] DOut2;

wire [7:0] DOut1;
wire clock;
reg [7:0] DOut2;

always @(posedge clock)
    DOut2 <= DOut1;

endmodule
```

MC_A:

```
module MC_A (
    IncA,
    Reset,
    clock,
    AddrA,
    WEA,
    DataInA,
    DOut1
);

input IncA, WEA, Reset, clock;
input [7:0] DataInA;
output [7:0] DOut1;
output [2:0] AddrA;

wire IncA, WEA, Reset, clock;
reg [2:0] AddrA;
reg [7:0] DOut1;
reg [7:0] MemA[0:7];

always @(posedge clock)
    begin
        if(!Reset)
            begin
                if(IncA && WEA && !(DataInA
=== 8'bz))
                    begin
                        MemA[AddrA] =
DataInA;
                        AddrA = AddrA + 1;
                    end
                else if(IncA && !WEA)
                    begin
                        DOut1 = MemA[AddrA];
                        AddrA = AddrA + 1;
                    end
                else if(!IncA && WEA
&& !(DataInA === 8'bz))
                    MemA[AddrA] = DataInA;
                else if(!IncA && !WEA)
                    DOut1 = MemA[AddrA];
            end
        else
            begin
                AddrA = 3'b000;
                DOut1 = 8'bz;
    end
```

```
end

endmodule
```

MC_B

```
module MC_B (
    IncB,
    Reset,
    clock,
    AddrB,
    WEB,
    DataInB
);

input IncB, WEB, Reset, clock;
input [7:0] DataInB;
output [1:0] AddrB;

wire IncB, WEB, Reset, clock;
reg [1:0] AddrB;
reg [7:0] MemB[0:3];

always @(posedge clock)
    begin
        if(!Reset)
            begin
                if((IncB == 1'b0 && WEB ==
1'b1) || (IncB == 1'b1 && WEB == 1'b0))
                    begin
                        if(WEB)
                            MemB[AddrB] <=
DataInB;
                        if(IncB == 1'b0 &&
WEB == 1'b1)
                            AddrB = AddrB +
1;
                    end
                else
                    AddrB <= 2'b00;
            end
        else
            AddrB <= 2'b00;
    end

endmodule
```

Controller:

```
module Controller(
    clock,
    Reset,
    IncA,
    IncB,
    WEA,
    WEB,
    counter
);

input clock, Reset;

output IncA, IncB, WEA, WEB;
output [4:0] counter;

wire clock, Reset;
reg IncA, IncB, WEA, WEB;
reg [4:0] counter = 5'b00000;

always @(posedge clock)
begin
    if(Reset == 1'b1 || counter == 5'b10010)
        counter = 5'b0;
    else
        counter = counter + 1;

    if(counter >= 5'd1 && counter <= 5'd8)
        WEA = 1'b1;
    else
        WEA = 1'b0;

    if(counter != 5'b10001 && counter !=
5'b10010 && counter != 5'b10011)
        IncA = 1;
    else
        IncA = 0;

    if(counter == 5'b01011 || counter ==
5'b01101 || counter == 5'b01111 || counter
== 5'b10001)
        WEB = 1;
    else
        WEB = 0;

    if(counter == 5'b01100 || counter ==
5'b01110 || counter == 5'b10000 || counter
== 5'b10010)
        IncB = 1;
    else
        IncB = 0;
end

endmodule
```