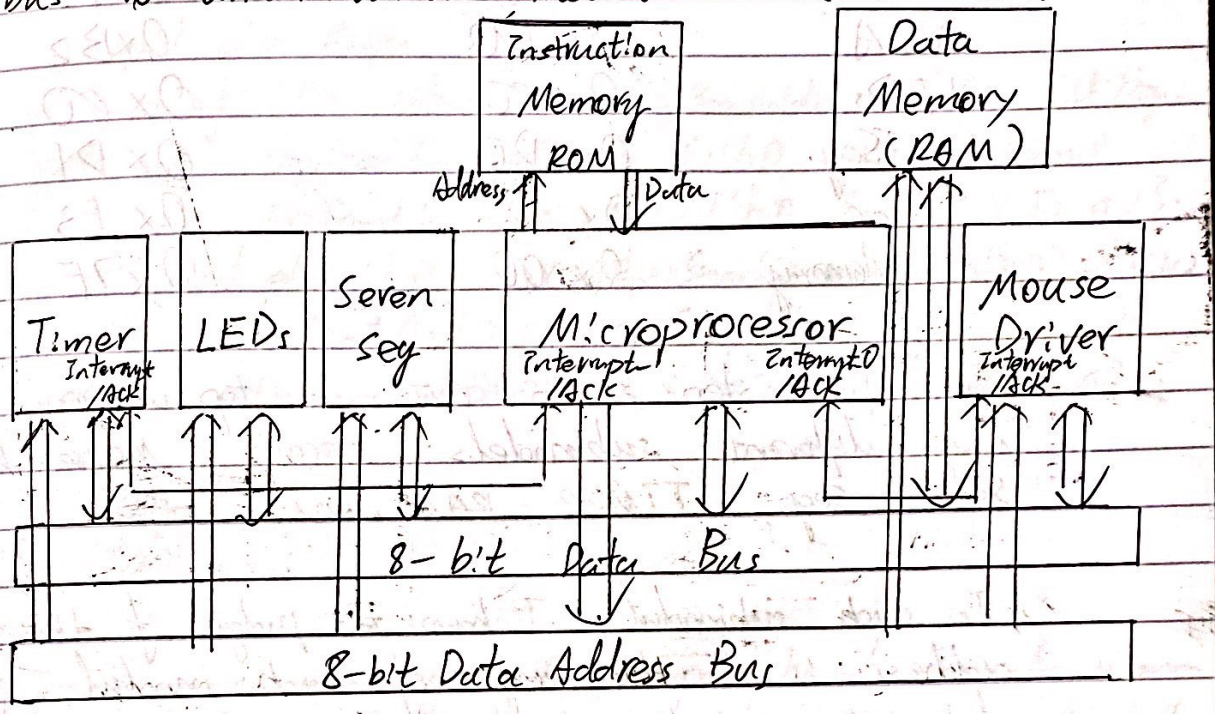
Microprocessor — Mouse

Then start to try to achieve mouse function with microprocessor. I First to draw a block diagram of FPGA-based system for mouse. The microprocessor will be the master of an 8-bit bus to which various modul will be connected.



The function of this project is the same as the last project. However, in the last project, ti different submodules connected directly, and the data flow from moduls to moduls.

but in the microprocessor, the whole system is controlled by microprocessor when a data flow from one modul to another, The data writes into ~~ROM~~ RAM (data memory) first, and the data write to the display modul. The whole process is controlled by the instructions in the ROM.

The memory mapping of different peripherals is shown below.

Peripheral	Base Address	High Address
IR Transmitter	0x90	0x90
Mouse	0xA0	0xA2
VGA	0xB0	0xB2
LEDs	0xC0	0xC0
SevenSeg	0xD0	0xD1
Timer	0xF0	0xF3
Data Memory	0x00	0x7F

The first step is to creat a top wrappor and wrap different submodels: Processor, Mouse Driver, Seven Seg, TIMER, RAM, ROM, LED.

In each submodul, I have to judge if the bus address is in the range of each modul. If the data is in the working rang of a modul

In the block of SevenSeg Display. We wanna display two 8-bit information togethor. However, the bus line is only 8-bit. Therefore, the SevenSeg Display has two bus address.

I assign base address of SevenSeg Display for D0 and high address for D1.


```
always (posedge CLK) begin
  if (RESET) begin
      DecCountAndDOT0 <= 5'h0;
      DecCountAndDOT1 <= 5'h0;
      DecCountAndDOT2 <= 5'h0;
      DecCountAndDOT3 <= 5'h0;
  end else begin
      if ((BUS_ADDR == SevenSegBaseAddr) && BUS_WE) begin
          DecCountAndDOT0 <= {1'b0, BUS_DATA[3:0]};
          DecCountAndDOT1 <= {1'b0, BUS_DATA[7:4]};
      end else if ((BUS_ADDR == SevenSegBaseAddr + 8'h01) && BUS_WE) begin
          DecCountAndDOT2 <= {1'b0, BUS_DATA[3:0]};
          DecCountAndDOT3 <= {1'b0, BUS_DATA[7:4]};
      end
  end
end
```

It is the same for LED peripheral. I only wanna to display 4 bit byte status, therefore 8-bit bus line is enough. The Base Address and High Address is the same for LEDs modul.

```
always (posedge CLK) begin
    if (RESET)
        LEDS <= 4'h0;
    else if ((BUS_ADDR == LEDBaseAddr) && BUS_WE)
        LEDS <= BUS_DATA[3:0];
end
```

For the MouseDriver Module, I wanna output three 8-bit data to the bus data line. However, the Abus data line is only 8-bit. Therefore I have to use three bus address for Mouse.

0xA0	Mouse Status Byte
0xA1	Mouse X byte
0xA2	Mouse Y byte

The code is as followings:

```
always (posedge CLK) begin
    if (RESET)
        MouseOut <= 8'h0;
    else begin
        case (BUS_ADDR)
            (MouseBaseAddr + 8'h00):
                MouseOut <= {4'h0, MouseStatus};
            (MouseBaseAddr + 8'h01):
                MouseOut <= MouseX;
            (MouseBaseAddr + 8'h02):
                MouseOut <= MouseY;
            default:
                MouseOut <= 8'hZZ;
        endcase
    end
end
```

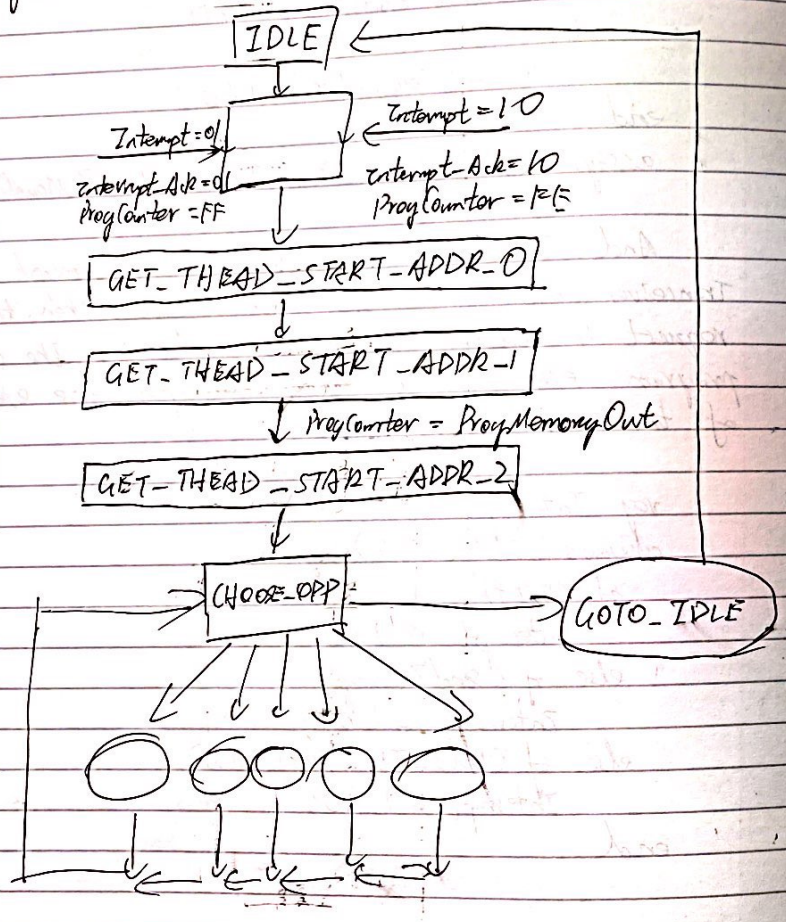
The BUS_DATA line is an inout line. Therefore, I have to deal with Interrupt timer output value. The BUS_DATA is in output model when the bus address is in the working rang of Mouse. It is High Impedence when it is input model.


```
reg TransmitMouseValue;
always (posedge CLK) begin
    if ((BUS_ADDR >= MouseBaseAddr) &&
        (BUS_ADDR <= (MouseBaseAddr + 8'h02)))
        TransmitMouseValue <= 1'b1;
    else
        TransmitMouseValue <= 1'b0;
end
assign BUS_DATA = (TransmitMouseValue) ? MouseOut[7:0] : 8'hZZ;
```

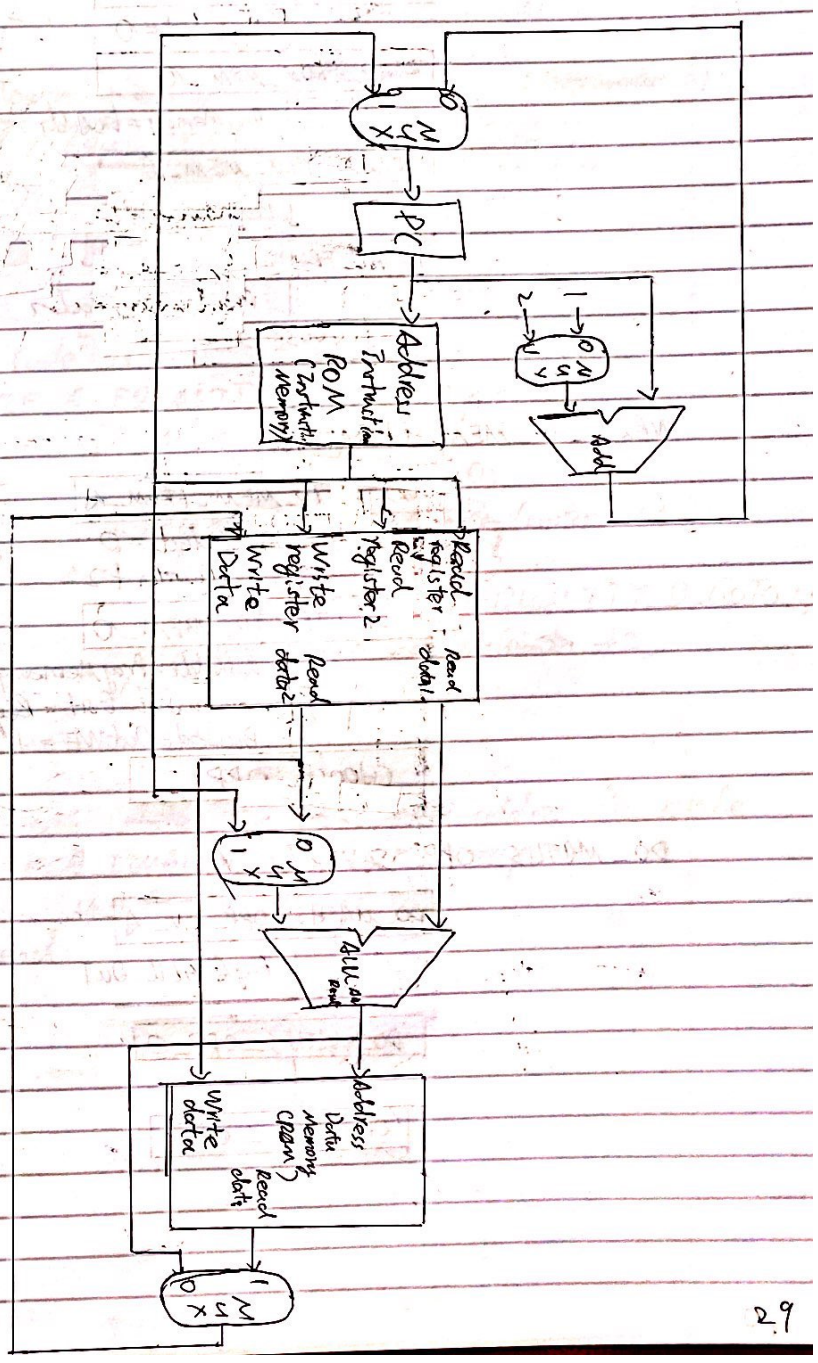
And I need to broadcast the Interrupt from Transceiver modul to the top Wrapper when the interrupt request is received by the microprocessor, the current program execution is suspended after the execution of the current instruction.

```
reg Interrupt;
always (posedge CLK) begin
    if (RESET)
        Interrupt <= 1'b0;
    else if (SendInterrupt)
        Interrupt <= 1'b1;
    else if (BUS_INTERRUPT_ACK)
        Interrupt <= 1'b0;
end
```

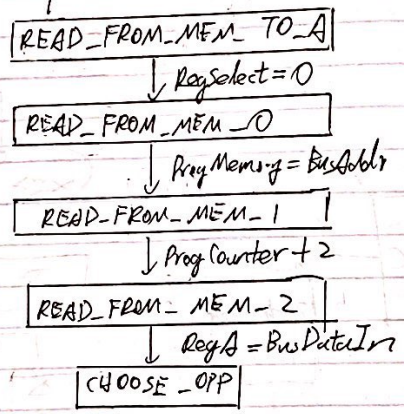
Then we to deal with Microprocessor part. The state Microprocessor is to fetch the instructions in the ROM and control the bus address to control the whole system. The state diagram of processor is:



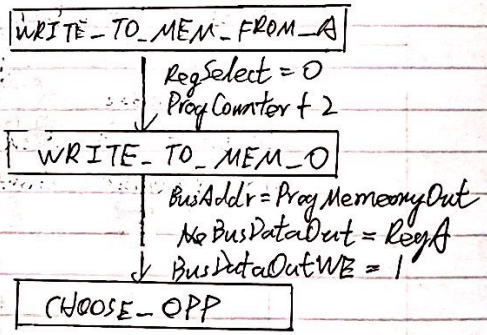
The Pipeline for the structure is:



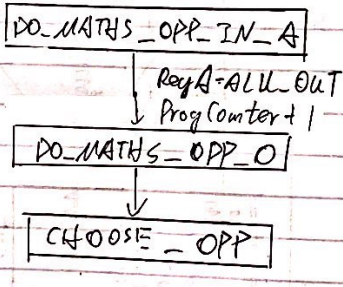
The state diagram for READ_FROM_MEM_TO_A:



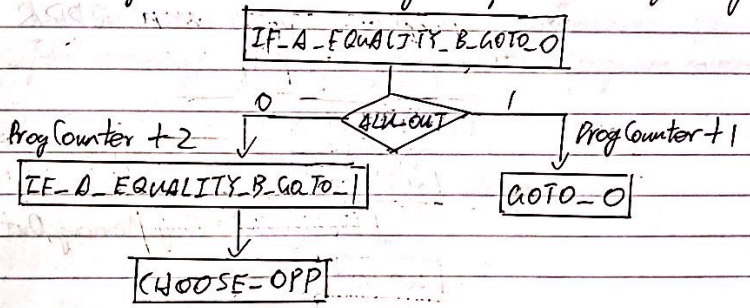
WRITE_TO_MEM_FROM_A



DO_MATHS_OPP_SAVE_IN_A:



Then I design the state diagram for Inv Equality:



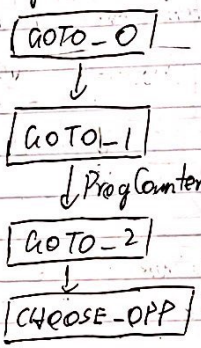
The code is:

```
IF_A_EQUALITY_B_GO_0: begin
    if(ALU_OUT) begin
        NextState = GOTO_0;
        NextProgCounter = CurrProgCounter + 1;
    end else begin
        NextState = IF_A_EQUALITY_B_GOTO_1;
        NextProgCounter = CurrProgCounter + 2;
    end
end
```

wait state for new prog address to settle

```
IF_A_EQUALITY_B_GOTO_1: begin
    NextState = CHOOSE_OPP;
end
```

GOTO: branch the to address ADDR, which means load program counter with ADDR



The code is:

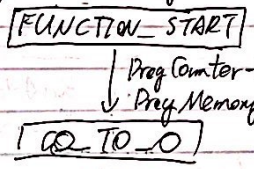
```
GOTO_0: begin    //g'in the for address to be read
    Next_State = GOTO_1;
end
GOTO_1: begin
    Next_State = GOTO_2;
    Next_ProgCounter = ProgMemoryOut;
end
```

Wait state: for new program address to settle

```
GOTO_2: begin
    Next_State = CHOOSE_OPP;
end
```

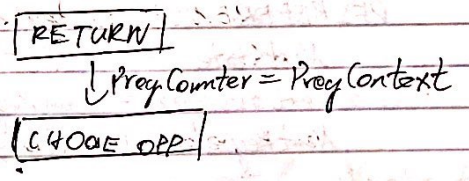
```
GOTO_IDLE: begin
    NextState = IDLE;
end
```


FUNCTION_CALL_ADDR: Branch to memory address ADDR. Save the next program address to execute from after returning from the function (program context)



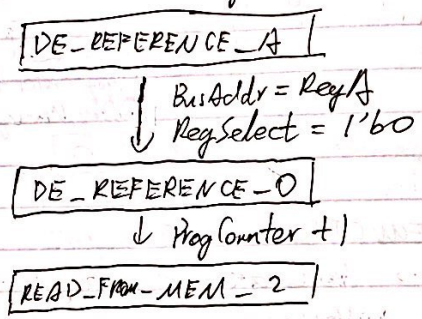

```
FUNCTION_START: begin
    NextState = GOTO_0;
    NextProgContext = CurrProgCounter + 1;
    NextProgCounter = ProgMemoryOut;
end
```

RETURN: Returns from a function call load program context to the program counter for next instruction execution.




```
RETURN: begin
    NextState = CHOOSE_OPP;
    NextProgCounter = CurrProgContext;
end
```

Dereference A: Read Memory address given by the value of register A and set the result as the new register value A ← [A]




```
DE_REFERENCE_A: begin
    NextState DE_REFERENCE_0;
    NextBusAddr = CurrRegA;
    NextRegSelect = 1'b0;
end

DE_REFERENCE_B: begin
    NextState = DE_REFERENCE_0;
    NextBusAddr = CurrRegB;
    NextRegSelect = 1'b1;
end

DE_REFERENCE_0: begin
    NextState = READ_FROM_MEM_2;
    NextProgCounter = CurrProgCounter +1;
end
```


Then to finish ROM to control the whole system.
The steps for the system is:

1. First, read mouse status to register A into A
2. store Display mouse status into LEDs
3. Read Mouse X coordinate into register A
4. Read Mouse Y coordinate into register B
5. Write Mouse X coordinate into 7-seg
6. Write Mouse Y coordinate into 7-seg

Therefore, the ROM ~~textxt~~ should be

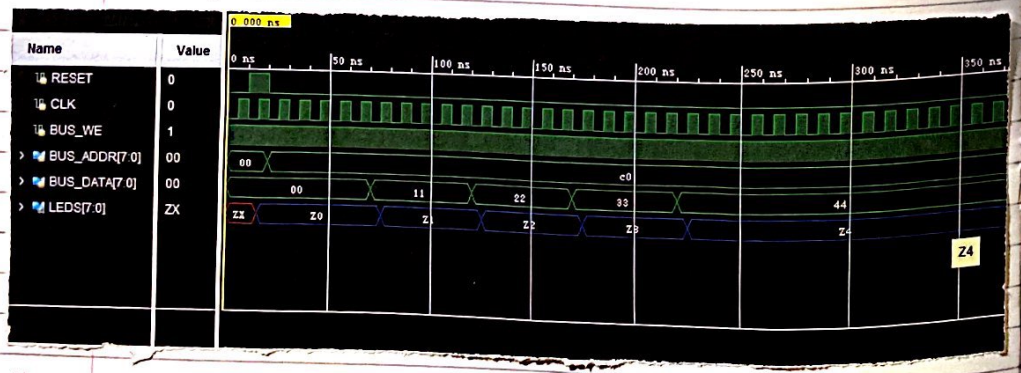
```
00// Read from memory to register A
B0 // The memory is Mouse status
02 // Write register A to memory
c0 // The memory address is LED
00// Read from memory to register A
A1// The memory address is Mouse X
01// Read from memory to register B
A2// The memory address is Mouse Y
02// Write register A to memory
1D1// The memory address is first and second 7-Seg
03// Write register B to memory
D0// The memory address is third and fourth 7-Seg
```

Simulation:
All the internal clock period set as 10nS, the same as 10MHz internal clock in FPGA

```
initial begin
    CLK = 0;
    forever #5 CLK = ~CLK;
end
```

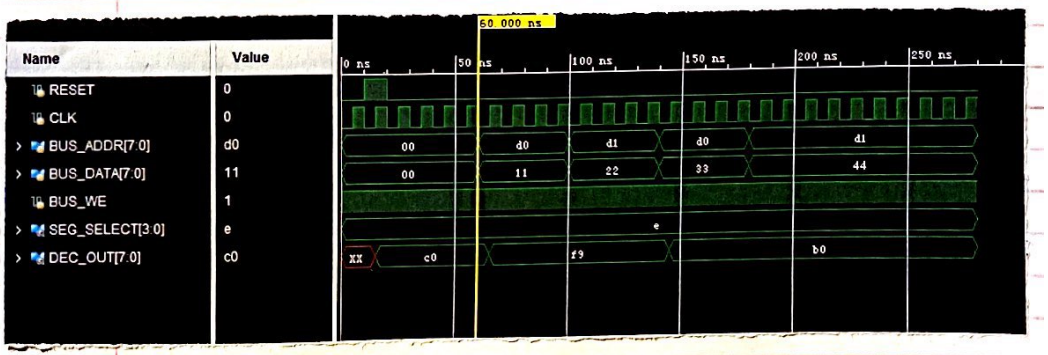
To simulate LED interface, I have to set the bus address as the LED base address, and increase bus data to simulate the LED display.

```
initial begin
    RESET = 0;
    BUS_ADDR = 8'h00;
    BUS_DATA = 8'h00;
    #10 RESET = 1;
    #10 RESET = 0;
    BUS_ADDR = 8'hC0;
    #50 BUS_DATA = 8'h11;
    #50 BUS_DATA = 8'h22;
    #50 BUS_DATA = 8'h33;
    #50 BUS_DATA = 8'h44;
end
```



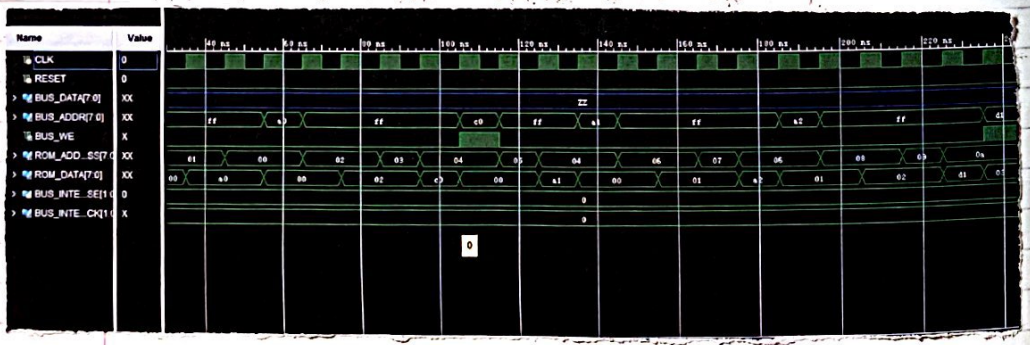
To simulate the SevenSeg peripheral, I ~~have~~ as SevenSeg has two bus address, I have to set bus address alternatively as D0 and D1, and set data on them. The bus write enable signal should always be 1.

```
initial begin
    BUS_WE = 1'b1;
end

initial begin
    #10 RESET = 1;
    #10 RESET = 0;
    #50 BUS_ADDR = 8'hD0;
        BUS_DATA = 8'h11;
    #50 BUS_ADDR = 8'hD1;
        BUS_DATA = 8'h22;
    #50 BUS_ADDR = 8'hD0;
        BUS_DATA = 8'h33;
    #50 BUS_ADDR = 8'hD1;
        BUS_DATA = 8'h44;
end
```



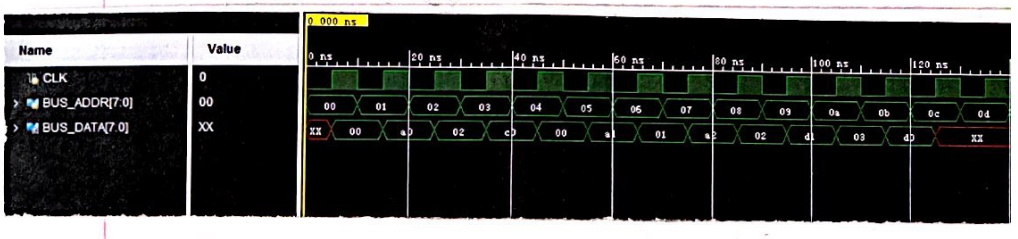
To simulate the MicroProcessor, I have to instantiate the processor model, ROM model and RAM model. And to test the function of interrupt signal, I set interrupt signal as 01 after 500ns.

```
initial begin
    RESET = 0;
    BUS_INTERRUPTS_RAISE = 2'b00;

    #10 RESET = 1;
    #10 RESET = 0;
    #500;              Wait for a period
    #10 RESET = 1;
    #10 RESET = 0;
    #10 BUS_INTERRUPTS_RAISE = 2'b01;
    #10 BUS_INTERRUPTS_RAISE = 2'b00;
end
```



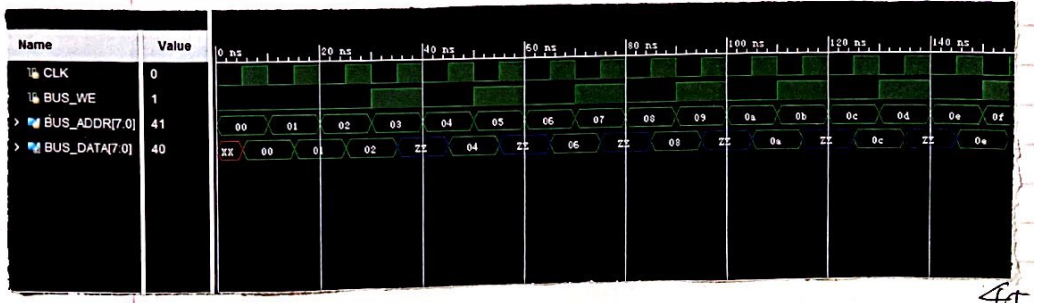
To simulate ROM, I just need to increase BUS_ADDR periodicity to extract data from instruction memory.

```
initial begin
    BUS_ADDR = 8'h00;
    forever #10 BUS_ADDR <= BUS_ADDR + 8'h01;
end
```

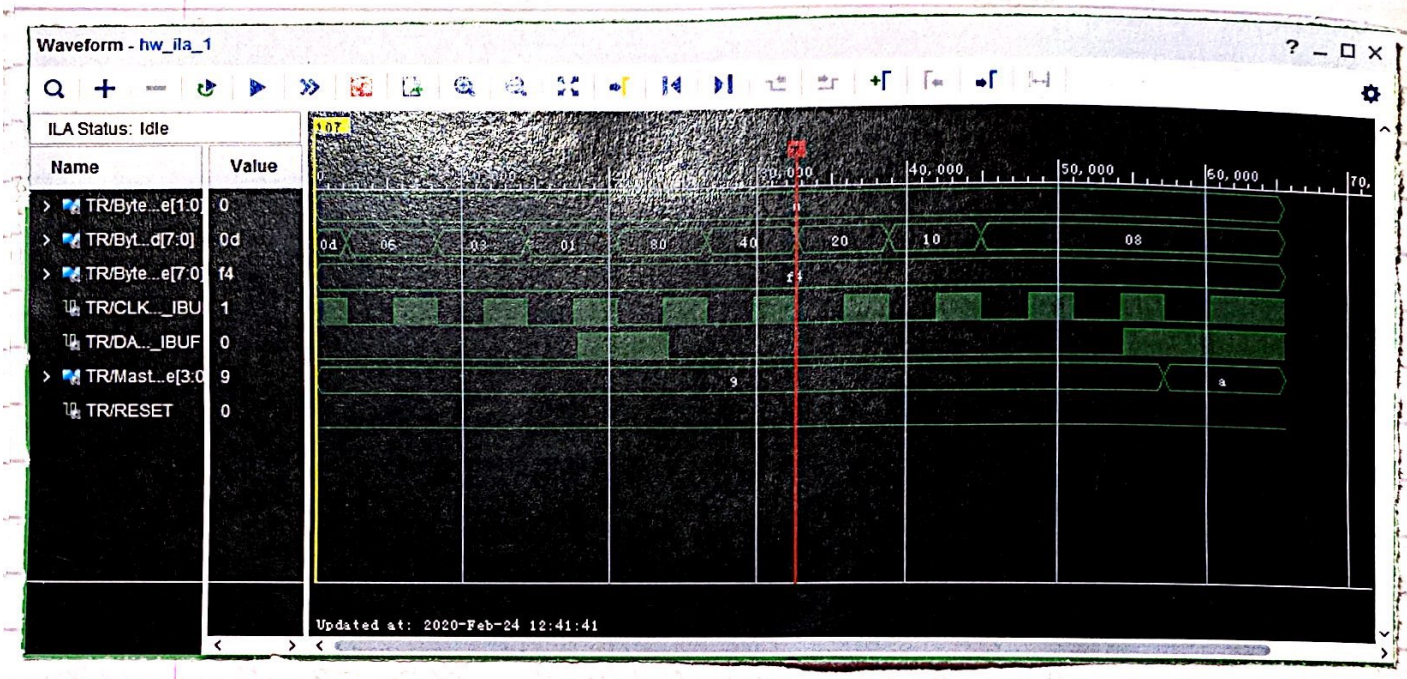


To simulate RAM, I the same as ROM, I need to increase BUS_ADDR periodicity to extract data from data memory. And to test the write/read function, I set the write_enable signal as 1 periodicity.

```
initial begin
    BUS_ADDR = 8'h00;
    forever #10 BUS_ADDR <= BUS_ADDR + 8'h01;
    BUS_WE = 0;
    forever #10 BUS_WE <= ~BUS_WE;
end
```



Finally, use ila to test the function of mouse driver. Because if use simulation to simulate the mouse driver, there will be error report of "multi-driver" and I don't know how to solve the problem.



After generate the bitstream, ~~to the~~ copy it to the FPGA. The FPGA works well.