HO CHI MINH UNIVERSITY OF TECHNOLOGY

COMPUTER SCIENCE & ENGINEERING

Mini-Project: ASM-FSM Implementing Algorithms in Hardware

Group 2:

Name	Student ID
Lê Tự Ngọc Minh	1952844
Trần Đại Tỷ	1953090

Teacher: Ngô Đức Minh

Introduction:

Algorithmic State Machine (ASM) charts are a design tool that allow the specification of digital systems in a form similar to a flow chart. An example of an ASM chart is shown in Figure 1. It represents a circuit that counts the number of bits set to 1 in an n-bit input A $(A = a_{n-1}a_{n-2}...a_1a_0)$. *The rectangular boxes in this diagram represent the states of the digital system, and actions specified inside of a state box occur on each active clock edge in this state. Transitions between states are specified by arrows. The diamonds in the ASM chart represent conditional tests, and the ovals represent actions taken only if the corresponding conditions are either true (on an arrow labeled 1) or false (on an arrow labeled 0).*

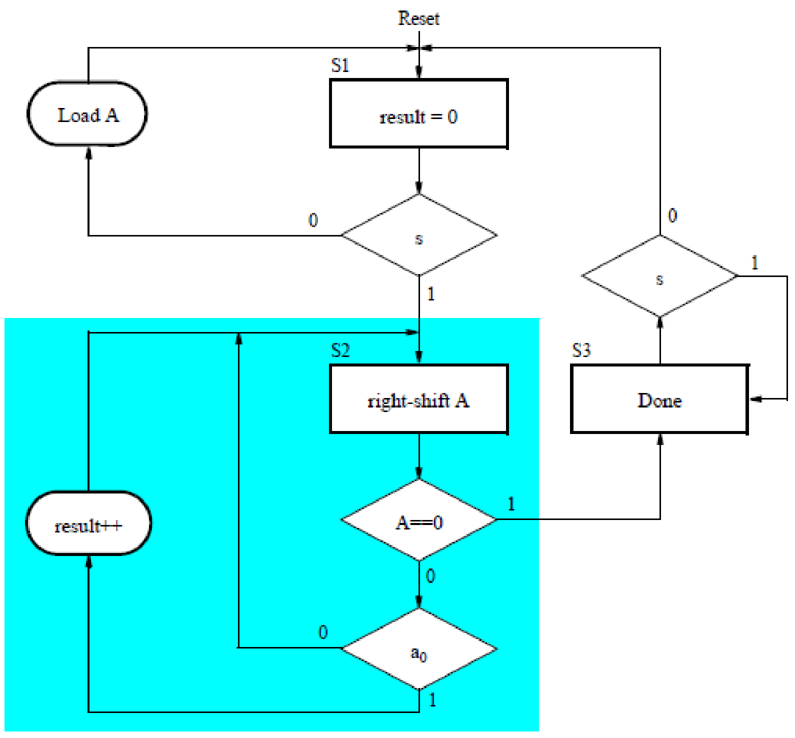


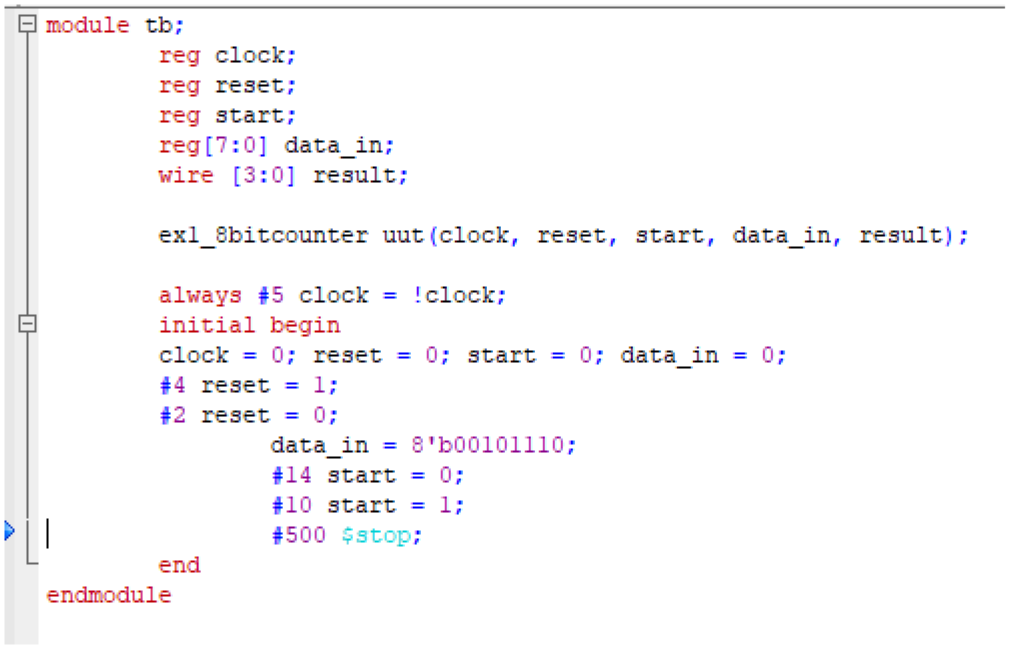
Figure 1: ASM chart for a bit counting circuit.

Part I:

Write VHDL code to implement the bit-counting circuit using the ASM chart shown in Figure 1 on a DE-series
board. Include in your VHDL code the datapath components needed, and make an FSM for the control circuit.
The inputs to your circuit should consist of an 8-bit input connected to slide switches SW_{7-0}, a synchronous reset
connected to KEY0, and a start signal (s) connected to switch SW9. Use the 50 MHz clock signal provided on the
board as the clock input for your circuit. Be sure to synchronize the s signal to the clock. Display the number of 1s
counted in the input data on the 7-segment display HEX0, and signal that the algorithm is finished by lighting up
LEDR9.

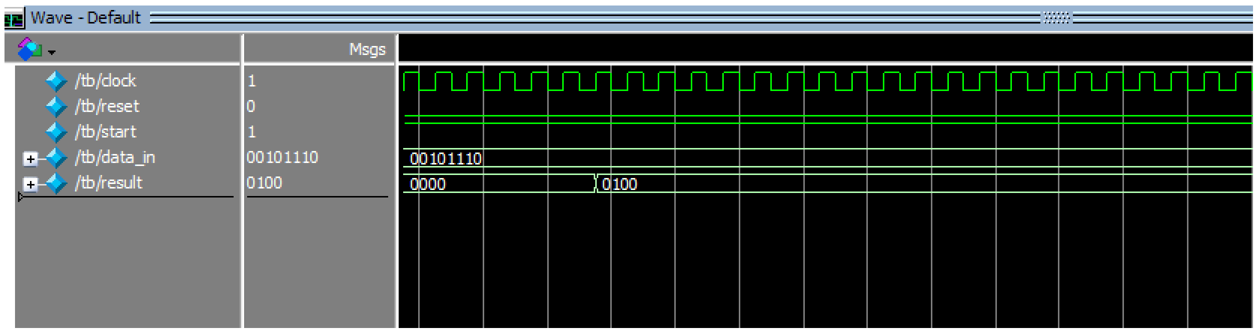
1. *Design and Implementation:*

```verilog
module ex1_8bitcounter(clock, reset, start, data_in, result);
   input clock, reset, start;
   input [7:0] data_in;
   output reg [3:0] result;
   reg[7:0] A;
   reg[3:0] count;
   reg[1:0] state, next_state;

   parameter s1 = 2'b00;
   parameter s2 = 2'b01;
   parameter s3 = 2'b10;
   always@(posedge clock)
   if(reset) state <= s1;
   else state <= next_state;

   always@(*) begin
   next_state = state;
   case (state)
   s1: if(start) next_state = s2;
   s2: if(A == 0) next_state = s3;
   s3: if(!start) next_state = s1;
   default: next_state = state;
   endcase
   end



   always@(posedge clock) begin
   case(state)
      s1: begin result <= 0; A<= data_in ; count = 0; end
      s2: begin A <= {1'b0,A[7:1]}; count <= count + A[0]; end
      s3: result <= count;
      default: begin A <= 0; count <=0 ; result <= 0; end
   endcase
   end
endmodule
```

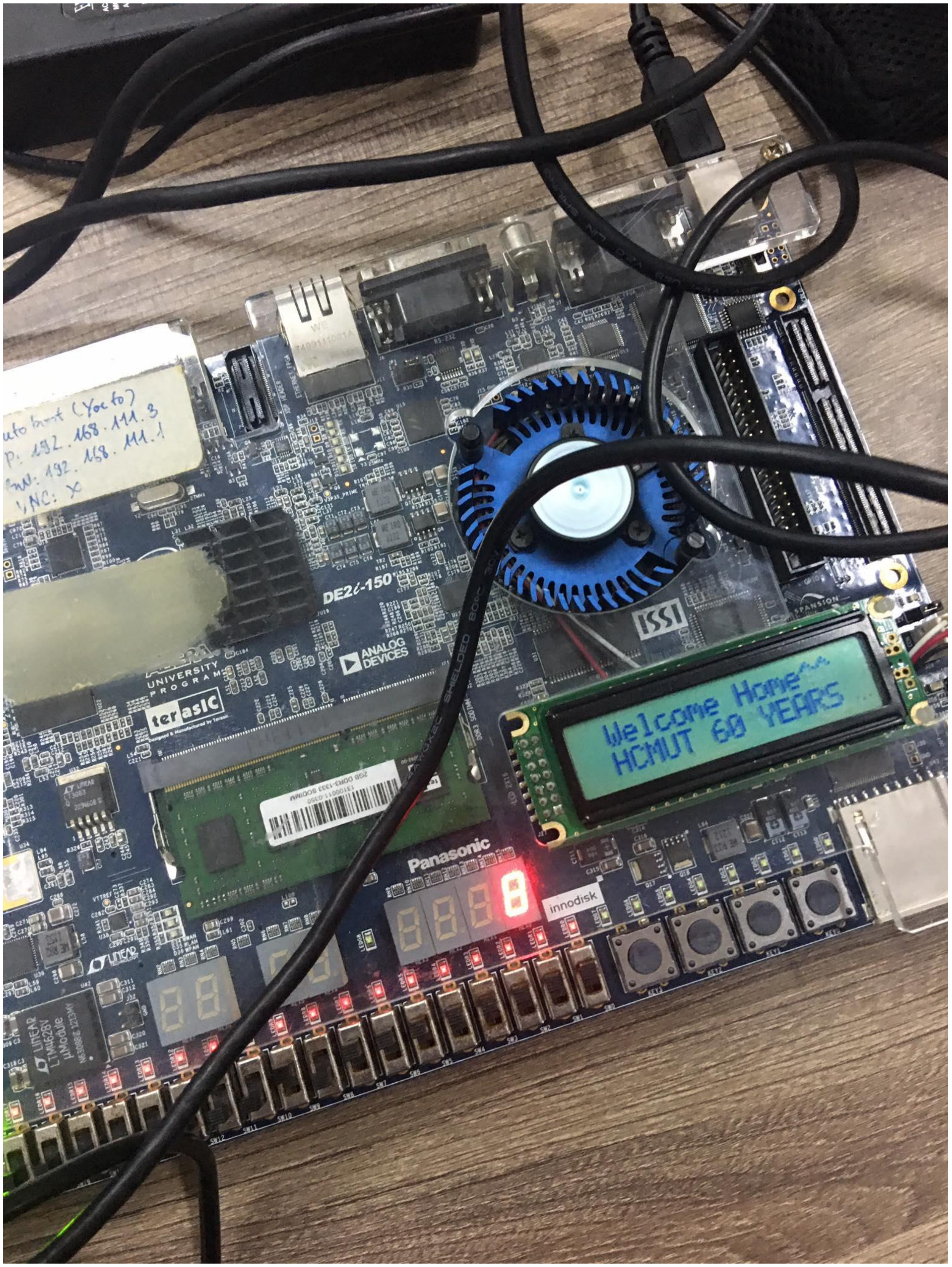


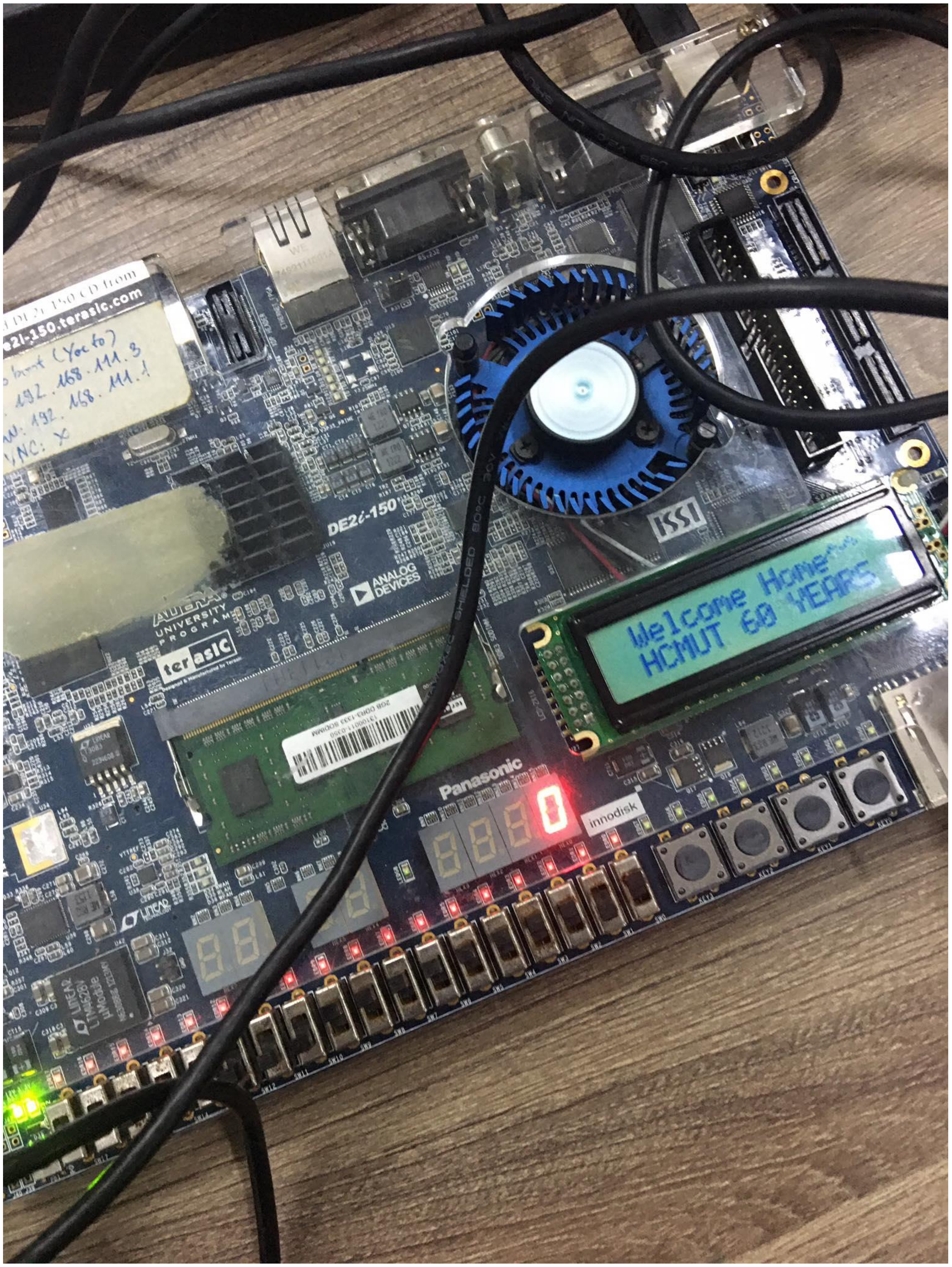
```
module tb;
        reg clock;
        reg reset;
        reg start;
        reg[7:0] data_in;
        wire [3:0] result;

        exl_8bitcounter uut(clock, reset, start, data_in, result);

        always #5 clock = !clock;
        initial begin
        clock = 0; reset = 0; start = 0; data_in = 0;
        #4 reset = 1;
        #2 reset = 0;
                data_in = 8'b00101110;
                #14 start = 0;
                #10 start = 1;
                #500 $stop;
        end
endmodule
```

2. *Experiment:*

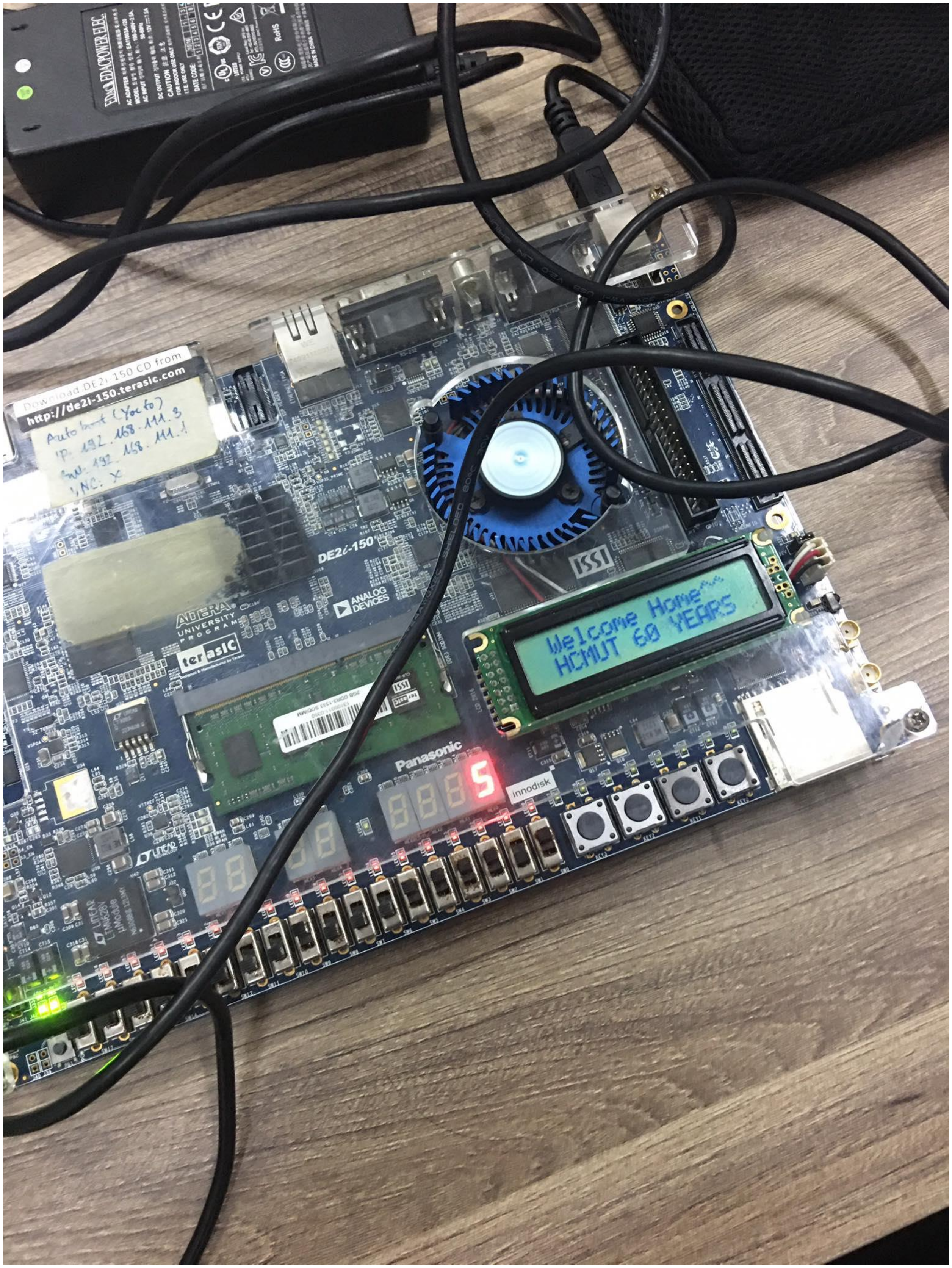


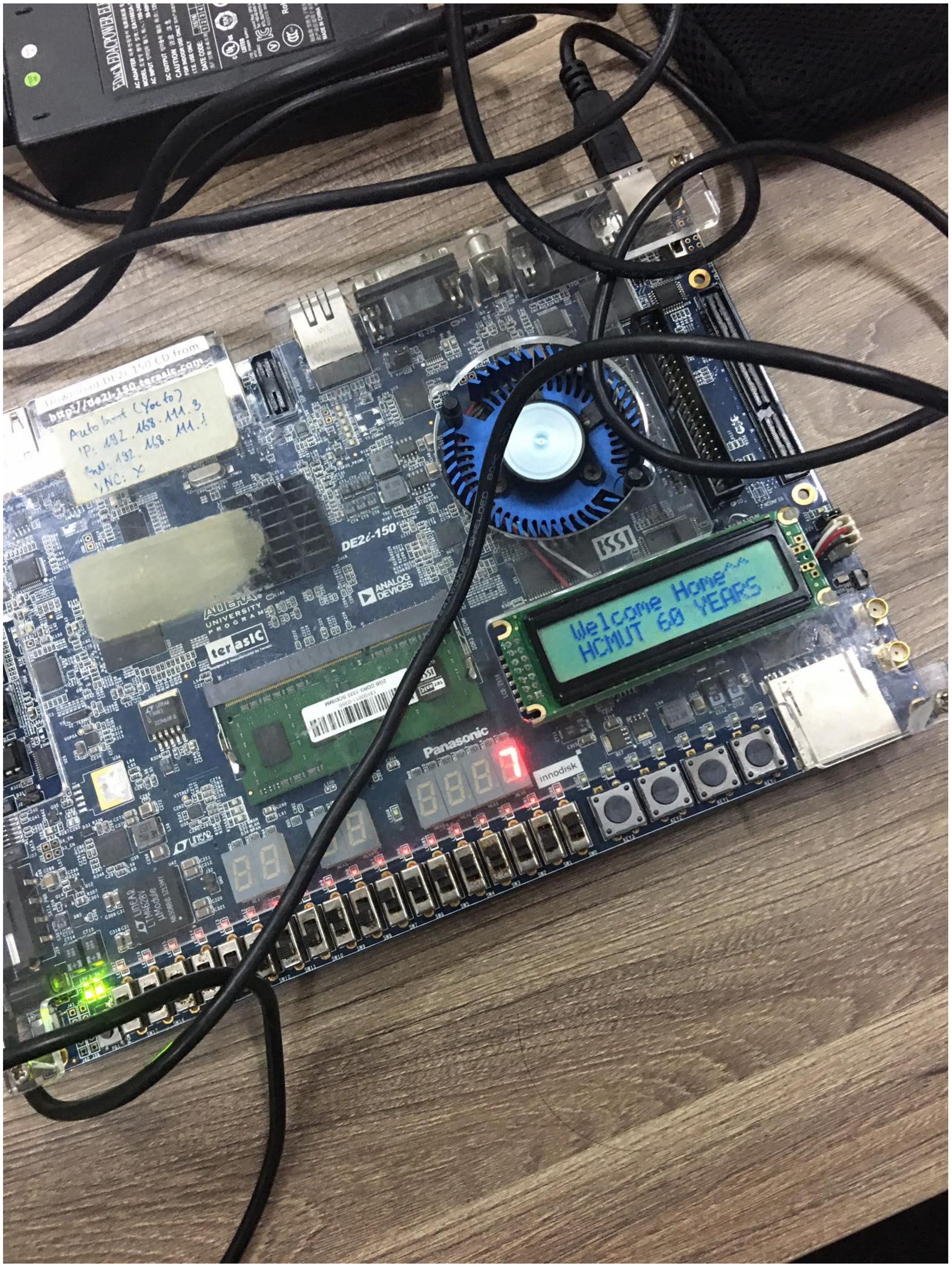
```
Wave - Default
                Msgs
/tb/clock       1
/tb/reset       0
/tb/start       1
/tb/data_in     00101110       00101110
/tb/result      0100           0000            0100
```


uto boot (Yocto)
P: 192.168.111.3
GW: 192.168.111.1
VNC: x
DE2i-150
ANALOG DEVICES
UNIVERSITY PROGRAM
terasIC
ISSI
Welcome Home^^
HCMUT 60 YEARS
Panasonic
innodisk



Welcome Home
HCMUT 60 YEARS
DE2i-150
de2i-150.terasic.com
terasIC
ANALOG DEVICES
ALTERA UNIVERSITY PROGRAM
Panasonic
innodisk
ISSI


Download DE2i-150 CD from
http://de2i-150.terasic.com
Auto boot (Yocto)
IP: 192.168.111.3
Gw: 192.168.111.1
VNC: X
DE2i-150
ALTERA UNIVERSITY PROGRAM
terasIC
ANALOG DEVICES
ISSI
Panasonic
innodisk
Welcome Home
HCMUT 60 YEARS



3. *Conclusion and future work:*

In this ASM chart, state S1 is the initial state. In this state the result is initialized to 0, and data is loaded into
a register A, until a start signal, s, is asserted. The ASM chart then transitions to state S2, where it increments
the result to count the number of 1's in register A. Since state S2 specifies a shifting operation, then A should be
implemented as a shift register. Also, since the result is incremented, then this variable should be implemented as
a counter. When register A contains 0 the ASM chart transitions to state S3, where it sets an output Done = 1 and
waits for the signal s to be deasserted.
A key distinction between ASM charts and flow charts is a concept known as implied timing. The implied timing
specifies that all actions associated with a given state take place only when the system is in that state when an
active clock edge occurs. For example, when the system is in state S1 and the start signal s becomes 1, then the
next active clock edge performs the following actions: initializes result to 0, and transitions to state S2. The action
right-shift A does not happen yet, because the system is not yet in state S2. For each active clock cycle in state
S2, the actions highlighted in Figure 1 take place, as follows: increment result if bit a0 = 1, change to state S3 if
A = 0 (or else remain in state S2), and shift A to the right.
The implementation of the bit counting circuit includes the counter to store the result and the shift register A, as
well as a finite state machine. The FSM is often referred to as the control circuit, and the other components as the
datapath circuit.

Part II:

We wish to implement a binary search algorithm, which searches through an array to locate an 8-bit value A specified via switches SW_{7-0}. A block diagram for the circuit is shown in Figure 2.

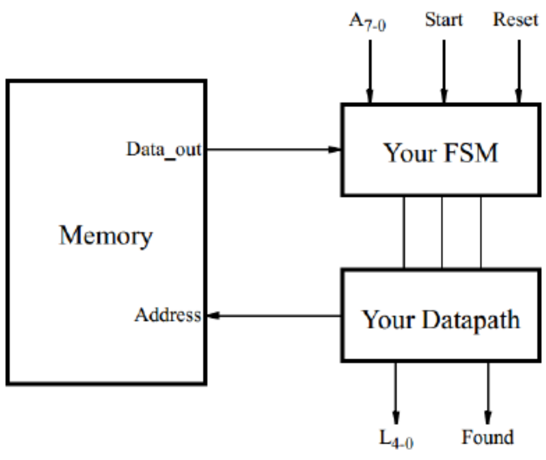


Figure 2: A block diagram for a circuit that performs a binary search.

The binary search algorithm works on a sorted array. Rather than comparing each value in the array to the one being sought, we first look at the middle element and compare the sought value to the middle element. If the middle element has a greater value, then we know that the element we seek must be in the first half of the array. Otherwise, the value we seek must be in the other half of the array. By applying this approach recursively, we can locate the sought element in only a few steps.

In this circuit, the array is stored in an on-chip memory instantiated using MegaWizard Plug-In Manager. To create the approriate memory block, use the the RAM: 1-PORT module from the MegaWizard Plug-In Manager as shown in Figure 3.

In the window in Figure 3, specify the Verilog HDL output file to be memory_block.v. When creating the memory block, you should also specify a memory initilization file to be my_array.mif, so that the memory contents can be set to contain an ordered array of numbers.

The circuit should produce a 5-bit value *L* that specifies the address in the memory where the number *A* is located. In addition, a signal *Found* should be set high to indicate that the number A was found in the memory, and set low otherwise.

Perform the following steps:

1. Create an ASM chart for the binary search algorithm. Keep in mind that it takes two clock cycles for the data to be read from memory. You may assume that the array has a fixed size of 32 elements.

2. Implement the FSM and the datapath for your circuit.

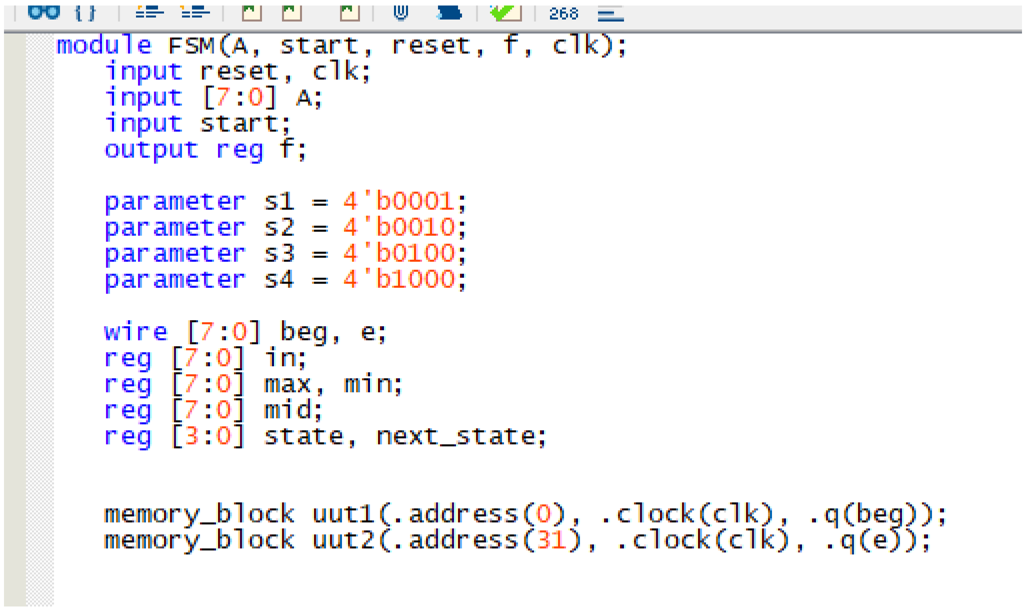
3. Connect your FSM and datapath to the memory block as shown in Figure 2.

4. Include in your project the necessary pin assignments to implement your circuit on the DE2-series board. Use switch SW_8 to drive the processor's *Run* input, use SW_7 to SW_0 to specify the value to be searched, use KEY_0 for *Resetn*, and use the board's 50 MHz clock signal as the *Clock* input. Connect $LEDR_4$ to $LEDR_0$ to show the address in memory of the number A, and $LEDG_0$ for the *Found* signal.

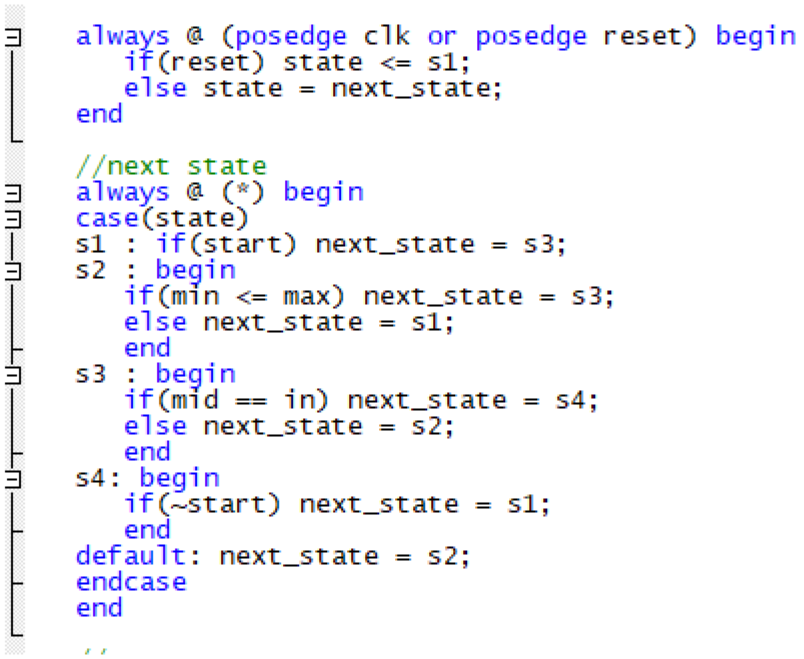
5. Create a file called my_array.mif and fill it with an ordered set of 32 eight-bit integer numbers. You can do this in Quartus II by choosing File > New... from the main menu and selecting Memory Initialization File. This will open a memory file editor, where the contents of the memory may be specified. After this file is created and/or modified, your design needs to be fully recompiled, and downloaded onto the DE2-series board for the changes to take effect.

1. *Design and Implementation:*

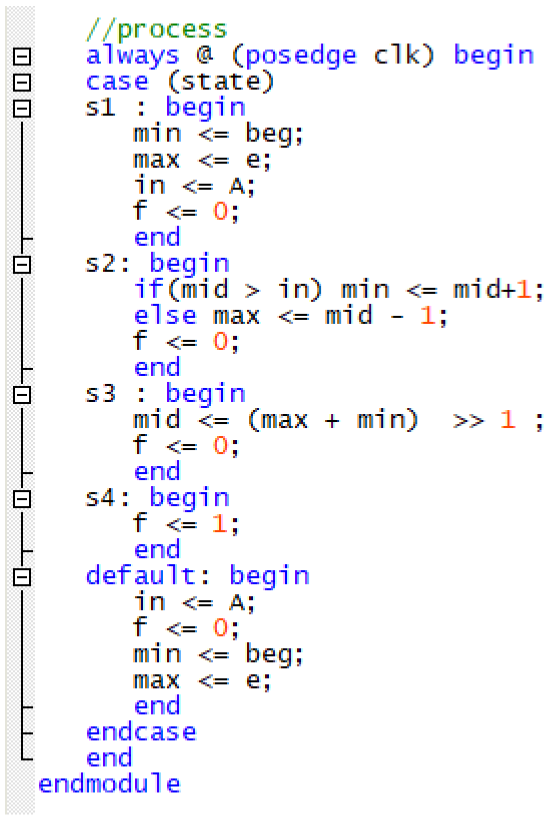
FSM:



```
module FSM(A, start, reset, f, clk);
   input reset, clk;
   input [7:0] A;
   input start;
   output reg f;

   parameter s1 = 4'b0001;
   parameter s2 = 4'b0010;
   parameter s3 = 4'b0100;
   parameter s4 = 4'b1000;

   wire [7:0] beg, e;
   reg [7:0] in;
   reg [7:0] max, min;
   reg [7:0] mid;
   reg [3:0] state, next_state;


   memory_block uut1(.address(0), .clock(clk), .q(beg));
   memory_block uut2(.address(31), .clock(clk), .q(e));
```

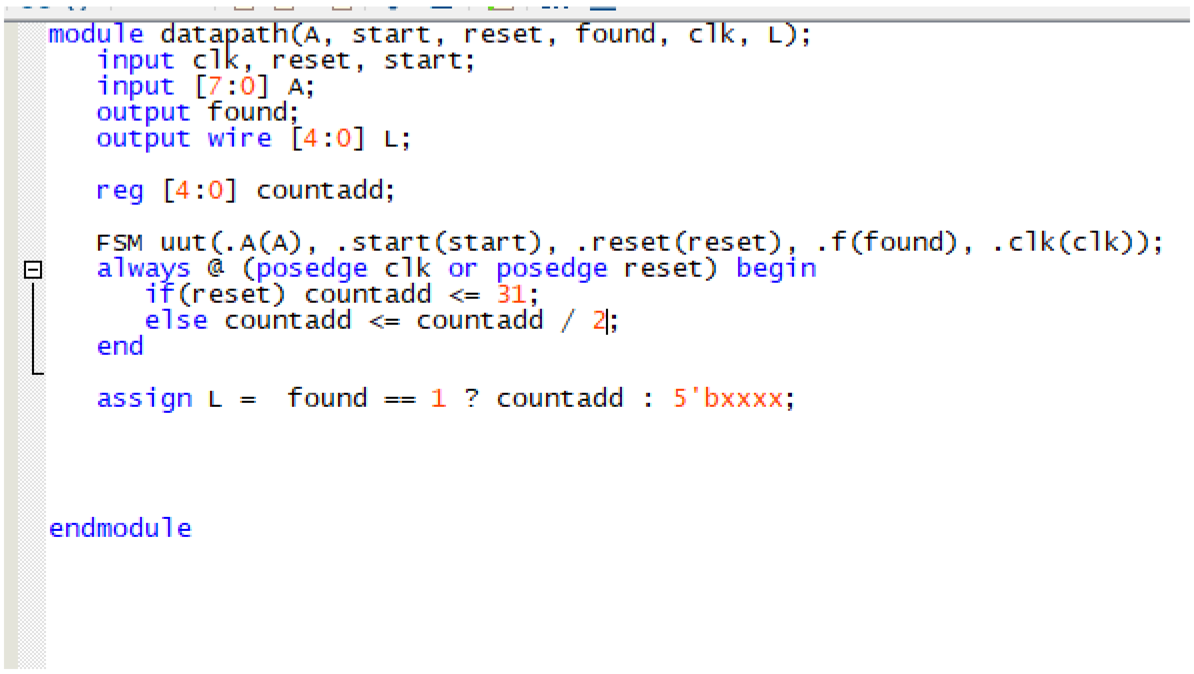


```
always @ (posedge clk or posedge reset) begin
   if(reset) state <= s1;
   else state = next_state;
end

//next state
always @ (*) begin
case(state)
s1 : if(start) next_state = s3;
s2 : begin
   if(min <= max) next_state = s3;
   else next_state = s1;
   end
s3 : begin
   if(mid == in) next_state = s4;
   else next_state = s2;
   end
s4: begin
   if(~start) next_state = s1;
   end
default: next_state = s2;
endcase
end
```



```
 //process
 always @ (posedge clk) begin
 case (state)
 s1 : begin
    min <= beg;
    max <= e;
    in <= A;
    f <= 0;
    end
 s2: begin
    if(mid > in) min <= mid+1;
    else max <= mid - 1;
    f <= 0;
    end
 s3 : begin
    mid <= (max + min)  >> 1 ;
    f <= 0;
    end
 s4: begin
    f <= 1;
    end
 default: begin
    in <= A;
    f <= 0;
    min <= beg;
    max <= e;
    end
 endcase
 end
endmodule
```

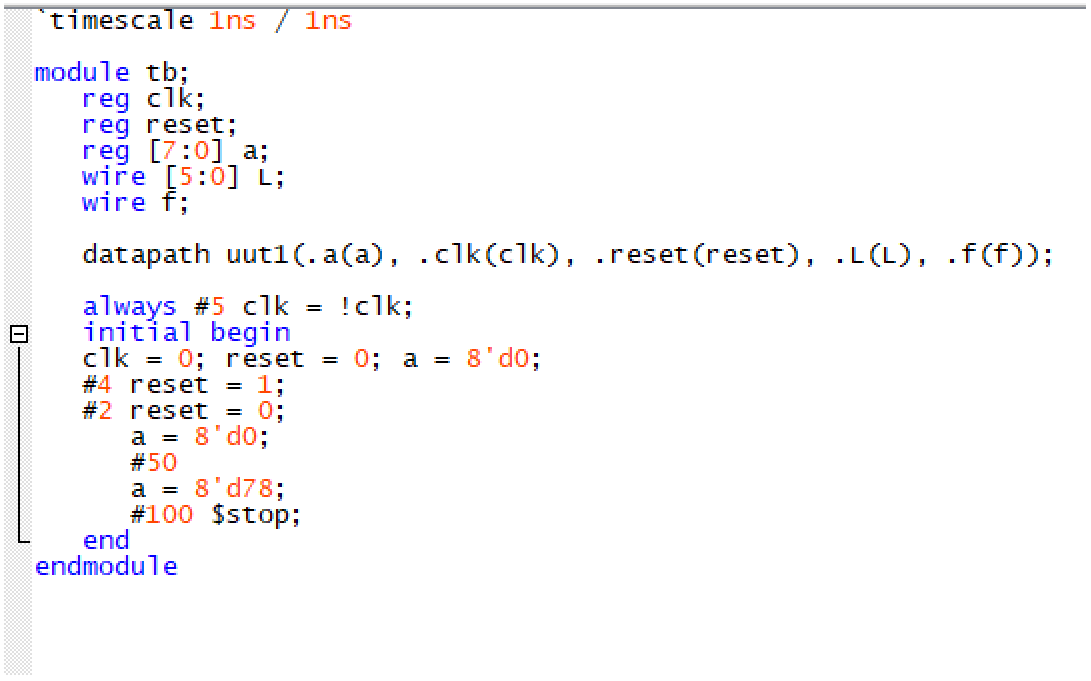
Datapath:



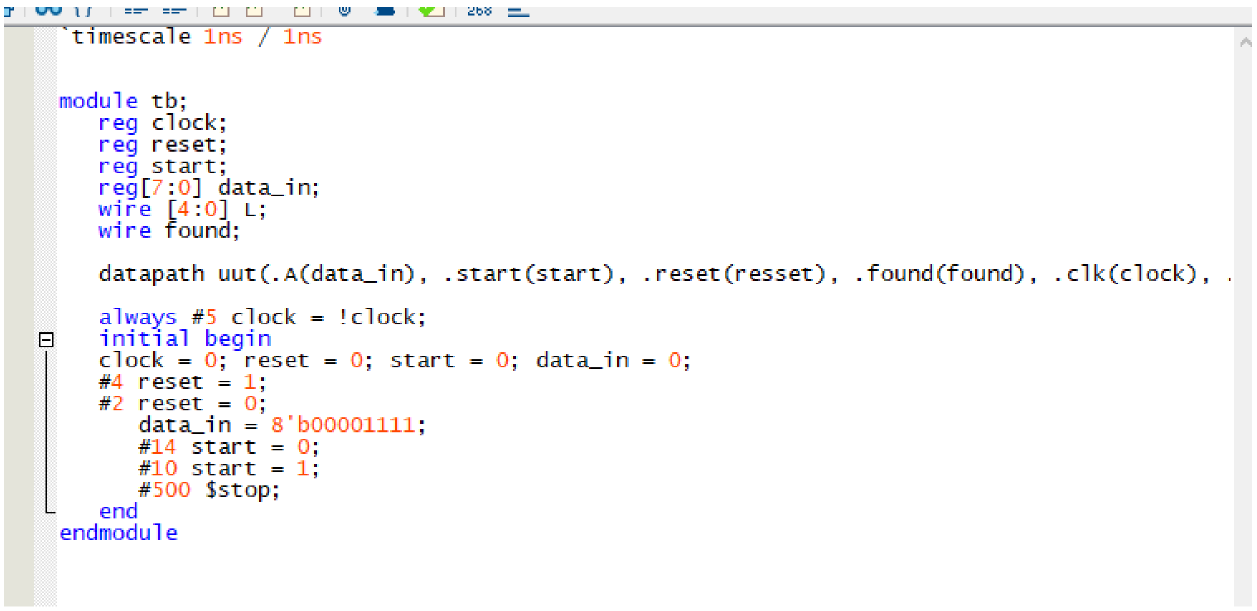
```
module datapath(A, start, reset, found, clk, L);
   input clk, reset, start;
   input [7:0] A;
   output found;
   output wire [4:0] L;

   reg [4:0] countadd;

   FSM uut(.A(A), .start(start), .reset(reset), .f(found), .clk(clk));
   always @ (posedge clk or posedge reset) begin
      if(reset) countadd <= 31;
      else countadd <= countadd / 2;
   end

   assign L =  found == 1 ? countadd : 5'bxxxx;



endmodule
```

2. *Exeperiment:*

Testbench:

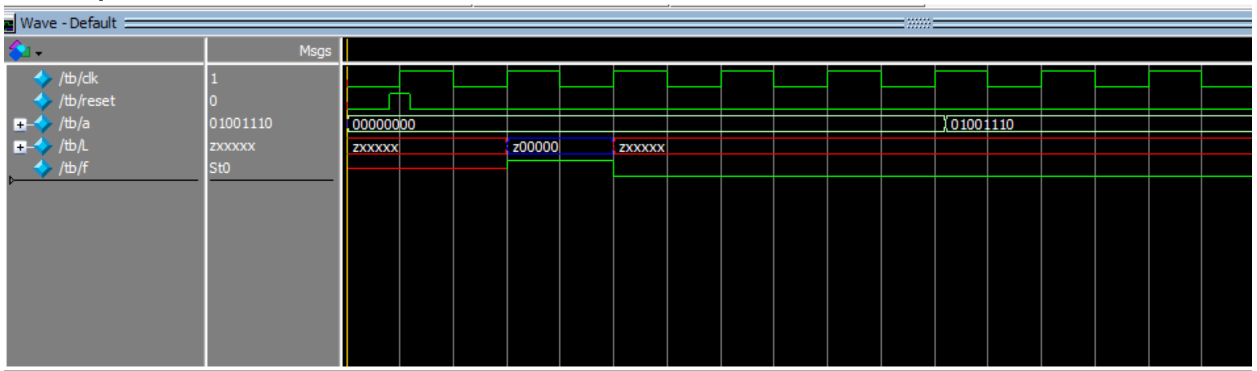


```
`timescale 1ns / 1ns

module tb;
   reg clk;
   reg reset;
   reg [7:0] a;
   wire [5:0] L;
   wire f;

   datapath uut1(.a(a), .clk(clk), .reset(reset), .L(L), .f(f));

   always #5 clk = !clk;
   initial begin
   clk = 0; reset = 0; a = 8'd0;
   #4 reset = 1;
   #2 reset = 0;
      a = 8'd0;
      #50
      a = 8'd78;
      #100 $stop;
   end
endmodule
```

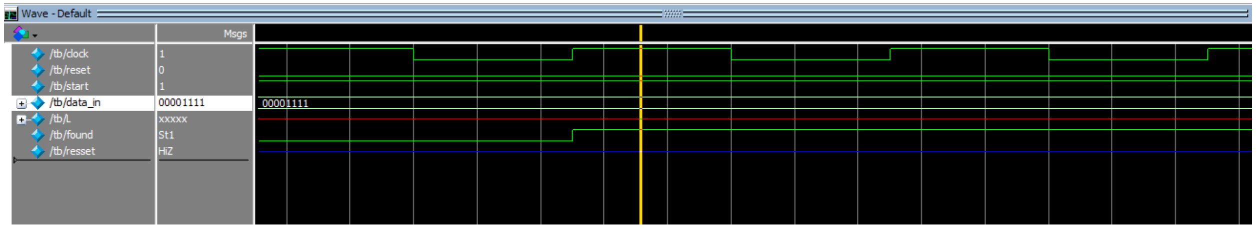


```
`timescale 1ns / 1ns


module tb;
   reg clock;
   reg reset;
   reg start;
   reg[7:0] data_in;
   wire [4:0] L;
   wire found;

   datapath uut(.A(data_in), .start(start), .reset(resset), .found(found), .clk(clock), .

   always #5 clock = !clock;
   initial begin
   clock = 0; reset = 0; start = 0; data_in = 0;
   #4 reset = 1;
   #2 reset = 0;
      data_in = 8'b00001111;
      #14 start = 0;
      #10 start = 1;
      #500 $stop;
   end
endmodule
```

Waveform:



```
Wave - Default
            Msgs
/tb/clk     1
/tb/reset   0
/tb/a       01001110
/tb/L       zxxxxx
/tb/f       St0
```



```
Wave - Default
               Msgs
/tb/clock      1
/tb/reset      0
/tb/start      1
/tb/data_in    00001111
/tb/L          xxxxx
/tb/found      St1
/tb/resset     HiZ
```

3. *Conclusion and Future work:*

Binary Search is applied on the sorted array or list of large size. It's time complexity of step makes it very fast as compared to other sorting algorithms. The only limitation is that the array or list of elements must be sorted for the binary search algorithm to work on it.