Team members: *Burt Zhao, Shangrong Li*
EE 371 Lab 4

Abstract
This lab report is based on the material of the lab 4 “Implementing Algorithms in Hardware”. We will be discussing the background, procedure, analysis, result of this lab following by a brief conclusion through this report. Different to the previous labs, this lab was completed by a group of two students. We will also include our strategy and result of working in team in our result and analysis portion of the report.

Introduction
The objective of this lab is to utilize the ASM to build a bit counter and binary search function in hardware. ASMD is important because it separates a module into two parts, the control path and the data path. This allowed us to write code separately in a efficient manner while keeping us on the same approach to the problem. Furthermore, coding the control path and data path separately makes the modules more organized and easier to our debugging processes.

Procedure
In task 1, we were given the ASMD of the binary counter. We would start by discussing what inputs and outputs are involved between the control and datapath. Then we would either picked the control module or the data path module to write. After writing and testing the two modules individually, we would write a wrapper module together to instantiate the two modules using the correct input and output port names. Task 2 takes the design process a step further, we would have to design our own ASMD chart based on the lab spec. Since the ASMD represents the core logic of the modules, we sat down and drew the ASMD as a team very slowly. We would discuss until we both agree and understand the behavior of the program in a each state before we can move on to the next thing. Manual simulation was performed for the looping part of the binary search to make sure every cases were covered. Then we would split the modules into two parts, controls and the rest of the modules. The control module controls signal traffics and sends out determines when to perform a certain operation by sending out the signals to the data path module. The datapath would be the module that receive the signal and actually do the work and calculations according to the received signals. The person who writes the datapath is also responsible for writing the display and ram modules, because these are the data that only goes and be manipulated in the data path module. About the display and ram modules, they are either taken straight from previous labs or we would have some minor changes in order match the needs for this lab. The ram module should have a size of 32x5 bits. We would also have a top module that instantiate all the sub-modules.

Analysis & Results
In this lab, we mainly wrote testbenches and use ModelSim to debug any unexpected errors. At the beginning, all the test benches worked individually. Most of the codes are bug-free after one or two trials. But when we put everything together, we found that the reset are not working correctly when start is turn on. Supposedly, it should output the correct value of address of the found value. However, if we keep press on the reset, the value toggles between the correct value and 0F. After digging back into the ASMD and looking at the simulation, we realized it program somewhat jump to the found state after the first binary search. After putting the test bench into different circumstances, we discovered that we would need

more clock cycles in some states compensate the timing difference between the states. To do this, we created states that do nothing and used them as buffer. The result after this fix was satisfying and the reset began to work as expected. Another debugging tools we used is SingalTap. We set the sensing signal to be the found signal, and we would be able to catch the output data and address when found is 1.

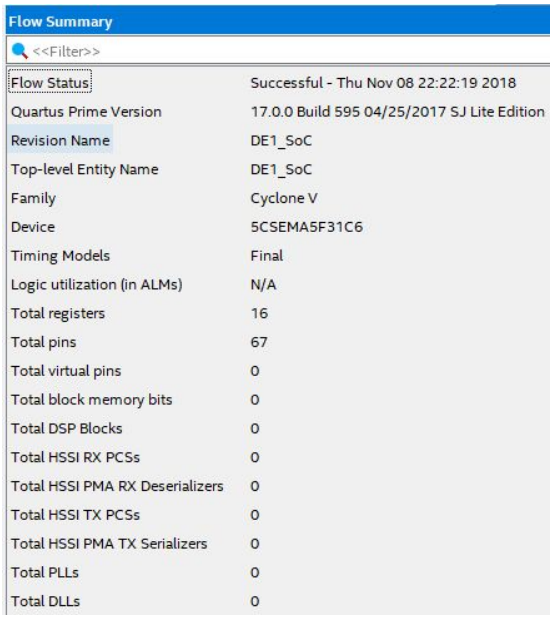
Conclusion

Overall, this lab perform as a great practice for the material given in lecture. It also provided a complete hand-on experience about ASMD design and implementation. Furthermore, the method of ASMD expanded my knowledge about coding in a team. At the same time, it's the first coding lab that I've ever done in my life in any area of study and it was a far better experience than what I expected before doing this lab.

Time - approximately 20 hours

Appendix

Flow summary:



Flow Summary

<<Filter>>

Flow Status	Successful - Thu Nov 08 22:22:19 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	16
Total pins	67
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 1: Task 1 flow summary

Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Nov 08 22:23:31 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	684
Total pins	67
Total virtual pins	0
Total block memory bits	3,328
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 2: Task 2 flow summary

ASMD chart:

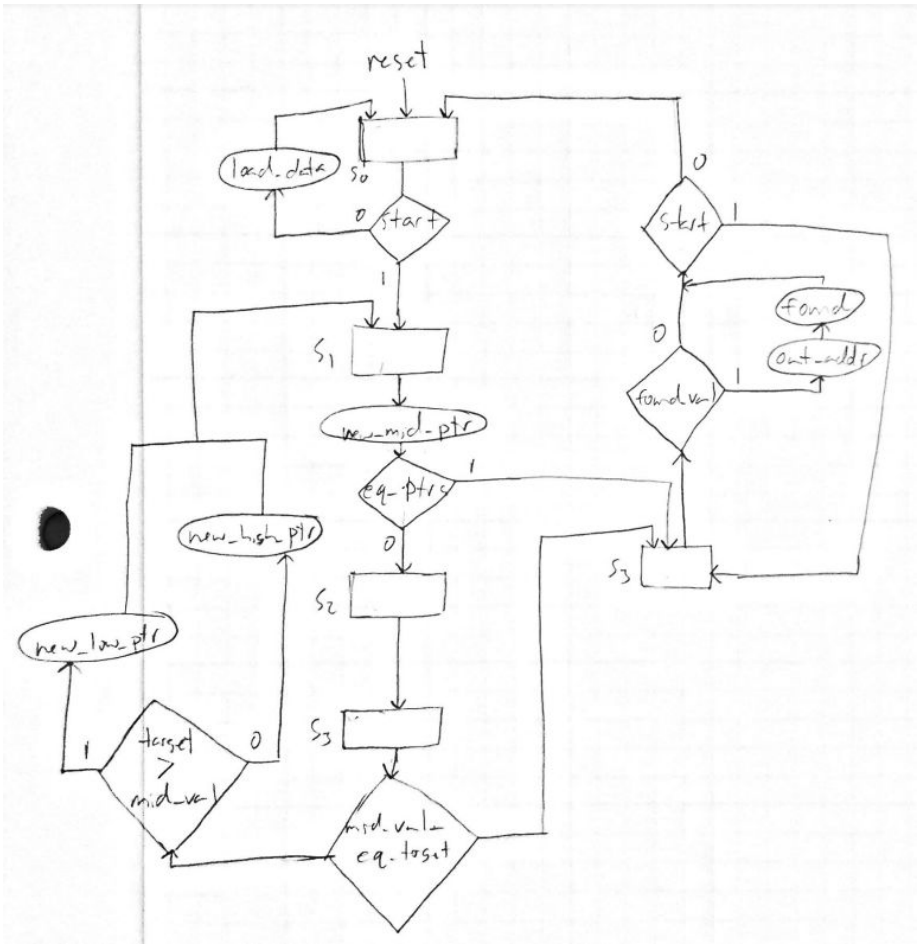


Figure 3: Task 2 ASMD chart

ModelSim:

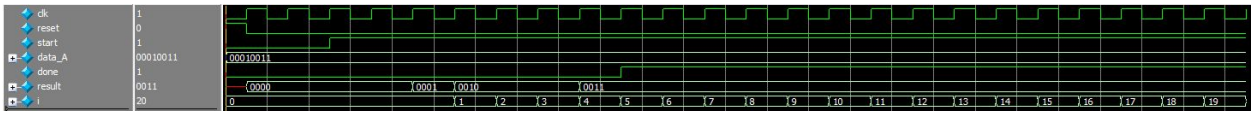


Figure 4: Task 1 ModelSim

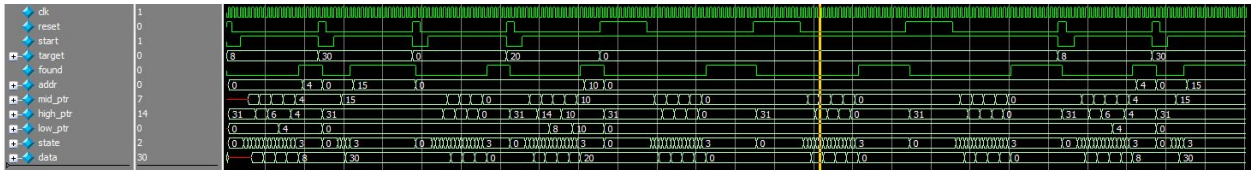


Figure 5: Task 2 ModelSim

SignalTap:

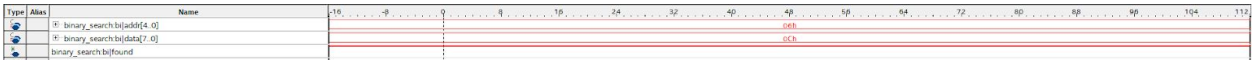


Figure 6: Task 2 signal tap

Code:

```
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
    input logic CLOCK_50; // 50MHz clock.
    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
    output logic [9:0] LEDR;
    input logic [3:0] KEY; // True when not pressed, False when pressed
    input logic [9:0] SW;


    // Top level inputs
    logic reset, start, sync;
    logic [7:0] data_A;

    // Top level outputs
    logic [3:0] result;

    // Key and switch assignments
    assign reset = ~KEY[0];
    assign data_A = {SW[7], SW[6], SW[5], SW[4], SW[3], SW[2], SW[1], SW[0]};

    // sync start
    flipflop U1(.D(SW[8]), .reset, .clk(CLOCK_50), .Q(sync));
    flipflop U2(.D(sync), .reset, .clk(CLOCK_50), .Q(start));

    // Bit counter
    count_bits c1(.clk(CLOCK_50), .reset, .start, .data_A, .done(LEDR[9]), .result);

    display d1(.in(result), .hex(HEX0));
endmodule
```

Code 1: Task 1 DE1_SoC.sv

```
module count_bits(
   input logic clk, reset, start,
   input logic [7:0] data_A,
   output logic done,
   output logic [3:0] result
   );

   // Signals from control to datapath
   logic load_data, inc_counter, r_shift;

   // Feedback from datapath to control
   logic A_eq_0, a0;

   // Control module
   t1_control cont(.clk, .reset, .start, .A_eq_0, .a0, .done, .load_data, .inc_counter, .r_shift);

   // Datapath module
   t1_datapath data(.clk, .reset, .data_A, .load_data, .inc_counter, .r_shift, .A_eq_0, .a0, .result);

endmodule

module count_bits_tb();
   logic clk, reset, start;
   logic [7:0] data_A;
   logic done;
   logic [3:0] result;

   count_bits dut(clk, reset, start, data_A, done, result);

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   integer i = 0;
   initial begin
      reset <= 1; start <= 0; data_A <= 19; @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      start <= 1; @(posedge clk); @(posedge clk);
      for (i = 0; i < 20; i++)
         @(posedge clk);
      $stop;
   end
endmodule
```

Code 2: Task 1 count_bits.sv

```
module t1_control(
   input logic clk, reset, start,
   input logic A_eq_0, a0,
   output logic done,
   output logic load_data, inc_counter, r_shift
   );

   logic [1:0] ps, ns;
   parameter s1 = 2'b00;
   parameter s2 = 2'b01;
   parameter s3 = 2'b10;

   // Next state logic
   always @(ps, start, A_eq_0)
      case (ps)
         s1:   if (start)
                  ns = s2;
               else
                  ns = s1;
         s2:   if (A_eq_0)
                  ns = s3;
               else
                  ns = s2;
         s3:   if (start)
                  ns = s3;
               else
                  ns = s1;
      endcase

   // Output logic
   always @(ps, start, A_eq_0, a0) begin
      load_data = 0;
      inc_counter = 0;
      r_shift = 0;
      done = 0;

      case (ps)
         s1:   if (~start)
                  load_data = 1;
         s2:   if (~A_eq_0) begin
                  if (a0)
                     inc_counter = 1;
                  r_shift = 1;
               end
         s3: done = 1;
      endcase
   end

   // DFFs
   always_ff @(posedge clk)
      if (reset)
         ps <= s1;
      else
         ps <= ns;
endmodule
```

Code 3: Task 1 t1_control.sv

```
module t1_control_tb();
   logic clk, reset, start;
   logic A_eq_0, a0;
   logic done;
   logic load_data, inc_counter, r_shift;

   t1_control dut(clk, reset, start, A_eq_0, a0, done, load_data, inc_counter, r_shift);

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      reset <= 1; start <= 0; A_eq_0 <= 0; a0 <= 0; @(posedge clk); @(posedge clk);
      reset <= 0; start <= 1; @(posedge clk); @(posedge clk);
      a0 <= 1; @(posedge clk); @(posedge clk);
      A_eq_0 <= 1; @(posedge clk); @(posedge clk);
      start <= 1; @(posedge clk); @(posedge clk);
      $stop;
   end
endmodule
```

Code 4: Task 1 t1_control_tb.sv

```
module t1_datapath(
   input logic clk, reset,
   input logic [7:0] data_A,
   input logic load_data, inc_counter, r_shift,
   output logic A_eq_0, a0,
   output logic [3:0] result
   );

   logic [7:0] A;

   always_ff @(posedge clk) begin
      if (reset) begin
         A <= 8'b0;
         result <= 4'b0;
      end
      if (load_data) begin
         A <= data_A;
         result <= 4'b0;
      end
      if (inc_counter)
         result <= result + 1;
      if (r_shift)
         A <= A >> 1;
   end

   // Feedback output to control
   assign A_eq_0 = (A == 8'b0);
   assign a0 = (A[0]);
endmodule

module t1_datapath_tb();
   logic clk, reset;
   logic [7:0] data_A;
   logic load_data, inc_counter, r_shift;
   logic A_eq_0, a0;
   logic [3:0] result;

   t1_datapath dut(clk, reset, data_A, load_data, inc_counter, r_shift, A_eq_0, a0, result);

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      reset <= 1; data_A <= 32; load_data <= 0; inc_counter <= 0; r_shift <= 0; @(posedge clk);
      reset <= 0; @(posedge clk);
      load_data <= 1; @(posedge clk); @(posedge clk);
      load_data <= 0; @(posedge clk); @(posedge clk);
      inc_counter <= 1; r_shift <= 1; @(posedge clk); @(posedge clk);
      inc_counter <= 0; r_shift <= 0; @(posedge clk); @(posedge clk);
      inc_counter <= 1; r_shift <= 1; @(posedge clk); @(posedge clk);
      inc_counter <= 0; r_shift <= 0; @(posedge clk); @(posedge clk);
      inc_counter <= 1; r_shift <= 1; @(posedge clk); @(posedge clk);
      inc_counter <= 0; r_shift <= 0; @(posedge clk); @(posedge clk);
      inc_counter <= 1; r_shift <= 1; @(posedge clk); @(posedge clk);
      inc_counter <= 0; r_shift <= 0; @(posedge clk); @(posedge clk);
      $stop;
   end
endmodule
```

Code 5: Task 1 t1_datapath.sv

```
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
    input logic CLOCK_50; // 50MHz clock.
    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
    output logic [9:0] LEDR;
    input logic [3:0] KEY; // True when not pressed, False when pressed
    input logic [9:0] SW;

    logic reset, start, sync;
    logic [7:0] target;
    logic [4:0] addr, mid_ptr, high_ptr, low_ptr;
    logic [2:0] state;
    logic [7:0] data;
    logic [7:0] hex1, hex0;

    assign reset = ~KEY[0];
    assign target = {SW[7], SW[6], SW[5], SW[4], SW[3], SW[2], SW[1], SW[0]};


    // sync start
    flipflop U1(.D(SW[8]), .reset, .clk(CLOCK_50), .Q(sync));
    flipflop U2(.D(sync), .reset, .clk(CLOCK_50), .Q(start));

    binary_search bi(
    .clk(CLOCK_50), .reset, .start,
    .target,
    .found(LEDR[9]),
    .addr, .mid_ptr, .high_ptr, .low_ptr,
    .state,
    .data
    );

    // Displays
    display d0(.in(addr[3:0]), .hex(HEX0));
    display d1(.in({3'b000, addr[4]}), .hex(HEX1));
    assign HEX5 = 7'b1111111;
    assign HEX4 = 7'b1111111;
    assign HEX3 = 7'b1111111;
    assign HEX2 = 7'b1111111;
endmodule
```

Code 6: Task 2 DE1_SoC.sv

```
module binary_search(
   input logic clk, reset, start,
   input logic [7:0] target,
   output logic found, // found also output of system
   output logic [4:0] addr, mid_ptr, high_ptr, low_ptr,
   output logic [2:0] state,
   output logic [7:0] data
   );

   // data to cont
   logic mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val;

   // cont to data
   logic load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr;


   t2_control control(
      .clk, .reset,
      .start, .mid_val_eq_target, .eq_ptrs, .target_gt_mid_val, .target_lt_mid_val, .found_val,
      .load_data, .new_high_ptr, .new_mid_ptr, .new_low_ptr, .out_addr, .found,
      .state
   );

   t2_datapath datapath(
      .clk, .reset,
      .target, .data,
      .load_data, .new_high_ptr, .new_mid_ptr, .new_low_ptr, .out_addr,
      .mid_val_eq_target, .eq_ptrs, .target_gt_mid_val, .target_lt_mid_val, .found_val,
      .addr, .mid_ptr, .high_ptr, .low_ptr
   );

   ram32x8 ram(.address(mid_ptr), .clock(clk), .din(8'bx), .wren(1'b0), .dout(data));
endmodule
```

Code 7: Task 2 binary_search.sv

```
`timescale 1 ps / 1 ps
module binary_search_tb();
   logic clk, reset, start;
   logic [7:0] target;
   logic found; // found also output of system
   logic [4:0] addr, mid_ptr, high_ptr, low_ptr;
   logic [2:0] state;
   logic [7:0] data;

   binary_search dut(
   clk, reset, start,
   target,
   found, // found also output of system
   addr, mid_ptr, high_ptr, low_ptr,
   state,
   data
   );

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   integer i;
   initial begin
      reset <= 1; start <= 0; target <= 8; @(posedge clk); @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      start <= 1;
      for (i = 0; i < 20; i++)
         @(posedge clk);
      reset <= 1; start <= 0; target <= 30; @(posedge clk); @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      start <= 1;
      for (i = 0; i < 20; i++)
         @(posedge clk);
      reset <= 1; start <= 0; target <= 0; @(posedge clk); @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      start <= 1;
      for (i = 0; i < 20; i++)
         @(posedge clk);
      reset <= 1; start <= 0; target <= 20; @(posedge clk); @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      start <= 1;
      for (i = 0; i < 20; i++)
         @(posedge clk);
      reset <= 1; start <= 1; target <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 10; i++)
         @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 25; i++)
         @(posedge clk);
      reset <= 1; start <= 1; target <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 10; i++)
         @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 25; i++)
         @(posedge clk);
      reset <= 1; start <= 1; target <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 10; i++)
         @(posedge clk);
      reset <= 0; @(posedge clk); @(posedge clk);
      for (i = 0; i < 25; i++)
         @(posedge clk);
```

```
        reset <= 1; start <= 0; target <= 8; @(posedge clk); @(posedge clk);
        reset <= 0; @(posedge clk); @(posedge clk);
        start <= 1;
        for (i = 0; i < 20; i++)
           @(posedge clk);
        reset <= 1; start <= 0; target <= 30; @(posedge clk); @(posedge clk);
        reset <= 0; @(posedge clk); @(posedge clk);
        start <= 1;
        for (i = 0; i < 20; i++)
           @(posedge clk);
        $stop;
    end
endmodule
```

Code 8: Task 2 binary_search_tb.sv

```
module t2_control(
   input logic clk, reset,
   input logic start, mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val,
   output logic load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr, found,
   output logic [2:0] state
   );

   logic [2:0] ps, ns;
   parameter s0 = 3'b000;
   parameter s1 = 3'b001;
   parameter s2 = 3'b010;
   parameter s3 = 3'b011;
   parameter s4 = 3'b100;

   // Next state logic
   always_comb
      case (ps)
         s0:   if (start)
                  ns = s1;
               else
                  ns = s0;
         s1:   if (eq_ptrs)
                  ns = s3;
               else
                  ns = s2;
         s2:   ns = s4;
         s3:   if (start)
                  ns = s3;
               else
                  ns = s0;
         s4:   if (mid_val_eq_target)
                  ns = s3;
               else
                  ns = s1;
      endcase

   // Output logic
   always_comb begin
      load_data = 0;
      new_high_ptr = 0;
      new_mid_ptr = 0;
      new_low_ptr = 0;
      out_addr = 0;
      found = 0;

      case (ps)
         s0:   if (~start)
                  load_data = 1;
         s1:   new_mid_ptr = 1;
         s3:   if (found_val) begin
                  found = 1;
                  out_addr = 1;
               end
         s4:   if (target_gt_mid_val)
                  new_low_ptr = 1;
               else if (target_lt_mid_val)
                  new_high_ptr = 1;
      endcase
   end

   assign state = ps;

   // DFFs
   always_ff @(posedge clk) begin
      if (reset)
         ps <= s0;
      else
         ps <= ns;
   end
endmodule
```

Code 9: Task 2 t2_control.sv

```
module t2_control_tb();
   logic clk, reset;
   logic start, mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val;
   logic load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr, found;
   logic [1:0] state;

   t2_control dut(
   clk, reset,
   start, mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val,
   load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr, found,
   state
   );

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      reset <= 1; start <= 0; mid_val_eq_target <= 0; eq_ptrs <= 0; target_gt_mid_val <= 0; found_val <= 0; @(posedge clk); @(posedge clk);
      reset <= 0; start <= 1; @(posedge clk); @(posedge clk);
      target_gt_mid_val <= 1; @(posedge clk); @(posedge clk);
      mid_val_eq_target <= 1; found_val <= 1; @(posedge clk); @(posedge clk);
      start <= 0; @(posedge clk); @(posedge clk);
      $stop;
   end
endmodule
```

Code 10: Task 2 t2_control_tb.sv

```
module t2_datapath(
   input logic clk, reset,
   input logic [7:0] target, data,
   input logic load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr,
   output logic mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val,
   output logic [4:0] addr, mid_ptr, high_ptr, low_ptr
   );

   logic [7:0] target_reg;

   // Data processing
   always_ff @(posedge clk) begin
      if (reset) begin
         target_reg <= 8'b0;
         addr <= 5'b0;
         high_ptr <= 31;
         low_ptr <= 0;
      end else begin
         if (load_data) begin
            target_reg <= target;
            high_ptr <= 31;
            low_ptr <= 0;
            addr <= 5'b0;
         end
         if (new_mid_ptr)
            mid_ptr <= ((high_ptr + low_ptr) / 2);
         else if (new_high_ptr)
            high_ptr <= mid_ptr - 1;
         else if (new_low_ptr)
            low_ptr <= mid_ptr + 1;
         if (out_addr)
            addr <= mid_ptr;
      end
   end

   // Feedback signals
   assign mid_val_eq_target = (data == target_reg);
   assign eq_ptrs = (high_ptr == low_ptr);
   assign target_gt_mid_val = (target_reg > data);
   assign target_lt_mid_val = (target_reg < data);
   assign found_val = (data == target_reg);
endmodule
```

Code 11: Task 2 t2_datapath.sv

```
module t2_datapath_tb();
   logic clk, reset;
   logic [7:0] target, data;
   logic load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr;
   logic mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val;
   logic [4:0] addr, mid_ptr, high_ptr, low_ptr;

   t2_datapath dut(
   clk, reset,
   target, data,
   load_data, new_high_ptr, new_mid_ptr, new_low_ptr, out_addr,
   mid_val_eq_target, eq_ptrs, target_gt_mid_val, target_lt_mid_val, found_val,
   addr, mid_ptr, high_ptr, low_ptr
   );

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      load_data <= 1; new_high_ptr <= 0; new_mid_ptr <= 0; new_low_ptr <= 0; out_addr <= 0; target <= 17; data <= 0; @(posedge clk); @(posedge clk);
      load_data <= 0; new_mid_ptr <= 1; @(posedge clk); @(posedge clk);
      new_high_ptr <= 1; @(posedge clk); @(posedge clk);
      new_high_ptr <= 0; new_low_ptr <= 1; @(posedge clk); @(posedge clk);
      out_addr <= 1; @(posedge clk); @(posedge clk);
      data <= 17; @(posedge clk); @(posedge clk);
      $stop;
   end
endmodule
```

Code 12: Task 2 t2_datapath_tb.sv

```
module flipflop(
   input logic D, reset, clk,
   output logic Q
   );

   always_ff @(posedge clk)
      if (reset)
         Q <= 0;
      else
         Q <= D;
endmodule
```

Code 13: flipflop.sv

```
module display(in, hex);
   input logic [3:0] in;
   output logic [6:0] hex;

   always @(in)
      case (in)
         0: hex = 7'b1000000; // 0
         1: hex = 7'b1111001; // 1
         2: hex = 7'b0100100; // 2
         3: hex = 7'b0110000; // 3
         4: hex = 7'b0011001; // 4
         5: hex = 7'b0010010; // 5
         6: hex = 7'b0000010; // 6
         7: hex = 7'b1111000; // 7
         8: hex = 7'b0000000; // 8
         9: hex = 7'b0010000; // 9
         10: hex = 7'b0001000;   // A
         11: hex = 7'b0000011;   // b
         12: hex = 7'b1000110;   // C
         13: hex = 7'b0100001;   // d
         14: hex = 7'b0000110;   // E
         15: hex = 7'b0001110;   // F
      endcase
endmodule
```

Code 14: display.sv