

```

1  /*****
2
3  bc_ctrl.sv
4
5  *****/
6
7  /*
8   * Connor Aksama
9   * 02/12/2023
10  * CSE 371
11  * Lab 4
12  */
13
14  /**
15   * Controller module for the bit counting system.
16
17   * Inputs:
18   *     start [1 bit] - Signal to begin the binary searching process. While start is 0
19   *                     and the algo is not in progress,
20   *                     the input value num is loaded into the system. When start is 1,
21   *                     the algo begins. Once the search
22   *                     is complete, the next value of num will be loaded once start is
23   *                     lowered to 0.
24   *     done [1 bit] - 1 if the algo is complete, 0 o.w.
25   *     clk [1 bit] - The clock to use for this module.
26   *     reset [1 bit] - Resets the system to its initial state before the search process
27   *                     has started.
28
29   * Outputs:
30   *     load_A [1 bit] - 1 if A should be sampled from the user during this cycle, 0 o.w.
31   *     rs_A [1 bit] - 1 if the next bit of the num should be counted, 0 o.w.
32  */
33  module bc_ctrl (
34      output logic load_A, rs_A
35      ,input logic start, done, clk, reset
36      );
37
38      typedef enum logic [1:0] { s_load, s_count, s_done } state;
39
40      state ps, ns;
41
42      logic start_reg;
43
44      always_comb begin
45          // Handle state transitions
46          case (ps)
47              s_load: begin
48                  if (start_reg) begin
49                      ns = s_count;
50                  end else begin
51                      ns = s_load;
52                  end
53              end
54              s_count: begin
55                  if (done) begin
56                      ns = s_done;
57                  end else begin
58                      ns = s_count;
59                  end
60              end
61              s_done: begin
62                  if (!start_reg) begin
63                      ns = s_load;
64                  end else begin
65                      ns = s_done;
66                  end
67              end
68          endcase
69      end
70  endmodule

```

```

66
67         endcase
68
69         // Control signals
70         load_A = (ps == s_load);
71         rs_A = (ps == s_count);
72
73     end
74
75     always_ff @(posedge clk) begin
76         // Register state, start
77         if (reset) begin
78             ps <= s_load;
79         end else begin
80             ps <= ns;
81         end
82
83         start_reg <= start;
84     end
85
86 endmodule // bc_ctrl
87
88 /*
89  * Testbench to test the functionality of the bc_ctrl module
90  */
91 module bc_ctrl_testbench();
92
93     logic start, done, clk, reset, load_A, rs_A;
94
95     bc_ctrl dut (.*);
96
97     parameter CLOCK_PERIOD = 100;
98     initial begin
99         clk <= 0;
100         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
101     end
102
103     initial begin
104         integer i;
105         // Hold in start state
106         @(posedge clk) reset <= 1'b1; start <= 1'b0; done <= 1'b0;
107         @(posedge clk); reset <= 1'b0;
108         @(posedge clk);
109
110         // Start algo
111         for (i = 0; i < 15; i++) begin
112             @(posedge clk) start <= 1'b1;
113         end
114
115         // Hold in done state
116         for (i = 0; i < 5; i++) begin
117             @(posedge clk) done <= 1'b1;
118         end
119
120         // Back to start state
121         start <= 1'b0;
122         for (i = 0; i < 5; i++) begin
123             @(posedge clk);
124         end
125
126         // Start algo
127         start <= 1'b1;
128         for (i = 0; i < 5; i++) begin
129             @(posedge clk);
130         end
131
132         $stop;
133     end
134

```

```

135     endmodule // bc_ctrl_testbench
136
137     /*****
138
139     bc_data.sv
140
141     *****/
142
143     /*
144     * Connor Aksama
145     * 02/12/2023
146     * CSE 371
147     * Lab 4
148     */
149
150     /**
151     * Datapath module for the bit counting system.
152
153     * Inputs:
154     *     a_in [8 bit] - The number to bit count. Latched when load_A is 1.
155     *     load_A [1 bit] - 1 if a_in should be sampled, 0 o.w.
156     *     rs_A [1 bit] - 1 if the next bit should be counted, 0 o.w.
157     *     clk [1 bit] - The clock to use for this module.
158
159     * Outputs:
160     *     done [1 bit] - 1 if the algo is complete, 0 o.w.
161     *     count [4 bit] - The number of 1s in the sampled a_in number. Valid if and only
162     if done is 1.
163     */
164     module bc_data (
165         output logic done
166         ,output logic [3:0] count
167         ,input logic load_A, rs_A, clk
168         ,input logic [7:0] a_in
169     );
170
171     logic [7:0] A;
172     logic [3:0] result;
173
174     always_ff @(posedge clk) begin
175         if (load_A) begin
176             // Sample A
177             result <= 4'b0;
178             A <= a_in;
179         end
180
181         if (rs_A) begin
182             // RS and count
183             A <= A >> 1;
184             if (A[0] == 1'b1) begin
185                 result <= result + 1'b1;
186             end
187         end
188
189         done <= (A == 0);
190         count <= result;
191     end
192
193     endmodule // bc_data
194
195     /**
196     * Testbench to test the functionality of the bc_data module.
197     */
198     module bc_data_testbench();
199
200         logic done, load_A, rs_A, clk;
201         logic [3:0] count;
202         logic [7:0] a_in;

```

```

203     bc_data dut (.*) ;
204
205     parameter CLOCK_PERIOD = 100;
206     initial begin
207         clk <= 0;
208         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
209     end
210
211     initial begin
212         integer i;
213         // Load value, hold in start
214         @(posedge clk) {load_A, rs_A} <= 2'b10; a_in <= 8'b10011111;
215         @(posedge clk);
216         @(posedge clk);
217         @(posedge clk);
218         @(posedge clk);
219         // Begin algo
220         for (i = 0; i < 15; i++) begin
221             @(posedge clk) {load_A, rs_A} <= 2'b01;
222         end
223         // Load value
224         @(posedge clk) {load_A, rs_A} <= 2'b10; a_in <= 8'b00000000;
225         @(posedge clk);
226         @(posedge clk);
227         @(posedge clk);
228         @(posedge clk);
229         // Begin
230         for (i = 0; i < 15; i++) begin
231             @(posedge clk) {load_A, rs_A} <= 2'b01;
232         end
233
234         $stop;
235     end
236
237 endmodule
238
239 /*****
240
241 bit_counter.sv
242
243 *****/
244
245 /*
246  * Connor Aksama
247  * 02/12/2023
248  * CSE 371
249  * Lab 4
250 */
251
252 /**
253  * Instantiates the controller and datapath modules and defines connections.
254
255  * Inputs:
256  *     num [8 bit] - Number to bit count.
257  *     start [1 bit] - Signal to begin the bit counting algo. While start is 0 and the
258  *                     algo is not in progress,
259  *                     the input value num is loaded into the system. When start is 1,
260  *                     the algo begins. Once the algo
261  *                     is complete, the next value of num will be loaded once start is
262  *                     lowered to 0.
263  *     clk [1 bit] - The clock to use for this module.
264  *     reset [1 bit] - Resets the system to its initial state before the algo has
265  *                     started.
266
267  * Outputs:
268  *     result [4 bit] - The number of 1s in the sampled num. This result value is valid if
269  *                     and only if done is raised.
270  *     If found and not_found are 0, the search is not complete.
271  *     done [1 bit] - 1 if the algo is complete, 0 o.w.

```

```

267  */
268  module bit_counter(
269      output logic [3:0] result
270      ,output logic done
271      ,input logic [7:0] num
272      ,input logic start, clk, reset
273  );
274
275      // Controller module
276      // In: start, done, clk, reset
277      // Out: load_A, rs_A
278      logic load_A, rs_A, d;
279      bc_ctrl controller (
280          .load_A(load_A)
281          ,.rs_A(rs_A)
282          ,.start(start)
283          ,.done(d)
284          ,.clk(clk)
285          ,.reset(reset)
286      );
287
288      // Datapath module
289      // In: load_A, rs_A, a_in, clk
290      // Out: done, count
291      bc_data datapath (
292          .done(d)
293          ,.count(result)
294          ,.load_A(load_A)
295          ,.rs_A(rs_A)
296          ,.a_in(num)
297          ,.clk(clk)
298      );
299
300      assign done = d;
301
302  endmodule // bit_counter
303
304  /*
305   * Testbench to test the functionality of the bit_counter module.
306   */
307  module bit_counter_testbench();
308
309      logic [3:0] result;
310      logic done;
311      logic [7:0] num;
312      logic start, clk, reset;
313
314      bit_counter dut (.*);
315
316      parameter CLOCK_PERIOD = 100;
317      initial begin
318          clk <= 0;
319          forever #(CLOCK_PERIOD / 2) clk <= ~clk;
320      end
321
322      initial begin
323          integer i;
324          // Load value
325          @(posedge clk) reset <= 1; num <= 8'b10000001; start <= 0;
326          @(posedge clk) reset <= 0;
327          @(posedge clk);
328          @(posedge clk);
329          @(posedge clk);
330          // Start count
331          @(posedge clk) start <= 1;
332
333          for (i = 0; i < 16; i++) begin
334              @(posedge clk);
335              end

```

```

336         // Load value
337         @(posedge clk) start <= 0; num <= 8'b0;
338         @(posedge clk);
339         @(posedge clk);
340         @(posedge clk);
341         // Start count
342         @(posedge clk) start <= 1;
343         for (i = 0; i < 5; i++) begin
344             @(posedge clk);
345         end
346
347         @(posedge clk);
348         @(posedge clk);
349
350         $stop;
351     end
352
353 endmodule // bit_counter_testbench
354
355 /*****
356
357 DE1_SoC_task1.sv
358
359 *****/
360
361 /*
362  * Connor Aksama
363  * 02/12/2023
364  * CSE 371
365  * Lab 4
366  */
367
368 /**
369  * Top-level module for Lab 4 Task 1. Instantiates binary search module and connects I/O
370  * to peripherals.
371
372  * Inputs:
373  *   SW [10 bit] - The 10 onboard switches, respectively.
374  *   KEY [4 bit] - The 4 onboard keys, respectively.
375  *   CLOCK_50 [1 bit] - The system clock to use for this module.
376
377  * Outputs:
378  *   HEX0 [7 bit] - Data to show on the HEX0 display, formatted in standard 7 segment
379  *   display format.
380  *   LEDR [10 bit] - Signal to output to 10 onboard LEDs, respectively.
381  */
382 module DE1_SoC_task1 (
383     output logic [6:0] HEX0
384     ,output logic [9:0] LEDR
385     ,input logic [3:0] KEY
386     ,input logic [9:0] SW
387     ,input logic CLOCK_50
388 );
389
390     // Bit counting module
391     // In: num, start, clk, reset
392     // Out: result, done
393     logic [3:0] result;
394     bit_counter bc (
395         .result(result)
396         ,.num(SW[7:0])
397         ,.done(LEDR[9])
398         ,.start(SW[9])
399         ,.clk(CLOCK_50)
400         ,.reset(~KEY[0])
401     );
402
403     // Seven-Segment display
404     // In: num

```

```

403         // Out: HEX0, HEX1(unused)
404         double_seg7 res_display (.HEX0(HEX0), .HEX1(), .num({4'b0, result}));
405
406     endmodule // DE1_SoC_task1
407
408     /*
409     * Testbench to test the functionality of the DE1_SoC_task1 module.
410     */
411     module DE1_SoC_task1_testbench();
412
413         logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
414         logic [9:0] LEDR;
415         logic [3:0] KEY;
416         logic [9:0] SW;
417         logic CLOCK_50, clk;
418
419         assign CLOCK_50 = clk;
420
421         DE1_SoC_task1 dut (.*);
422
423         parameter CLOCK_PERIOD = 100;
424         initial begin
425             clk <= 0;
426             forever #(CLOCK_PERIOD / 2) clk <= ~clk;
427         end
428
429         initial begin
430             integer i;
431
432             // Load value
433             @(posedge clk) KEY[0] <= 1'b0; SW[7:0] <= 8'b10000001; SW[9] <= 0;
434             @(posedge clk); KEY[0] <= 1'b1;
435             // Start count
436             @(posedge clk) SW[9] <= 1;
437
438             for (i = 0; i < 16; i++) begin
439                 @(posedge clk);
440             end
441             // Load value
442             @(posedge clk) SW[9] <= 0; SW[7:0] <= 8'b0;
443             @(posedge clk);
444             // Start count
445             @(posedge clk) SW[9] <= 1;
446             for (i = 0; i < 5; i++) begin
447                 @(posedge clk);
448             end
449
450             @(posedge clk);
451             @(posedge clk);
452
453             $stop;
454         end
455
456     endmodule // DE1_SoC_task1_testbench
457
458     /*****
459
460     double_seg7.sv
461
462     *****/
463
464     /*
465     * Connor Aksama
466     * 02/12/2023
467     * CSE 371
468     * Lab 4
469     */
470
471     /**

```

```

472 * Defines data for 2-digit hexadecimal HEX display given a 8-bit unsigned integer
473
474 * Inputs:
475 *     num [8 bit] - An unsigned integer value [0x0-0x7F] to display
476
477 * Outputs:
478 *     HEX0 [7 bit] - A HEX display for the least significant digit of the input num,
479 *     formatted in standard seven segment display format
480 *     HEX1 [7 bit] - A HEX display for the most significant digit of the input num,
481 *     formatted in standard seven segment display format
482 */
483 module double_seg7(
484     output logic [6:0] HEX0, HEX1
485     ,input logic [7:0] num
486 );
487
488 // Drive HEX output signals given num
489 always_comb begin
490     // Light HEX0 using LSD of num
491     case (num[3:0])
492         //      Light: 6543210
493         0: HEX0 = ~7'b0111111; // 0
494         1: HEX0 = ~7'b0000110; // 1
495         2: HEX0 = ~7'b1011011; // 2
496         3: HEX0 = ~7'b1001111; // 3
497         4: HEX0 = ~7'b1100110; // 4
498         5: HEX0 = ~7'b1101101; // 5
499         6: HEX0 = ~7'b1111101; // 6
500         7: HEX0 = ~7'b0000111; // 7
501         8: HEX0 = ~7'b1111111; // 8
502         9: HEX0 = ~7'b1101111; // 9
503         10: HEX0 = ~7'b1110111; // A
504         11: HEX0 = ~7'b1111100; // b
505         12: HEX0 = ~7'b1011000; // c
506         13: HEX0 = ~7'b1011110; // d
507         14: HEX0 = ~7'b1111001; // E
508         15: HEX0 = ~7'b1110001; // F
509     default: HEX0 = 7'bx;
510 endcase
511
512 // Light HEX1 using MSD of num
513 case (num >> 4)
514     //      Light: 6543210
515     0: HEX1 = ~7'b0111111; // 0
516     1: HEX1 = ~7'b0000110; // 1
517     2: HEX1 = ~7'b1011011; // 2
518     3: HEX1 = ~7'b1001111; // 3
519     4: HEX1 = ~7'b1100110; // 4
520     5: HEX1 = ~7'b1101101; // 5
521     6: HEX1 = ~7'b1111101; // 6
522     7: HEX1 = ~7'b0000111; // 7
523     8: HEX1 = ~7'b1111111; // 8
524     9: HEX1 = ~7'b1101111; // 9
525     10: HEX1 = ~7'b1110111; // A
526     11: HEX1 = ~7'b1111100; // b
527     12: HEX1 = ~7'b1011000; // c
528     13: HEX1 = ~7'b1011110; // d
529     14: HEX1 = ~7'b1111001; // E
530     15: HEX1 = ~7'b1110001; // F
531     default: HEX1 = 7'bx;
532 endcase
533
534 end
535
536 endmodule // double_seg7
537
538 /*
539 * Tests the functionality of the double_seg7 module.
540 */

```



```

539 module double_seg7_testbench();
540
541     logic [6:0] HEX0, HEX1;
542     logic [7:0] num;
543
544     double_seg7 dut (.HEX0, .HEX1, .num);
545
546     initial begin
547
548         integer i;
549
550         // Check HEX displays for integers 0x0-0xff
551         for (i = 0; i <= 8'hFF; i++) begin
552             #10 num = i;
553         end
554         #50;
555     end
556
557 endmodule // double_seg7_testbench
558
559 /*****
560
561 binary_search.sv
562
563 *****/
564
565 /*
566  * Connor Aksama
567  * 02/12/2023
568  * CSE 371
569  * Lab 4
570  */
571
572 /**
573  * Instantiates the controller and datapath modules and defines connections.
574
575  * Inputs:
576  *     num [8 bit] - An unsigned integer value to search for in the system's RAM.
577  *     start [1 bit] - Signal to begin the binary searching process. While start is 0
578  * and the search is not in progress,
579  *         the input value num is loaded into the system. When start is 1,
580  * the search begins. Once the search
581  *         is complete, the next value of num will be loaded once start is
582  * lowered to 0.
583  *     clk [1 bit] - The clock to use for this module.
584  *     reset [1 bit] - Resets the system to its initial state before the search process
585  * has started.
586
587  * Outputs:
588  *     index [5 bit] - The index of the given num in the system's RAM. This index value is
589  * valid if and only if found is raised.
590  *         If found and not_found are 0, the search is not complete.
591  *     found [1 bit] - 1 if the search is complete and num was found in the system's RAM. 0
592  * otherwise.
593  *     not_found [1 bit] - 1 if the search is complete and num was not found in the
594  * system's RAM. 0 otherwise.
595  */
596 module binary_search (
597     output logic [4:0] index
598     ,output logic found, not_found
599     ,input logic [7:0] num
600     ,input logic start, clk, reset
601 );
602
603     // Controller Module
604     // In: a_eq_t, s_zero, start, clk, reset
605     // Out: load_A, try_S
606     logic load_A, try_S, a_eq_t, s_zero;
607     bs_ctrl controller (

```

```

601         .load_A(load_A)
602         ,.try_S(try_S)
603         ,.start(start)
604         ,.a_eq_t(a_eq_t)
605         ,.s_zero(s_zero)
606         ,.clk(clk)
607         ,.reset(reset)
608     );
609
610     // Datapath module
611     // In: A_in (user input), load_A, try_S (from controller), clk
612     // Out: index (answer), found, not_found (done & success?)
613     bs_data datapath (
614         .index(index)
615         ,.found(found)
616         ,.not_found(not_found)
617         ,.a_eq_t(a_eq_t)
618         ,.s_zero(s_zero)
619         ,.A_in(num)
620         ,.load_A(load_A)
621         ,.try_S(try_S)
622         ,.clk(clk)
623     );
624
625 endmodule // binary_search
626
627 /*
628  * The testbench module to test the functionality of the binary_search module.
629  */
630 `timescale 1 ps / 1 ps
631 module binary_search_testbench();
632
633     logic [4:0] index;
634     logic found, not_found;
635     logic [7:0] num;
636     logic start, clk, reset;
637
638     binary_search dut (.*);
639
640     parameter CLOCK_PERIOD = 100;
641     initial begin
642         clk <= 0;
643         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
644     end
645
646     initial begin
647         integer i;
648
649         @(posedge clk) reset <= 1'b1; start <= 1'b0; num <= 11;
650         @(posedge clk) reset <= 1'b0;
651         @(posedge clk);
652         @(posedge clk);
653         @(posedge clk) start <= 1'b1;
654
655         for (i = 0; i < 15; i++) begin
656             @(posedge clk);
657         end
658
659         @(posedge clk) start <= 1'b0; num <= 55;
660         @(posedge clk);
661         @(posedge clk);
662         @(posedge clk) start <= 1'b1;
663
664         for (i = 0; i < 15; i++) begin
665             @(posedge clk);
666         end
667         $stop;
668     end
669 end

```

```

670     endmodule // binary_search_testbench
671
672     /*****
673
674     bs_ctrl.sv
675
676     *****/
677
678     /*
679     * Connor Aksama
680     * 02/12/2023
681     * CSE 371
682     * Lab 4
683     */
684
685     /**
686     * Controller module for the binary search system.
687
688     * Inputs:
689     *     start [1 bit] - Signal to begin the binary searching process. While start is 0
690     *     and the search is not in progress,
691     *     the input value num is loaded into the system. When start is 1,
692     *     the search begins. Once the search
693     *     is complete, the next value of num will be loaded once start is
694     *     lowered to 0.
695     *     a_eq_t [1 bit] - 1 if the search is complete and the number was found in RAM, 0
696     *     o.w.
697     *     s_zero [1 bit] - 1 if the search is complete and the number was not found in
698     *     RAM, 0 o.w.
699     *     clk [1 bit] - The clock to use for this module.
700     *     reset [1 bit] - Resets the system to its initial state before the search process
701     *     has started.
702
703     * Outputs:
704     *     load_A [1 bit] - 1 if A should be sampled from the user during this cycle, 0 o.w.
705     *     try_S [1 bit] - 1 if the next bit index of the RAM address should be tested
706     *     during this cycle, 0 o.w.
707     */
708     module bs_ctrl (
709         output logic load_A, try_S
710         ,input logic start, a_eq_t, s_zero, clk, reset
711     );
712
713         typedef enum logic [1:0] { s_start, s_load, s_search, s_done } state;
714
715         state ps, ns;
716
717         logic start_reg;
718
719         always_comb begin
720             // Handle state transitions
721             case (ps)
722                 s_start: begin
723                     if (start_reg) begin
724                         ns = s_load;
725                     end else begin
726                         ns = s_start;
727                     end
728                 end
729                 s_load: begin
730                     ns = s_search;
731                 end
732                 s_search: begin
733                     if (a_eq_t | s_zero) begin
734                         ns = s_done;
735                     end else begin
736

```

```

732         ns = s_load;
733     end
734 end
735
736     s_done: begin
737         if (~start_reg) begin
738             ns = s_start;
739         end else begin
740             ns = s_done;
741         end
742     end
743
744 endcase
745
746 // Control signals to datapath
747 load_A = (ps == s_start);
748
749 try_S = (ps == s_search);
750 end
751
752 always_ff @(posedge clk) begin
753     // Register state and start input
754     if (reset) begin
755         ps <= s_start;
756     end else begin
757         ps <= ns;
758     end
759
760     start_reg <= start;
761 end
762
763 endmodule // bs_ctrl
764
765 /*
766  * Module to test the functionality of the bs_ctrl module
767  */
768 module bs_ctrl_testbench();
769
770     logic load_A, try_S;
771     logic start, a_eq_t, s_zero, clk, reset;
772
773     bs_ctrl dut (.*);
774
775     parameter CLOCK_PERIOD = 100;
776     initial begin
777         clk <= 0;
778         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
779     end
780
781     initial begin
782         integer i;
783
784         // Hold in start state
785         start <= 1'b0; a_eq_t <= 1'b0; s_zero <= 1'b0; reset <= 1'b1;
786         @(posedge clk) reset <= 1'b0;
787         @(posedge clk);
788         @(posedge clk);
789         @(posedge clk);
790         // Begin the search process
791         @(posedge clk) start <= 1'b1;
792
793         for (i = 0; i < 10; i++) begin
794             @(posedge clk);
795         end
796         // Search success
797         a_eq_t <= 1'b1;
798         // Hold in done state
799         for (i = 0; i < 5; i++) begin
800             @(posedge clk);

```

```

801     end
802     // Go back to start
803     start <= 1'b0; a_eq_t <= 1'b0; s_zero <= 1'b0;
804     for (i = 0; i < 5; i++) begin
805         @(posedge clk);
806     end
807     // Search
808     @(posedge clk) start <= 1'b1;
809
810     for (i = 0; i < 10; i++) begin
811         @(posedge clk);
812     end
813     // Search failure
814     s_zero <= 1'b1;
815     for (i = 0; i < 5; i++) begin
816         @(posedge clk);
817     end
818
819     $stop;
820 end
821
822 endmodule // bs_ctrl_testbench
823
824 /*****
825
826 bs_data.sv
827
828 *****/
829
830 /*
831  * Connor Aksama
832  * 02/12/2023
833  * CSE 371
834  * Lab 4
835  */
836
837 /**
838  * Datapath module for the binary search system.
839
840  * Inputs:
841  *     A_in [8 bit] - The number to search for in the RAM. Latched when load_A is true
842  *     load_A [1 bit] - 1 if A_in should be sampled, 0 o.w.
843  *     try_S [1 bit] - 1 if the next bit in the index should be tested, 0 o.w.
844  *     clk [1 bit] - The clock to use for this module.
845
846  * Outputs:
847  *     index [5 bit] - The index of the sampled A_in value in the RAM. Valid if and
848  *     only if found is 1.
849  *     a_eq_t [1 bit] - Feedback if the sampled A_in value is equal to the currently
850  *     read value from RAM. 1 if eq, 0 o.w.
851  *     s_zero [1 bit] - Feedback if all bits have been tested in the index. 1 if true,
852  *     0 o.w.
853  *     found [1 bit] - 1 if the search is complete and the sampled A_in was found in
854  *     RAM, 0 o.w.
855  *     not_found [1 bit] - 1 if the search is complete and the sampled A_in was not
856  *     found in RAM, 0 o.w.
857  */
858 module bs_data (
859     output logic [4:0] index
860     ,output logic a_eq_t, s_zero, found, not_found
861     ,input logic [7:0] A_in
862     ,input logic load_A, try_S, clk
863 );
864
865     logic [4:0] I, S;
866     logic [7:0] T, A;
867
868     // RAM block
869     // In: address/I(ndex), clk

```

```

865 // Out: data/wren - unused; q - data
866 ram32x8 ram (
867     .address(I)
868     ,.clock(clk)
869     ,.data(8'b0)
870     ,.wren(1'b0)
871     ,.q(T)
872 );
873
874 assign a_eq_t = (A == T);
875 assign s_zero = (S == '0);
876
877 always_ff @(posedge clk) begin
878
879     if (load_A) begin
880         S <= 5'b10000;
881         I <= 5'b10000;
882         A <= A_in;
883     end
884
885     if (try_S) begin
886         S <= S >> 1;
887         if (T > A) begin
888             I <= (I ^ S) | (S >> 1);
889         end else if (T < A) begin
890             I <= I | (S >> 1);
891         end
892     end
893
894     index <= I;
895     found <= (A == T);
896     not_found <= (A != T) & (S == 0);
897 end
898
899 endmodule // bs_data
900
901 /*
902  * Testbench to test the functionality of the bs_data module
903  */
904 `timescale 1 ps / 1 ps
905 module bs_data_testbench();
906
907     logic [4:0] index;
908     logic found, not_found, a_eq_t, s_zero;
909     logic [7:0] A_in;
910     logic load_A, try_S, clk;
911
912     bs_data dut (.*);
913
914     parameter CLOCK_PERIOD = 100;
915     initial begin
916         clk <= 0;
917         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
918     end
919
920     initial begin
921         integer i;
922
923         // Load value
924         A_in <= 13; load_A <= 1'b1; try_S <= 1'b0;
925
926         // Search
927         for (i = 0; i < 10; i++) begin
928             @(posedge clk); load_A <= 1'b0; try_S <= 1'b0;
929             @(posedge clk); load_A <= 1'b0; try_S <= 1'b1;
930         end
931
932         // Load
933         @(posedge clk) A_in <= 33; load_A <= 1'b1; try_S <= 1'b0;

```

```

934         // Search
935         for (i = 0; i < 10; i++) begin
936             @(posedge clk); load_A <= 1'b0; try_S <= 1'b0;
937             @(posedge clk); load_A <= 1'b0; try_S <= 1'b1;
938         end
939
940         $stop;
941
942     end
943
944 endmodule // bs_data_testbench
945
946 /*****
947
948 DE1_SoC_task2.sv
949
950 *****/
951
952 /*
953  * Connor Aksama
954  * 02/12/2023
955  * CSE 371
956  * Lab 4
957  */
958
959 /**
960  * Top-level module for Lab 4 Task 2. Instantiates binary search module and connects I/O
961  * to peripherals.
962  *
963  * Inputs:
964  *   SW [10 bit] - The 10 onboard switches, respectively.
965  *   KEY [4 bit] - The 4 onboard keys, respectively.
966  *   CLOCK_50 [1 bit] - The system clock to use for this module.
967  *
968  * Outputs:
969  *   HEX0 [7 bit] - Data to show on the HEX0 display, formatted in standard 7 segment
970  *   display format.
971  *   HEX1 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
972  *   display format.
973  *   LEDR [10 bit] - Signal to output to 10 onboard LEDs, respectively.
974  */
975 module DE1_SoC_task2 (
976     output logic [6:0] HEX0, HEX1
977     ,output logic [9:0] LEDR
978     ,input logic [9:0] SW
979     ,input logic [3:0] KEY
980     ,input logic CLOCK_50
981 );
982
983     assign LEDR[7:0] = 8'b0;
984
985     // Seven-Segment display
986     // In: num
987     // Out: HEX0, HEX1
988     logic [7:0] index;
989     double_seg7 addr (
990         .HEX0(HEX0)
991         ,.HEX1(HEX1)
992         ,.num(index)
993     );
994
995     // Binary search module
996     // In: num (switch input), start (switch), clk, reset (key)
997     // Out: index (final answer), found (success), not_found (failure)
998     binary_search bs (
999         .index(index)
1000         ,.found(LEDR[9])
1001         ,.not_found(LEDR[8])
1002         ,.num(SW[7:0])

```

```

1000         ,.start(SW[9])
1001         ,.clk(CLOCK_50)
1002         ,.reset(~KEY[0])
1003     );
1004
1005 endmodule // DE1_SoC_task2
1006
1007 /*
1008  * Testbench to test the functionality of the DE1_SoC_task2 module
1009  */
1010 `timescale 1 ps / 1 ps
1011 module DE1_SoC_task2_testbench();
1012
1013     logic [6:0] HEX0, HEX1;
1014     logic [9:0] LEDR;
1015     logic [9:0] SW;
1016     logic [3:0] KEY;
1017     logic CLOCK_50, clk;
1018
1019     assign CLOCK_50 = clk;
1020
1021     DE1_SoC_task2 dut (.*);
1022
1023     parameter CLOCK_PERIOD = 100;
1024     initial begin
1025         clk <= 0;
1026         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
1027     end
1028
1029     initial begin
1030         integer i;
1031
1032         // Load num
1033         @(posedge clk) KEY[0] <= 1'b0; SW[9] <= 1'b0; SW[7:0] <= 8'b00001011;
1034         @(posedge clk) KEY[0] <= 1'b1;
1035         @(posedge clk);
1036         @(posedge clk);
1037         // Start
1038         @(posedge clk) SW[9] <= 1'b1;
1039         // Search
1040         for (i = 0; i < 15; i++) begin
1041             @(posedge clk);
1042         end
1043
1044         // Load
1045         @(posedge clk) SW[9] <= 1'b0; SW[7:0] <= 8'b00110111;
1046         @(posedge clk);
1047         @(posedge clk);
1048         // Start
1049         @(posedge clk) SW[9] <= 1'b1;
1050         // Search
1051         for (i = 0; i < 15; i++) begin
1052             @(posedge clk);
1053         end
1054         $stop;
1055     end
1056
1057 endmodule // DE1_SoC_task2_testbench
1058

```