```
1  `timescale 1ns/1ps
2  `include "cmach_recipes.svh"
3
4  module lcdIp_Top (
5      input  wire        CLOCK_50,
6      input  wire        KEY,            // KEY0 (active-low) -> START
7
8      input  wire        BTN_FLAVOR,      // if these are also KEY buttons, they are active-low
9      input  wire        BTN_TYPE,
10     input  wire        BTN_SIZE,
11     input  wire        BTN_START,       // unused (KEY0 is start), kept for pin-compat
12
13     // 2-bit "level" inputs (same encoding as PAPER_LEVEL):
14     // 00=NOT INSTALLED (or hard empty), 01=EMPTY, 10=ALMOST EMPTY, 11=OK
15     input  wire [1:0]  PAPER_LEVEL,
16     input  wire [1:0]  BIN0_LEVEL,
17     input  wire [1:0]  BIN1_LEVEL,
18     input  wire [1:0]  ND_LEVEL,
19     input  wire [1:0]  CH_LEVEL,
20
21     input  wire [1:0]  W_PRESSURE,
22     input  wire        W_TEMP,
23     input  wire        STATUS,
24
25     output wire        HEAT_EN,
26     output wire        POUROVER_EN,
27     output wire        WATER_EN,
28     output wire        GRINDER_0_EN,
29     output wire        GRINDER_1_EN,
30     output wire        PAPER_EN,
31     output wire        COCOA_EN,
32     output wire        CREAMER_EN,
33
34     output wire [7:0]  LCD_DATA,
35     output wire        LCD_RS,
36     output wire        LCD_RW,
37     output wire        LCD_EN,
38     output wire        LCD_ON,
39     output wire        LCD_BLON
40 );
41
42     assign LCD_ON   = 1'b1;
43     assign LCD_BLON = 1'b1;
44
45     //============================================================
46     // Internal power-on reset (POR)
47     //============================================================
48     localparam [21:0] POR_CYCLES = 22'd1_000_000; // ~20ms @50MHz
49     reg [21:0] por_cnt = 22'd0;
50     wire rst = (por_cnt < POR_CYCLES);
```

```
51
52      always @(posedge CLOCK_50) begin
53          if (por_cnt < POR_CYCLES)
54              por_cnt <= por_cnt + 22'd1;
55      end
56
57      //=============================================================
58      // Buttons: treat board keys as active-low => invert to active-high
59      // KEY0 is START
60      //=============================================================
61      wire btn_flavor_lvl = ~BTN_FLAVOR;
62      wire btn_type_lvl   = ~BTN_TYPE;
63      wire btn_size_lvl   = ~BTN_SIZE;
64      wire btn_start_lvl  = ~KEY;         // KEY0 start (active-low on board)
65
66      //=============================================================
67      // Recipes
68      //=============================================================
69      logic [$bits(coffee_recipe_t)-1:0] recipes [0:14];
70      cmach_recp u_recipes (.recipes(recipes));
71
72      //=============================================================
73      // Coffee control system
74      //=============================================================
75      wire       cur_flavor;
76      wire [2:0] cur_type;
77      wire [1:0] cur_size;
78      wire [1:0] sys_state;
79
80      wire [15:0] coffee_err_mask;
81      wire [2:0]  brew_phase;
82      wire [4:0]  brew_progress16;
83
84      coffeeSystem #(
85          .CLK_HZ(50_000_000),
86          .SPEEDUP_DIV(1)
87      ) u_coffee (
88          .clk(CLOCK_50),
89          .rst(rst),
90
91          .btn_flavor(btn_flavor_lvl),
92          .btn_type  (btn_type_lvl),
93          .btn_size  (btn_size_lvl),
94          .btn_start (btn_start_lvl),
95
96          .PAPER_LEVEL(PAPER_LEVEL),
97          .BIN0_LEVEL (BIN0_LEVEL),
98          .BIN1_LEVEL (BIN1_LEVEL),
99          .ND_LEVEL   (ND_LEVEL),
100         .CH_LEVEL   (CH_LEVEL),
```

```
101
102         .W_PRESSURE(W_PRESSURE),
103         .W_TEMP(W_TEMP),
104         .STATUS(STATUS),
105
106         .recipes(recipes),
107
108         .cur_flavor(cur_flavor),
109         .cur_type(cur_type),
110         .cur_size(cur_size),
111         .sys_state(sys_state),
112
113         .HEAT_EN(HEAT_EN),
114         .POUROVER_EN(POUROVER_EN),
115         .WATER_EN(WATER_EN),
116         .GRINDER_0_EN(GRINDER_0_EN),
117         .GRINDER_1_EN(GRINDER_1_EN),
118         .PAPER_EN(PAPER_EN),
119         .COCOA_EN(COCOA_EN),
120         .CREAMER_EN(CREAMER_EN),
121
122         .err_mask(coffee_err_mask),
123         .brew_phase(brew_phase),
124         .brew_progress16(brew_progress16)
125     );
126
127     //============================================================
128     // LCD IP interface
129     //============================================================
130     reg  [1:0] op;       // 2'b00 = INSTR (RS=0), 2'b01 = DATA (RS=1)
131     reg        send;
132     reg  [7:0] din;
133     wire       busy, systemReady;
134
135     lcdIp u_lcd (
136         .clk          (CLOCK_50),
137         .userOp       (op),
138         .send         (send),
139         .reset        (rst),
140         .inputCommand (din),
141         .lcd_data     (LCD_DATA),
142         .lcd_rs       (LCD_RS),
143         .lcd_rw       (LCD_RW),
144         .lcd_e        (LCD_EN),
145         .busy         (busy),
146         .systemReady  (systemReady)
147     );
148
149     //============================================================
150     // PLEASE ENJOY screen (shows briefly after brew completes)
```

```
151     //============================================================
152     reg [7:0] enjoy_l1 [0:15] = '{"P","l","e","a","s","e"," ","E","n","j","o","y","!"," ","
    "," "};
153     reg [7:0] enjoy_l2 [0:15] = '{"T","h","a","n","k"," ","y","o","u"," "," "," "," "," ","
    "," "};
154 
155     // 1-second tick (also used for err/warn cycling)
156     localparam [31:0] DISP_TICKS = 32'd50_000_000;
157     reg [31:0] disp_cnt;
158     wire disp_tick = (disp_cnt == (DISP_TICKS-1));
159 
160     always @(posedge CLOCK_50) begin
161         if (rst) disp_cnt <= 32'd0;
162         else if (disp_tick) disp_cnt <= 32'd0;
163         else disp_cnt <= disp_cnt + 32'd1;
164     end
165 
166     reg [1:0] sys_state_d;
167     reg       enjoy_active;
168     reg [1:0] enjoy_secs_left;
169 
170     always @(posedge CLOCK_50) begin
171         if (rst) begin
172             sys_state_d     <= 2'd0;
173             enjoy_active    <= 1'b0;
174             enjoy_secs_left <= 2'd0;
175         end else begin
176             sys_state_d <= sys_state;
177 
178             // detect transition BREW(2) -> SELECT(0)
179             if ((sys_state_d == 2'd2) && (sys_state == 2'd0)) begin
180                 enjoy_active    <= 1'b1;
181                 enjoy_secs_left <= 2'd2; // show for 2 seconds
182             end else if (enjoy_active && disp_tick) begin
183                 if (enjoy_secs_left <= 2'd1) begin
184                     enjoy_active    <= 1'b0;
185                     enjoy_secs_left <= 2'd0;
186                 end else begin
187                     enjoy_secs_left <= enjoy_secs_left - 2'd1;
188                 end
189             end
190 
191             // any new selection input cancels enjoy screen immediately
192             if (enjoy_active && (btn_flavor_lvl || btn_type_lvl || btn_size_lvl ||
    btn_start_lvl))
193                 enjoy_active <= 1'b0;
194         end
195     end
196 
197     //============================================================
198     // ERROR/WARN cycling selection
```

```
199     //============================================================
200     // Must match coffeeSystem.sv error bit numbers
201     localparam int E_PAPER_NOT_INST = 0;
202     localparam int E_PAPER_EMPTY    = 1;
203     localparam int E_NO_COFFEE0     = 2;
204     localparam int E_NO_COFFEE1     = 3;
205     localparam int E_NO_CREAMER     = 4;
206     localparam int E_NO_CHOC        = 5;
207     localparam int E_PRESS_ERR      = 6;
208     localparam int E_PRESS_HIGH     = 7;
209     localparam int E_STATUS_ERR     = 8;
210 
211     // warnings we want to cycle (not blocking)
212     localparam int W_FILTER_ALMOST  = 0;
213     localparam int W_LOW_PRESSURE   = 1;
214     localparam int W_BIN0_EMPTY     = 2;
215     localparam int W_BIN1_EMPTY     = 3;
216     localparam int W_ND_EMPTY       = 4;
217     localparam int W_CH_EMPTY       = 5;
218 
219     logic [15:0] warn_mask;
220 
221     always @(*) begin
222         warn_mask = 16'b0;
223 
224         if (PAPER_LEVEL == 2'b10) warn_mask[W_FILTER_ALMOST] = 1'b1; // almost empty
225         if (W_PRESSURE  == 2'b00) warn_mask[W_LOW_PRESSURE]  = 1'b1; // low pressure
226 
227         // ingredient "almost empty" warnings are ONLY when level==2'b10
228         if (BIN0_LEVEL == 2'b10) warn_mask[W_BIN0_EMPTY] = 1'b1;
229         if (BIN1_LEVEL == 2'b10) warn_mask[W_BIN1_EMPTY] = 1'b1;
230         if (ND_LEVEL   == 2'b10) warn_mask[W_ND_EMPTY]   = 1'b1;
231         if (CH_LEVEL   == 2'b10) warn_mask[W_CH_EMPTY]   = 1'b1;
232     end
233 
234     wire err_present  = |coffee_err_mask;
235     wire warn_present = |warn_mask;
236 
237     // --------- Quartus-safe priority encoder (NO LOOPS) ---------
238     function automatic [3:0] first_set16(input logic [15:0] m);
239         begin
240             if      (m[0])  first_set16 = 4'd0;
241             else if (m[1])  first_set16 = 4'd1;
242             else if (m[2])  first_set16 = 4'd2;
243             else if (m[3])  first_set16 = 4'd3;
244             else if (m[4])  first_set16 = 4'd4;
245             else if (m[5])  first_set16 = 4'd5;
246             else if (m[6])  first_set16 = 4'd6;
247             else if (m[7])  first_set16 = 4'd7;
248             else if (m[8])  first_set16 = 4'd8;
```

```
            else if (m[9])  first_set16 = 4'd9;
            else if (m[10]) first_set16 = 4'd10;
            else if (m[11]) first_set16 = 4'd11;
            else if (m[12]) first_set16 = 4'd12;
            else if (m[13]) first_set16 = 4'd13;
            else if (m[14]) first_set16 = 4'd14;
            else if (m[15]) first_set16 = 4'd15;
            else            first_set16 = 4'd0;
        end
    endfunction

    function automatic [3:0] next_set16(input logic [15:0] m, input logic [3:0] cur);
        logic [31:0] mm;
        logic [31:0] rot;
        logic [15:0] r16;
        logic [4:0]  sh;
        logic [3:0]  idx;
        begin
            if (m == 16'b0) begin
                next_set16 = cur;
            end else begin
                mm = {m, m};
                sh = {1'b0, cur} + 5'd1;
                rot = (mm >> sh);
                r16 = rot[15:0];

                if (r16 == 16'b0) begin
                    next_set16 = first_set16(m);
                end else begin
                    idx = first_set16(r16);
                    next_set16 = (cur + 4'd1 + idx) & 4'hF;
                end
            end
        end
    endfunction

    reg warn_show;
    reg [3:0] err_code_cur;
    reg [3:0] warn_code_cur;

    always @(posedge CLOCK_50) begin
        if (rst) begin
            warn_show     <= 1'b0;
            err_code_cur  <= 4'd0;
            warn_code_cur <= 4'd0;
        end else if (disp_tick) begin
            if (err_present) begin
                warn_show <= 1'b0;

                if (!coffee_err_mask[err_code_cur])
```

```
299                     err_code_cur <= first_set16(coffee_err_mask);
300                 else
301                     err_code_cur <= next_set16(coffee_err_mask, err_code_cur);
302 
303             end else if (warn_present) begin
304                 warn_show <= ~warn_show;
305 
306                 if (!warn_show) begin
307                     if (!warn_mask[warn_code_cur])
308                         warn_code_cur <= first_set16(warn_mask);
309                     else
310                         warn_code_cur <= next_set16(warn_mask, warn_code_cur);
311                 end
312 
313             end else begin
314                 warn_show <= 1'b0;
315             end
316         end
317     end
318 
319     function automatic [3:0] map_err_code_to_msg(input logic [3:0] code);
320         begin
321             case (code)
322                 E_PAPER_EMPTY:    map_err_code_to_msg = 4'd2;
323                 E_PAPER_NOT_INST: map_err_code_to_msg = 4'd3;
324                 E_PRESS_ERR:      map_err_code_to_msg = 4'd8;
325                 E_PRESS_HIGH:     map_err_code_to_msg = 4'd9;
326                 E_STATUS_ERR:     map_err_code_to_msg = 4'd11;
327 
328                 E_NO_COFFEE0:     map_err_code_to_msg = 4'd12;
329                 E_NO_COFFEE1:     map_err_code_to_msg = 4'd13;
330                 E_NO_CHOC:        map_err_code_to_msg = 4'd14;
331                 E_NO_CREAMER:     map_err_code_to_msg = 4'd15;
332 
333                 default:          map_err_code_to_msg = 4'd11;
334             endcase
335         end
336     endfunction
337 
338     function automatic [3:0] map_warn_code_to_msg(input logic [3:0] code);
339         begin
340             case (code)
341                 W_FILTER_ALMOST: map_warn_code_to_msg = 4'd1;
342                 W_LOW_PRESSURE:  map_warn_code_to_msg = 4'd10;
343 
344                 W_BIN0_EMPTY:    map_warn_code_to_msg = 4'd4;
345                 W_BIN1_EMPTY:    map_warn_code_to_msg = 4'd5;
346                 W_ND_EMPTY:      map_warn_code_to_msg = 4'd6;
347                 W_CH_EMPTY:      map_warn_code_to_msg = 4'd7;
348 
```

```
                default:         map_warn_code_to_msg = 4'd1;
            endcase
        end
    endfunction

    wire [3:0] msg_sel =
        (enjoy_active)              ? 4'd0 : // enjoy overrides inside get_byte()
        (err_present)               ? map_err_code_to_msg(err_code_cur) :
        (warn_present && warn_show) ? map_warn_code_to_msg(warn_code_cur) :
                                      4'd0;

    //============================================================
    // LCD message ROMs
    //============================================================
    reg [7:0] m1_l1 [0:15]  = '{"P","a","p","e","r"," ","F","i","l","t","e","r","
","A","l","m"};
    reg [7:0] m1_l2 [0:15]  = '{"s","t","r","E","m","p","t","y"," "," "," "," "," "," "," ","
"};
    reg [7:0] m2_l1 [0:15]  = '{"P","a","p","e","r"," ","F","i","l","t","e","r"," "," "," ","
"};
    reg [7:0] m2_l2 [0:15]  = '{"E","m","p","t","y"," "," "," "," "," "," "," "," "," "," ","
"};
    reg [7:0] m3_l1 [0:15]  = '{"P","a","p","e","r"," ","F","i","l","t","e","r","
","N","O","T"};
    reg [7:0] m3_l2 [0:15]  = '{"I","N","S","T","A","L","L","E","D"," "," "," "," "," "," ","
"};
    reg [7:0] m4_l1 [0:15]  = '{"C","o","f","f","e","e"," ","B","i","n"," ","0","
","A","l","m"};
    reg [7:0] m4_l2 [0:15]  = '{"s","t","r","E","m","p","t","y"," "," "," "," "," "," "," ","
"};
    reg [7:0] m5_l1 [0:15]  = '{"C","o","f","f","e","e"," ","B","i","n"," ","1","
","A","l","m"};
    reg [7:0] m5_l2 [0:15]  = '{"s","t","r","E","m","p","t","y"," "," "," "," "," "," "," ","
"};
    reg [7:0] m6_l1 [0:15]  = '{"N","o","n","-","D","a","i","r","y"," ","C","r","m","r","
","A"};
    reg [7:0] m6_l2 [0:15]  = '{"l","m","s","t","r","E","m","p","t","y"," "," "," "," "," ","
"};
    reg [7:0] m7_l1 [0:15]  = '{"C","h","o","c","o","l","a","t","e"," ","P","w","d","r","
","A"};
    reg [7:0] m7_l2 [0:15]  = '{"l","m","s","t","r","E","m","p","t","y"," "," "," "," "," ","
"};
    reg [7:0] m8_l1 [0:15]  = '{"W","a","t","e","r"," ","P","r","e","s","s","u","r","e"," ","
"};
    reg [7:0] m8_l2 [0:15]  = '{"E","r","r","o","r","!"," "," "," "," "," "," "," "," "," ","
"};
    reg [7:0] m9_l1 [0:15]  = '{"H","i","g","h","r","W","a","t","e","r"," "," "," "," "," ","
"};
    reg [7:0] m9_l2 [0:15]  = '{"P","r","e","s","s","u","r","e"," "," "," "," "," "," "," ","
"};
    reg [7:0] m10_l1[0:15]  = '{"L","o","w","r","W","a","t","e","r"," "," "," "," "," "," ","
"};
```

```
382     reg [7:0] m10_l2[0:15]  = '{"P","r","e","s","s","u","r","e"," "," "," "," "," "," "," "," "
    "};
383     reg [7:0] m11_l1[0:15]  = '{"H","a","r","d","w","a","r","e"," ","S","t","a","t","u","s","
    "};
384     reg [7:0] m11_l2[0:15]  = '{"E","r","r","o","r"," ","-"," ","S","e","r","v","i","c","e","
    "};
385
386     reg [7:0] m12_l1[0:15]  = '{"N","o"," ","C","o","f","f","e","e"," ","B","i","n"," ","0","
    "};
387     reg [7:0] m12_l2[0:15]  = '{"C","a","n","n","o","t"," ","e","x","e","c","u","t","e"," ","
    "};
388
389     reg [7:0] m13_l1[0:15]  = '{"N","o"," ","C","o","f","f","e","e"," ","B","i","n"," ","1","
    "};
390     reg [7:0] m13_l2[0:15]  = '{"C","a","n","n","o","t"," ","e","x","e","c","u","t","e"," ","
    "};
391
392     reg [7:0] m14_l1[0:15]  = '{"N","o"," ","C","h","o","c","o","l","a","t","e"," "," "," ","
    "};
393     reg [7:0] m14_l2[0:15]  = '{"C","a","n","n","o","t"," ","e","x","e","c","u","t","e"," ","
    "};
394
395     reg [7:0] m15_l1[0:15]  = '{"N","o"," ","C","r","e","a","m","e","r"," "," "," "," "," ","
    "};
396     reg [7:0] m15_l2[0:15]  = '{"C","a","n","n","o","t"," ","e","x","e","c","u","t","e"," ","
    "};
397
398     //============================================================
399     // Normal 2-line message generator (selection + state/progress)
400     //============================================================
401     function automatic [39:0] drink5(input [2:0] t);
402         begin
403             case (t)
404                 3'd0: drink5 = {"M","O","C","H","A"};
405                 3'd1: drink5 = {"L","A","T","T","E"};
406                 3'd2: drink5 = {"E","S","P","R"," "};
407                 3'd3: drink5 = {"A","M","E","R"," "};
408                 3'd4: drink5 = {"D","R","I","P"," "};
409                 default: drink5 = {"U","N","K","N"," "};
410             endcase
411         end
412     endfunction
413
414     function automatic [31:0] size4(input [1:0] s);
415         begin
416             case (s)
417                 2'd0: size4 = {"1","0","o","z"};
418                 2'd1: size4 = {"1","6","o","z"};
419                 2'd2: size4 = {"2","0","o","z"};
420                 default: size4 = {"?","?","o","z"};
421             endcase
```

```
422         end
423     endfunction
424 
425     function automatic [127:0] mk_line1(input logic f, input logic [2:0] t, input logic [1:0]
    s);
426         logic [7:0] fch;
427         begin
428             fch = (f) ? "2" : "1";
429             mk_line1 = {"C", fch, " ", drink5(t), " ", size4(s), " ", " ", " "};
430         end
431     endfunction
432 
433     function automatic [31:0] phase4(input logic [2:0] ph);
434         begin
435             case (ph)
436                 3'd0: phase4 = {"P","A","P","R"};
437                 3'd1: phase4 = {"G","R","N","D"};
438                 3'd2: phase4 = {"C","O","C","O"};
439                 3'd3: phase4 = {"P","O","U","R"};
440                 3'd4: phase4 = {"W","A","T","R"};
441                 default: phase4 = {"D","O","N","E"};
442             endcase
443         end
444     endfunction
445 
446     function automatic [95:0] bar12(input logic [3:0] filled);
447         begin
448             bar12 = {
449                 (filled> 0 ? "#" : "-"),
450                 (filled> 1 ? "#" : "-"),
451                 (filled> 2 ? "#" : "-"),
452                 (filled> 3 ? "#" : "-"),
453                 (filled> 4 ? "#" : "-"),
454                 (filled> 5 ? "#" : "-"),
455                 (filled> 6 ? "#" : "-"),
456                 (filled> 7 ? "#" : "-"),
457                 (filled> 8 ? "#" : "-"),
458                 (filled> 9 ? "#" : "-"),
459                 (filled>10 ? "#" : "-"),
460                 (filled>11 ? "#" : "-")
461             };
462         end
463     endfunction
464 
465     function automatic [127:0] mk_line2(input logic [1:0] st, input logic [2:0] ph, input
    logic [4:0] prog16);
466         logic [3:0] filled12;
467         logic [7:0] tmp_fill;
468         begin
469             case (st)
```

```
470                 2'd0: mk_line2 = {"P","r","e","s","s"," ","S","T","A","R","T"," "," "," "," "," "};
471                 2'd1: mk_line2 = {"H","E","A","T","I","N","G",".",".",".","."," "," "," "," "," "};
472                 2'd2: begin
473                     if (prog16 >= 5'd16) filled12 = 4'd12;
474                     else begin
475                         tmp_fill = (prog16 * 8'd12) / 8'd16;
476                         filled12 = tmp_fill[3:0];
477                     end
478                     mk_line2 = { bar12(filled12), phase4(ph) };
479                 end
480                 default: mk_line2 = {"R","E","A","D","Y"," "," "," "," "," "," "," "," "," "," "," "," "};
481             endcase
482         end
483     endfunction
484 
485     function automatic [7:0] byte16(input [127:0] s, input [4:0] k);
486         begin
487             byte16 = s[8*(15-k) +: 8];
488         end
489     endfunction
490 
491     function automatic [7:0] get_byte;
492         input [3:0] sel;
493         input       line; // 0=line1, 1=line2
494         input [4:0] k;
495         logic [127:0] L1, L2;
496         begin
497             // enjoy override
498             if (enjoy_active) begin
499                 get_byte = line ? enjoy_l2[k] : enjoy_l1[k];
500             end else if (sel == 4'd0) begin
501                 L1 = mk_line1(cur_flavor, cur_type, cur_size);
502                 L2 = mk_line2(sys_state, brew_phase, brew_progress16);
503                 get_byte = (line) ? byte16(L2, k) : byte16(L1, k);
504             end else begin
505                 case (sel)
506                     4'd1:  get_byte = line ? m1_l2[k]  : m1_l1[k];
507                     4'd2:  get_byte = line ? m2_l2[k]  : m2_l1[k];
508                     4'd3:  get_byte = line ? m3_l2[k]  : m3_l1[k];
509                     4'd4:  get_byte = line ? m4_l2[k]  : m4_l1[k];
510                     4'd5:  get_byte = line ? m5_l2[k]  : m5_l1[k];
511                     4'd6:  get_byte = line ? m6_l2[k]  : m6_l1[k];
512                     4'd7:  get_byte = line ? m7_l2[k]  : m7_l1[k];
513                     4'd8:  get_byte = line ? m8_l2[k]  : m8_l1[k];
514                     4'd9:  get_byte = line ? m9_l2[k]  : m9_l1[k];
515                     4'd10: get_byte = line ? m10_l2[k] : m10_l1[k];
516                     4'd11: get_byte = line ? m11_l2[k] : m11_l1[k];
517                     4'd12: get_byte = line ? m12_l2[k] : m12_l1[k];
```

```
518                     4'd13: get_byte = line ? m13_l2[k] : m13_l1[k];
519                     4'd14: get_byte = line ? m14_l2[k] : m14_l1[k];
520                     4'd15: get_byte = line ? m15_l2[k] : m15_l1[k];
521                     default: get_byte = " ";
522                 endcase
523             end
524         end
525     endfunction
526 
527     //============================================================
528     // LCD init/write FSM
529     //============================================================
530     localparam [5:0]
531         S_PWRUP     = 6'd0,
532         S_WAIT1     = 6'd2,
533         S_WAIT2     = 6'd4,
534         S_WAIT3     = 6'd6,
535         S_WAIT_DISPOFF   = 6'd8,
536         S_WAIT_CLR        = 6'd10,
537         S_WAIT_ENTRY      = 6'd12,
538         S_WAIT_DISPON     = 6'd14,
539         S_WAITON          = 6'd16,
540         S_SET_L1    = 6'd17,  S_WAIT_L1 = 6'd18,
541         S_WRITE_L1  = 6'd19,  S_WAIT_WL1= 6'd20,
542         S_SET_L2    = 6'd21,  S_WAIT_L2 = 6'd22,
543         S_WRITE_L2  = 6'd23,  S_WAIT_WL2= 6'd24,
544         S_IDLE      = 6'd25,
545         S_CLR_SW    = 6'd26,  S_WAIT_ENTRY_SW= 6'd27;
546 
547     reg [5:0]  state = S_PWRUP;
548     reg [31:0] dly = 0;
549     reg [4:0]  idx = 0;
550 
551     localparam DLY_15MS   = 32'd750_000;
552     localparam DLY_5MS    = 32'd250_000;
553     localparam DLY_100US  = 32'd5_000;
554     localparam DLY_CMD    = 32'd5_000;
555     localparam DLY_CLEAR  = 32'd75_000;
556 
557     // Change detection
558     reg [3:0] prev_msg_sel;
559     reg       prev_flavor;
560     reg [2:0] prev_type;
561     reg [1:0] prev_size;
562     reg [1:0] prev_sys_state;
563     reg [2:0] prev_brew_phase;
564     reg [4:0] prev_brew_prog;
565 
566     reg       prev_enjoy_active;
567     wire      enjoy_changed = (enjoy_active != prev_enjoy_active);
```

```
568
569     wire msg_changed = (msg_sel != prev_msg_sel);
570
571     wire normal_changed = (msg_sel == 4'd0) &&
572                           ((prev_flavor     != cur_flavor) ||
573                            (prev_type       != cur_type)   ||
574                            (prev_size       != cur_size)   ||
575                            (prev_sys_state  != sys_state)  ||
576                            (prev_brew_phase != brew_phase) ||
577                            (prev_brew_prog  != brew_progress16));
578
579     reg change_detected;
580
581     always @(posedge CLOCK_50) begin
582         if (rst) begin
583             prev_msg_sel    <= 4'd0;
584             prev_flavor     <= 1'b0;
585             prev_type       <= 3'd0;
586             prev_size       <= 2'd0;
587             prev_sys_state  <= 2'd0;
588             prev_brew_phase <= 3'd0;
589             prev_brew_prog  <= 5'd0;
590             prev_enjoy_active <= 1'b0;
591             change_detected <= 1'b0;
592         end else begin
593             if (msg_changed || normal_changed || enjoy_changed)
594                 change_detected <= 1'b1;
595             else if (state == S_CLR_SW)
596                 change_detected <= 1'b0;
597
598             prev_msg_sel    <= msg_sel;
599             prev_flavor     <= cur_flavor;
600             prev_type       <= cur_type;
601             prev_size       <= cur_size;
602             prev_sys_state  <= sys_state;
603             prev_brew_phase <= brew_phase;
604             prev_brew_prog  <= brew_progress16;
605             prev_enjoy_active <= enjoy_active;
606         end
607     end
608
609     always @(posedge CLOCK_50) begin
610         if (rst) begin
611             state <= S_PWRUP;
612             dly   <= 0;
613             idx   <= 0;
614             op    <= 2'b00;
615             din   <= 8'h00;
616             send  <= 1'b0;
617         end else begin
```

```
618             send <= 1'b0;
619
620             case (state)
621                 S_PWRUP: begin
622                     if (dly < DLY_15MS) dly <= dly + 1;
623                     else begin
624                         din   <= 8'h30; op <= 2'b00; send <= 1'b1;
625                         dly   <= 0; state <= S_WAIT1;
626                     end
627                 end
628
629                 S_WAIT1: begin
630                     if (!busy && dly >= DLY_5MS) begin
631                         din   <= 8'h30; op <= 2'b00; send <= 1'b1;
632                         dly   <= 0; state <= S_WAIT2;
633                     end else dly <= dly + 1;
634                 end
635
636                 S_WAIT2: begin
637                     if (!busy && dly >= DLY_100US) begin
638                         din   <= 8'h30; op <= 2'b00; send <= 1'b1;
639                         dly   <= 0; state <= S_WAIT3;
640                     end else dly <= dly + 1;
641                 end
642
643                 S_WAIT3: begin
644                     if (!busy && dly >= DLY_100US) begin
645                         din   <= 8'h38; op <= 2'b00; send <= 1'b1;
646                         dly   <= 0; state <= S_WAIT_DISPOFF;
647                     end else dly <= dly + 1;
648                 end
649
650                 S_WAIT_DISPOFF: begin
651                     if (!busy && dly >= DLY_CMD) begin
652                         din   <= 8'h08; op <= 2'b00; send <= 1'b1;
653                         dly   <= 0; state <= S_WAIT_CLR;
654                     end else dly <= dly + 1;
655                 end
656
657                 S_WAIT_CLR: begin
658                     if (!busy && dly >= DLY_CMD) begin
659                         din   <= 8'h01; op <= 2'b00; send <= 1'b1;
660                         dly   <= 0; state <= S_WAIT_ENTRY;
661                     end else dly <= dly + 1;
662                 end
663
664                 S_WAIT_ENTRY: begin
665                     if (!busy && dly >= DLY_CLEAR) begin
666                         din   <= 8'h06; op <= 2'b00; send <= 1'b1;
667                         dly   <= 0; state <= S_WAIT_DISPON;
```

```
668                     end else dly <= dly + 1;
669                 end
670 
671                 S_WAIT_DISPON: begin
672                     if (!busy && dly >= DLY_CMD) begin
673                         din   <= 8'h0C; op <= 2'b00; send <= 1'b1;
674                         dly   <= 0; state <= S_WAITON;
675                     end else dly <= dly + 1;
676                 end
677 
678                 S_WAITON: begin
679                     if (!busy && dly >= DLY_CMD) begin
680                         state <= S_SET_L1;
681                     end else dly <= dly + 1;
682                 end
683 
684                 S_SET_L1: begin
685                     if (!busy) begin
686                         din <= 8'h80; op <= 2'b00; send <= 1'b1;
687                         dly <= 0; idx <= 0; state <= S_WAIT_L1;
688                     end
689                 end
690 
691                 S_WAIT_L1: begin
692                     if (!busy && dly >= DLY_CMD) state <= S_WRITE_L1;
693                     else dly <= dly + 1;
694                 end
695 
696                 S_WRITE_L1: begin
697                     if (!busy) begin
698                         din <= get_byte(msg_sel, 1'b0, idx);
699                         op  <= 2'b01; send <= 1'b1;
700                         dly <= 0; state <= S_WAIT_WL1;
701                     end
702                 end
703 
704                 S_WAIT_WL1: begin
705                     if (!busy && dly >= DLY_CMD) begin
706                         if (idx < 5'd15) begin
707                             idx <= idx + 5'd1; state <= S_WRITE_L1;
708                         end else begin
709                             idx <= 0; state <= S_SET_L2;
710                         end
711                     end else dly <= dly + 1;
712                 end
713 
714                 S_SET_L2: begin
715                     if (!busy) begin
716                         din <= 8'hC0; op <= 2'b00; send <= 1'b1;
717                         dly <= 0; state <= S_WAIT_L2;
```

```
718                     end
719                 end
720
721                 S_WAIT_L2: begin
722                     if (!busy && dly >= DLY_CMD) state <= S_WRITE_L2;
723                     else dly <= dly + 1;
724                 end
725
726                 S_WRITE_L2: begin
727                     if (!busy) begin
728                         din <= get_byte(msg_sel, 1'b1, idx);
729                         op  <= 2'b01; send <= 1'b1;
730                         dly <= 0; state <= S_WAIT_WL2;
731                     end
732                 end
733
734                 S_WAIT_WL2: begin
735                     if (!busy && dly >= DLY_CMD) begin
736                         if (idx < 5'd15) begin
737                             idx <= idx + 5'd1; state <= S_WRITE_L2;
738                         end else begin
739                             idx <= 0; state <= S_IDLE;
740                         end
741                     end else dly <= dly + 1;
742                 end
743
744                 S_IDLE: begin
745                     if (change_detected && !busy) state <= S_CLR_SW;
746                 end
747
748                 S_CLR_SW: begin
749                     if (!busy) begin
750                         din <= 8'h01; op <= 2'b00; send <= 1'b1;
751                         dly <= 0; state <= S_WAIT_ENTRY_SW;
752                     end
753                 end
754
755                 S_WAIT_ENTRY_SW: begin
756                     if (!busy && dly >= DLY_CLEAR) state <= S_SET_L1;
757                     else dly <= dly + 1;
758                 end
759
760                 default: state <= S_PWRUP;
761             endcase
762         end
763     end
764
765 endmodule
```