src\coffeeSystem.sv

```
1  `timescale 1ns/1ps
2  `include "cmach_recipes.svh"
3
4  //============================================================
5  // Debounce Module Import
6  //============================================================
7  module coffeeSystem #(
8      parameter int CLK_HZ      = 50_000_000,
9      parameter int SPEEDUP_DIV = 1
10 )(
11     input  logic        clk,
12     input  logic        rst,
13
14     input  logic        btn_flavor,
15     input  logic        btn_type,
16     input  logic        btn_size,
17     input  logic        btn_start,
18
19     input  logic [1:0]  PAPER_LEVEL,
20     input  logic [1:0]  BIN0_LEVEL,
21     input  logic [1:0]  BIN1_LEVEL,
22     input  logic [1:0]  ND_LEVEL,
23     input  logic [1:0]  CH_LEVEL,
24
25     input  logic [1:0]  W_PRESSURE,
26     input  logic        W_TEMP,
27     input  logic        STATUS,
28
29     input  logic [$bits(coffee_recipe_t)-1:0] recipes [0:14],
30
31     output logic        cur_flavor,
32     output logic [2:0]  cur_type,
33     output logic [1:0]  cur_size,
34
35     output logic [1:0]  sys_state,
36
37     output logic        HEAT_EN,
38     output logic        POUROVER_EN,
39     output logic        WATER_EN,
40     output logic        GRINDER_0_EN,
41     output logic        GRINDER_1_EN,
42     output logic        PAPER_EN,
43
44     output logic        COCOA_EN,
45     output logic        CREAMER_EN,
46
47     output logic [15:0] err_mask,
48
```

```
49      output logic [2:0]  brew_phase,
50      output logic [4:0]  brew_progress16
51  );
52
53  //=============================================================
54  // Add Debounced Button Signals
55  //=============================================================
56  logic db_flavor, db_type, db_size, db_start;
57
58  debounce #(.CLK_HZ(CLK_HZ)) DB0 (.clk(clk), .rst(rst), .noisy(btn_flavor),
    .clean(db_flavor));
59  debounce #(.CLK_HZ(CLK_HZ)) DB1 (.clk(clk), .rst(rst), .noisy(btn_type),   .clean(db_type));
60  debounce #(.CLK_HZ(CLK_HZ)) DB2 (.clk(clk), .rst(rst), .noisy(btn_size),   .clean(db_size));
61  debounce #(.CLK_HZ(CLK_HZ)) DB3 (.clk(clk), .rst(rst), .noisy(btn_start),  .clean(db_start));
62
63  //=============================================================
64  // Error bit mapping
65  //=============================================================
66  localparam int E_PAPER_NOT_INST = 0;
67  localparam int E_PAPER_EMPTY    = 1;
68  localparam int E_NO_COFFEE0     = 2;
69  localparam int E_NO_COFFEE1     = 3;
70  localparam int E_NO_CREAMER     = 4;
71  localparam int E_NO_CHOC        = 5;
72  localparam int E_PRESS_ERR      = 6;
73  localparam int E_PRESS_HIGH     = 7;
74  localparam int E_STATUS_ERR     = 8;
75
76  //=============================================================
77  // Rising edge detection for *debounced* buttons
78  //=============================================================
79  logic bf_d, bt_d, bs_d, bstart_d;
80  logic bf_p, bt_p, bs_p, bstart_p;
81  logic btns_armed;
82
83  always_ff @(posedge clk or posedge rst) begin
84      if (rst) begin
85          bf_d       <= 1'b0;
86          bt_d       <= 1'b0;
87          bs_d       <= 1'b0;
88          bstart_d   <= 1'b0;
89          btns_armed <= 1'b0;
90      end else begin
91          bf_d     <= db_flavor;
92          bt_d     <= db_type;
93          bs_d     <= db_size;
94          bstart_d <= db_start;
95          btns_armed <= 1'b1;
96      end
97  end
```

```
 98
 99 assign bf_p     = btns_armed & (db_flavor & ~bf_d);
100 assign bt_p     = btns_armed & (db_type   & ~bt_d);
101 assign bs_p     = btns_armed & (db_size   & ~bs_d);
102 assign bstart_p = btns_armed & (db_start  & ~bstart_d);
103
104 //============================================================
105 // MAIN STATE MACHINE
106 //============================================================
107 typedef enum logic [1:0] { S_SELECT=2'd0, S_WAIT=2'd1, S_BREW=2'd2 } state_t;
108 state_t state;
109
110 logic        flavor;
111 logic [2:0]  dType;
112 logic [1:0]  dSize;
113
114 logic        flavor_run;
115 logic [2:0]  dType_run;
116 logic [1:0]  dSize_run;
117
118 always_comb begin
119     sys_state = state;
120     if (state == S_SELECT) begin
121         cur_flavor = flavor;
122         cur_type   = dType;
123         cur_size   = dSize;
124     end else begin
125         cur_flavor = flavor_run;
126         cur_type   = dType_run;
127         cur_size   = dSize_run;
128     end
129 end
130
131 //============================================================
132 // Recipe lookup
133 //============================================================
134 logic [3:0] rIndex;
135 coffee_recipe_t recipe_live;
136
137 always_comb begin
138     rIndex      = ({1'b0,dType} * 4'd3) + {2'b0,dSize};
139     recipe_live = coffee_recipe_t'(recipes[rIndex]);
140 end
141
142 coffee_recipe_t recipe_lat;
143
144 //============================================================
145 // Unpack recipe
146 //============================================================
147 logic       r_load_filter, r_add_creamer;
```

```
148 logic [3:0] r_pour_time, r_hot_water_time, r_grinder_time, r_cocoa_time;
149 logic       _unused_HP;
150
151 always_comb begin
152     {r_load_filter, _unused_HP, r_pour_time, r_hot_water_time,
153      r_grinder_time, r_cocoa_time, r_add_creamer} = recipe_lat;
154 end
155
156 logic       lv_load_filter, lv_add_creamer;
157 logic [3:0] lv_pour_time, lv_hot_water_time, lv_grinder_time, lv_cocoa_time;
158 logic       lv_unused_HP;
159
160 always_comb begin
161     {lv_load_filter, lv_unused_HP, lv_pour_time, lv_hot_water_time,
162      lv_grinder_time, lv_cocoa_time, lv_add_creamer} = recipe_live;
163 end
164
165 //============================================================
166 // System error mask
167 //============================================================
168 logic [15:0] sys_err_mask;
169
170 always_comb begin
171     sys_err_mask = 16'b0;
172
173     if (PAPER_LEVEL == 2'b00)      sys_err_mask[E_PAPER_NOT_INST] = 1'b1;
174     else if (PAPER_LEVEL == 2'b01) sys_err_mask[E_PAPER_EMPTY]    = 1'b1;
175
176     if (W_PRESSURE == 2'b11)      sys_err_mask[E_PRESS_ERR]  = 1'b1;
177     else if (W_PRESSURE == 2'b10) sys_err_mask[E_PRESS_HIGH] = 1'b1;
178
179     if (STATUS == 1'b0) sys_err_mask[E_STATUS_ERR] = 1'b1;
180 end
181
182 wire sys_error_condition = |sys_err_mask;
183
184 //============================================================
185 // Ingredient empty decode
186 //============================================================
187 wire bin0_empty = (BIN0_LEVEL == 2'b00) || (BIN0_LEVEL == 2'b01);
188 wire bin1_empty = (BIN1_LEVEL == 2'b00) || (BIN1_LEVEL == 2'b01);
189 wire nd_empty   = (ND_LEVEL   == 2'b00) || (ND_LEVEL   == 2'b01);
190 wire ch_empty   = (CH_LEVEL   == 2'b00) || (CH_LEVEL   == 2'b01);
191
192 //============================================================
193 // Ingredient errors
194 //============================================================
195 logic [15:0] ing_fail_mask;
196 logic [15:0] ing_err_latch;
197
```

```
198 always_comb begin
199     ing_fail_mask = 16'b0;
200 
201     if (lv_grinder_time != 4'd0) begin
202         if (!flavor && bin0_empty) ing_fail_mask[E_NO_COFFEE0] = 1'b1;
203         if ( flavor && bin1_empty) ing_fail_mask[E_NO_COFFEE1] = 1'b1;
204     end
205 
206     if ((lv_cocoa_time != 4'd0) && ch_empty) ing_fail_mask[E_NO_CHOC] = 1'b1;
207     if (lv_add_creamer && nd_empty)          ing_fail_mask[E_NO_CREAMER] = 1'b1;
208 end
209 
210 assign err_mask = sys_err_mask | ing_err_latch;
211 
212 //============================================================
213 // Time constants
214 //============================================================
215 localparam int TICKS_PER_SEC = (CLK_HZ / SPEEDUP_DIV);
216 localparam int ING_ERR_HOLD_SECS = 2;
217 
218 logic [31:0] ing_tick_cnt;
219 logic [3:0]  ing_secs_left;
220 
221 //============================================================
222 // Brewing phase machine
223 //============================================================
224 typedef enum logic [2:0] {
225     PH_PAPER=3'd0, PH_GRIND=3'd1, PH_COCOA=3'd2, PH_POUR=3'd3, PH_WATER=3'd4, PH_DONE=3'd5
226 } phase_t;
227 
228 phase_t phase;
229 assign brew_phase = phase;
230 
231 logic [31:0] tick_cnt;
232 logic [3:0]  sec_left;
233 
234 //============================================================
235 // Duration helper functions
236 //============================================================
237 function automatic [3:0] phase_duration(input phase_t p);
238     case (p)
239         PH_PAPER: phase_duration = (r_load_filter) ? 4'd1 : 4'd0;
240         PH_GRIND: phase_duration = r_grinder_time;
241         PH_COCOA: phase_duration = r_cocoa_time;
242         PH_POUR:  phase_duration = r_pour_time;
243         PH_WATER: phase_duration = r_hot_water_time;
244         default:  phase_duration = 4'd0;
245     endcase
246 endfunction
247 
```

```
248 function automatic phase_t next_phase(input phase_t p);
249     case (p)
250         PH_PAPER: next_phase = PH_GRIND;
251         PH_GRIND: next_phase = PH_COCOA;
252         PH_COCOA: next_phase = PH_POUR;
253         PH_POUR:  next_phase = PH_WATER;
254         PH_WATER: next_phase = PH_DONE;
255         default:  next_phase = PH_DONE;
256     endcase
257 endfunction
258 
259 function automatic phase_t first_nonzero_phase(input phase_t start_p);
260     phase_t p = start_p;
261     for (int k = 0; k < 6; k++) begin
262         if (p == PH_DONE) begin end
263         else if (phase_duration(p) != 4'd0) begin end
264         else p = next_phase(p);
265     end
266     return p;
267 endfunction
268 
269 //============================================================
270 // Progress 0..16
271 //============================================================
272 logic [7:0] total_sec;
273 logic [7:0] elapsed_sec;
274 
275 function automatic [7:0] calc_total_seconds(input coffee_recipe_t r);
276     logic ld, ac;
277     logic [3:0] pt, hw, gt, ct;
278     logic hp;
279     {ld, hp, pt, hw, gt, ct, ac} = r;
280     calc_total_seconds = (ld ? 8'd1 : 8'd0) + pt + hw + gt + ct;
281     if (calc_total_seconds == 8'd0) calc_total_seconds = 8'd1;
282 endfunction
283 
284 function automatic [4:0] calc_progress16(input [7:0] el, input [7:0] tot);
285     int unsigned q = (el * 16) / ((tot == 0) ? 1 : tot);
286     if (q > 16) q = 16;
287     return q[4:0];
288 endfunction
289 
290 //============================================================
291 // MAIN sequential block
292 //============================================================
293 always_ff @(posedge clk or posedge rst) begin
294     if (rst) begin
295         state          <= S_SELECT;
296 
297         flavor         <= 1'b0;
```

```
298         dType          <= 3'd0;
299         dSize          <= 2'd0;
300
301         flavor_run     <= 1'b0;
302         dType_run      <= 3'd0;
303         dSize_run      <= 2'd0;
304
305         recipe_lat     <= '0;
306
307         phase          <= PH_PAPER;
308         tick_cnt       <= 32'd0;
309         sec_left       <= 4'd0;
310
311         total_sec      <= 8'd1;
312         elapsed_sec    <= 8'd0;
313         brew_progress16<= 5'd0;
314
315         ing_err_latch  <= 16'd0;
316         ing_tick_cnt   <= 32'd0;
317         ing_secs_left  <= 4'd0;
318
319     end else begin
320
321         // INGREDIENT ERROR LATCH LOGIC
322         if (sys_error_condition) begin
323             ing_err_latch <= 16'd0;
324             ing_tick_cnt  <= 32'd0;
325             ing_secs_left <= 4'd0;
326         end else if (state != S_SELECT) begin
327             ing_err_latch <= 16'd0;
328             ing_tick_cnt  <= 32'd0;
329             ing_secs_left <= 4'd0;
330         end else begin
331             if (bf_p || bt_p || bs_p) begin
332                 ing_err_latch <= 16'd0;
333                 ing_tick_cnt  <= 32'd0;
334                 ing_secs_left <= 4'd0;
335             end else if (ing_err_latch != 16'd0) begin
336                 if (ing_secs_left == 4'd0) begin
337                     ing_err_latch <= 16'd0;
338                     ing_tick_cnt  <= 32'd0;
339                 end else begin
340                     if (ing_tick_cnt == (TICKS_PER_SEC-1)) begin
341                         ing_tick_cnt <= 32'd0;
342                         if (ing_secs_left <= 4'd1) begin
343                             ing_err_latch <= 16'd0;
344                             ing_secs_left <= 4'd0;
345                         end else begin
346                             ing_secs_left <= ing_secs_left - 4'd1;
347                         end
```

```
348                     end else begin
349                         ing_tick_cnt <= ing_tick_cnt + 32'd1;
350                     end
351                 end
352             end else begin
353                 ing_tick_cnt  <= 32'd0;
354                 ing_secs_left <= 4'd0;
355             end
356         end
357 
358         // SYSTEM ERROR → RESET TO SELECT
359         if (sys_error_condition) begin
360             state           <= S_SELECT;
361             phase           <= PH_PAPER;
362             tick_cnt        <= 32'd0;
363             sec_left        <= 4'd0;
364             elapsed_sec     <= 8'd0;
365             brew_progress16 <= 5'd0;
366 
367         end else begin
368 
369             // SELECTION HANDLING
370             if (state == S_SELECT) begin
371                 if (bf_p) flavor <= ~flavor;
372                 if (bt_p) dType  <= (dType == 3'd4) ? 3'd0 : (dType + 3'd1);
373                 if (bs_p) dSize  <= (dSize == 2'd2) ? 2'd0 : (dSize + 2'd1);
374             end
375 
376             case (state)
377 
378                 //-------------------------------------------------------------
379                 // SELECT
380                 //-------------------------------------------------------------
381                 S_SELECT: begin
382                     if (bstart_p) begin
383                         if (ing_fail_mask != 16'b0) begin
384                             ing_err_latch <= ing_fail_mask;
385                             ing_tick_cnt  <= 32'd0;
386                             ing_secs_left <= ING_ERR_HOLD_SECS[3:0];
387 
388                             elapsed_sec      <= 8'd0;
389                             brew_progress16  <= 5'd0;
390                             phase            <= PH_PAPER;
391                             sec_left         <= 4'd0;
392                             tick_cnt         <= 32'd0;
393 
394                         end else begin
395                             flavor_run <= flavor;
396                             dType_run  <= dType;
397                             dSize_run  <= dSize;
```

```
398
399                         recipe_lat      <= recipe_live;
400                         total_sec       <= calc_total_seconds(recipe_live);
401                         elapsed_sec     <= 8'd0;
402                         brew_progress16 <= 5'd0;
403
404                         phase           <= PH_PAPER;
405                         sec_left        <= 4'd0;
406                         tick_cnt        <= 32'd0;
407
408                         ing_err_latch   <= 16'd0;
409                         ing_tick_cnt    <= 32'd0;
410                         ing_secs_left   <= 4'd0;
411
412                         state           <= S_WAIT;
413                     end
414                 end
415             end
416
417             //------------------------------------------------------------
418             // WAIT FOR WATER TEMP
419             //------------------------------------------------------------
420             S_WAIT: begin
421                 if (W_TEMP == 1'b1) begin
422                     state       <= S_BREW;
423                     tick_cnt    <= 32'd0;
424                     elapsed_sec <= 8'd0;
425                     brew_progress16 <= 5'd0;
426
427                     phase    <= first_nonzero_phase(PH_PAPER);
428                     sec_left <= phase_duration(first_nonzero_phase(PH_PAPER));
429                 end
430             end
431
432             //------------------------------------------------------------
433             // BREW PROCESS
434             //------------------------------------------------------------
435             S_BREW: begin
436                 if (phase == PH_DONE) begin
437                     state           <= S_SELECT;
438                     phase           <= PH_PAPER;
439                     tick_cnt        <= 32'd0;
440                     sec_left        <= 4'd0;
441                     elapsed_sec     <= total_sec;
442                     brew_progress16 <= 5'd16;
443
444                 end else begin
445                     if (tick_cnt == (TICKS_PER_SEC - 1)) begin
446                         tick_cnt <= 32'd0;
447
```

```
448                             if (elapsed_sec < total_sec)
449                                 elapsed_sec <= elapsed_sec + 8'd1;
450 
451                             brew_progress16 <= calc_progress16(
452                                 ((elapsed_sec < total_sec) ? (elapsed_sec + 8'd1) :
    elapsed_sec),
453                                 total_sec
454                             );
455 
456                             if (sec_left != 4'd0)
457                                 sec_left <= sec_left - 4'd1;
458 
459                             if (sec_left == 4'd1) begin
460                                 phase    <= first_nonzero_phase(next_phase(phase));
461                                 sec_left <=
    phase_duration(first_nonzero_phase(next_phase(phase)));
462                             end
463 
464                         end else begin
465                             tick_cnt <= tick_cnt + 32'd1;
466                         end
467                     end
468                 end
469 
470             endcase
471         end
472     end
473 end
474 
475 //============================================================
476 // Output logic
477 //============================================================
478 always_comb begin
479     HEAT_EN      = 1'b0;
480     POUROVER_EN  = 1'b0;
481     WATER_EN     = 1'b0;
482     GRINDER_0_EN = 1'b0;
483     GRINDER_1_EN = 1'b1;
484     PAPER_EN     = 1'b0;
485     COCOA_EN     = 1'b0;
486     CREAMER_EN   = 1'b0;
487 
488     if (!sys_error_condition) begin
489 
490         if ((state != S_SELECT) && (W_TEMP == 1'b0))
491             HEAT_EN = 1'b1;
492 
493         if (state == S_BREW) begin
494             if (sec_left != 4'd0) begin
495                 case (phase)
```

```
496                     PH_PAPER: PAPER_EN = 1'b1;
497                     PH_GRIND: begin
498                         if (!flavor_run) GRINDER_0_EN = 1'b1;
499                         else             GRINDER_1_EN = 1'b1;
500                     end
501                     PH_COCOA: COCOA_EN = 1'b1;
502                     PH_POUR:  POUROVER_EN = 1'b1;
503                     PH_WATER: WATER_EN = 1'b1;
504                 endcase
505             end
506 
507             if (r_add_creamer)
508                 CREAMER_EN = 1'b1;
509         end
510     end
511 end
512 
513 endmodule
514 
```