

Alinx AX7035t 开发板实验

——基于埃氏筛法的 6 位素数计算与显示

```
1  //////////////////////////////////////
2  module top
3  (
4      input          clk,
5      input          rstn,
6      output [3:0]   led,
7      input  [3:0]   key,
8      output [5:0]   seg_sel,
9      output [7:0]   seg_dig
10 );
11 wire [3:0]         key_signal;
12 wire [3:0]         key_pulse;
13 wire              rstn_signal;
14 wire              tick;
15 wire              one_second;
16 wire              select;
17 wire              reset;
18 wire [47:0]        seg;
19 wire [19:0]         cnt_20b;
20 wire [23:0]         cnt_24d;
21
22 //1秒计时器
23 Count_to_one_second timer(clk,one_second);
24
25 //按键除抖及脉冲
26 Killshake Killshake(clk,rstn,rstn_signal);
27 genvar j;
28 generate for(j = 0; j < 4; j = j + 1) begin
29     Killshake Killshake (clk,key[j],key_signal[j]);
30     Edgedetect Edgedetect (key_signal[j],clk,key_pulse[j]);
31 end
32 endgenerate
33
34 //Led控制及模式选择
35 ledcontrol ledcontrol(clk,rstn_signal,key_pulse,led);
36 modecontrol modecontrol(clk,rstn_signal,one_second,led,key_pulse,reset,select,tick);
37
38 //埃氏筛法
39 isprime solver(clk,reset,tick,select,cnt_20b);
40
41 //显示模块
42 binary_20b_to_bcd_6d transformer(cnt_20b,cnt_24d);
43 genvar i;
44 generate for(i=0; i<6; i=i+1) begin
45     led7seg_decode d(cnt_24d[i*4 +: 4], 1'b1, seg[i*8 +: 8]);
```

```
46     end
47 endgenerate
48 seg_driver #(6) driver(clk, rstn_signal, 6'b111111, seg, seg_sel, seg_dig);
49
50 endmodule
51 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
52 module ledcontrol
53 (
54     input          clk,
55     input          rstn_signal,
56     input  [3:0]   key_pulse,
57     output [3:0]   led
58 );
59
60 reg    [3:0]   led_r;
61 always @(posedge clk or negedge rstn_signal) begin
62     if(~rstn_signal) begin
63         led_r <= 4'b1110;
64     end
65     else if(~key_pulse[0]) begin
66         led_r <= 4'b1110;
67     end
68     else if(~key_pulse[1]) begin
69         led_r <= 4'b1101;
70     end
71     else if(~key_pulse[2]) begin
72         led_r <= 4'b1011;
73     end
74     else if(~key_pulse[3]) begin
75         led_r <= 4'b0111;
76     end
77 end
78 assign led = led_r;
79 endmodule
80 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
81 module modecontrol
82 (
83     input          clk,
84     input          rstn_signal,
85     input          one_second,
86     input  [3:0]   led,
87     input  [3:0]   key_pulse,
88     output         reset,
89     output         select,
90     output         tick
91 );
92
93 reg          tick_r;
94 reg          select_reg;
95 reg  [1:0]   reset_r;
```

```

96
97     always @(posedge clk) begin
98         if(~led[0]) begin
99             tick_r <= one_second;
100             select_reg<=1;
101         end
102         else if(~led[1])begin
103             tick_r <= one_second;
104             select_reg<=0;
105         end
106         else if(~led[2])begin
107             tick_r <= 1;
108             select_reg<=1;
109         end
110         else if(~led[3])begin
111             tick_r <= 1;
112             select_reg<=0;
113         end
114         else begin
115             tick_r<=0;
116         end
117     end
118
119     always @(posedge clk) begin
120         reset_r<={reset_r[0],(&key_pulse)&rstn_signal};
121     end
122
123     assign tick = tick_r;
124     assign select=select_reg;
125     assign reset=reset_r[1];
126
127 endmodule
128 //////////////////////////////////////
129 module Edgedetect
130 (
131     input    key,
132     input    clk,
133     output   pulse
134 );
135
136     reg      key_prev;
137     reg      pulse_reg;
138
139     always @(posedge clk) begin
140         key_prev <= key;
141         if (key_prev & ~key)
142             pulse_reg <= 0;
143         else
144             pulse_reg <= 1;
145     end

```

```

146 assign pulse = pulse_reg;
147
148
149 endmodule
150 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
151 module Killshake
152 (
153     input    clk,
154     input    key,
155     output   signal
156 );
157
158 parameter DEBOUNCE_TIME = 1000000;
159 reg [19:0] count;
160 reg        key_state;
161 reg        signal_reg;
162
163 always @(posedge clk) begin
164     if (key == key_state) begin
165         if (count < DEBOUNCE_TIME)
166             count <= count + 1;
167         else
168             signal_reg <= key_state;
169         end
170     else begin
171         count <= 0;
172         key_state <= key;
173     end
174 end
175
176 assign signal = signal_reg;
177
178 endmodule
179 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
180 module seg_driver #(parameter NPorts=8)
181 (
182     input                clk, rstn,
183     input [NPorts-1:0]   valid_i,
184     input [NPorts*8-1:0] seg_i,
185     output reg [NPorts-1:0] valid_o,
186     output [7:0]          seg_o
187 );
188
189 reg [14:0] cnt;
190 reg [NPorts-1:0] sel;
191 always @(posedge clk or negedge rstn)
192     if(~rstn)
193         cnt <= 0;
194     else
195         cnt <= cnt + 1;

```

```

196
197 always @(posedge clk or negedge rstn)
198     if(~rstn)
199         sel <= 0;
200     else if(cnt == 0)
201         sel <= (sel == NPorts - 1) ? 0 : sel + 1;
202
203 always @(sel, valid_i) begin
204     valid_o = {NPorts{1'b1}};
205     valid_o[sel] = ~valid_i[sel];
206 end
207
208 assign seg_o = ~seg_i[sel*8+:8];
209
210 endmodule
211
212 //////////////////////////////////////
213 module led7seg_decode
214 (
215     input [3:0] digit,
216     input valid,
217     output reg [7:0] seg
218 );
219 always @(digit)
220     if(valid)
221         case(digit)
222             0: seg = 8'b00111111; //0
223             1: seg = 8'b00000110; //1
224             2: seg = 8'b01011011; //2
225             3: seg = 8'b01001111; //3
226             4: seg = 8'b01100110; //4
227             5: seg = 8'b01101101; //5
228             6: seg = 8'b01111101; //6
229             7: seg = 8'b00000111; //7
230             8: seg = 8'b01111111; //8
231             9: seg = 8'b01101111; //9
232             default: seg = 0;
233         endcase
234     else seg = 8'd0;
235 endmodule
236
237 //////////////////////////////////////
238 module isprime #(parameter N=999999)
239 (
240     input clk,rstn,tick,
241     input select,
242     output [19:0] cnt_20b
243 );
244 reg [19:0] w_addr;
245 reg w_data;
246 reg wea;
247 reg [19:0] r_addr;

```

```
246     wire                r_data;
247
248     reg [19:0]          i;
249     reg [19:0]          j;
250     reg                 en;
251     reg                 done;
252
253
254     reg [19:0]          cnt_temp_reg;
255     reg [19:0]          cnt_20b_reg;
256
257     reg [2:0]           timer;
258     reg                 hold;
259
260     always @(posedge clk or negedge rstn) begin
261         if(!rstn) begin
262             cnt_temp_reg<=(select)?2:N;
263             cnt_20b_reg<=(select)?2:N;
264             wea<=0;
265             i<=2;
266             j<=0;
267             en<=0;
268             done<=0;
269             timer<=0;
270             hold<=1;
271         end
272         else if (i*i<=N) begin
273             if(en==0)begin
274                 r_addr<=i;
275                 if(timer>2)begin
276                     timer<=0;
277                     hold<=0;
278                 end
279                 else begin
280                     timer<=timer+1;
281                     hold<=1;
282                 end
283                 if(!hold) begin
284                     if (r_data==0) begin
285                         en<=1;
286                         j<=i+i;
287                     end
288                     else begin
289                         i<=i+1;
290                     end
291                 end
292             end
293             else if(en==1) begin
294                 if(j<N)begin
295                     wea<=1;
```

```
296         w_addr<=j;
297         w_data<=1;
298         j<=j+i;
299     end
300     else begin
301         wea<=0;
302         en<=0;
303         i<=i+1;
304     end
305 end
306 end
307 else begin
308     done<=1;
309     if(done) begin
310         if (((cnt_temp_reg<N)&(select))|((cnt_temp_reg>=2)&(~select))) begin
311             r_addr<=cnt_temp_reg;
312             if(hold) begin
313                 if(timer>2)begin
314                     timer<=0;
315                     hold<=0;
316                 end
317                 else begin
318                     timer<=timer+1;
319                 end
320             end
321             else begin
322                 if ((~r_data)&tick) begin
323                     cnt_20b_reg<=cnt_temp_reg;
324                     cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
325                     hold<=1;
326                 end
327                 else if ((r_data)) begin
328                     cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
329                     hold<=1;
330                 end
331                 else if((~r_data)&(~tick)) begin
332                     cnt_temp_reg<=cnt_temp_reg;
333                 end
334             end
335         end
336     end
337 end
338 end
339 assign cnt_20b=cnt_20b_reg;
340 ram_ip ram_ip_inst_1
341 (
342     .clka      (clk      ),
343     .wea       (wea      ),
344     .addra     (w_addr   ),
345     .dina      (w_data   ),
```

```

346 .clkb      (clk      ),
347 .addrb     (r_addr   ),
348 .doutb     (r_data   )
349 );
350 endmodule
351 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
352 module binary_20b_to_bcd_6d #(parameter N = 20,parameter M = 24)
353 (
354     input  [N-1:0]  input_20b,
355     output [M-1:0]  output_6d
356 );
357 reg [3:0]          digits [5:0];
358
359 integer i;
360 always @(input_20b) begin
361     for (i = 0; i < 6; i = i+1) begin
362         digits[i] = 4'd0;
363     end
364     for(i = N-1; i >= 0; i = i-1) begin //加3移位法
365         if (digits[0] >= 4'b0101) digits[0] = digits[0] + 4'b0011;
366         if (digits[1] >= 4'b0101) digits[1] = digits[1] + 4'b0011;
367         if (digits[2] >= 4'b0101) digits[2] = digits[2] + 4'b0011;
368         if (digits[3] >= 4'b0101) digits[3] = digits[3] + 4'b0011;
369         if (digits[4] >= 4'b0101) digits[4] = digits[4] + 4'b0011;
370         if (digits[5] >= 4'b0101) digits[5] = digits[5] + 4'b0011;
371
372         digits[5][3:0] = {digits[5][2:0], digits[4][3]};
373         digits[4][3:0] = {digits[4][2:0], digits[3][3]};
374         digits[3][3:0] = {digits[3][2:0], digits[2][3]};
375         digits[2][3:0] = {digits[2][2:0], digits[1][3]};
376         digits[1][3:0] = {digits[1][2:0], digits[0][3]};
377         digits[0][3:0] = {digits[0][2:0], input_20b[i]};
378     end
379 end
380
381 assign output_6d ={digits[5],digits[4],digits[3],digits[2],digits[1],digits[0]};
382
383 endmodule
384 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
385 module Count_to_one_second #(parameter Count_To = 50_000_000)
386 (
387     input  clk,
388     output one_second
389 );
390
391 reg [31:0]  counter;
392 reg        one_second_r;
393
394 always @(posedge clk) begin
395     if((counter < Count_To) && ~one_second_r) begin

```



```
396         counter <= counter + 1;
397     end
398     else if((counter < Count_To) && one_second_r) begin
399         one_second_r <= 0;
400         counter <= counter + 1;
401     end
402     else begin
403         counter <= 0;
404         one_second_r <= 1;
405     end
406 end
407
408 assign one_second = one_second_r;
409
410 endmodule
```