数字逻辑电路

课程设计

**2023. 9.4~2023.9.8（第1周）**

**课题： 模拟自动售货机**

**学号： 19210230 19210231**

**姓名： 袁睿 徐洋**

**指导老师： 黄为民老师**

## 实验目的

1. 设计一个模拟自动售货机，此机能出售1元、2元、5元、10元的四种商品。出售哪种商品可有顾客按动相应的一个按键即可，并同时用数码管显示出此商品的价格。

2. 顾客投入硬币的钱数也是有1元、2元、5元、10元四种，但每次只能投入其中的一种硬币，此操作通过按动相应的一个按键来模拟，并同时用数码管将投币额显示出来。

3．顾客投币后，按一次确认键，如果投币额不足时则报警，报警时间3秒（可用LED灯模拟报警）。如果投币额足够时自动送出货物（送出的货物用相应不同的LED灯显示来模拟），同时多余的钱应找回，找回的钱数用数码管显示出来。

4．顾客一旦按动确认键3秒后，自动售货机即可自动恢复到初始状态，此时才允许顾客进行下一次购货操作。

5．售货机还应具有供商家使用的累加卖货额的功能，累加的钱数要用数码管显示，显示2位即可。此累加器只有商家可以控制清零。

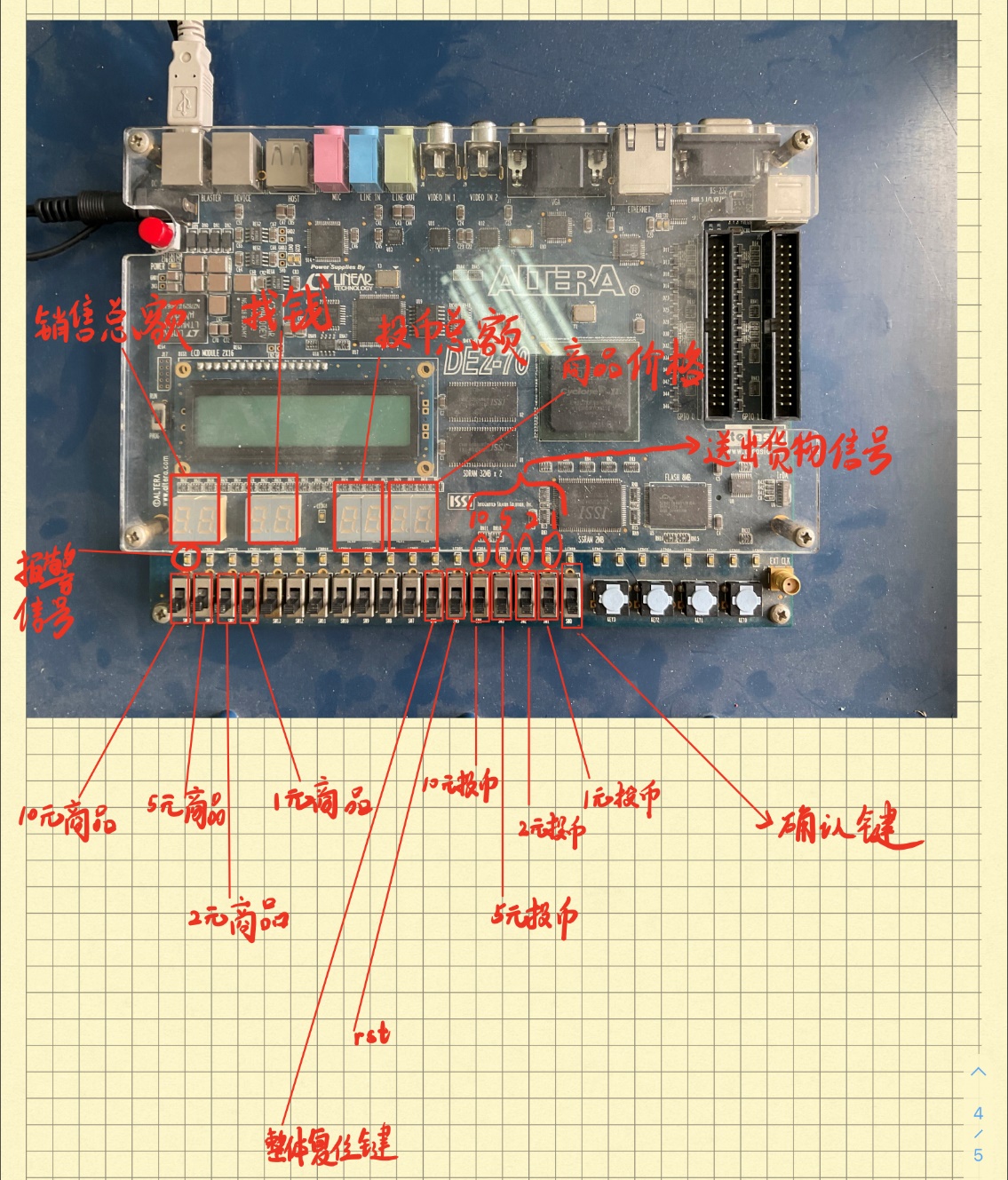
6. 此售货机要设有一个由商家控制的整体复位控制。

## 实验原理

整个数字系统的核心是控制模块（Control Module），其思想基于有限状态机，通过在不同状态的转换中完成选择、投币、找零计算、投币不足时报警、交易成功状态的转换等操作。在控制模块输出的需要显示在七段数码管上的数据需要有一个将二进制转换为BCD码的识别转换模块（BCDconvertModule），最后这些处理好的数据通过显示模块（DisplayModule）进行在七段数码管上显示。对于时钟信号clk，其频率是50MHZ，需要一个分频模块（clk\_1hz）来将频率降为1hz。整个系统需要一个计时模块（集成在控制模块，作为一个等待状态实现）来实现按下确认键后三秒复位/警报三秒的功能。

## 实现功能

1. 功能按键



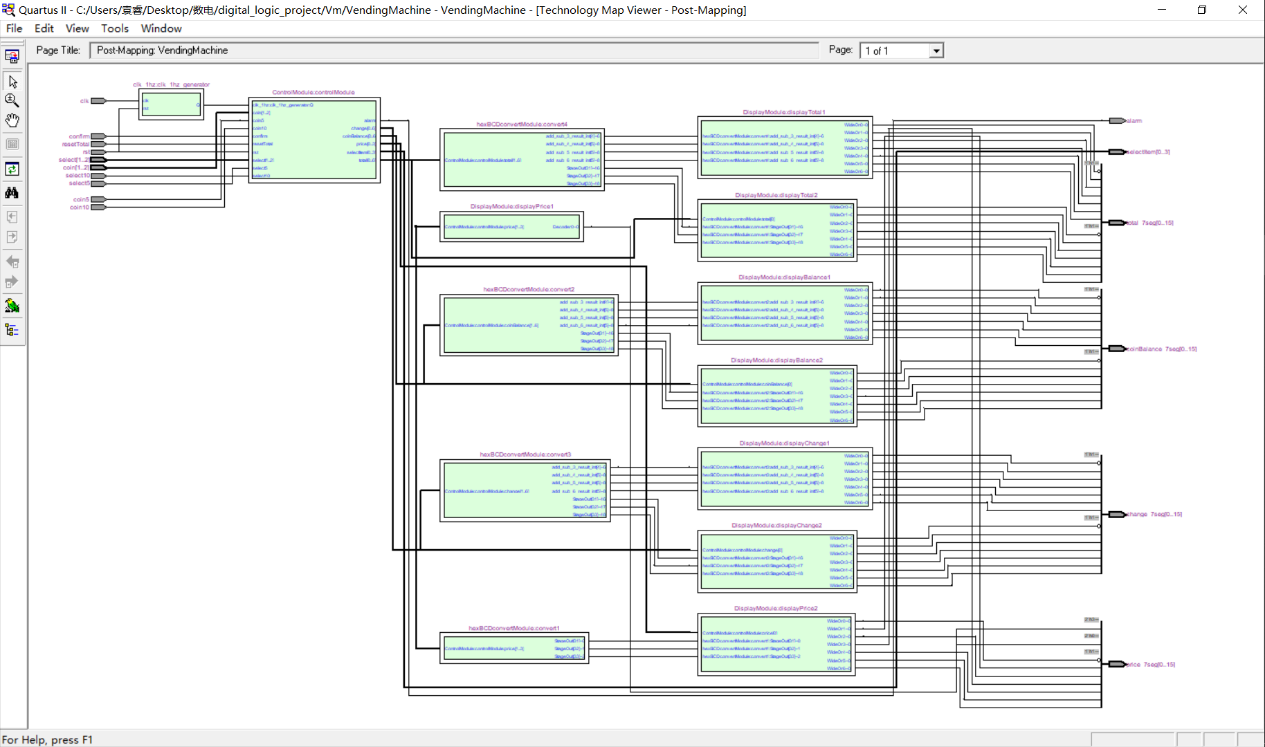
1. 功能描述

将复位rst键（或者称为使能端）先调至1（低电平使能，调至低电平时会清空当前选择商品的价格，投入硬币的总量，但是不会清空总销售额，此时整个数字系统暂停运作），此时选择一个价格的商品（对应开关上拨后下拨记为一次选择，后续开关同理），该商品的价格会在对应的数码管上显示，此时系统等待我们投币，我们选择需要投币的数额，上拨开关再下拨记为一次投币，每次只能投入一种硬币，数码管会实时显示投币总额。按下确认按钮后，会出现两种情况：1）当投币数额大于等于商品价格时，对应商品的LED灯点亮表示出货，数码管显示需要找客户的钱，同时更新累加总销售额。三秒后自动售货机恢复初始状态，可以进行下一次选择商品并购买的操作。2）当投币额小于商品价格时，警报LED灯亮起，找钱的数额即为投入硬币的数额，代表售货机将钱退给客户，本次购买失败。三秒后售货机恢复初始状态。

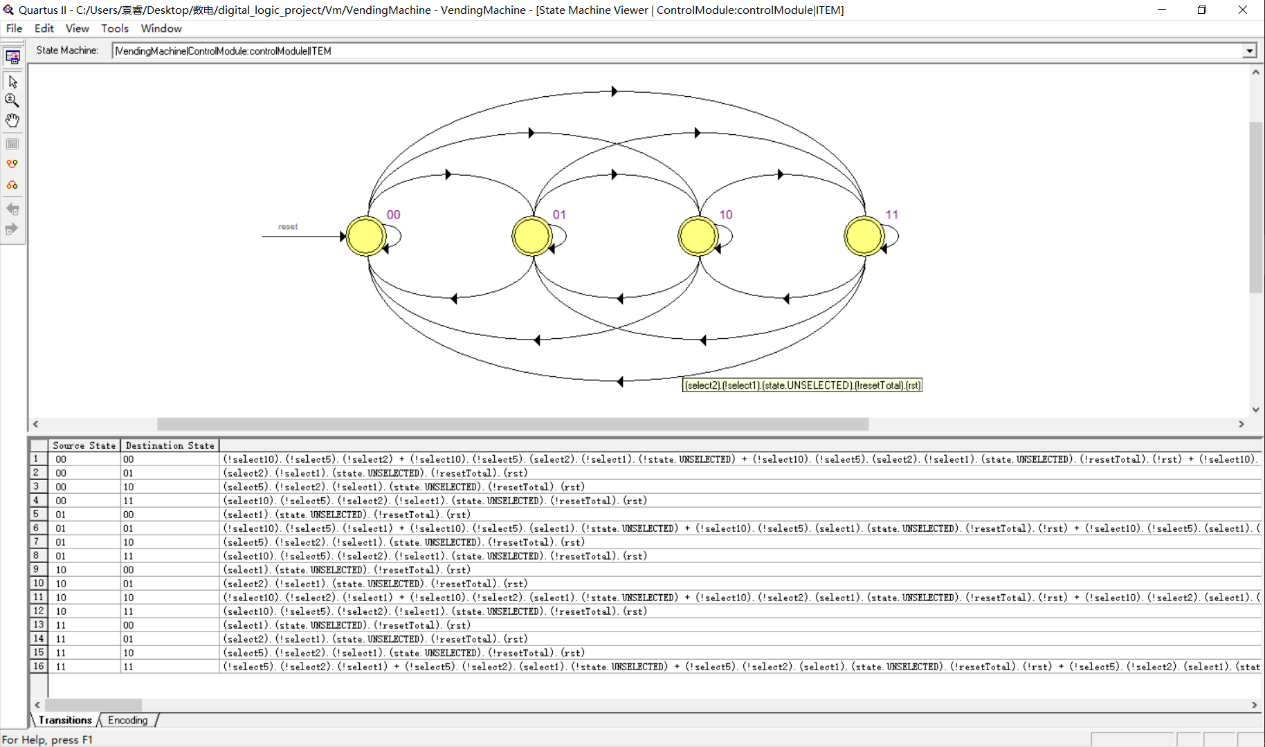
对于总销售额，在自动售货机正常运行时会不断累加，只有当商家拨动整体复位按钮时才会清零，代表商家将赚到的钱从售货机中拿出。

## 分析与设计思路

1. 电路图



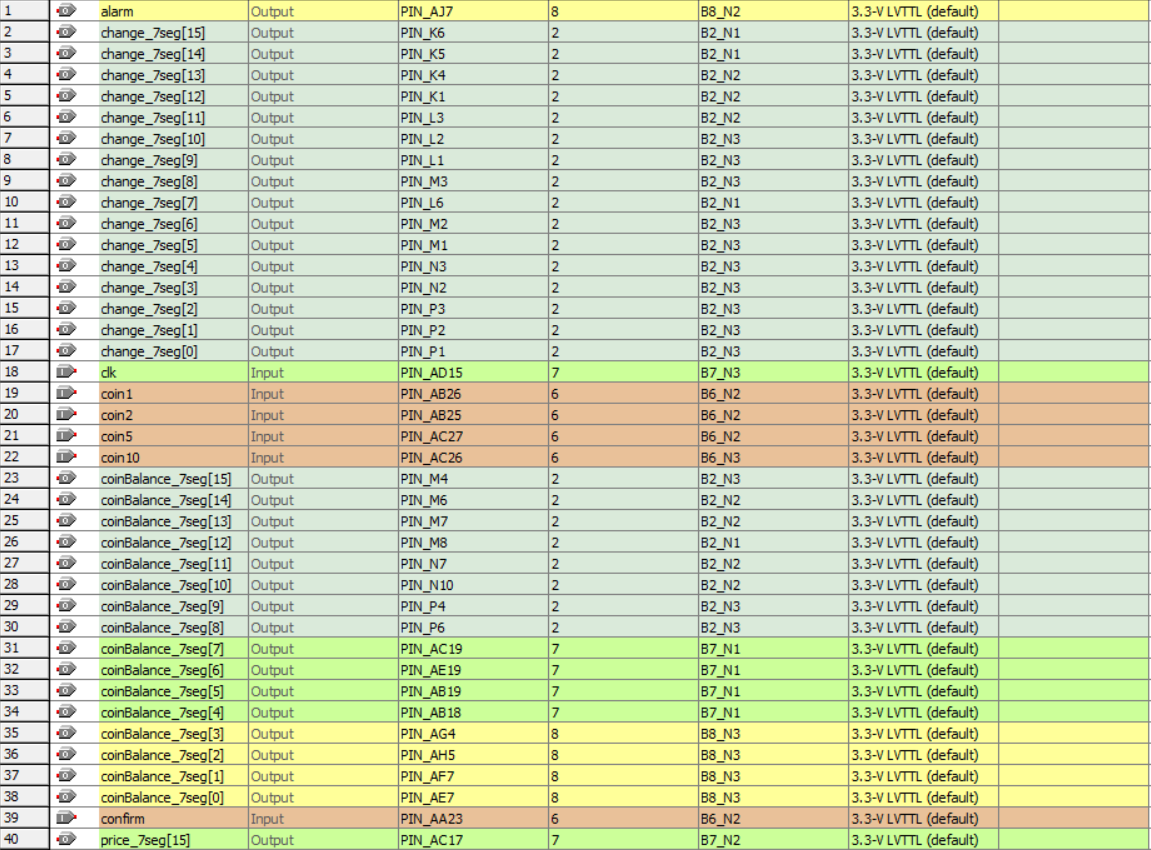
1. 状态转换图

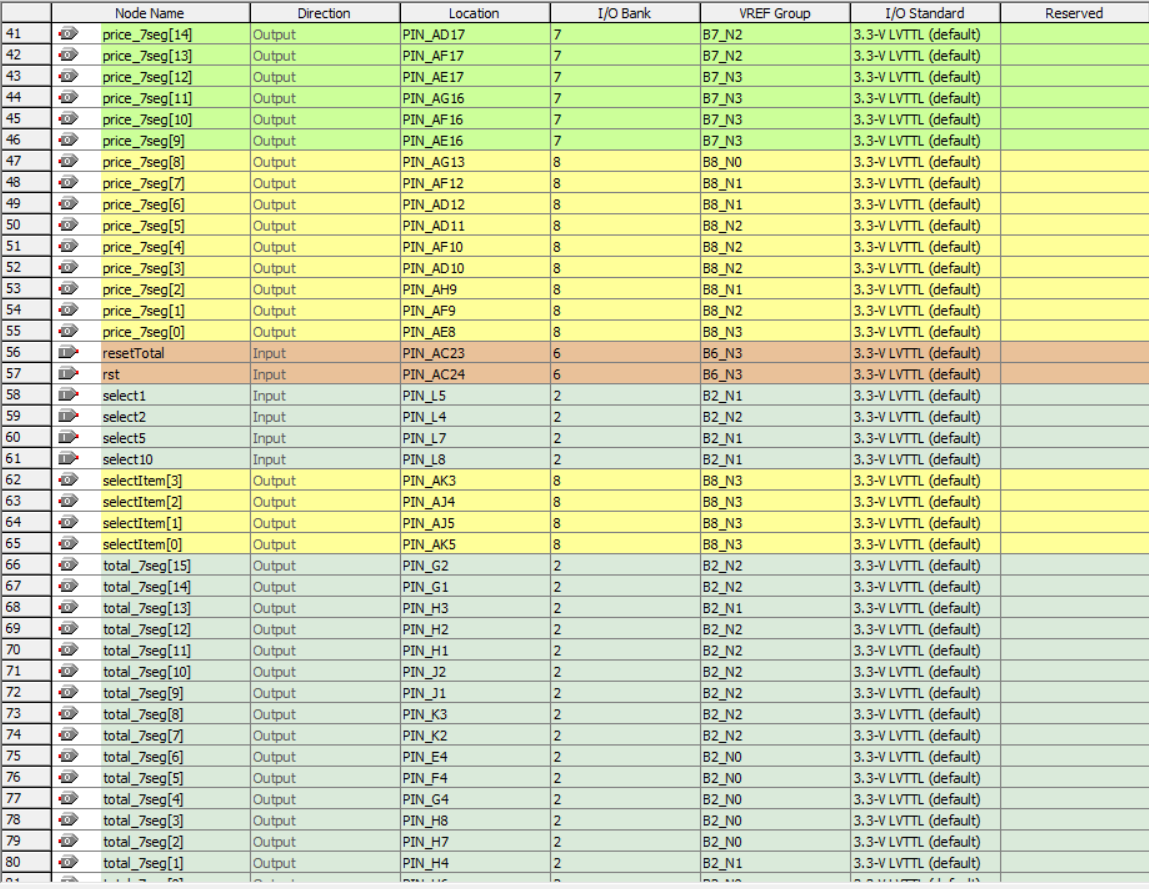


## 设计代码

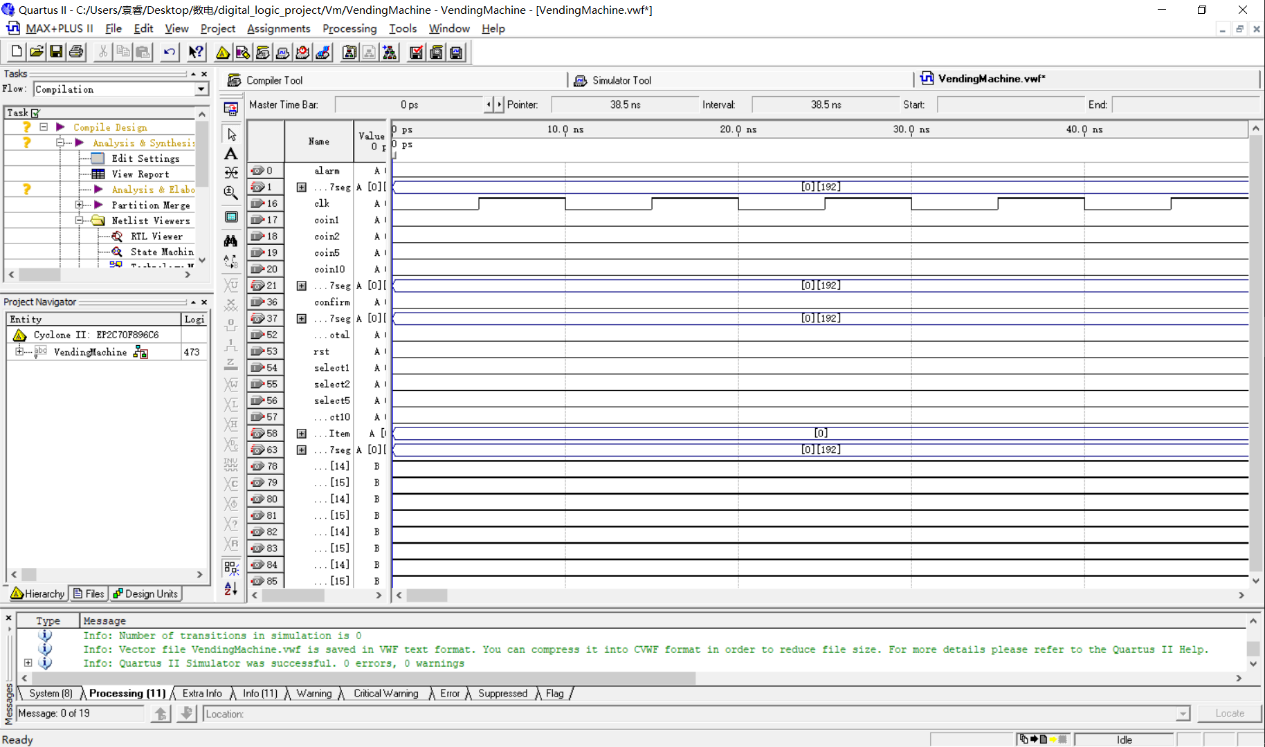
1. 分频模块
2. module clk\_1hz(CLK, RST, clk\_1Hz);
3. input CLK, RST;
4. output clk\_1Hz;
5. wire clk\_1MHz, clk\_100KHz, clk\_10KHz, clk\_1KHz, clk\_100Hz, clk\_10Hz;
6. //assign clock = {clk\_1MHz, clk\_100KHz, clk\_10KHz, clk\_1KHz, clk\_100Hz, clk\_10Hz, clk\_1Hz};
7. divide\_by\_50 d6 (clk\_1MHz, CLK, RST);
8. divide\_by\_10 d5 (clk\_100KHz, clk\_1MHz, RST);
9. divide\_by\_10 d4 (clk\_10KHz, clk\_100KHz, RST);
10. divide\_by\_10 d3 (clk\_1KHz, clk\_10KHz, RST);
11. divide\_by\_10 d2 (clk\_100Hz, clk\_1KHz, RST);
12. divide\_by\_10 d1 (clk\_10Hz, clk\_100Hz, RST);
13. divide\_by\_10 d0 (clk\_1Hz, clk\_10Hz, RST);
15. endmodule
17. module divide\_by\_10 (Q, CLK, RST,);
18. input CLK, RST;
19. output Q;
20. reg Q;
21. reg [2:0] count;
23. always @(posedge CLK or negedge RST)
24. begin
25. **if** (~RST)
26. begin
27. Q <= 1'b0;
28. count <= 3'b000;
29. end
30. **else** **if** (count < 4)
31. begin
32. count <= count + 1'b1;
33. end
34. **else**
35. begin
36. count <= 3'b000;
37. Q <= ~Q;
38. end
39. end
41. endmodule
43. module divide\_by\_50 (Q, CLK, RST);
44. input CLK, RST;
45. output Q;
46. reg Q;
47. reg [4:0] count;
49. always @(posedge CLK or negedge RST)
50. begin
51. **if** (~RST)
52. begin
53. Q <= 1'b0;
54. count <= 5'b00000;
55. end
56. **else** **if** (count < 24)
57. begin
58. count <= count + 1'b1;
59. end
60. **else**
61. begin
62. count <= 5'b00000;
63. Q <= ~Q;
64. end
65. end
67. endmodule
68. 控制模块
69. module ControlModule(
70. input wire clk,
71. input wire rst,
72. //input wire [1:0] state,
73. //input wire [3:0] price,
74. //input wire [3:0] coinBalance,
75. // input wire [1:0] selectedProduct,
76. input wire coin1,
77. input wire coin2,
78. input wire coin5,
79. input wire coin10,
80. input wire select1,
81. input wire select2,
82. input wire select5,
83. input wire select10,
84. input wire confirm,
85. input wire resetTotal,
86. output reg[6:0] price,
87. output reg[6:0] coinBalance,
88. output reg[3:0] selectItem,
89. output reg alarm,
90. // output reg vendSuccess,
91. output reg[6:0] change,
92. output reg[6:0] total
93. );
94. parameter UNSELECTED = 2'b00, SELECTED= 2'b01, WAIT\_3SEC = 2'b10;
95. // temp var
96. reg[1:0] ITEM; // 0 -> 1  1 -> 2  2 -> 5  3 -> 10
97. //reg [6:0] coinBalance\_ ;
98. //reg[6:0] change\_;
99. //reg[6:0] total\_;
100. reg[1:0] state = UNSELECTED;
102. reg[1:0] counter = 0;
104. always @(posedge clk or negedge rst)
105. begin
106. **if**(~rst)
107. begin
108. price <= 7'h0;
109. //coinBalance\_ <= 7'd0;
110. alarm <= 1'b0;
111. state <= UNSELECTED;
112. coinBalance <= 7'b0;
113. selectItem <= 7'b0;
114. change <= 7'b0;
115. //total <= 0;
116. end
117. **else** **if**(resetTotal)
118. begin
119. //total\_ <= 7'b0;
120. total <= 7'b0;
121. end
122. **else**
123. begin
124. **case**(state)
125. UNSELECTED:
126. begin
127. // init
128. price <= 7'b0;
129. //coinBalance\_ <= 7'b0;
130. alarm <= 1'b0;
131. state <= UNSELECTED;
132. coinBalance <= 7'b0;
133. selectItem <= 4'b0;
134. change <= 7'b0;
135. counter <= 0;
136. // select
137. **if**(select1)
138. begin
139. price <= 7'b0000001;
140. state <= SELECTED;
141. ITEM <= 2'd0;
142. end
143. **else** **if**(select2)
144. begin
145. price <= 7'b0000010;
146. state <= SELECTED;
147. ITEM <= 2'd1;
148. end
149. **else** **if**(select5)
150. begin
151. price <= 7'b0000101;
152. state <= SELECTED;
153. ITEM <= 2'd2;
154. end
155. **else** **if**(select10)
156. begin
157. price <= 7'b0001010;
158. state <= SELECTED;
159. ITEM <= 2'd3;
160. end
161. end
162. SELECTED:
163. begin
164. **if**(coin1)
165. begin
166. //coinBalance\_ <= coinBalance\_ + 1;
167. coinBalance <= coinBalance + 1;
168. //hexBCDconvertModule converter1(.hexNum(coinBalance), .hexBCD(coinBalance\_BCD));
169. end
170. **else** **if**(coin2)
171. begin
172. //coinBalance\_ <= coinBalance\_ + 2;
173. coinBalance <= coinBalance + 2;
174. //hexBCDconvertModule converter2(.hexNum(coinBalance), .hexBCD(coinBalance\_BCD));
175. end
176. **else** **if**(coin5)
177. begin
178. //coinBalance\_ <= coinBalance\_ + 5;
179. coinBalance <= coinBalance + 5;
180. //hexBCDconvertModule converter3(.hexNum(coinBalance), .hexBCD(coinBalance\_BCD));
181. end
182. **else** **if**(coin10)
183. begin
184. //coinBalance\_ <= coinBalance\_ + 10;
185. coinBalance <= coinBalance + 10;
186. //hexBCDconvertModule converter4(.hexNum(coinBalance), .hexBCD(coinBalance\_BCD));
187. end
188. **if**(confirm)
189. begin
190. **if**(price <= coinBalance)
191. begin
192. //change\_ <= coinBalance - price;
193. //hexBCDconvertModule converter5(.hexNum(change), .hexBCD(change\_BCD));
194. total <= total + price;
195. change <= coinBalance - price;
196. //hexBCDconvertModule converter6(.hexNum(total), .hexBCD(total\_BCD));
197. //total <= total\_;
198. //change <= change\_;
199. **case**(ITEM)
200. 0: selectItem <= 4'b0001;
201. 1: selectItem <= 4'b0010;
202. 2: selectItem <= 4'b0100;
203. 3: selectItem <= 4'b1000;
204. endcase
205. //total <= total\_;
206. state <= WAIT\_3SEC;
207. end
208. **else**
209. begin
210. change <= coinBalance;
211. state <= WAIT\_3SEC;
212. alarm <= 1'b1;
214. //selectItem <= selectItem & 4'b0000;
215. end
216. end
217. end
218. // time counter  3 sec
219. WAIT\_3SEC:
220. begin
221. **if**(~rst) counter <= 0;
222. **if**(counter == 3)
223. begin
224. counter <= 0;
225. **if**(~alarm)
226. state <= UNSELECTED;
227. **else**
228. begin
229. state <= UNSELECTED;
230. end
231. end
232. **else**
233. counter <= counter + 1;
234. end
235. **default**: state <= UNSELECTED;
236. endcase
237. end
238. end
239. endmodule
240. BCD码转换模块
241. module BCDconvertModule(input[6:0] Num, output reg[7:0] BCD);
242. always@(Num)
243. begin
244. BCD[7:4] = Num / 10;
245. BCD[3:0] = Num % 10;
246. end
247. endmodule
248. 显示模块
249. module DisplayModule(input [3:0]displayNum, output reg[7:0]seg);
250. always @(displayNum)
251. begin
252. seg[7] = 1'b1;
253. case(displayNum)
254. 4'h0: seg[6:0] = ~7'h3F;
255. 4'h1: seg[6:0] = ~7'h06;
256. 4'h2: seg[6:0] = ~7'h5B;
257. 4'h3: seg[6:0] = ~7'h4F;
258. 4'h4: seg[6:0] = ~7'h66;
259. 4'h5: seg[6:0] = ~7'h6D;
260. 4'h6: seg[6:0] = ~7'h7D;
261. 4'h7: seg[6:0] = ~7'h07;
262. 4'h8: seg[6:0] = ~7'h7F;
263. 4'h9: seg[6:0] = ~7'h6F;
264. 4'ha: seg[6:0] = ~7'h77;
265. 4'hb: seg[6:0] = ~7'h7C;
266. 4'hc: seg[6:0] = ~7'h39;
267. 4'hd: seg[6:0] = ~7'h5E;
268. 4'he: seg[6:0] = ~7'h79;
269. 4'hf: seg[6:0] = ~7'h71;
270. endcase
271. end
272. endmodule
273. 顶层模块
274. module VendingMachine(
275. input wire clk,           // 时钟信号
276. input wire rst,           // 复位信号
277. input wire coin1,         // 1元硬币输入信号
278. input wire coin2,         // 2元硬币输入信号
279. input wire coin5,         // 5元硬币输入信号
280. input wire coin10,        // 10元硬币输入信号
281. input wire select1,       // 选择商品1信号
282. input wire select2,       // 选择商品2信号
283. input wire select5,       // 选择商品5信号
284. input wire select10,      // 选择商品10信号
285. input wire confirm,       // 确认购买信号
286. input wire resetTotal,    // 清零销售总额信号
288. output[15:0] price\_7seg, // the price of item, printed in 7seg
289. output[15:0] change\_7seg,
290. output[15:0] coinBalance\_7seg,
291. output[15:0] total\_7seg,
292. output[3:0] selectItem,  // LED reveal the successfully bought item
293. output alarm       // 报警信号
294. // output vendSuccess,  // 购买成功信号
295. // output [3:0] change,  // 找零信号
296. // output [3:0] total    // 销售总额
298. );
299. wire clk\_1hz;
301. wire[7:0] price\_BCD;
302. wire[7:0] coinBalance\_BCD;
303. wire[7:0] change\_BCD;
304. wire[7:0] total\_BCD;
306. wire[6:0] price;
307. wire[6:0] coinBalance;
308. wire[6:0] change;
309. wire[6:0] total;
311. clk\_1hz clk\_1hz\_generator(.CLK(clk), .RST(rst), .clk\_1Hz(clk\_1hz));
313. ControlModule controlModule(
314. .clk(clk\_1hz),
315. //.clk(clk),
316. .rst(rst),
317. //.state(state),
318. //.price(price),
319. //.coinBalance(coinBalance),
320. .coin1(coin1),
321. .coin2(coin2),
322. .coin5(coin5),
323. .coin10(coin10),
324. .select1(select1),
325. .select2(select2),
326. .select5(select5),
327. .select10(select10),
328. .confirm(confirm),
329. .resetTotal(resetTotal),
330. .price(price),
331. .coinBalance(coinBalance),
332. .selectItem(selectItem),
333. .alarm(alarm),
334. //.vendSuccess(vendSuccess),
335. .change(change),
336. .total(total));
338. BCDconvertModule convert1(.Num(price),
339. .BCD(price\_BCD));
340. BCDconvertModule convert2(.Num(coinBalance),
341. .BCD(coinBalance\_BCD));
342. BCDconvertModule convert3(.Num(change),
343. .BCD(change\_BCD));
344. BCDconvertModule convert4(.Num(total),
345. .BCD(total\_BCD));
347. // 1 -**>** shi wei  2 -**>** ge wei
348. DisplayModule displayPrice1(.displayNum(price\_BCD[7:4]),
349. .seg(price\_7seg[15:8]));
350. DisplayModule displayPrice2(.displayNum(price\_BCD[3:0]),
351. .seg(price\_7seg[7:0]));
352. DisplayModule displayBalance1(.displayNum(coinBalance\_BCD[7:4]),
353. .seg(coinBalance\_7seg[15:8]));
354. DisplayModule displayBalance2(.displayNum(coinBalance\_BCD[3:0]),
355. .seg(coinBalance\_7seg[7:0]));
356. DisplayModule displayChange1(.displayNum(change\_BCD[7:4]),
357. .seg(change\_7seg[15:8]));
358. DisplayModule displayChange2(.displayNum(change\_BCD[3:0]),
359. .seg(change\_7seg[7:0]));
360. DisplayModule displayTotal1(.displayNum(total\_BCD[7:4]),
361. .seg(total\_7seg[15:8]));
362. DisplayModule displayTotal2(.displayNum(total\_BCD[3:0]),
363. .seg(total\_7seg[7:0]));
364. endmodule

## 管脚分配



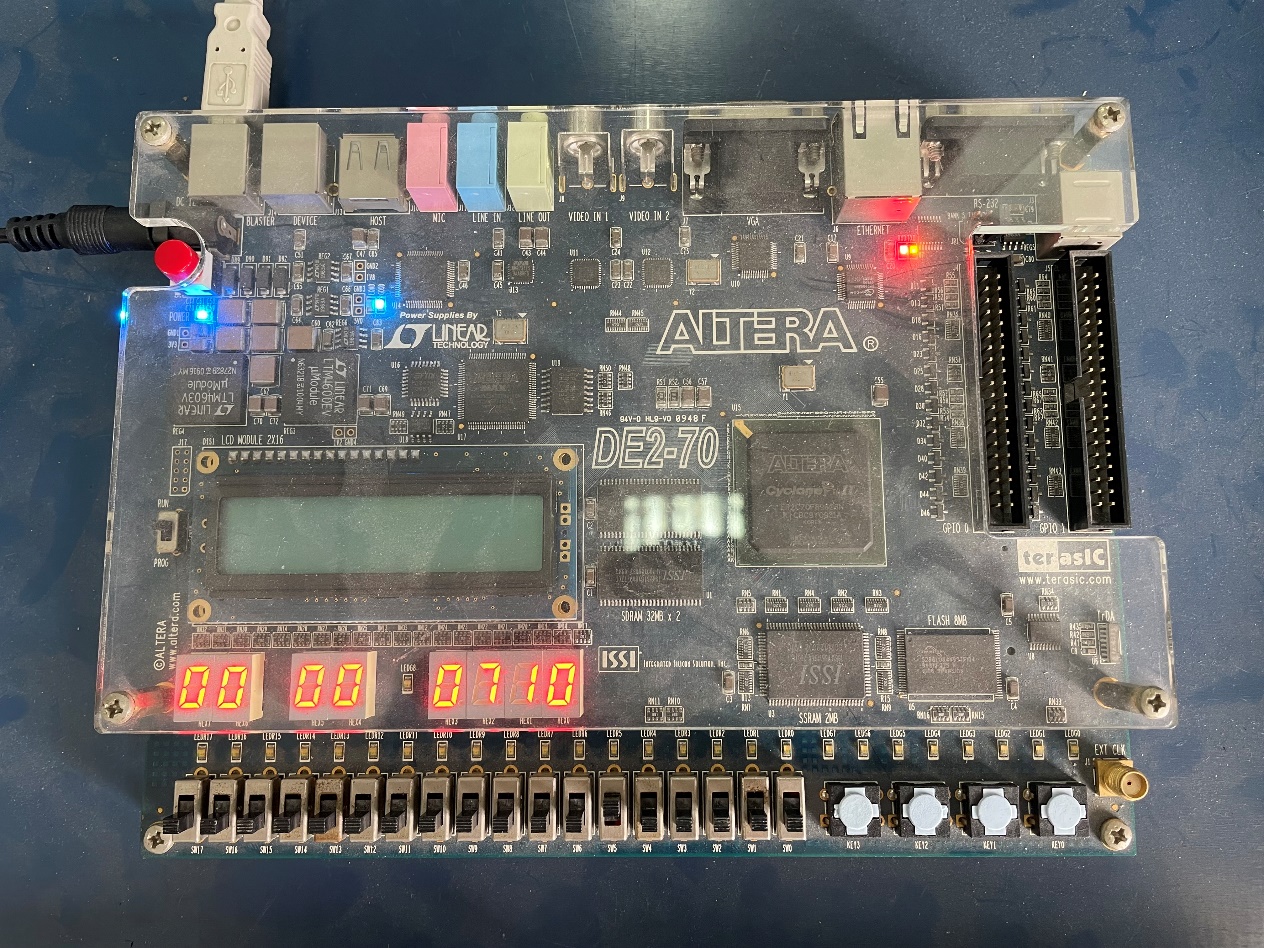


## 仿真结果

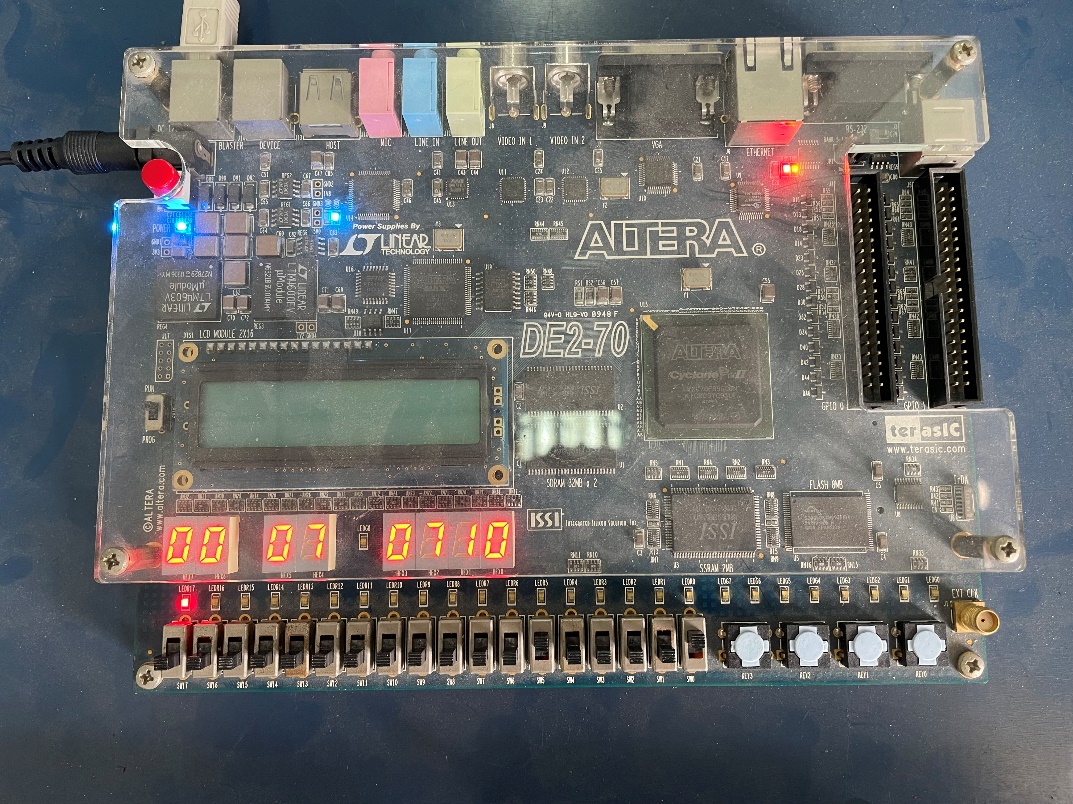


## 实验结果展示

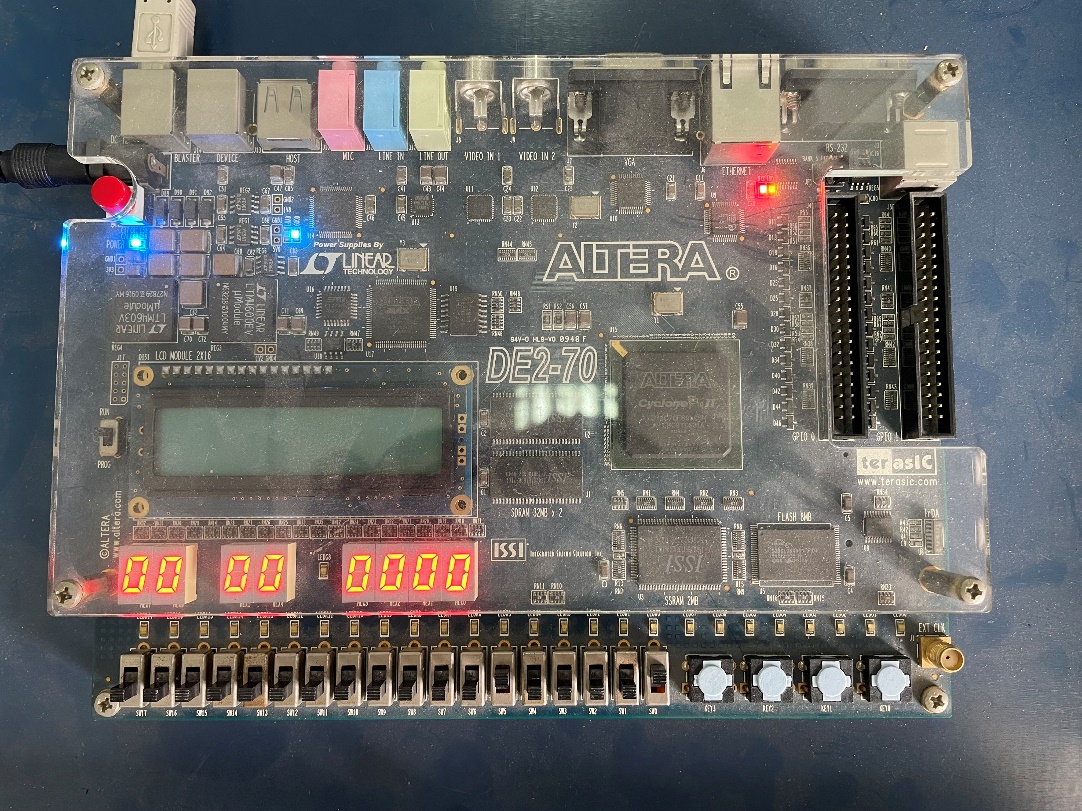
1）选择10元商品后投币7元（尚未按下确认键）



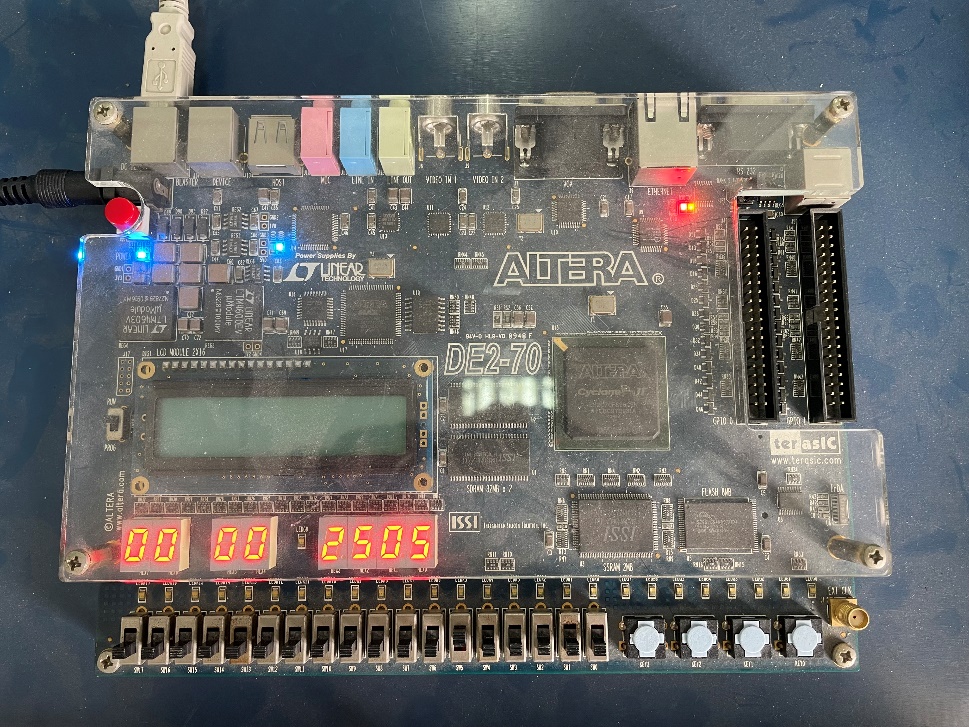
2）选择10元商品投币7元后按下确认，发出警报，此时找钱为7元，意为投币不足，把钱退给客户



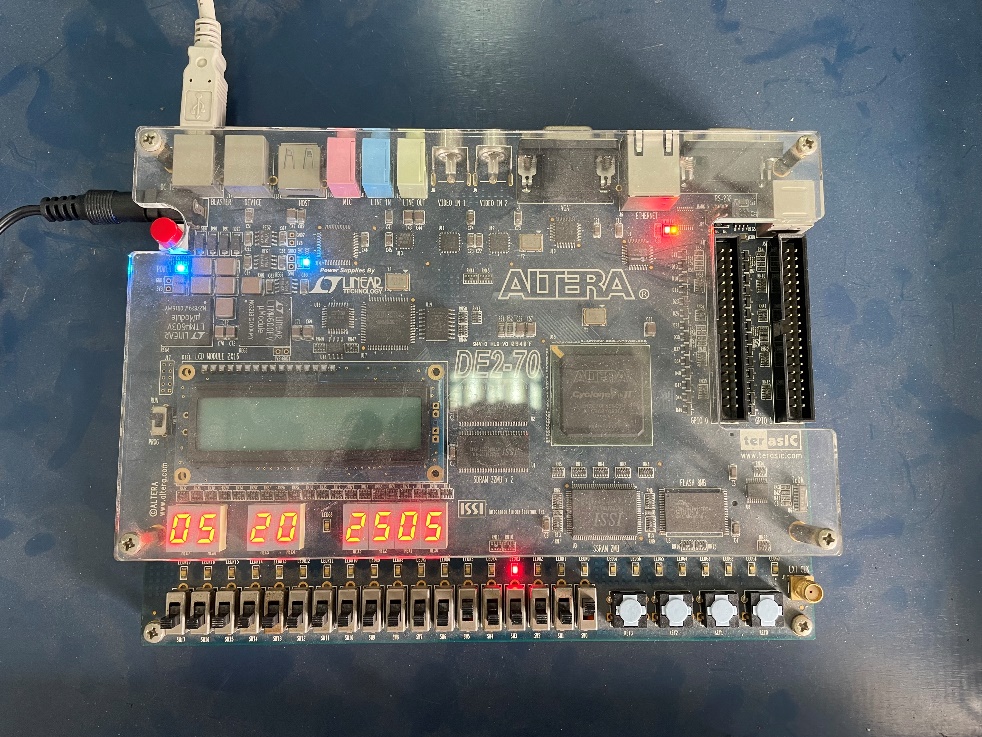
3）报警三秒后恢复初始状态



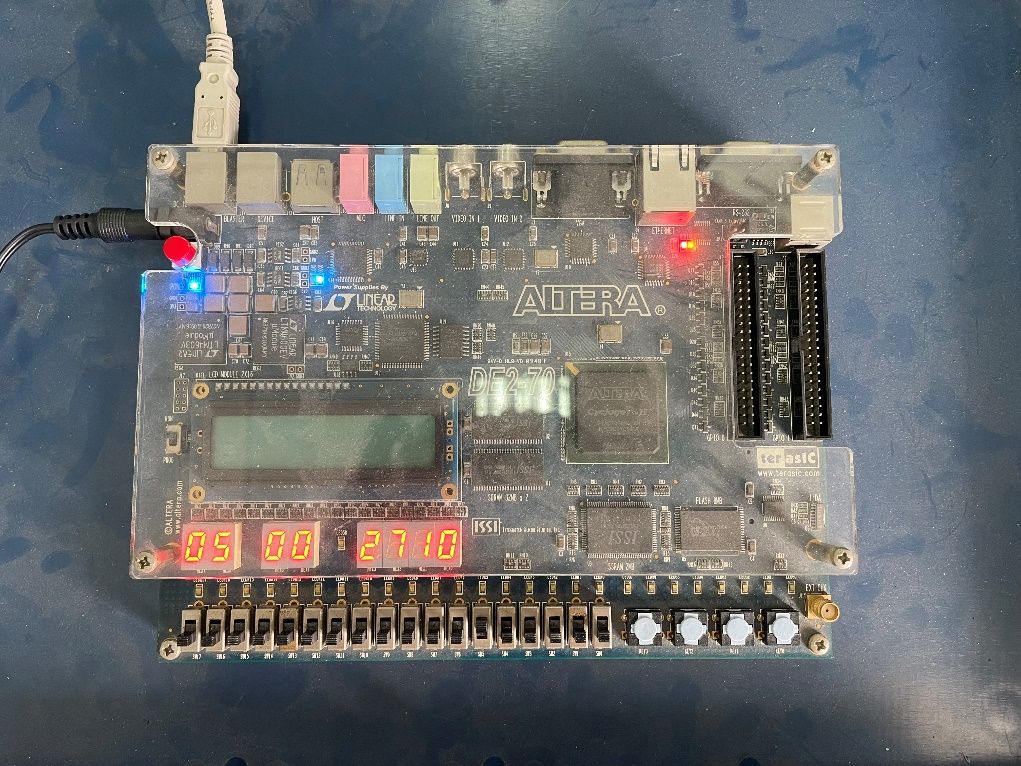
4）选择五元商品投币25元，尚未按下确认按键



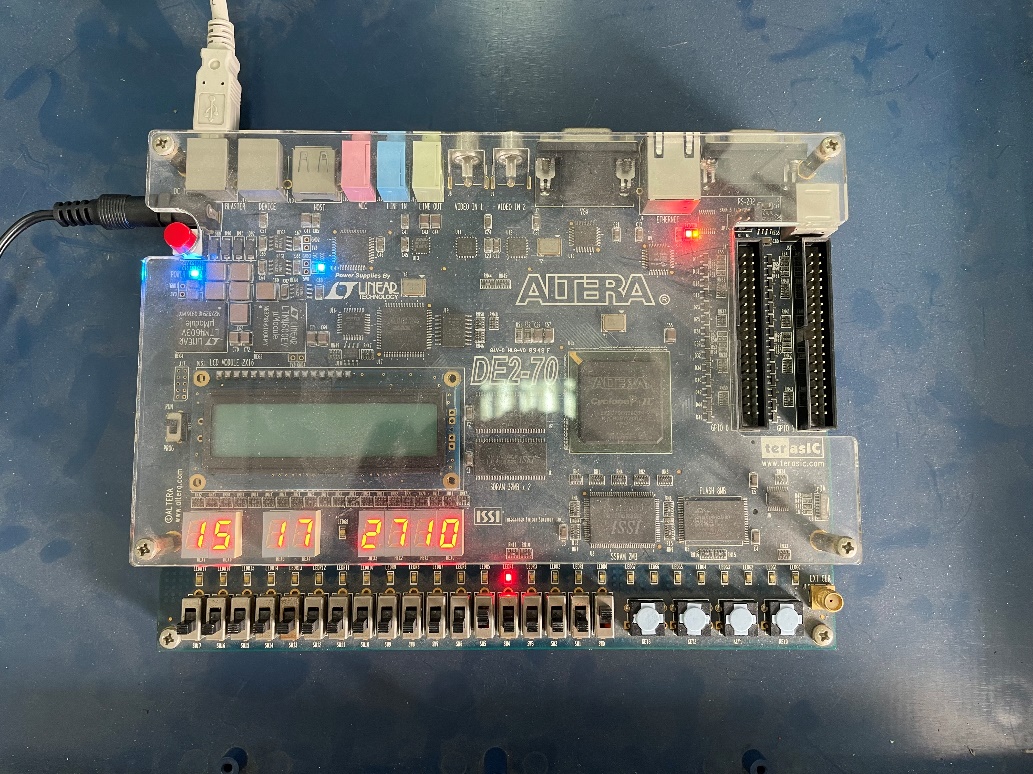
5）选择五元商品投币25元，按下确认按键，此时对应五元商品的LED灯亮起，找钱显示20元，总销售额从零元变为5元



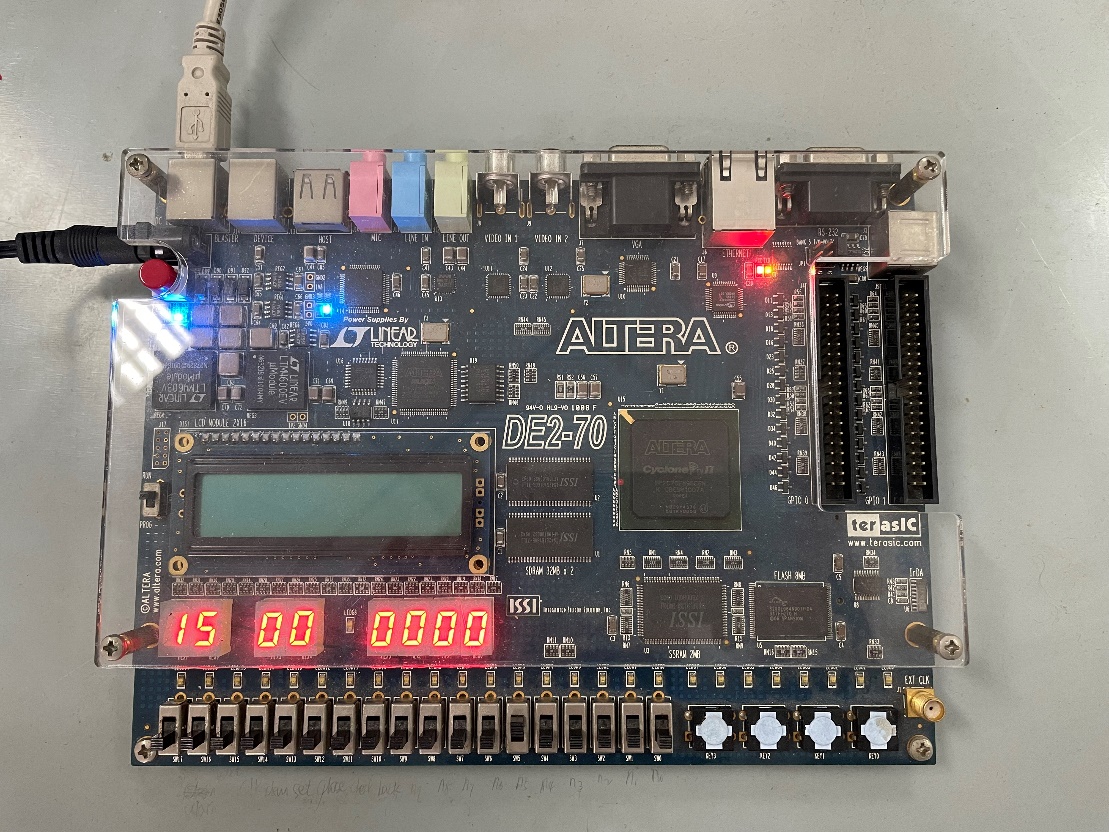
6）三秒归零后，选择10元商品投币27元，尚未按下确认按键，此时销售额还是五元



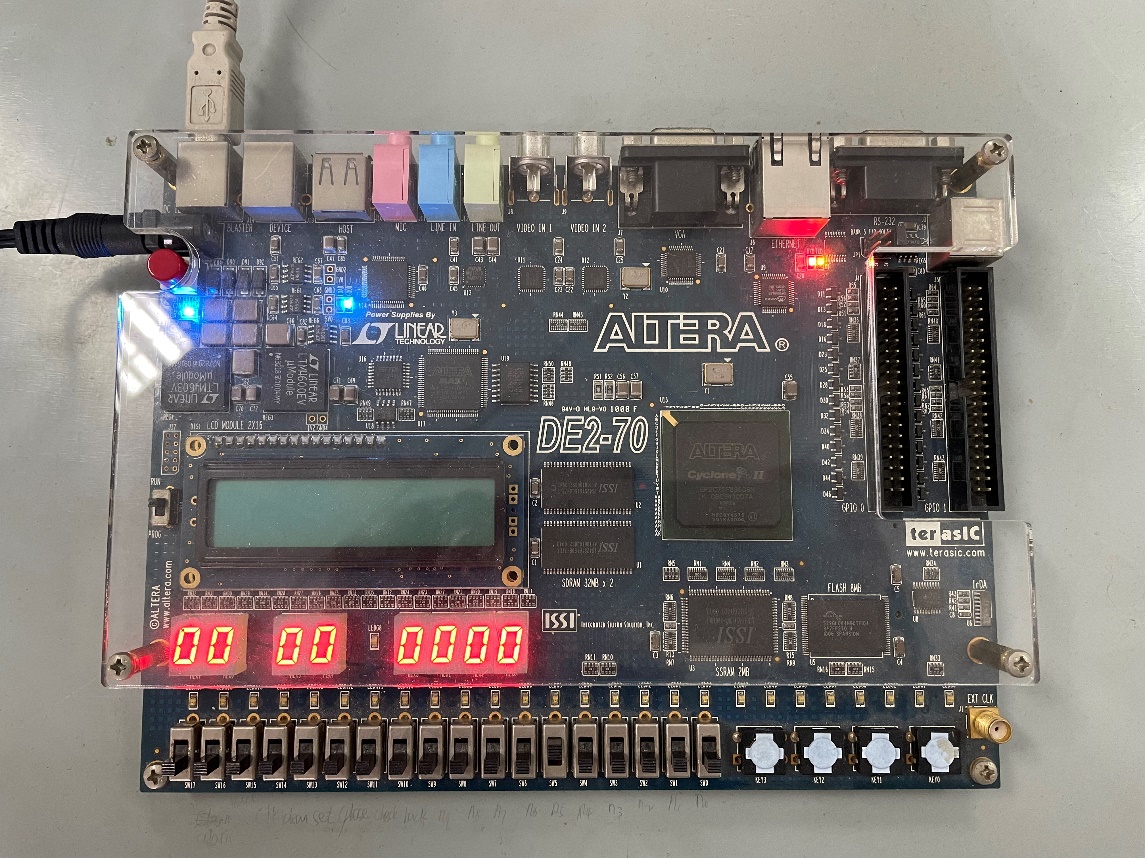
7）按下确认键，此时找钱为17元，总销售额从5元变为15元，10元商品对应的LED灯亮起



8）三秒后归零，此时总销售额为15元



9）此时，按下商家的总体复位键清空总销售额，意为商家从售货机中取出赚到的钱，此时售货机回到最初始状态。



## 总结与感悟

设计自动售货机器，是第一次我们学着用分多模块的方式小组协同开发较大规模的verilog程序（400~500行代码）。设计一个自动售货机需要考虑多个组件和模块的协作。我们学会了如何系统性地思考问题，分析需求，设计解决方案，并将各个模块有机地整合在一起。

其实整个开发过程是有趣的：在设计时，我们确定了有几个模块，各个模块负责什么功能，约定各个模块间如何交互。团队各自负责好自己的模块，单独完成某个模块功能的过程其实是比较枯燥的，我们只是管中窥豹般将规定的输入处理后，进行规定的输出（设计时定好的“协议”），最有趣的部分是编写顶层模块，此时我们拥有了所有的功能模块，此时我们按照约定的“蓝图”，如同拼装乐高积木般将各个模块连接好。一个完整的系统在眼前徐徐展开。

最折磨人部分当属编译成功后将程序下到板子上调试的时候，编写顶层模块时的愉悦还未消退时，板子上映射出的程序的现实狠狠的打脸，总是会出现“奇怪的”输出，我们得不断不断的根据这些奇怪的输出调整我们的程序，再编译，再录入程序，再调试，循环往复。此时我们就会感谢分模块编写带给我们的馈赠，因为板子上出现的“奇怪”输出很容易就能定位到是哪一个模块出现了问题再“对症下药”，节省了许多时间和精力。

通过这个项目，我们不仅完成了一个有趣的自动售货机模拟器，还积累了宝贵的经验，提高了团队合作和问题解决能力。这种实践经验将对我们未来的学习和职业发展产生积极的影响。希望我们能继续探索和创新，将这些知识和经验应用到更多的项目中。

## 参考资料

[1] 康华光. 电子技术基础(数字部分) [M ]. 第五版

[2]Verilog HDL程序设计教程 （第一版） 王金明主编

[3] 夏宇闻. 复杂数字电路与系统的Verilog HDL设计技术 [M ].北京: 北京航空航天大学出版社, 1998

[4] 张明. V erilog HDL 实用教程[M ]. 成都: 电子科技大学出版社, 1999

[5] FPGA设计及应用（第二版）西安电子科技大学出版社