**University of Essex**

**CE323 Assignment**

**Advanced Embedded Systems Design**

**Home Alarm System**

Name: Yu Wenlu

Essex ID: 1909135

[wy19403@essex.ac.uk](mailto:wy19403@essex.ac.uk)

CONTENT

[Requirement Form 2](#_Toc130573537)

[Class Diagram 2](#_Toc130573538)

[State Diagram 3](#_Toc130573539)

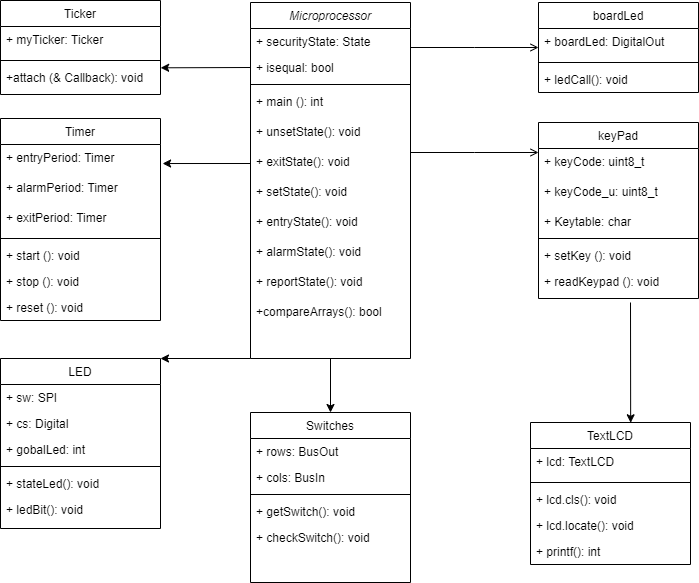
[Sequence Diagram 4](#_Toc130573540)

[Reference 5](#_Toc130573541)

# Requirement Form

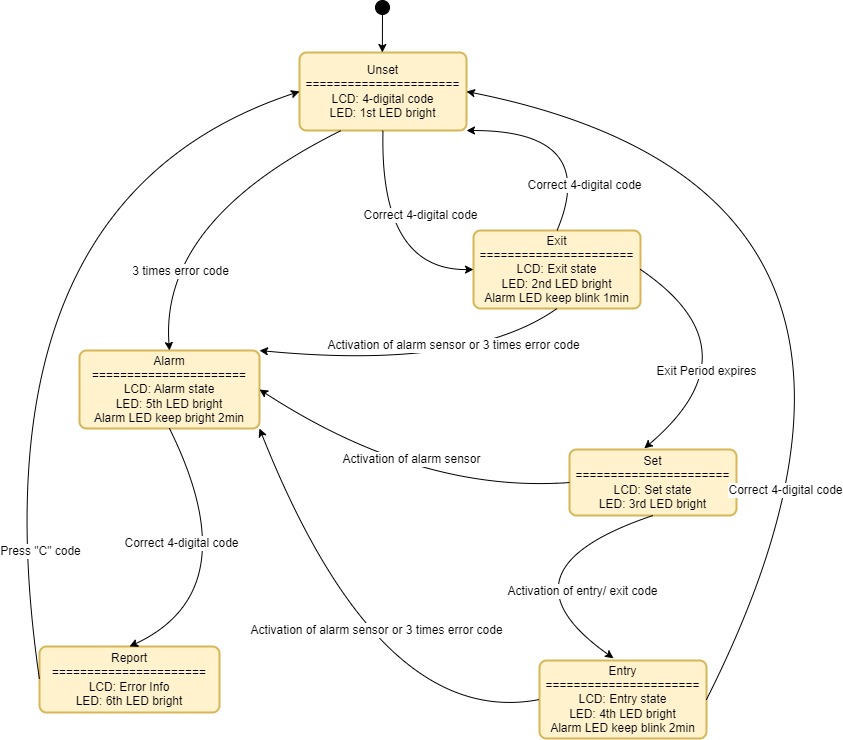
|  |  |
| --- | --- |
| Name | Home Alarm System |
| Purpose | Sense intruders and send out alarm information |
| Inputs | Board's keyboard,8 switches |
| Outputs | 8 \* LED, 1\* LCD screen, 1 \* alarm-LED |
| Functions | The system uses a TextLCD display, a keypad, and LEDs to indicate the current state of the system, which can be one of six possible states: UNSET, EXIT, SET, ENTRY, ALARM, or REPORT. |
| Performance | The program uses a state machine to handle the transitions between states based on user input and sensor input. The state machine is implemented as an enum with six possible values. There are also several timer and ticker objects to manage the timing of various actions in the system. |
| Manufacture costs | ￡75.89 |
| Physical size/weight | No more than 5 x 3.5 ich , 200 g |
| Power | 221.1 mV |

# Class Diagram



A class diagram is a type of diagram that shows the structure of a system by using classes, attributes, operations and relationships.

# State Diagram



A state machine diagram is a type of diagram that shows the behavior of a system by using states, transitions and events [1]. A state is a condition or situation that an object can be in [2]. A transition is a change from one state to another triggered by an event [3]. An event is something that happens and affects the system [3].

# Sequence Diagram

图示, 示意图

描述已自动生成

# Reference

|  |  |
| --- | --- |
| [1] | V. Paradigm., “What is State Machine Diagram?,” [online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/. [Access data: 24 3 2023]. |
| [2] | Lucidchart., “State Machine Diagram Tutorial,” [online]. Available: https://www.lucidchart.com/pages/uml-state-machine-diagram . [Access data: 23 3 2023]. |
| [3] | Wikipedia, “UML state machine,” [online]. Available: https://en.wikipedia.org/wiki/UML\_state\_machine. [Access data: 24 3 2023]. |

**Appendix**

1. */\* @Copyright University of Essex 2017*
2. \* @Author: NicoleYu wy19403@essex.ac.uk
3. \* @Date: 2023-03-03 22:20:33
4. \* @LastEditors: NicoleYu wy19403@essex.ac.uk
5. \* @LastEditTime: 2023-03-22 16:11:55
6. \* @FilePath: \mbed-os-example-blinky\assignment.cpp
7. \* @FILE - main.c
8. mbed LPC1768 application.
9. \* @DESCRIPTION
10. hardware: mbed LPC1768, an extension board.
11. Use a terminal program such as 'HyperTerminal' to communicate with the application.
12. The serial port configuration is shown below.
13. 9600 Baud
14. 8 data bits
15. No parity bits
16. One stop bit
17. No hardware flow control
18. =========================================================\*/
19. #include "mbed.h"
20. #include "TextLCD.h"
21. *//---------Board initialization declaration----------*
22. TextLCD lcd(p15, p16, p17, p18, p19, p20); *// Create a TextLCD object with pins for rs, e, d4-d7*
23. BusOut rows(p26,p25,p24);*// Create a TextLCD object with pins for rs, e, d4-d7*
24. BusIn cols(p14,p13,p12,p11);*// Create a BusIn object with pins for columns*
25. SPI sw(p5, p6, p7);*// Create an SPI object with pins for mosi, miso and sclk*
26. DigitalOut cs(p8); *// Create a DigitalOut object with pin for cs*
27. DigitalOut boardLed(LED1);*// Create a DigitalOut object with pin for boardLed*
28. Ticker myTicker;*// Create Ticker objects for light, monitorZones, blink, blinkk and bright*
29. Timer exitPeriod, entryPeriod, alarmPeriod;*// Create Timer objects for exitPeriod, entryPeriod and alarmPeriod*
30. char Keytable[] = { 'F', 'E', 'D', 'C',
31. '3', '6', '9', 'B',
32. '2', '5', '8', '0',
33. '1', '4', '7', 'A'
34. };
35. *//----------State declaration------------------------*
36. enum State {UNSET, EXIT, SET, ENTRY, ALARM, REPORT} securityState; *// State machine setting*
37. const char\* StateNames[] = {"UNSET", "EXIT", "SET", "ENTRY", "ALARM", "REPORT"}; *// In order to display the current status on the lcd screen conveniently*
38. *//----------Personalization declaration------------------*
39. uint8\_t keyCode[4] = {'1','2','3','4'}; *// Correct password*
40. uint8\_t keyCode\_u[4] = {'1','2','3','4'}; *// User input password*
41. *//----------Intermediate variable declaration-----------------*
42. int judge=0;*//'judge' variable used to judge the state. 0=unset, 1=other, 2=unset, 3=alarm.*
43. int gobalLed = 0;*// gobalLed is ALARMLED on the development board LED*
44. int counter = 0; *// counter is 4-digit code attempt times, and can not more than 3.*
45. *//-----------Function declaration-----------------*
46. void initLeds();
47. char getKey();
48. int getSwitch();
49. void readKeypad();
50. bool compareArrays(const uint8\_t\* keyCode, const uint8\_t\* keyCode\_u);
51. void ledBit(int order);
52. void checkSwitches();
53. void setKey(void);
54. void stateLed();
55. void ledCall();
56. void unsetState();
57. void exitState();
58. void setState();
59. void entryState();
60. void alarmState();
61. void reportState();
62. int main();
63. *//-------------------Code begining-------------------------*
64. *// Initialize the LED*
65. void initLeds() {
66. cs = 0;*// latch must start low*
67. sw.format(16,0);*// SPI 16 bit data, low state, high going clock*
68. sw.frequency(1000000);*// 1MHz clock rate*
69. }
70. char getKey(){
71. int i,j;
72. char ch=' ';
74. for (i = 0; i <= 3; i++) {
75. rows = i;
76. for (j = 0; j <= 3; j++) {
77. if (((cols ^ 0x00FF)  & (0x0001<<j)) != 0) {
78. ch = Keytable[(i \* 4) + j];
79. }
80. }
81. }
82. return ch;
83. }
84. void readKeypad(){
85. char b = ' ';
86. b=getKey();
87. wait\_ns(10000);
88. if (b != ' '){
89. setKey();
90. }
91. }
92. *// Verify that the password is correct*
93. bool compareArrays(const uint8\_t\* keyCode, const uint8\_t\* keyCode\_u) {
94. for (int i = 0; i < 3; i++) {
95. if (keyCode[i] != keyCode\_u[i]) {
96. return 0;
97. }
98. }
99. return 1;
100. }
101. *//controlling the state of one or more LEDs using a shift register.*
102. void ledBit(int order) {
103. order = (((8-order) & 0x0007) + 1) \* 2; *//offset of led state in 'gobalLed'*
104. gobalLed = gobalLed & ((0x0003 << order) ^ 0xffff); *// clear and set led state*
105. sw.write((gobalLed & 0x03ff) | ((gobalLed & 0xa800) >> 1) | ((gobalLed & 0x5400) << 1));
106. cs = 1;                                        *// latch pulse start*
107. cs = 0;
108. }
109. void checkSwitches(){
110. initLeds();
111. int switches = getSwitch();
112. if ((securityState != UNSET)&& (securityState != ALARM) && (securityState != REPORT)&&(num != 0x80)&&(num !=0x00)){
114. judge = 1;
115. counter = 0;
116. securityState = ALARM;
117. exitPeriod.stop();
118. exitPeriod.reset();
119. lcd.cls();
120. alarmState();
121. }
122. }
123. int getSwitch()
124. {
125. int switches = cols;
126. switches = switches\*17
127. for(int i=0;i<=7;i++){
128. if ((switches & 0x0001<<i)!=0){
129. gobalLed  = gobalLed | (0x0003 << i\*2); }
130. else {
131. gobalLed  = gobalLed & ((0x0003 << i\*2) ^ 0xffff); }
132. }
133. sw.write(gobalLed);
134. cs = 1;
135. cs = 0;
136. return switches;
137. *//([1])*
138. }
139. void setKey(void){
140. int a;
141. char b = ' ';
142. judge =0;
143. b=getKey();
144. wait\_ns(10000);
145. lcd.cls();
146. lcd.locate(0,0);
147. lcd.printf("STATE: %s", StateNames[securityState]);
148. lcd.locate(0,1);
149. lcd.printf("Code:\_\_\_\_");
150. for(a=0;a<4;a++){
151. b=getKey();
152. wait\_ns(10000);
153. if (judge != 0){
154. judge = 0;
155. lcd.cls();
156. return;
157. }
158. switch(b){
159. case ' ':
160. a--;
161. break;
162. case 'C':
163. if(a>0){
164. a=a-2;
165. lcd.locate(6+a,1);
166. lcd.putc('\_');
167. }
168. else if(a==0){
169. a--;
170. }
171. break;
172. case 'B':
173. default:
174. lcd.locate(5+a,1);
175. lcd.putc('\*');
176. keyCode\_u[a]=b;
177. break;
178. }    *// [3]*
179. }
180. b=getKey();
181. wait\_ns(10000);
182. while (b != 'B'){
183. if (judge != 0){
184. judge = 0;
185. lcd.cls();
186. return;
187. }
188. lcd.locate(0,1);
189. lcd.printf("Press B to set");
190. b=getKey();
191. wait\_ns(100);
192. }
193. if(bool isEqual = compareArrays(keyCode, keyCode\_u)){
194. if(securityState == UNSET){
195. securityState = EXIT;
196. lcd.cls();
197. }
198. else if(securityState == EXIT){
199. securityState = UNSET;
200. exitPeriod.stop();
201. exitPeriod.reset();
202. boardLed = 0;
203. lcd.cls();
204. }
205. else if(securityState == ENTRY){
206. securityState = UNSET;
207. entryPeriod.stop();
208. entryPeriod.reset();
209. boardLed = 0;
210. lcd.cls();
211. }
212. else if(securityState == ALARM){
213. securityState = REPORT;
214. alarmPeriod.stop();
215. alarmPeriod.reset();
216. lcd.cls();
217. reportState();
218. }
219. counter = 0;
220. }
221. else {
222. counter = counter++;
223. if(counter >= 3){
224. counter = 0;
225. securityState = ALARM;
226. lcd.cls();
227. alarmState();
228. }
229. else{
230. lcd.locate(0,1);
231. lcd.printf("Remain attempt:%d", 3-counter);
232. wait\_ns(10000);
233. }
234. }
235. }
236. void stateLed(){
237. switch(securityState)
238. {
239. case UNSET :
240. ledBit(1);
241. break;
242. case EXIT :
243. ledBit(2);
244. break;
245. case SET :
246. ledBit(3);
247. break;
248. case ENTRY :
249. ledBit(4);
250. break;
251. case ALARM :
252. ledBit(5);
253. break;
254. case REPORT :
255. ledBit(6);
256. break;
257. }
258. }
259. *//controlling the sort of security system with LED lights and an LCD screen.*
260. void ledCall(){
261. if ((exitPeriod.read()<=10) && (securityState == EXIT) &&(securityState == ENTRY)){
262. boardLed = !boardLed;
263. }
264. else if((exitPeriod.read()>10)){
265. exitPeriod.stop();
266. exitPeriod.reset();
267. counter = 0;
268. if (securityState == EXIT){
269. judge = 2;
270. securityState = SET;
271. boardLed = 0;
272. lcd.cls();
273. setState();
274. }
275. else if(securityState == ENTRY){
276. judge = 3;
277. securityState = ALARM;
278. lcd.cls();
279. alarmState();
280. }
282. }
283. else if (securityState == ALARM){
284. alarmPeriod.start();
285. boardLed = 1;
286. if(alarmPeriod.read()>20){
287. boardLed = 0;
288. alarmPeriod.stop();
289. alarmPeriod.reset();
290. }
291. }
292. }
293. *// unset state function*
294. void unsetState(){
295. lcd.locate(0,0);
296. lcd.printf("STATE: UNSET");
297. ledBit(1);
298. readKeypad();
299. }
300. *// exit state function*
301. void exitState(){
302. exitPeriod.start();
303. lcd.locate(0,0);
304. lcd.printf("STATE: EXIT");
305. ledBit(2);
306. readKeypad();
307. }
308. *// set state function*
309. void setState(){
310. lcd.locate(0,0);
311. lcd.printf("STATE: SET");
312. ledBit(3);
313. if(num == 0x80){
314. counter = 0;
315. securityState = ENTRY;
316. lcd.cls();
317. }
318. }
319. *// entry state function*
320. void entryState(){
321. entryPeriod.start();
322. lcd.locate(0,0);
323. lcd.printf("STATE: ENTRY");
324. ledBit(4);
325. readKeypad();
326. }
327. *//alarm state function*
328. void alarmState(){
329. alarmPeriod.start();
330. lcd.locate(0,0);
331. lcd.printf("STATE: ALARM");
332. ledBit(5);
333. readKeypad();
334. }
335. *//report state function*
336. void reportState(){
337. char cls=' ';
338. if ( (checkSwitches() & 0x03) != 0 )
339. lcd.printf("full set error");
340. if ( (checkSwitches() & 0x0c) != 0 )
341. lcd.printf("set error");
342. if ( (checkSwitches() & 0x30) != 0 )
343. lcd.printf("entry error");
344. if ( (checkSwitches() & 0xc0) != 0 )
345. lcd.printf("exit error");
346. else
347. lcd.printf("code error");
348. lcd.locate(0,1);
349. lcd.printf("C key to clear");
350. cls = getKey();
351. wait\_ns(10000);
352. if (cls == 'C'){
353. securityState = UNSET;
354. lcd.cls();
355. }
356. ledBit(6);
357. }
358. int main() {
360. initLeds(); *//initialize leds*
361. lcdRefresh(); *// initial lcd clear*
362. securityState = UNSET; *//initial state*
363. monitorZones.attach\_us(&checkSwitches, 0.1); *//check zones every 100ms*
364. myTicker.attach(&ledCall, 0.1);*// check LED*
365. while(1)
366. { *// State machine judgment*
367. switch(securityState)
368. {
369. case UNSET :
370. unsetState();
371. break;
372. case EXIT :
373. exitState();
374. break;
375. case SET :
376. setState();
377. break;
378. case ENTRY :
379. entryState();
380. break;
381. case ALARM :
382. alarmState();
383. break;
384. case REPORT :
385. reportState();
386. break;
387. *//[4]*
388. }
389. }
391. }
392. *// reference*
393. *// [1]"home\_alarm\_simple - | Mbed," mbed import http://os.mbed.com/users/liruihao2008/code/home\_alarm\_simple/.*
394. *// [2]"CE323\_Lab4"*
395. *// [3]"CE323\_Lab6"*
396. *// [4]"CE323\_Lab7"*