INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Software for Embedded Systems

MEIC-A 2017-2018 – 1st Semester

Project Report

Lab04 - Control of traffic lights at a roundabout

Group 06

João Carlos Santos Fernandes
joaocsfernandes@tecnico.ulisboa.pt
87786

Jorge Heleno
jorge.heleno@ist.utl.pt
79042

1. Introduction

Embedded Systems are a continuously grown area of I.T., nowadays the presence of embedded systems on the quotidian life its huge. The normal home appliances now come with processors and can be connected in a network to allow the user to control them with a smartphone. Also cars now have hundreds of processors doing the work that in the past was done all by mechanic structures. A example of embedded system that we deal daily it's the semaphores on the streets, a system that its present in our daily routine as pedestrians or as drivers.

The goal of this project was develop a embedded system that simulates the traffic lights at a roundabout. For that we develop 3 modules. The semaphore of the entry, the semaphore on the roundabout and the controller that will manage multiple entry's. The project was developed in Arduino and using the I2C communication to communicate between multiple entry's.

2. Design of The Circuit

The circuit that we implemented can be seen in Figure 1 - Main Circuit.

- **Controller:**
  - **Input:**
    - **A0** – Connected to the potentiometer to read the value that will define the period of the roundabout.
    - **D13** – Connected to a pulse button to read the state to turn ON/OFF the controller.
  - **Output:**
    - **D12** – Connected to the LED that presents the state of the controller (blinking to turned on, off to turned off).
- **Semaphore A (Entry):**
  - **Input:**
    - **A1** – Connected to the Red LED branch to read the value for error detecting purposes.
    - **D07**– Connected to a pulse button to read the state to reduce the remain time of the period by half (Pedestrians Button).
  - **Output:**
    - **D11** – Connected to the Red LED of the semaphore.

        - **D10** – Connected to the Yellow LED of the semaphore.
        - **D09** – Connected to the Green LED of the semaphore.
        - **D08** – Connected to the Green LED of the pedestrian's semaphore.
- **Semaphore B (Roundabout):**
    - **Input:**
        - **A2** – Connected to the Red LED branch to read the value for error detecting purposes.
    - **Output:**
        - **D06** – Connected to the Red LED of the semaphore.
        - **D05** – Connected to the Yellow LED of the semaphore.
        - **D04** – Connected to the Green LED of the semaphore.
- **Entry Controller:**
    - **Input:**
        - **D03** – Highest Bit of the controller entry number.
        - **D02** – Lowest Bit of the controller entry number.

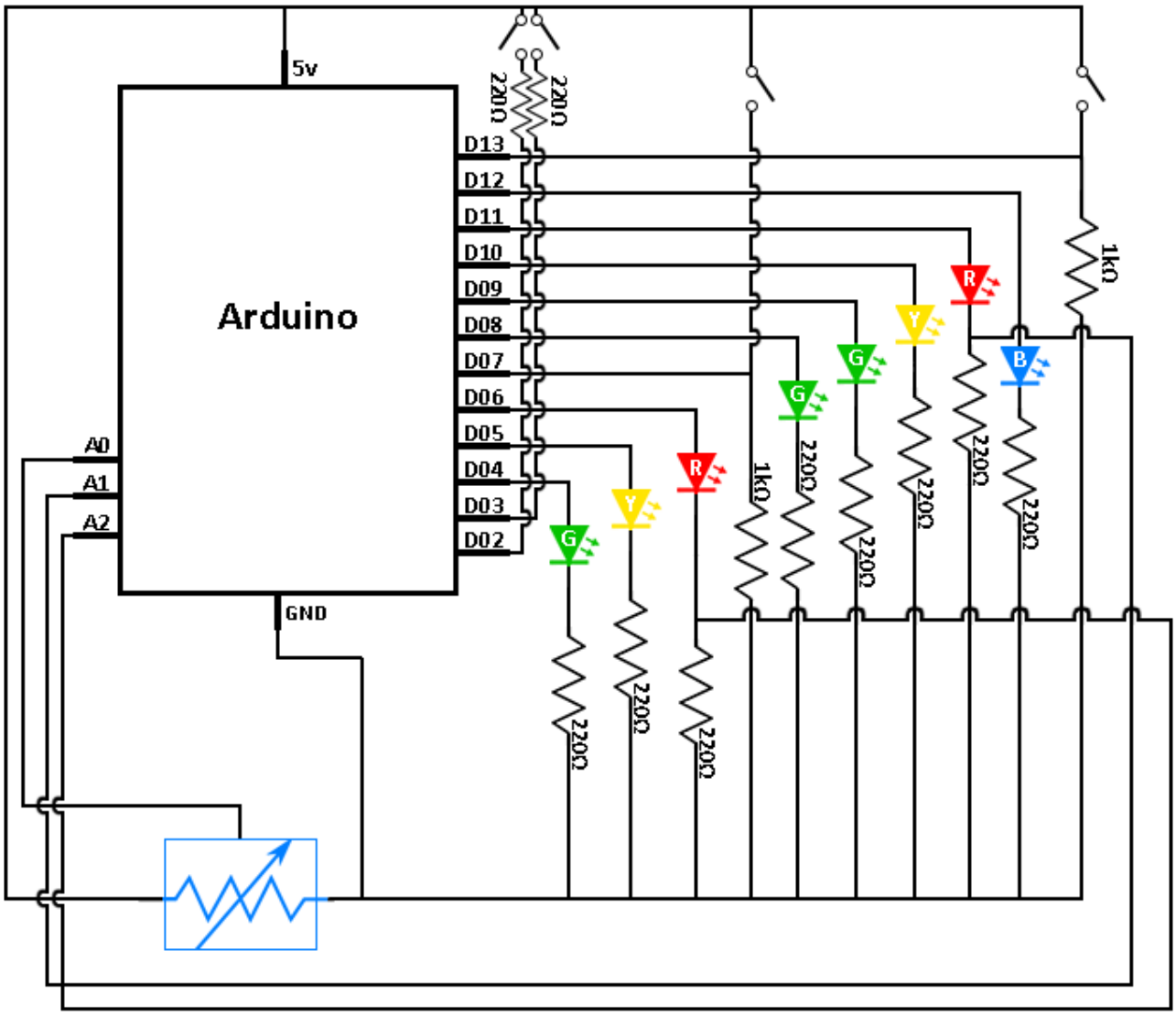


Figure 1 - Main Circuit

3. Architecture

a. Controller

The controller follows a round robin architecture. In each cycle it checks if the button to turn on was pressed, if there exists persistent errors that will shut down the controller, read the value of the potentiometer and maps the value to the period [2s, 15s], if the controller received a status that a pedestrian button as been pressed it will reduce the remaining period time, if the period time was elapsed it starts a new period, if the ping period time was elapsed it will ping the entry's again, if the errors period time was elapsed it will count the errors that persist among this period, if the controller was turned off it checks if last cycle it was off and if it wasn't sends the off message. Our controller pseudocode its something like:

```
void controller() {
	checkButtonPressed();
	checkPersistentErrors();
	if(controllerOn) {
		readPeriodTimeFromThePotentiometer();
		if(pedestriansButtonPressed) {
			reduceRemainingPeriodTime();
		}
		if(periodElapsed) {
			startNewPeriod();
		}
		if(pingPeriodElapsed) {
			sendNewPeriod();
		}
		if(errorPeriodElapsed) {
			countErrors();
		}
	} else {
		if(notOffLastCycle) {
			sendOffMessage();
		}
	}
}
```

b. Semaphores

The semaphores follow a round robin with interrupts architecture with a state machine approach. The interrupts change variables and based on that the semaphore changes its status, the semaphore also has an internal state that only him can achieve that is the "error" state, the semaphore flags the error state when it founds a error on a LED. Also the semaphore has a watchdog to see if the controller was turned off and didn't announce that, the watchdog has a pre-defined period and if it fails the test it will turn off the semaphore. Our semaphore pseudocode its something like:

```
void interruptHandler() {
	watchdogMessageReceive = true;

	if(messageReceiveType == OFF) {
		setSempahoreOff()
	}
	if(messageReceiveType == GREEN) {
		setSempahoreOpen ()
	}
	if(messageReceiveType == RED) {
		setSempahoreClose()
	}
	if(messageReceiveType == PING) {
		generateStatusMessage();
	}
}
```

```
void semaphore() {
	if(stateOff || stateError) {
		blinkYellow();
	}
	if(stateOn){
		if(inTransitionTime){
			yellowLight();
		} else {
			if(open) {
				greenLight();
			} else {
				redLight();
				pedestriansGreenLight();
			}
		}

	}

	if(stateError) {
		tryRecover();
	}

	if(watchdogTimeElapsed) {
		if(watchdogMessaReceive) {
			resetWatchdog();
		} else {
			setSemaphoreOff();
		}
	}
}
```

4. Safety and Fault Tolerance

Since a semaphore it's a critical system we implemented some safety and fault tolerance features to prevent accidents from happening.

a. Controller

On the controller side we implemented error detection and timeouts. The error detection feature follows the requirements. On the status message every semaphore sent the information about the errors in the LED's. The controller will save that error states in an array to know which entry has an error. If an error persists more that 2 periods, the controller will send a shutdown to all entry's and shutdown himself. The timeouts are used when it reads messages from the wire library. If the slaves don't respond in a pre-defined period, it will timeout and the flag for errors in this entry will be turned on.

b. Semaphore

On the semaphore side we implemented error detection, error recovering and a watchdog. The error detection is made by reading with an analogic pin the state of the LED circuit, so when we expect a LED its turn on we expect to read a certain value on the analogic pin when its turn of we expect other value. When the read doesn't get the value expected we turn on the error flag and the semaphore starts blinking, in the status message it start s sending to the controller the error in the LED's for it to shut down all entry's. The error recovering is made by trying to turn on the LED that its defect and read the value of it circuits again, if the value becomes the expected the error flag its turned off, if not the error flag keeps turn on. The watchdog as mentioned above its used to keep track of the controller connection so if the controller stops sending messages it will turn off the semaphore because something its wrong.

5. Interoperable Changes

When we connected our Arduino to the other groups Arduinos we needed to do some changes to our code, the changes are explained in this section and how they affected our initial approach.

- **Changed the *"char"* type to *"uint8_t"*:**

  The groups that have been pair with us didn't used chars to represent the bytes of the messages, they used unsigned integers of 8 bits, so since we have made some comparisons and attributions using char's we needed to change our code to use uint8_t variables and arrays instead. That was a small change that doesn't affect anything in our initial approach.

- **Changed the *period of ping:***

  The groups that have been pair with us have a very small ping period, so their semaphore safety system was making their semaphores go off since they didn't receive pings in the time they expected so, we needed to make our period ping smaller. That changed didn't affected anything in our initial approach.

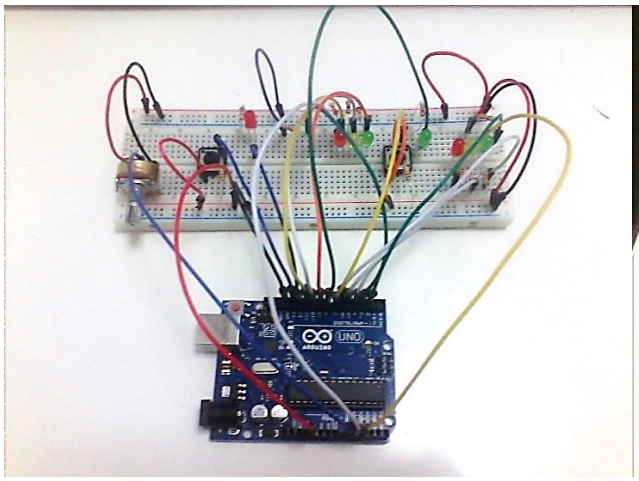
- **Changed the *period of error counting:***

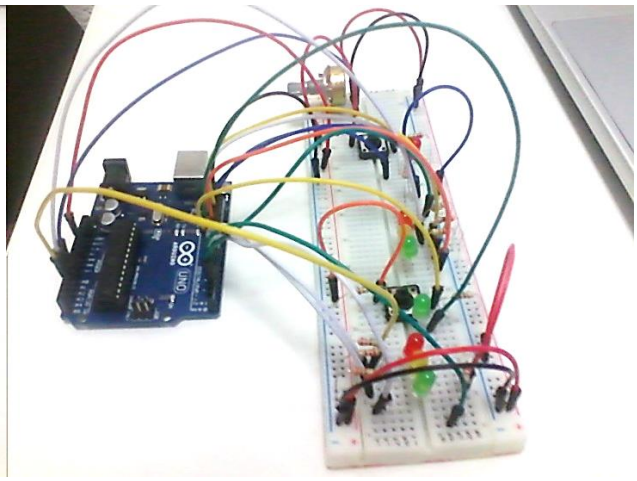
  The groups that have been pair with us have a very small error counting period, so our controller safety system was being much slower on shut down the all system then theirs. Because of that we needed to reduce our error counting period. That changed affected our initial approach because now the error period it's too fast and we cant use the error recovery of our semaphores since the controller tells it to turn off before our semaphores have time to recover.

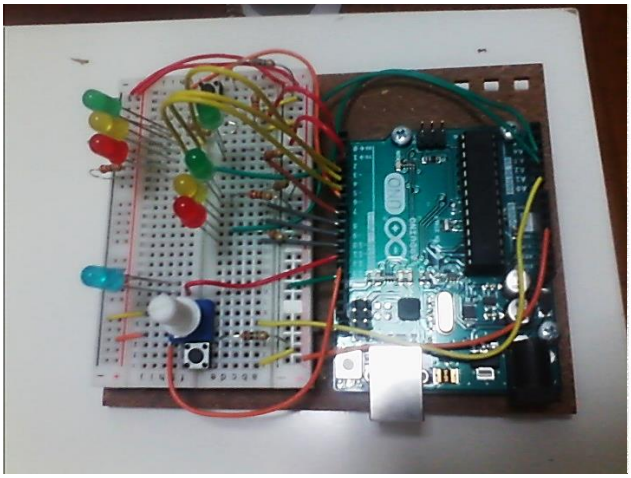
6. Conclusion

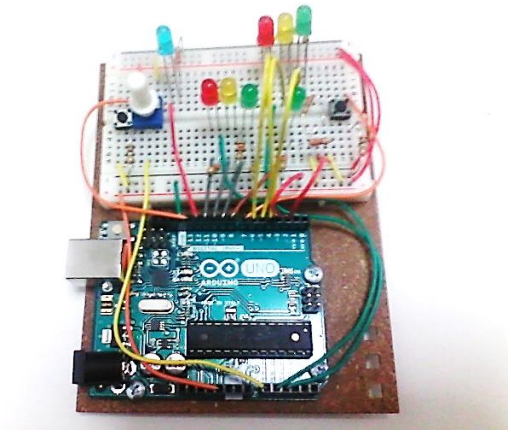
In this report we present our design decisions for the implementation of this embedded system. We think our system match all the requirements and have some nice extra features like the error recovering. The integration was made very easily only requiring some adjustments to the values that the multiple groups were using and the approach of message that each one was using. Annexed to this report we sent the code of our solution and the photos of our circuits implemented in ours Arduino's.

ANEXO









```
1  #include <Wire.h>
2
3  // MESSAGES
4  #define RED 0
5  #define GREEN 1
6  #define OFF 2
7  #define PING 3
8  #define ACK 4
9  #define STATUS 5
10
11 // SEMAPHORES
12 #define YELLOW_TRANSITION_TIME 500
13 #define YELLOW_BLINKING_PERIOD 1000
14 #define WATCHDOG_TIMER 300
15
16 // CONTROLLER
17 #define WIRE_MODE false
18 #define MAX_ENTRY 4
19 #define PING_PERIOD 100
20 #define ERRORS_PERIOD 200
21
22
23 // SEMAPHORE A
24 const int A_redLedPin = 11;
25 const int A_yellowLedPin = 10;
26 const int A_greenLedPin = 9;
27 const int A_redStatusPin = A1;
28 const int A_pedestriansLedPin = 8;
29 const int A_pedestriansButtonPin = 7;
30
31 boolean A_redLight = false;
32 boolean A_yellowLight = false;
33 boolean A_greenLight = false;
34 boolean A_pedestriansLight = false;
35
36 boolean A_pedestriansButtonPressed = false;
37
38 int A_status = 0; //0 - OFF | 1 - OPEN | 2 - CLOSE
39 boolean A_error = false;
40
41 long A_lastTime = 0;
42 long A_watchdogTimer = 0;
43 boolean A_watchdog = true;
44 // -------------------------------
45
46 // SEMAPHORE B
47 const int B_redLedPin = 6;
48 const int B_yellowLedPin = 5;
49 const int B_greenLedPin = 4;
50 const int B_redStatusPin = A2;
51
52 boolean B_redLight = false;
53 boolean B_yellowLight = false;
54 boolean B_greenLight = false;
55
56 int B_status = 0; //0 - OFF | 1 - OPEN | 2 - CLOSE
```

```
57  boolean B_error = false;
58
59  long B_lastTime = 0;
60  long B_watchdogTimer = 0;
61  boolean B_watchdog = true;
62  // --------------------------------
63
64  // ENTRY CONTROLLER
65  const int entryLSPin = 2;
66  const int entryHSPin = 3;
67
68  int entryNumber;
69  bool isController = false;
70
71  uint8_t messageReceive[4] = {0, 0, 0, 0};
72  uint8_t messageSent[5] = {0, 0, 0, 0, 0};
73  int mSLength = 0;
74  // --------------------------------
75
76  // CONTROLLER
77  const int potentiometerPin = A0;
78  const int controllerButtonPin = 13;
79  const int controllerLedPin = 12;
80
81  int entryOpen = 1;
82
83  bool controllerOn = false;
84  bool sentOff = true;
85  bool reduceTime = false;
86
87  uint8_t messageController[5] = {0, 0, 0, 0, 0};
88
89  int persistentErrors = 0;
90  bool entryErrors[5] = {false, false, false, false, false};
91
92  long controller_period = 2000;
93  long controller_lastTimePeriod = 0;
94  long controller_lastTimePing = 0;
95  long controller_lastTimeBlink = 0;
96  long controller_lastTimeErrors = 0;
97  // --------------------------------
98
99  void setup() {
100   Serial.begin(9600);
101
102   entryNumber = getEntryNumber();
103
104   Serial.println("ENTRY:");
105   Serial.println(entryNumber);
106   Serial.println("------");
107
108   if(entryNumber == 1) {
109     isController = true;
110   }
111
112   pinMode(A_redLedPin, OUTPUT);
```

```
113   pinMode(A_yellowLedPin, OUTPUT);
114   pinMode(A_greenLedPin, OUTPUT);
115   pinMode(A_redStatusPin, INPUT);
116
117   pinMode(A_pedestriansLedPin, OUTPUT);
118   pinMode(A_pedestriansButtonPin, INPUT);
119
120   pinMode(B_redLedPin, OUTPUT);
121   pinMode(B_yellowLedPin, OUTPUT);
122   pinMode(B_greenLedPin, OUTPUT);
123   pinMode(B_redStatusPin, INPUT);
124
125   pinMode(controllerButtonPin, INPUT);
126   pinMode(controllerLedPin, OUTPUT);
127
128   pinMode(entryLSPin, INPUT);
129   pinMode(entryHSPin, INPUT);
130
131   Wire.begin(entryNumber);
132   Wire.onReceive(receiveEvent);
133   Wire.onRequest(requestEvent);
134 }
135
136 void loop() {
137   if(isController) {
138     controller();
139   }
140
141   semaphoreA();
142
143   semaphoreB();
144 }
145
146 //-------------------------------------| SEMAPHORE A                    ↵
      |-------------------------------------------------------
147
148 void semaphoreA() {
149   long currentTime = millis();
150
151   if(A_status == 0 || A_error) {
152     if(currentTime - A_lastTime >= YELLOW_BLINKING_PERIOD) {
153       A_yellowLight = !A_yellowLight;
154       A_lastTime = currentTime;
155     }
156     A_redLight = false;
157     A_greenLight = false;
158     A_pedestriansLight = false;
159   } else {
160     if(currentTime - A_lastTime < YELLOW_TRANSITION_TIME) {
161       A_redLight = false;
162       A_yellowLight = true;
163       A_greenLight = false;
164       A_pedestriansLight = false;
165     } else {
166       if(A_status == 1) {
167         A_redLight = false;
```

```
168           A_yellowLight = false;
169           A_greenLight = true;
170           A_pedestriansLight = false;
171           if(digitalRead(A_pedestriansButtonPin) == HIGH) {
172             A_pedestriansButtonPressed = true;
173           }
174         } else if (A_status == 2) {
175           A_redLight = true;
176           A_yellowLight = false;
177           A_greenLight = false;
178           A_pedestriansLight = true;
179         }
180       }
181     }
182
183     if(A_redLight) {
184       digitalWrite(A_redLedPin, HIGH);
185     } else {
186       digitalWrite(A_redLedPin, LOW);
187     }
188     if(A_yellowLight) {
189       digitalWrite(A_yellowLedPin, HIGH);
190     } else {
191       digitalWrite(A_yellowLedPin, LOW);
192     }
193     if(A_greenLight) {
194       digitalWrite(A_greenLedPin, HIGH);
195     } else {
196       digitalWrite(A_greenLedPin, LOW);
197     }
198     if(A_pedestriansLight) {
199       digitalWrite(A_pedestriansLedPin, HIGH);
200     } else {
201       digitalWrite(A_pedestriansLedPin, LOW);
202     }
203
204     if(A_status == 2 && (A_redLight || A_error)) {
205         digitalWrite(A_redLedPin, HIGH);
206         if(analogRead(A_redStatusPin) > 450) {
207           A_error = false;
208         } else {
209           digitalWrite(A_redLedPin, LOW);
210           A_error = true;
211         }
212     }
213
214     if(A_status > 0 && currentTime >= A_watchdogTimer + WATCHDOG_TIMER) {
215       if(!A_watchdog) {
216         Serial.println("IN WATCHDOG A");
217         setSemaphoreAOff();
218       } else {
219         A_watchdog = false;
220       }
221       A_watchdogTimer = currentTime;
222     }
223   }
```

```
224
225 void setSemaphoreAOff() {
226   if(A_status != 0) {
227     A_status = 0;
228     A_lastTime = millis();
229     A_yellowLight = true;
230     A_error = false;
231     A_watchdogTimer = millis();
232     A_watchdog = true;
233   }
234 }
235
236 void setSemaphoreAOpen() {
237   if(A_status != 1) {
238     A_status = 1;
239     A_lastTime = millis();
240   }
241 }
242
243 void setSemaphoreAClose() {
244   if(A_status != 2) {
245     A_status = 2;
246     A_lastTime = millis();
247   }
248 }
249
250 boolean isSemaphoreAError() {
251   return A_error;
252 }
253
254 //-------------------------------------| SEMAPHORE B                    ↵
    |-----------------------------------------------------
255
256 void semaphoreB() {
257  long currentTime = millis();
258   if(B_status == 0 || B_error) {
259     if(currentTime - B_lastTime >= YELLOW_BLINKING_PERIOD) {
260       B_yellowLight = !B_yellowLight;
261       B_lastTime = currentTime;
262     }
263     B_redLight = false;
264     B_greenLight = false;
265   } else {
266     if(currentTime - B_lastTime < YELLOW_TRANSITION_TIME) {
267       B_redLight = false;
268       B_yellowLight = true;
269       B_greenLight = false;
270     } else {
271       if(B_status == 1) {
272         B_redLight = false;
273         B_yellowLight = false;
274         B_greenLight = true;
275       } else if(B_status == 2) {
276         B_redLight = true;
277         B_yellowLight = false;
278         B_greenLight = false;
```

```
279         }
280       }
281     }
282
283     if(B_redLight) {
284       digitalWrite(B_redLedPin, HIGH);
285     } else {
286       digitalWrite(B_redLedPin, LOW);
287     }
288     if(B_yellowLight) {
289       digitalWrite(B_yellowLedPin, HIGH);
290     } else {
291       digitalWrite(B_yellowLedPin, LOW);
292     }
293     if(B_greenLight) {
294       digitalWrite(B_greenLedPin, HIGH);
295     } else {
296       digitalWrite(B_greenLedPin, LOW);
297     }
298
299     if(B_status == 2 && (B_redLight || B_error)) {
300         digitalWrite(B_redLedPin, HIGH);
301         if(analogRead(B_redStatusPin) > 450) {
302           B_error = false;
303         } else {
304           digitalWrite(B_redLedPin, LOW);
305           B_error = true;
306         }
307     }
308
309     if(B_status > 0 && currentTime >= B_watchdogTimer + WATCHDOG_TIMER) {
310       if(!B_watchdog) {
311         Serial.println("IN WATCHDOG B");
312         setSemaphoreBOff();
313       } else {
314         B_watchdog = false;
315       }
316       B_watchdogTimer = currentTime;
317     }
318   }
319
320   void setSemaphoreBOff() {
321     if(B_status != 0) {
322       B_status = 0;
323       B_lastTime = millis();
324       B_yellowLight = true;
325       B_error = false;
326       B_watchdog = true;
327       B_watchdogTimer = millis();
328     }
329   }
330
331   void setSemaphoreBOpen() {
332     if(B_status != 1) {
333       B_status = 1;
334       B_lastTime = millis();
```

```
335   }
336 }
337
338 void setSemaphoreBClose() {
339   if(B_status != 2) {
340     B_status = 2;
341     B_lastTime = millis();
342   }
343 }
344
345 boolean isSemaphoreBError() {
346   return B_error;
347 }
348
349 //-------------------------------------| ENTRY CONTROLLER                    ↵
      |-----------------------------------------------------
350
351 int getEntryNumber() {
352   int entry = 0;
353   if(digitalRead(entryHSPin) == HIGH) {
354     entry = 1;
355   }
356   entry = entry << 1;
357   if(digitalRead(entryLSPin) == HIGH) {
358     entry = entry + 1;
359   }
360
361   return entry + 1;
362 }
363
364 void receiveMessage(uint8_t* message) {
365   uint8_t testIntegrity = 0;
366
367   for(int i = 0; i < 3; i++) {
368     testIntegrity = testIntegrity + message[i];
369   }
370
371   A_watchdog = true;
372   B_watchdog = true;
373
374   if(testIntegrity == message[3]) {
375     if(message[1] == RED) {
376       Serial.println("RED MESSAGE RECEIVED");
377       setSemaphoreAClose();
378       setSemaphoreBOpen();
379       generateMessage(entryNumber, ACK, 0);
380     } else if (message[1] == GREEN) {
381       Serial.println("GREEN MESSAGE RECEIVED");
382       setSemaphoreBClose();
383       setSemaphoreAOpen();
384       generateMessage(entryNumber, ACK, 0);
385     } else if (message[1] == OFF) {
386       Serial.println("OFF MESSAGE RECEIVED");
387       setSemaphoreAOff();
388       setSemaphoreBOff();
389       generateMessage(entryNumber, ACK, 0);
```

```
390       } else if (message[1] == PING) {
391         Serial.println("PING MESSAGE RECEIVED");
392         generateStatusMessage(entryNumber, STATUS, 0, isSemaphoreAError(),
              isSemaphoreBError(), A_pedestriansButtonPressed);
393         A_pedestriansButtonPressed = false;
394       }
395     }
396
397     if(entryNumber == 1) {
398       receiveMessageController(messageSent, mSLength);
399     }
400
401     A_watchdog = true;
402     B_watchdog = true;
403   }
404
405   void generateMessage(uint8_t sender, uint8_t messageType, uint8_t destination)
        {
406     messageSent[0] = sender;
407     messageSent[1] = messageType;
408     messageSent[2] = destination;
409     messageSent[3] = sender + messageType + destination;
410
411     mSLength = 4;
412   }
413
414   void generateStatusMessage(int sender, char messageType, int destination,
        boolean errorSemaphoreA, boolean errorSemaphoreB, boolean buttonPressed) {
415     uint8_t statusMessage = 0x00;
416
417     if(errorSemaphoreA) {
418       statusMessage = statusMessage | 0xE0;
419     }
420
421     if(errorSemaphoreB) {
422       statusMessage = statusMessage | 0x1C;
423     }
424
425     if(buttonPressed) {
426       statusMessage = statusMessage | 0x02;
427     }
428
429     messageSent[0] = sender;
430     messageSent[1] = messageType;
431     messageSent[2] = destination;
432     messageSent[3] = statusMessage;
433     messageSent[4] = sender + messageType + destination + statusMessage;
434
435     mSLength = 5;
436   }
437
438   void requestEvent(){
439     Wire.write(messageSent, mSLength);
440   }
441
442   void receiveEvent() {
```

```
443   int i = 0;
444   while (0 < Wire.available()) {
445     messageReceive[i] = Wire.read();
446     i++;
447   }
448
449   receiveMessage(messageReceive);
450 }
451
452 //-------------------------------------| CONTROLLER
      |-----------------------------------------------------
453 void controller() {
454   int changeController = false;
455   while(digitalRead(controllerButtonPin) == HIGH) {
456     if(!changeController) {
457       controllerOn = !controllerOn;
458     }
459     changeController = true;
460     sentOff = false;
461     entryOpen = 1;
462   }
463
464   // Go Off if Errors Persist
465   if (persistentErrors > 2) {
466     controllerOn = false;
467     sentOff = false;
468     persistentErrors = 0;
469   }
470
471   if(controllerOn) {
472     // -----------| READ PERIOD |------------
473     int angleRead = analogRead(potentiometerPin);
474     controller_period = map(angleRead, 0, 1023, 2000, 15000);
475
476     long currentTime = millis();
477
478     // --------------| PERIOD |--------------
479     if(currentTime >= controller_lastTimePeriod + controller_period ||
        changeController) {
480       //See if Button Has Been Pressed
481       if(reduceTime) {
482         long remainingTime = (controller_lastTimePeriod + controller_period) -
             currentTime;
483         controller_lastTimePeriod -= remainingTime / 2;
484         reduceTime = false;
485       }
486
487       //Close Entrys
488       for(int i = 0; i < MAX_ENTRY; i++) {
489         if(i != entryOpen - 1) {
490           sendMessage(generateMessageController(0, RED, i+1));
491         }
492       }
493
494       //Open Entry
495       sendMessage(generateMessageController(0, GREEN, entryOpen));
```

```
496
497       if(entryOpen == MAX_ENTRY) {
498         entryOpen = 1;
499       } else {
500         entryOpen++;
501       }
502
503       controller_lastTimePeriod = millis();
504     }
505
506     // --------------| PINGS |--------------
507     if(currentTime >= controller_lastTimePing + PING_PERIOD ||
        changeController) {
508
509       for(int i = 0; i < MAX_ENTRY; i++) {
510           sendMessage(generateMessageController(0, PING, i+1));
511       }
512
513       controller_lastTimePing = millis();
514     }
515
516     // --------------| ERRORS |--------------
517     if(currentTime >= controller_lastTimeErrors + ERRORS_PERIOD ||
        changeController) {
518
519       //See if Errors Exists
520       for(int i = 1; i <= MAX_ENTRY; i++) {
521         if(entryErrors[i]) {
522           persistentErrors++;
523           break;
524         }
525         if(i == MAX_ENTRY) {
526           persistentErrors = 0;
527         }
528       }
529
530       controller_lastTimeErrors = millis();
531     }
532
533   } else { // --------------| OFF |--------------
534     digitalWrite(controllerLedPin, LOW);
535     if(!sentOff) {
536       for(int i = 0; i < MAX_ENTRY; i++) {
537         sendMessage(generateMessageController(0, OFF, i+1));
538       }
539       sentOff = true;
540     }
541   }
542 }
543
544 uint8_t* generateMessageController(uint8_t sender, uint8_t messageType,
      uint8_t destination) {
545   messageController[0] = sender;
546   messageController[1] = messageType;
547   messageController[2] = destination;
548   messageController[3] = sender + messageType + destination;
```

```
549
550   return  messageController;
551 }
552
553 void sendMessage(uint8_t* message) {
554   blinkLedCommunication();
555   if(message[2] == 1) {
556     receiveMessage(message);
557   } else {
558     if(WIRE_MODE) {
559       sendWireMessage(message);
560     }
561   }
562   blinkLedCommunication();
563 }
564
565 void sendWireMessage(uint8_t* message){
566   int entry = message[2];
567   int mLength = 0;
568   if(message[1] == PING) {
569     mLength = 5;
570   } else {
571     mLength = 4;
572   }
573   Wire.beginTransmission(entry);
574   Wire.write(message, 4);
575   Wire.endTransmission();
576
577   Wire.requestFrom(entry, mLength);
578   int i = 0;
579
580   long startTime = millis();
581   long timeout = startTime + 200;
582
583   while (0 < Wire.available() && millis() < timeout) {
584     messageController[i] = Wire.read();
585     i++;
586   }
587
588   if(i < mLength) {
589     entryErrors[entry] = true;
590   }
591
592   receiveMessageController(messageController, mLength);
593 }
594
595 void receiveMessageController(uint8_t* message, int mLength) {
596   uint8_t testIntegrity = 0;
597
598   blinkLedCommunication();
599
600   for(int i = 0; i < mLength-1; i++) {
601     testIntegrity = testIntegrity + message[i];
602   }
603
604   int entry = message[0];
```

```
605
606   if(testIntegrity == message[mLength-1]) {
607     if(message[1] == ACK) {
608       entryErrors[entry] = false;
609       return;
610     } else if (message[1] == STATUS) {
611       uint8_t statusM = message[3];
612       uint8_t statusLeds = statusM & 0xFC;
613       uint8_t statusButton = statusM & 0x02;
614
615       if(statusLeds != 0) {
616         entryErrors[entry] = true;
617       } else {
618         entryErrors[entry] = false;
619       }
620
621       if(statusButton == 0x02) {
622         reduceTime = true;
623       }
624     }
625   } else {
626      entryErrors[entry] = true;
627   }
628
629   blinkLedCommunication();
630 }
631
632 void blinkLedCommunication() {
633   long currentTime = millis();
634
635   // --------------| BLINKING |-----------
636   if(currentTime >= controller_lastTimeBlink + 100) {
637     digitalWrite(controllerLedPin, !digitalRead(controllerLedPin));
638     controller_lastTimeBlink = currentTime;
639   }
640 }
641
642
```