

# Student Declaration of Authorship

<b>Course code and name:</b>	B31DG – Embedded Software
<b>Type of assessment:</b>	Individual
<b>Coursework Title:</b>	Assignment 3
<b>Student Name:</b>	Jack B Mavor
<b>Student ID Number:</b>	H00298763

**Declaration of authorship. By signing this form:**

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.
- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the [University's website](#), and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on [Academic Integrity and Plagiarism](#)

**Student Signature** (type your name): Jack Burgoyne Mavor

**Date:** 14/04/2022

Copy this page and insert it into your coursework file in front of your title page.  
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

**Your work will not be marked if a signed copy of this form is not included with your submission.**

# Embedded software – Assignment 3

15/04/2022

Jack B. Mavor

H0028763

Github repo: [B31DG\\_AS3\\_jbm5](#) (source files and revision log available)

Demo video: [Demo video](#)

## Problem

In this assignment the systems built in assignment 2 are rebuilt using a real time operating system (RTOS). The RTOS will carry out the following series of tasks at varying frequencies, these tasks and their rates are as follows:

Task	Frequency (Hz)	Period (ms)	Description
1	50	20*	Output a basic digital signal that will be used as a basic watchdog
2	5	200	Check a push button signal
3	1	1000	Measure the frequency of a square waveform
4	24	42*	Read an analogue input signal
5	24	42*	Filter the analogue signal
6	10	100	Sleep the computer for 1000 ticks using the “__asm__ __volatile__ (“nop”);” command
7	3	333*	Determine an error code based on the current filtered analogue signal
8	3	333*	Output the error code via an LED
9	0.2	5000	Output various data points in a CSV style serial output

\*Rounded to the nearest millisecond

## Development

While there are many open source RTOS's available, for this project I will be using “freeRTOS” as the ESP32 Arduino library includes a version of it built in. As most of the tasks have identical functionality to their assignment 2 counterparts their core functionality will remain the same, however these functions have been modified to avoid using global variables as much as possible. Where a task still requires access to variables it has been adjusted to either use Mutex's or Queue's. For any task requiring access the “data” struct and its components, a mutex is used to prevent memory access issues (this is shown in the bellow code snippet), without this protection the memory assigned to the struct variable could become corrupted due to being accessed by multiple tasks at the same time.

```
174 // Mutex protection statement
175 // prevents memory access issues
176 // attempts to take the mutex access token
177 // only continues if the access token is available
178 if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
179     duration = pulseIn(PULSE_IN, LOW);
180     data.frequency = 1000000.0 / (duration * 2);
181
182     // return mutex access token
183     xSemaphoreGive(data_mut);
184 }
```

Figure 1: Task 3 using a mutex to determine if it's safe to access the struct

For any task that required one way transfer of a single variable a queue was used, this was used for 2 cases, for passing the analogue voltage value from task 4 to task 5, and for passing the averaged analogue voltage from task 5 to task 7. In both cases a 1 entry queue was used as only the most recent versions of these data points were required by the other tasks.

```

229 // Mutex protection statement
230 // prevents memory access issues
231 // attempts to take the mutex access token
232 // only continues if the access token is available
233 if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
234
235     for (int i = 0; i < 4; i++)
236     {
237         y += analogs[i];
238     }
239
240     y = y / 4;
241
242     data.analog = y;
243
244     xQueueSend(average_queue, &y, 100);
245
246     // return mutex access token
247     xSemaphoreGive(data_mut);
248 }

```

Figure 2: task 5 adding the average value to the queue

```

281 // tasks next item from average queue and assign is to variable x
282 if (xQueueReceive(average_queue, &x, 100)){
283     if ( x > (3.3 / 2))
284     {
285         error_code = 1;
286     }
287     else
288     {
289         error_code = 0;
290     }
291 }

```

Figure 3: Task 7 receiving and dequeuing the average value from the queue

While the completed code contains very little in the way of wasted CPU usage, there are a couple of tasks that required the usage of either delays directly or functions that use delays. In particular, task 1 and task 3.

- Task 1 uses the “delayMicrosecond();” function which, due to the length of the delay, this should not extend the amount of time that the task uses the CPU.
- Task 3 uses the “pulseIn();” function, this function measures the width of a pulse by waiting until a pulse has completed.

While steps were taken to try and ensure that memory usage was kept to a minimum, such as minimising the number of global variables and redundant variables, some additional avenues for memory waste reduction were not implemented during this project. The main method of memory optimisation that was not appropriately implemented was determining the “high water mark” for each of the tasks. This would have been done using the “uxTaskGetStackHighWaterMark()” function built into freeRTOS, this function returns a value representing the number of bytes (while this is task in normal freeRTOS would be in words, the ESP version of the function uses bytes) between the tasks usage and the stack size. After using this function, a task’s stack size would then be shrunk by half that number, this is repeated until the stack size has been minimised while also not overflowing.

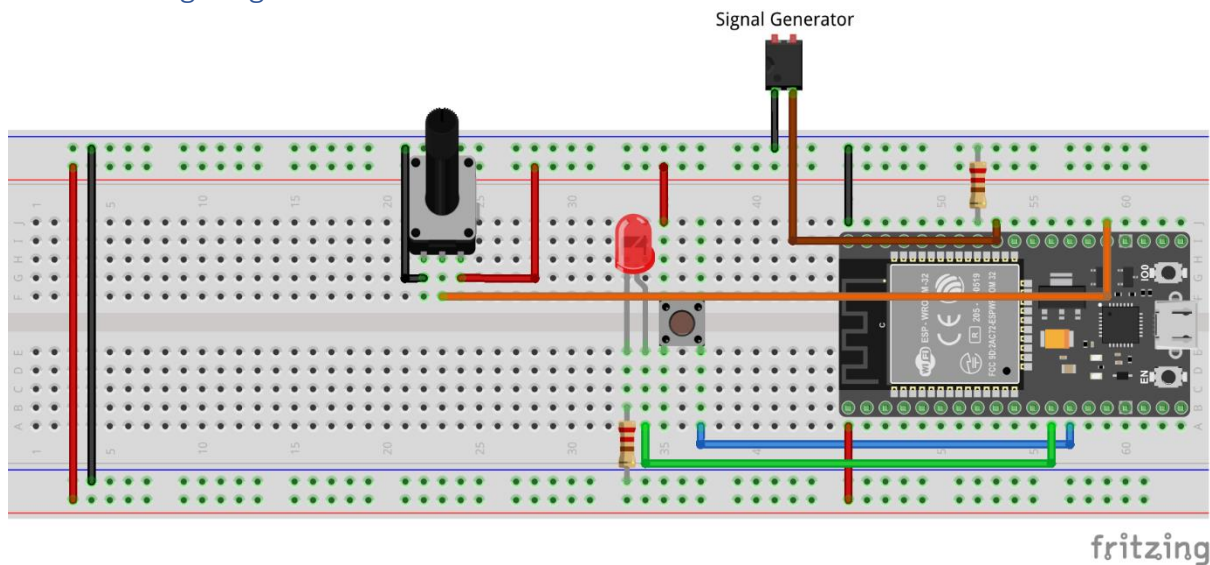
Additionally, the task priorities were based upon knowledge of which ones are likely to be more important to the overall functionality of the project as well as some basic testing. However, these priorities could be improved through further testing and additional analysis of the possible schedules.

# Appendix

## A. Commit History

Commits on Apr 15, 2022		
Minor comment corrections jbreaper committed 13 hours ago		d8aa34d <>
added declaration of authorship jbreaper committed 13 hours ago		9a8be7d <>
commenting and minor code changes jbreaper committed 13 hours ago		76aa84a <>
Commits on Apr 14, 2022		
task updates jbreaper committed yesterday	Verified	5f4f202 <>
readme error correction jbreaper committed yesterday	Verified	95bd0b2 <>
added demo jbreaper committed yesterday	Verified	b8feab9 <>
added demo jbreaper committed yesterday	Verified	fa66df7 <>
Create .gitignore.bak jbreaper committed yesterday		4d4cf05 <>
minor update to gitignore jbreaper committed yesterday		881cb90 <>
Commits on Apr 12, 2022		
Commits on Apr 12, 2022		
Final Debugging Completed jbreaper committed 3 days ago		1c4a7ea <>
Commits on Apr 8, 2022		
most code tested and adjusted jbreaper committed 7 days ago		92ade88 <>
Commits on Apr 4, 2022		
Corrective changes jbreaper committed 12 days ago		09d7f56 <>
Commits on Mar 24, 2022		
Minor Change jbreaper committed 22 days ago		587acff <>
gitignot jbreaper committed 22 days ago		d3880eb <>
Commits on Mar 23, 2022		
Added Tasks to project jbreaper committed 23 days ago		5b8e15a <>
Commits on Mar 19, 2022		
Basic frame work and constants created jbreaper committed 28 days ago		3389bba <>
Commits on Mar 16, 2022		
Initial project setup jbreaper committed on 16 Mar		9723313 <>
Initial commit jbreaper committed on 16 Mar		ae75084 <>
<a href="#">Newer</a> <a href="#">Older</a>		

## B. Wiring Diagram



## C. Code

```

1  /* Embedded Software - Assignment 3
2  * This project was programmed for the ESP32 microcontroller, it uses freeRTOS to run a series of tasks
3  * based on the provided brief.
4  *
5  * -----
6  * BRIEF
7  * -----
8  * For this assignment the tasks created in assignment 2 were recreated to utilise the freeRTOS framework.
9  * An additional requirement was added for this assignment, that being that in order for task 9 to print
10 * the data, the button from task 2 must be pressed.
11 * This the RTOS system must complete the following tasks at the related frequencies.
12 *
13 * | Task | Description | Frequency | Periodicity |
14 * |-----|-----|-----|-----|
15 * | 1 | Output a (digital) watchdog waveform (with same length and period of the 'Normal' operation of Sig8 in Assignment 1). Timings should be within 5%. | 50Hz | 20ms |
16 * | 2 | Monitor one digital input (to be connected to a pushbutton/switch or a signal generator for students using Proteus). | 5Hz | 200ms |
17 * | 3 | Measure the frequency of a 3.3v square wave signal. The frequency will be in the range 500Hz to 1000Hz and the signal will be a standard square wave (50% duty cycle). Accuracy to 2.5% is acceptable. | 1Hz | 1000ms |
18 * | 4 | Read one analogue input. The analogue input must be connected to a maximum of 3.3V, using a potentiometer. | 24Hz | 42ms |
19 * | 5 | Compute filtered analogue value, by averaging the last 4 readings. | 24Hz | 42ms |
20 * | 6 | Execute 1000 times the following instruction:
21 * |   |   __asm__ __volatile__ ("nop");
22 * |   |   The statement could be repeated using a single loop, or broken down into multiple loops (e.g. to be executed in different slots of the cyclic executive).
23 * |   |   |
24 * |   |   |
25 * |   |   |
26 * |   |   |
27 * |   |   |
28 * |   |   |
29 * |   |   |
30 * |   |   |
31 * |   |   |
32 * |   |   |
33 * |   |   |
34 * |   |   |
35 * |   |   |
36 * | 7 | Perform the following check: if (average_analogue_in > half of maximum range for analogue input):
37 * |   |   error_code = 1
38 * |   |   else:
39 * |   |   error_code = 0
40 * |   |   |
41 * |   |   |
42 * | 8 | Visualise error_code using an LED.
43 * |   |   |
44 * |   |   |
45 * | 9 | Check if the push button is currently in it's pressed state, if it is log the following information every five (5) seconds (in comma separated format, e.g. CSV) to the serial port:
46 * |   |   - State of the digital input (pushbutton / switch);
47 * |   |   - Frequency value (Hz, as an integer);
48 * |   |   - Filtered analogue input.
49 * |   |   |
50 * |   |   |
51 * |   |   |
52 * |   |   |
53 * |   |   |
54 * |   |   |
55 * |   |   |
56 * |   |   |
57 * |   |   |
58 * |   |   |
59 * |   |   |
60 * |   |   |
61 * |   |   |
62 * |   |   |
63 * |   |   |
64 * |   |   |
65 * |   |   |
66 * |   |   |
67 * |   |   |
68 * |   |   |
69 * |   |   |
70 * |   |   |
71 * |   |   |
72 * |   |   |
73 * |   |   |
74 * |   |   |
75 * |   |   |
76 * |   |   |
77 * |   |   |
78 * |   |   |
79 * |   |   |
80 * |   |   |
81 * |   |   |
82 * |   |   |
83 * |   |   |
84 * |   |   |
85 * |   |   |
86 * |   |   |
87 * |   |   |
88 * |   |   |
89 * |   |   |
90 * |   |   |
91 * |   |   |
92 * |   |   |
93 * |   |   |
94 * |   |   |
95 * |   |   |
96 * |   |   |
97 * |   |   |
98 * |   |   |
99 * |   |   |
100 * |   |   |
101 * |   |   |
102 * |   |   |
103 * |   |   |
104 * |   |   |
105 * |   |   |
106 * |   |   |
107 * |   |   |
108 * |   |   |
109 * |   |   |
110 * |   |   |
111 * |   |   |
112 * |   |   |
113 * |   |   |
114 * |   |   |
115 * |   |   |
116 * |   |   |
117 * |   |   |
118 * |   |   |
119 * |   |   |
120 * |   |   |
121 * |   |   |
122 * |   |   |
123 * |   |   |
124 * |   |   |
125 * |   |   |
126 * |   |   |
127 * |   |   |
128 * |   |   |
129 * |   |   |
130 * |   |   |
131 * |   |   |
132 * |   |   |
133 * |   |   |
134 * |   |   |
135 * |   |   |
136 * |   |   |
137 * |   |   |
138 * |   |   |
139 * |   |   |
140 * |   |   |
141 * |   |   |
142 * |   |   |
143 * |   |   |
144 * |   |   |
145 * |   |   |
146 * |   |   |
147 * |   |   |
148 * |   |   |
149 * |   |   |
150 * |   |   |
151 * |   |   |
152 * |   |   |
153 * |   |   |
154 * |   |   |
155 * |   |   |
156 * |   |   |
157 * |   |   |
158 * |   |   |
159 * |   |   |
160 * |   |   |
161 * |   |   |
162 * |   |   |
163 * |   |   |
164 * |   |   |
165 * |   |   |
166 * |   |   |
167 * |   |   |
168 * |   |   |
169 * |   |   |
170 * |   |   |
171 * |   |   |
172 * |   |   |
173 * |   |   |
174 * |   |   |
175 * |   |   |
176 * |   |   |
177 * |   |   |
178 * |   |   |
179 * |   |   |
180 * |   |   |
181 * |   |   |
182 * |   |   |
183 * |   |   |
184 * |   |   |
185 * |   |   |
186 * |   |   |
187 * |   |   |
188 * |   |   |
189 * |   |   |
190 * |   |   |
191 * |   |   |
192 * |   |   |
193 * |   |   |
194 * |   |   |
195 * |   |   |
196 * |   |   |
197 * |   |   |
198 * |   |   |
199 * |   |   |
200 * |   |   |
201 * |   |   |
202 * |   |   |
203 * |   |   |
204 * |   |   |
205 * |   |   |
206 * |   |   |
207 * |   |   |
208 * |   |   |
209 * |   |   |
210 * |   |   |
211 * |   |   |
212 * |   |   |
213 * |   |   |
214 * |   |   |
215 * |   |   |
216 * |   |   |
217 * |   |   |
218 * |   |   |
219 * |   |   |
220 * |   |   |
221 * |   |   |
222 * |   |   |
223 * |   |   |
224 * |   |   |
225 * |   |   |
226 * |   |   |
227 * |   |   |
228 * |   |   |
229 * |   |   |
230 * |   |   |
231 * |   |   |
232 * |   |   |
233 * |   |   |
234 * |   |   |
235 * |   |   |
236 * |   |   |
237 * |   |   |
238 * |   |   |
239 * |   |   |
240 * |   |   |
241 * |   |   |
242 * |   |   |
243 * |   |   |
244 * |   |   |
245 * |   |   |
246 * |   |   |
247 * |   |   |
248 * |   |   |
249 * |   |   |
250 * |   |   |
251 * |   |   |
252 * |   |   |
253 * |   |   |
254 * |   |   |
255 * |   |   |
256 * |   |   |
257 * |   |   |
258 * |   |   |
259 * |   |   |
260 * |   |   |
261 * |   |   |
262 * |   |   |
263 * |   |   |
264 * |   |   |
265 * |   |   |
266 * |   |   |
267 * |   |   |
268 * |   |   |
269 * |   |   |
270 * |   |   |
271 * |   |   |
272 * |   |   |
273 * |   |   |
274 * |   |   |
275 * |   |   |
276 * |   |   |
277 * |   |   |
278 * |   |   |
279 * |   |   |
280 * |   |   |
281 * |   |   |
282 * |   |   |
283 * |   |   |
284 * |   |   |
285 * |   |   |
286 * |   |   |
287 * |   |   |
288 * |   |   |
289 * |   |   |
290 * |   |   |
291 * |   |   |
292 * |   |   |
293 * |   |   |
294 * |   |   |
295 * |   |   |
296 * |   |   |
297 * |   |   |
298 * |   |   |
299 * |   |   |
300 * |   |   |
301 * |   |   |
302 * |   |   |
303 * |   |   |
304 * |   |   |
305 * |   |   |
306 * |   |   |
307 * |   |   |
308 * |   |   |
309 * |   |   |
310 * |   |   |
311 * |   |   |
312 * |   |   |
313 * |   |   |
314 * |   |   |
315 * |   |   |
316 * |   |   |
317 * |   |   |
318 * |   |   |
319 * |   |   |
320 * |   |   |
321 * |   |   |
322 * |   |   |
323 * |   |   |
324 * |   |   |
325 * |   |   |
326 * |   |   |
327 * |   |   |
328 * |   |   |
329 * |   |   |
330 * |   |   |
331 * |   |   |
332 * |   |   |
333 * |   |   |
334 * |   |   |
335 * |   |   |
336 * |   |   |
337 * |   |   |
338 * |   |   |
339 * |   |   |
340 * |   |   |
341 * |   |   |
342 * |   |   |
343 * |   |   |
344 * |   |   |
345 * |   |   |
346 * |   |   |
347 * |   |   |
348 * |   |   |
349 * |   |   |
350 * |   |   |
351 * |   |   |
352 * |   |   |
353 * |   |   |
354 * |   |   |
355 * |   |   |
356 * |   |   |
357 * |   |   |
358 * |   |   |
359 * |   |   |
360 * |   |   |
361 * |   |   |
362 * |   |   |
363 * |   |   |
364 * |   |   |
365 * |   |   |
366 * |   |   |
367 * |   |   |
368 * |   |   |
369 * |   |   |
370 * |   |   |
371 * |   |   |
372 * |   |   |
373 * |   |   |
374 * |   |   |
375 * |   |   |
376 * |   |   |
377 * |   |   |
378 * |   |   |
379 * |   |   |
380 * |   |   |
381 * |   |   |
382 * |   |   |
383 * |   |   |
384 * |   |   |
385 * |   |   |
386 * |   |   |
387 * |   |   |
388 * |   |   |
389 * |   |   |
390 * |   |   |
391 * |   |   |
392 * |   |   |
393 * |   |   |
394 * |   |   |
395 * |   |   |
396 * |   |   |
397 * |   |   |
398 * |   |   |
399 * |   |   |
400 * |   |   |
401 * |   |   |
402 * |   |   |
403 * |   |   |
404 * |   |   |
405 * |   |   |
406 * |   |   |
407 * |   |   |
408 * |   |   |
409 * |   |   |
410 * |   |   |
411 * |   |   |
412 * |   |   |
413 * |   |   |
414 * |   |   |
415 * |   |   |
416 * |   |   |
417 * |   |   |
418 * |   |   |
419 * |   |   |
420 * |   |   |
421 * |   |   |
422 * |   |   |
423 * |   |   |
424 * |   |   |
425 * |   |   |
426 * |   |   |
427 * |   |   |
428 * |   |   |
429 * |   |   |
430 * |   |   |
431 * |   |   |
432 * |   |   |
433 * |   |   |
434 * |   |   |
435 * |   |   |
436 * |   |   |
437 * |   |   |
438 * |   |   |
439 * |   |   |
440 * |   |   |
441 * |   |   |
442 * |   |   |
443 * |   |   |
444 * |   |   |
445 * |   |   |
446 * |   |   |
447 * |   |   |
448 * |   |   |
449 * |   |   |
450 * |   |   |
451 * |   |   |
452 * |   |   |
453 * |   |   |
454 * |   |   |
455 * |   |   |
456 * |   |   |
457 * |   |   |
458 * |   |   |
459 * |   |   |
460 * |   |   |
461 * |   |   |
462 * |   |   |
463 * |   |   |
464 * |   |   |
465 * |   |   |
466 * |   |   |
467 * |   |   |
468 * |   |   |
469 * |   |   |
470 * |   |   |
471 * |   |   |
472 * |   |   |
473 * |   |   |
474 * |   |   |
475 * |   |   |
476 * |   |   |
477 * |   |   |
478 * |   |   |
479 * |   |   |
480 * |   |   |
481 * |   |   |
482 * |   |   |
483 * |   |   |
484 * |   |   |
485 * |   |   |
486 * |   |   |
487 * |   |   |
488 * |   |   |
489 * |   |   |
490 * |   |   |
491 * |   |   |
492 * |   |   |
493 * |   |   |
494 * |   |   |
495 * |   |   |
496 * |   |   |
497 * |   |   |
498 * |   |   |
499 * |   |   |
500 * |   |   |
501 * |   |   |
502 * |   |   |
503 * |   |   |
504 * |   |   |
505 * |   |   |
506 * |   |   |
507 * |   |   |
508 * |   |   |
509 * |   |   |
510 * |   |   |
511 * |   |   |
512 * |   |   |
513 * |   |   |
514 * |   |   |
515 * |   |   |
516 * |   |   |
517 * |   |   |
518 * |   |   |
519 * |   |   |
520 * |   |   |
521 * |   |   |
522 * |   |   |
523 * |   |   |
524 * |   |   |
525 * |   |   |
526 * |   |   |
527 * |   |   |
528 * |   |   |
529 * |   |   |
530 * |   |   |
531 * |   |   |
532 * |   |   |
533 * |   |   |
534 * |   |   |
535 * |   |   |
536 * |   |   |
537 * |   |   |
538 * |   |   |
539 * |   |   |
540 * |   |   |
541 * |   |   |
542 * |   |   |
543 * |   |   |
544 * |   |   |
545 * |   |   |
546 * |   |   |
547 * |   |   |
548 * |   |   |
549 * |   |   |
550 * |   |   |
551 * |   |   |
552 * |   |   |
553 * |   |   |
554 * |   |   |
555 * |   |   |
556 * |   |   |
557 * |   |   |
558 * |   |   |
559 * |   |   |
560 * |   |   |
561 * |   |   |
562 * |   |   |
563 * |   |   |
564 * |   |   |
565 * |   |   |
566 * |   |   |
567 * |   |   |
568 * |   |   |
569 * |   |   |
570 * |   |   |
571 * |   |   |
572 * |   |   |
573 * |   |   |
574 * |   |   |
575 * |   |   |
576 * |   |   |
577 * |   |   |
578 * |   |   |
579 * |   |   |
580 * |   |   |
581 * |   |   |
582 * |   |   |
583 * |   |   |
584 * |   |   |
585 * |   |   |
586 * |   |   |
587 * |   |   |
588 * |   |   |
589 * |   |   |
590 * |   |   |
591 * |   |   |
592 * |   |   |
593 * |   |   |
594 * |   |   |
595 * |   |   |
596 * |   |   |
597 * |   |   |
598 * |   |   |
599 * |   |   |
600 * |   |   |
601 * |   |   |
602 * |   |   |
603 * |   |   |
604 * |   |   |
605 * |   |   |
606 * |   |   |
607 * |   |   |
608 * |   |   |
609 * |   |   |
610 * |   |   |
611 * |   |   |
612 * |   |   |
613 * |   |   |
614 * |   |   |
615 * |   |   |
616 * |   |   |
617 * |   |   |
618 * |   |   |
619 * |   |   |
620 * |   |   |
621 * |   |   |
622 * |   |   |
623 * |   |   |
624 * |   |   |
625 * |   |   |
626 * |   |   |
627 * |   |   |
628 * |   |   |
629 * |   |   |
630 * |   |   |
631 * |   |   |
632 * |   |   |
633 * |   |   |
634 * |   |   |
635 * |   |   |
636 * |   |   |
637 * |   |   |
638 * |   |   |
639 * |   |   |
640 * |   |   |
641 * |   |   |
642 * |   |   |
643 * |   |   |
644 * |   |   |
645 * |   |   |
646 * |   |   |
647 * |   |   |
648 * |   |   |
649 * |   |   |
650 * |   |   |
651 * |   |   |
652 * |   |   |
653 * |   |   |
654 * |   |   |
655 * |   |   |
656 * |   |   |
657 * |   |   |
658 * |   |   |
659 * |   |   |
660 * |   |   |
661 * |   |   |
662 * |   |   |
663 * |   |   |
664 * |   |   |
665 * |   |   |
666 * |   |   |
667 * |   |   |
668 * |   |   |
669 * |   |   |
670 * |   |   |
671 * |   |   |
672 * |   |   |
673 * |   |   |
674 * |   |   |
675 * |   |   |
676 * |   |   |
677 * |   |   |
678 * |   |   |
679 * |   |   |
680 * |   |   |
681 * |   |   |
682 * |   |   |
683 * |   |   |
684 * |   |   |
685 * |   |   |
686 * |   |   |
687 * |   |   |
688 * |   |   |
689 * |   |   |
690 * |   |   |
691 * |   |   |
692 * |   |   |
693 * |   |   |
694 * |   |   |
695 * |   |   |
696 * |   |   |
697 * |   |   |
698 * |   |   |
699 * |   |   |
700 * |   |   |
701 * |   |   |
702 * |   |   |
703 * |   |   |
704 * |   |   |
705 * |   |   |
706 * |   |   |
707 * |   |   |
708 * |   |   |
709 * |   |   |
710 * |   |   |
711 * |   |   |
712 * |   |   |
713 * |   |   |
714 * |   |   |
715 * |   |   |
716 * |   |   |
717 * |   |   |
718 * |   |   |
719 * |   |   |
720 * |   |   |
721 * |   |   |
722 * |   |   |
723 * |   |   |
724 * |   |   |
725 * |   |   |
726 * |   |   |
727 * |   |   |
728 * |   |   |
729 * |   |   |
730 * |   |   |
731 * |   |   |
732 * |   |   |
733 * |   |   |
734 * |   |   |
735 * |   |   |
736 * |   |   |
737 * |   |   |
738 * |   |   |
739 * |   |   |
740 * |   |   |
741 * |   |   |
742 * |   |   |
743 * |   |   |
744 * |   |   |
745 * |   |   |
746 * |   |   |
747 * |   |   |
748 * |   |   |
749 * |   |   |
750 * |   |   |
751 * |   |   |
752 * |   |   |
753 * |   |   |
754 * |   |   |
755 * |   |   |
756 * |   |   |
757 * |   |   |
758 * |   |   |
759 * |   |   |
760 * |   |   |
761 * |   |   |
762 * |   |   |
763 * |   |   |
764 * |   |   |
765 * |   |   |
766 * |   |   |
767 * |   |   |
768 * |   |   |
769 * |   |   |
770 * |   |   |
771 * |   |   |
772 * |   |   |
773 * |   |   |
774 * |   |   |
775 * |   |   |
776 * |   |   |
777 * |   |   |
778 * |   |   |
779 * |   |   |
780 * |   |   |
781 * |   |   |
782 * |   |   |
783 * |   |   |
784 * |   |   |
785 * |   |   |
786 * |   |   |
787 * |   |   |
788 * |   |   |
789 * |   |   |
790 * |   |   |
791 * |   |   |
792 * |   |   |
793 * |   |   |
794 * |   |   |
795 * |   |   |
796 * |   |   |
797 * |   |   |
798 * |   |   |
799 * |   |   |
800 * |   |   |
801 * |   |   |
802 * |   |   |
803 * |   |   |
804 * |   |   |
805 * |   |   |
806 * |   |   |
807 * |   |   |
808 * |   |   |
809 * |   |   |
810 * |   |   |
811 * |   |   |
812 * |   |   |
813 * |   |   |
814 * |   |   |
815 * |   |   |
816 * |   |   |
817 * |   |   |
818 * |   |   |
819 * |   |   |
820 * |   |   |
821 * |   |   |
822 * |   |   |
823 * |   |   |
824 * |   |   |
825 * |   |   |
826 * |   |   |
827 * |   |   |
828 * |   |   |
829 * |   |   |
830 * |   |   |
831 * |   |   |
832 * |   |   |
833 * |   |   |
834 * |   |   |
835 * |   |   |
836 * |   |   |
837 * |   |   |
838 * |   |   |
839 * |   |   |
840 * |   |   |
841 * |   |   |
842 * |   |   |
843 * |   |   |
844 * |   |   |
845 * |   |   |
846 * |   |   |
847 * |   |   |
848 * |   |   |
849 * |   |   |
850 * |   |   |
851 * |   |   |
852 * |   |   |
853 * |   |   |
854 * |   |   |
855 * |   |   |
856 * |   |   |
857 * |   |   |
858 * |   |   |
859 * |   |   |
860 * |   |   |
861 * |   |   |
862 * |   |   |
863 * |   |   |
864 * |   |   |
865 * |   |   |
866 * |   |   |
867 * |   |   |
868 * |   |   |
869 * |   |   |
870 * |   |   |
871 * |   |   |
872 * |   |   |
873 * |   |   |
874 * |   |   |
875 * |   |   |
876 * |   |   |
877 * |   |   |
878 * |   |   |
879 * |   |   |
880 * |   |   |
881 * |   |   |
882 * |   |   |
883 * |   |   |
884 * |   |   |
885 * |   |   |
886 * |   |   |
887 * |   |   |
888 * |   |   |
889 * |   |   |
890 * |   |   |
891 * |   |   |
892 * |   |   |
893 * |   |   |
894 * |   |   |
895 * |   |   |
896 * |   |   |
897 * |   |   |
898 * |   |   |
899 * |   |   |
900 * |   |   |
901 * |   |   |
902 * |   |   |
903 * |   |   |
904 * |   |   |
905 * |   |   |
906 * |   |   |
907 * |   |   |
908 * |   |   |
909 * |   |   |
910 * |   |   |
911 * |   |   |
912 * |   |   |
913 * |   |   |
914 * |   |   |
915 * |   |   |
916 * |   |   |
917 * |   |   |
918 * |   |   |
919 * |   |   |
920 * |   |   |
921 * |   |   |
922 * |   |   |
923 * |   |   |
924 * |   |   |
925 * |   |   |
926 * |   |   |
927 * |   |   |
928 * |   |   |
929 * |   |   |
930 * |   |   |
931 * |   |   |
932 * |   |   |
933 * |   |   |
934 * |   |   |
935 * |   |   |
936 * |   |   |
937 * |   |   |
938 * |   |   |
939 * |   |   |
940 * |   |   |
941 * |   |   |
942 * |   |   |
943 * |   |   |
944 * |   |   |
945 * |   |   |
946 * |   |   |
947 * |   |   |
948 * |   |   |
949 * |   |   |
950 * |   |   |
951 * |   |   |
952 * |   |   |
953 * |   |   |
954 * |   |   |
955 * |   |   |
956 * |   |   |
957 * |   |   |
958 * |   |   |
959 * |   |   |
960 * |   |   |
961 * |   |   |
962 * |   |   |
963 * |   |   |
964 * |   |   |
965 * |   |   |
966 * |   |   |
967 * |   |   |
968 * |   |   |
969 * |   |   |
970 * |   |   |
971 * |   |   |
972 * |   |   |
973 * |   |   |
974 * |   |   |
975 * |   |   |
976 * |   |   |
977 * |   |   |
978 * |   |   |
979 * |   |   |
980 * |   |   |
981 * |   |   |
982 * |   |   |
983 * |   |   |
984 * |   |   |
985 * |   |   |
986 * |   |   |
987 * |   |   |
988 * |   |   |
989 * |   |   |
990 * |   |   |
991 * |   |   |
992 * |   |   |
993 * |   |   |
994 * |   |   |
995 * |   |   |
996 * |   |   |
997 * |   |   |
998 * |   |   |
999 * |   |   |
1000 * |   |   |

```

```

71  * has been rounded to 42 as the ESP32's
72  * freeRTOS tick rate is 1ms
73  */
74  #define R_T1 20          // 50 Hz
75  #define R_T2 200        // 5 Hz
76  #define R_T3 1000       // 1 Hz
77  ~ /* while hz to ms gives 41.666666...this
78  * has been rounded to 42 as the ESP32's
79  * freeRTOS tick rate is 1ms
80  */
81  #define R_T4 42          // 24 Hz
82  #define R_T5 42          // 24 Hz
83  #define R_T6 100        // 10 Hz
84  ~ /* while hz to ms gives 333.33333... this
85  * has been rounded to 333 as the ESP32's
86  * freeRTOS tick rate is 1ms
87  */
88  #define R_T7 333         // 3 Hz
89  #define R_T8 333         // 3 Hz
90  #define R_T9 1000        // 0.2 Hz
91
92  ~ // alternate to vtaskdelay that allows the delay to
93  // be defined using milliseconds rather than ticks
94  #define TaskDelay(x) vTaskDelay(x/portTICK_PERIOD_MS)
95
96  // prototype functions for tasks
97  void task_1(void *pvParameters);
98  void task_2(void *pvParameters);
99  void task_3(void *pvParameters);
100 void task_4(void *pvParameters);
101 void task_5(void *pvParameters);
102 void task_6(void *pvParameters);
103 void task_7(void *pvParameters);
104 void task_8(void *pvParameters);
105 void task_9(void *pvParameters);
106
107 // Defined Queue handles for analog signal
108 // and averaged analogue signal queues
109 static QueueHandle_t analog_queue;
110 static QueueHandle_t average_queue;
111
112 // Mutex to protect the "data" struct variable
113 static SemaphoreHandle_t data_mut;
114
115 // error code variable for low analogue voltage check
116 volatile int error_code;
117
118 // Struct to collect all data to be output in task 9
119 struct Data
120 {
121     bool button = false;
122     int frequency = 0;
123     float analog = 0;
124 } data;
125
126 // generate pulse of width 50us
127 void task_1(void *pvParameters)
128 {
129     (void) pvParameters;
130     for (;;)
131     {
132         digitalWrite(WD, HIGH);
133         // 50 microsecond delay
134         delayMicroseconds(50);
135         digitalWrite(WD, LOW);
136
137         // delays task for rate
138         // before restarting
139         TaskDelay(R_T1);
140     }
141 }
142
143 // read input of a button on pin PB1
144 void task_2(void *pvParameters)
145 {
146     (void) pvParameters;
147     for (;;)
148     {
149         // Mutex protection statement
150         // prevents memory access issues
151         // attempts to take the mutex access token
152         // only continues if the access token is available
153         if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
154             data.button = digitalRead(PB1);
155
156             // return mutex access token
157             xSemaphoreGive(data_mut);
158         }
159
160         // delays task for rate
161         // before restarting
162         TaskDelay(R_T2);
163     }
164 }
165
166 // determine frequency of digital signal on pin PULSE_IN
167 void task_3(void *pvParameters)
168 {
169     float duration = 0;
170
171     (void) pvParameters;
172     for (;;)
173     {
174         // Mutex protection statement
175         // prevents memory access issues

```

```

176 // attempts to take the mutex access token
177 // only continues if the access token is available
178 if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
179     duration = pulseIn(PULSE_IN, LOW);
180     data.frequency = 1000000.0 / (duration * 2);
181
182     // return mutex access token
183     xSemaphoreGive(data_mut);
184 }
185
186 // delays task for rate
187 // before restarting
188 TaskDelay(R_T3);
189 }
190 }
191
192 // read analogue input on pin A_IN
193 void task_4(void *pvParameters)
194 {
195     int x = 0;
196
197     (void) pvParameters;
198     for (;;)
199     {
200         x = analogRead(A_IN);
201         xQueueSend(analog_queue, &x, 100);
202
203         // delays task for rate
204         // before restarting
205         TaskDelay(R_T4);
206     }
207 }
208
209 // Average last 4 analog input readings
210 void task_5(void *pvParameters)
211 {
212     float analogs[4] = {0, 0, 0, 0};
213
214     (void) pvParameters;
215     for (;;)
216     {
217         int x = 0;
218         float y = 0;
219
220         if(xQueueReceive(analog_queue, &x, 100)){
221             for (int i = 1; i < 4; i++)
222             {
223                 analogs[i - 1] = analogs[i];
224             }
225
226             analogs[3] = x * (3.3 / 4095);
227         }
228
229         // Mutex protection statement
230         // prevents memory access issues
231         // attempts to take the mutex access token
232         // only continues if the access token is available
233         if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
234
235             for (int i = 0; i < 4; i++)
236             {
237                 y += analogs[i];
238             }
239
240             y = y / 4;
241
242             data.analog = y;
243
244             xQueueSend(average_queue, &y, 100);
245
246             // return mutex access token
247             xSemaphoreGive(data_mut);
248         }
249
250         // delays task for rate
251         // before restarting
252         TaskDelay(R_T5);
253     }
254 }
255
256 // use "__asm__ __volatile__ ("nop");" 1000 times
257 void task_6(void *pvParameters)
258 {
259     (void) pvParameters;
260     for (;;)
261     {
262         for (int i = 0; i < 1000; i++)
263         {
264             __asm__ __volatile__ ("nop");
265         }
266
267         // delays task for rate
268         // before restarting
269         TaskDelay(R_T6);
270     }
271 }
272
273 // determine error code based on average analogue reading
274 void task_7(void *pvParameters)
275 {
276     (void) pvParameters;
277     for (;;)
278     {
279         float x = 0;
280

```

```

281 // tasks next item from average queue and assign is to variable x
282 if (xQueueReceive(average_queue, &x, 100)){
283     if ( x > (3.3 / 2))
284     {
285         error_code = 1;
286     }
287     else
288     {
289         error_code = 0;
290     }
291 }
292
293 // delays task for rate
294 // before restarting
295 TaskDelay(R_T7);
296 }
297 }
298
299 // light LED based on error code
300 void task_8(void *pvParameters)
301 {
302     (void) pvParameters;
303     for (;;)
304     {
305         digitalWrite(LED, error_code);
306
307         // delays task for rate
308         // before restarting
309         TaskDelay(R_T8);
310     }
311 }
312
313 // print; button PB1 state, Frequency of PULSE_IN, and average of analogue input A_IN
314 // This data is presented in a CSV format
315 void task_9(void *pvParameters)
316 {
317     (void) pvParameters;
318     for (;;)
319     {
320         // Mutex protection statement
321         // prevents memory access issues
322         // attempts to take the mutex access token
323         // only continues if the access token is available
324         if(xSemaphoreTake(data_mut, portMAX_DELAY) == pdTRUE){
325             if (data.button == 1)
326             {
327                 Serial.print(data.button);
328                 Serial.print(", \t\t\t");
329                 Serial.print(data.frequency);
330                 Serial.print(", \t\t");
331                 Serial.print(data.analog);
332                 Serial.print("\n");
333             }
334
335             // return mutex access token
336             xSemaphoreGive(data_mut);
337         }
338
339         // delays task for rate
340         // before restarting
341         TaskDelay(R_T9);
342     }
343 }
344
345 void setup()
346 {
347     Serial.begin(115200);
348
349     // setup pins
350     pinMode(LED, OUTPUT);
351     pinMode(LED, OUTPUT);
352     pinMode(PB1, INPUT);
353     pinMode(A_IN, INPUT);
354     pinMode(PULSE_IN, INPUT);
355
356     // header to assist with reading the CSV formatted output
357     Serial.println("-----");
358     Serial.println("Switch, \tFrequency, \tInput");
359     Serial.println("-----");
360
361     //setup mutex in order to protect the "data" struct
362     data_mut = xSemaphoreCreateMutex();
363
364     //setup queues for analog readings and averaged analogue readings
365     analog_queue = xQueueCreate(1, sizeof(float));
366     average_queue = xQueueCreate(1, sizeof(float));
367
368     // setup FreeRTOS tasks for each of the task functions
369     /* xTaskCreate(
370      *     task function,
371      *     name for debugging purposes,
372      *     stack size,
373      *     pvParameter
374      */
375
376     xTaskCreate(
377         task_1,
378         "task 1",
379         512,
380         NULL,
381         4,
382         NULL);
383
384     xTaskCreate(
385         task_2,

```



```

386     "task 2",
387     512,
388     NULL,
389     3,
390     NULL);
391
392     xTaskCreate(
393         task_3,
394         "task 3",
395         1024,
396         NULL,
397         3,
398         NULL);
399
400     xTaskCreate(
401         task_4,
402         "task 4",
403         1024,
404         NULL,
405         2,
406         NULL);
407
408     xTaskCreate(
409         task_5,
410         "task 5",
411         1024,
412         NULL,
413         1,
414         NULL);
415
416     xTaskCreate(
417         task_6,
418         "task 6",
419         512,
420         NULL,
421         3,
422         NULL);
423
424     xTaskCreate(
425         task_7,
426         "task 7",
427         1024,
428         NULL,
429         4,
430         NULL);
431
432     xTaskCreate(
433         task_8,
434         "task 8",
435         512,
436         NULL,
437         3,
438         NULL);
439
440     xTaskCreate(
441         task_9,
442         "task 9",
443         1024,
444         NULL,
445         4,
446         NULL);
447 }
448
449 void loop(){}
450

```