Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	B31DG - Embedded Software
Type of assessment:	**Individual**
Coursework Title:	Assignment 2
Student Name:	Jack B Mavor
Student ID Number:	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: *11/03/2022*

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.

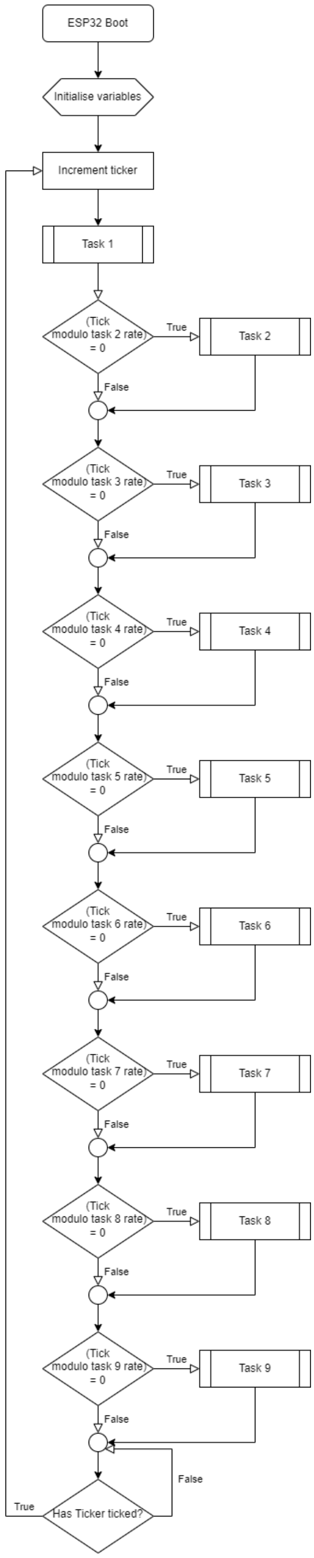
Embedded software – Assignment 2

11/03/2022

Jack B. Mavor

H0028763

Github repo: B31DG_AS2_jbm5 (source files and revision log available)



Problem

For this assignment a basic cyclic executive system was created for the ESP32 board. This was done primarily using the ESP32 version of the Ticker.h library. The cyclic executive system will carry out a series of tasks at varying frequencies, these tasks and their rates are as follows:

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

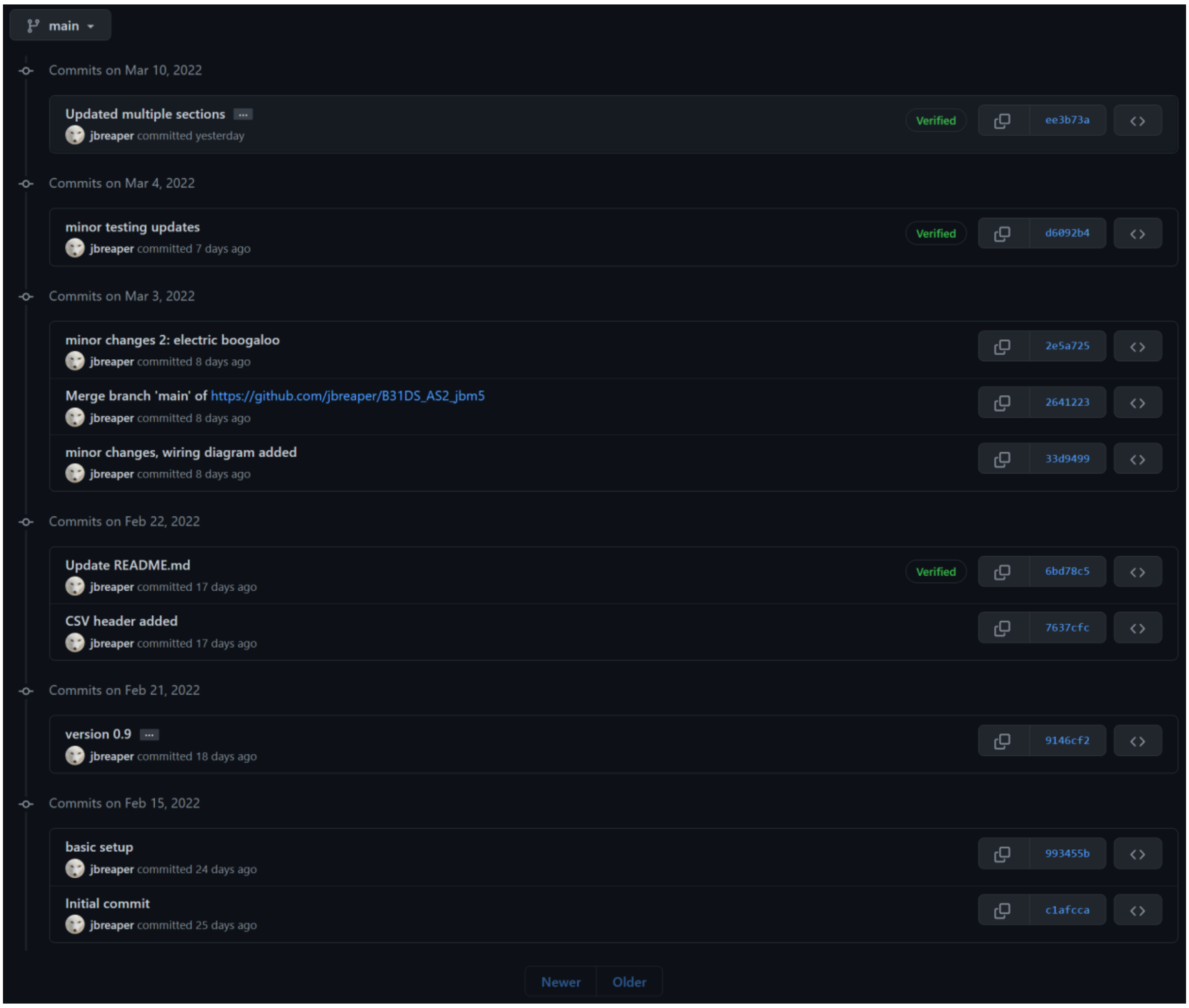
Development

In order to determine a frequency at which the ticker should activate at the lowest common multiple of the task frequencies was determined, this gave a frequency of 120Hz. Meaning that the timing for the ticker should be set to either 120Hz or a multiple of it, in order to minimise rounding errors while still providing enough time between ticks to complete required tasks the frequency of 240Hz. Meaning that the ticker should be triggered every 4.1666… seconds, however as the ticker function can only offer full millisecond timings this was rounded to 4 seconds.

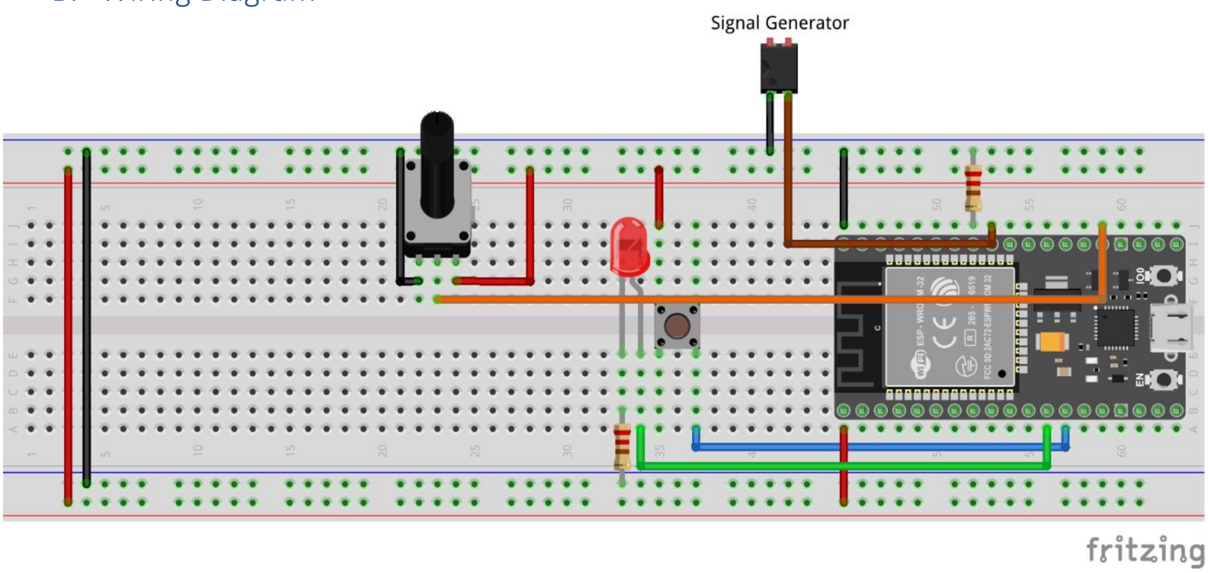
With the ticker frequency determined the function that the ticker calls can then be created. This was done by taking the rates of the absolute rate of the tasks and converting it to relative rates based upon the frequency of the ticker.

Appendix

A. Commit History



B. Wiring Diagram



C. Code

```
/* Embbeded Software - Assignment 2
 *  This project was programmed for for the ESP32 microcontroller, it uses a single "Ticker" object to create a cyclic
 *  executive based on the provided brief.
 *
 *  -------
 *   BRIEF
 *  -------
 *  For this assignment a cyclic executive was created.
 *  This executive must complete the following tasks at the related frequencies.
 * ______________________________________________________________________________________________________________________
 * |  Task  | Description                                                             | Frequency   | Periodicity |
 * ______________________________________________________________________________________________________________________
 * |  1     | Output a (digital) watchdog waveform (with same length and period of the ‘Normal’ | each cycle  |   1ms       |
 * |        | operation of SigB in Assignment 1). Timings should be within 5%.         |             |             |
 * ______________________________________________________________________________________________________________________
 * |  2     | Monitor one digital input (to be connected to a pushbutton/switch or a signal |     5Hz     |   200ms     |
 * |        | generator for students using Proteus).                                   |             |             |
 * ______________________________________________________________________________________________________________________
 * |  3     | Measure the frequency of a 3.3v square wave signal. The frequency will be in the |     1Hz     |   1000ms    |
 * |        | range 500Hz to 1000Hz and the signal will be a standard square wave (50% duty |             |             |
 * |        | cycle). Accuracy to 2.5% is acceptable.                                  |             |             |
 * ______________________________________________________________________________________________________________________
 * |  4     | Read one analogue input. The analogue input must be connected to a maximum of |     24Hz    |   42ms      |
 * |        | 3.3V, using a potentiometer.                                             |             |             |
 * ______________________________________________________________________________________________________________________
 * |  5     | Compute filtered analogue value, by averaging the last 4 readings.       |     24Hz    |   42ms      |
 * |        |                                                                          |             |             |
 * ______________________________________________________________________________________________________________________
 * |  6     | Execute 1000 times the following instruction:                            |     10Hz    |   100ms     |
 * |        | __asm__ __volatile__ ("nop");                                            |             |             |
 * |        | 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).  |             |             |
 * ______________________________________________________________________________________________________________________
 * |  7     | Perform the following check:if (average_analogue_in > half of maximum range for |     3Hz     |   333ms     |
 * |        | analogue input):                                                         |             |             |
 * |        |     error_code = 1                                                       |             |             |
 * |        | else:                                                                    |             |             |
 * |        |     error_code = 0                                                       |             |             |
 * ______________________________________________________________________________________________________________________
 * |  8     | Visualise error_code using an LED.                                       |     3Hz     |   333ms     |
 * |        |                                                                          |             |             |
 * ______________________________________________________________________________________________________________________
 * |  9     | Log the following information every five (5) seconds (in comma separated |     0.3Hz   |   5000ms    |
 * |        | format, e.g. CSV) to the serial port:                                    |             |             |
 * |        |   - State of the digital input (pushbutton / switch);                    |             |             |
 * |        |   - Frequency value (Hz, as an integer);                                 |             |             |
 * |        |   - Filtered analogue input.                                             |             |             |
 * ______________________________________________________________________________________________________________________
 */
#include <Ticker.h>

// pin assignments
#define LED         14    // Pin G14, Used to output error signal
#define WD          19    // Pin G19, Used to output the watchdog signal
#define PB1         12    // Pin G12, Used to read the state of a push button
#define A_IN        2     // Pin G0,  Used to read an analogue input signal
#define PULSE_IN    18    // Pin G18, Used to read a digital input signal

// using lowest common multiple of the frequencies, a timing to allow all
// tasks to be triggered was calculated, 8.33333333...ms
// to decrease the loss through rounding, this was then halved to 4.166...
#define T_CLK       4

// Rate of task (ticks)
#define R_T2        50        // 200  /4
#define R_T3        250       // 1000 /4
// while hz to ms gives 41.666666... for multiple
// reasons this has been rounded to 42
#define R_T4        10        // 41.666../4
#define R_T5        10        // 41.666../4
#define R_T6        25        // 100  /4
// while hz to ms gives 333.33333... for multiple
// reasons this has been rounded to 333
#define R_T7        83        // 333.33.../4
#define R_T8        83        // 333.33.../4
#define R_T9        1250      // 5000/4
```

```
Ticker ticker;

volatile int tick;

bool button_1 = false;

float frequency_in = 0;

float analogue_in;
float average_analogue_in = 0;
float analogues[4];

int error_code;

void setup() {
  Serial.begin(57600);
  tick = 0;
  analogues[0] = 0;
  analogues[1] = 0;
  analogues[2] = 0;
  analogues[3] = 0;
  pinMode(LED, OUTPUT);
  pinMode(WD, OUTPUT);
  pinMode(PB1, INPUT);
  pinMode(A_IN, INPUT);
  pinMode(PULSE_IN, INPUT);
  ticker.attach_ms(T_CLK, tick_up);
  Serial.print("\nSwitch, \tFrequency, \tInput \n");
}

// Activates at rate determined by the Ticker
// upon activation, determine which functions are to run
// on this tick.
void tick_up() {
  tick ++;

  task_1();
  if((tick%R_T2) == 0)          task_2();
  if((tick%R_T3) == 0)          task_3();
  if((tick%R_T4) == 0)          task_4();
  if((tick%R_T5) == 5)          task_5();
  if((tick%R_T6) == 0)          task_6();
  if((tick%R_T7) == 0)          task_7();
  if((tick%R_T8) == 40)         task_8();
  if((tick%R_T9) == 0)          task_9();
}

// generate pulse of with 50us
void task_1() {
  digitalWrite(WD, HIGH);
  delayMicroseconds(50);
  digitalWrite(WD, LOW);
}

// read input of a button on pin PB1
void task_2() {
  button_1 = digitalRead(PB1);
}

// determine frequency of digital signal on pin PULSE_IN
void task_3() {
  float high;
  high = pulseIn(PULSE_IN, LOW);
  frequency_in = 1000000.0 / (high * 2);
}

// read analogue input on pin A_IN
void task_4() {
  for (int i = 1; i < 4; i++) {
    analogues[i - 1] = analogues[i];
  }

  analogues[3] = 4095 - analogRead(A_IN);
}

// Average last 4 analog input readings
void task_5() {
```

```
  average_analogue_in = 0;

  for (int i = 0; i < 4; i++) {
    average_analogue_in += analogues[i];
  }

  average_analogue_in = average_analogue_in / 4;
}

// use "__asm__ __volatile__ ("nop");" 1000 times
void task_6() {
  for (int i = 0; i < 1000; i++) {
    __asm__ __volatile__ ("nop");
  }
}

// determine error code based on average analogue reading
void task_7() {
  if (average_analogue_in > (4095/ 2)) {
    error_code = 1;
  } else {
    error_code = 0;
  }
}

// light LED based on error code
void task_8() {
  digitalWrite(LED, error_code);
}

// print; button PB1 state, Frequency of PULSE_IN, and average of alalogue input A_IN
// This data is presented in a CSV format
void task_9() {
  Serial.print(button_1);
  Serial.print(", \t");
  Serial.print(frequency_in);
  Serial.print(", \t");
  Serial.print(((average_analogue_in*3.3)/4095));
  Serial.print("\n");
}

// not used as the ticker triggers all functions
void loop() {}
```