**ASSIGNMENT 3**

#ifndef \_\_FREERTOS\_H\_\_

#define \_\_FREERTOS\_H\_\_

 // Pin numbers

#define T1\_PIN        10  // Task 1 output signal pin

#define T2\_PIN        2   // Task 2 input measure signal pin

#define T3\_PIN        3   // Task 3 input measure signal pin

#define T4\_ANIN\_PIN   5   // Task 4 analogue input pin

#define T4\_LED\_PIN    0   // Task 4 LED output pin

#define T6\_PIN        4   // Task 6 push button input pin

#define T7\_PIN        1   // Task 7 LED output pin

 // Task periods (ms)

#define TASK1\_P       4

#define TASK2\_P       20

#define TASK3\_P       8

#define TASK4\_P       20

#define TASK5\_P       100

#define TASK6\_P       10

#define TASK7\_P       8

// Task parameters

#define BAUD\_RATE       9600  // Baud rate

#define TASK2\_TIMEOUT   3100  // Timeout for pulseIn = worst case in us

#define TASK2\_MINFREQ   333   // Task 2 minimum frequency of waveform in Hz

#define TASK2\_MAXFREQ   1000  // Task 2 maximum frequency of waveform in Hz

#define TASK3\_TIMEOUT   2100  // Timeout for pulseIn = worst case in us

#define TASK3\_MINFREQ   500   // Task 3 minimum frequency of waveform in Hz

#define TASK3\_MAXFREQ   1000  // Task 3 maximum frequency of waveform in Hz

#define NUM\_PARAMS      4     // Task 4 length of array for storing past measurements

#define TASK4\_THRESH    2048  // Task 4 threshold for turning LED on

#define TASK5\_MIN       0     // Task 5 lower bound of range

#define TASK5\_MAX       99    // Task 5 upper bound of range

// Helpful typedefs

typedef unsigned char uint8;

typedef unsigned int uint16;

typedef unsigned long uint32;

// Data structure for holding task 2 & 3 frequencies

typedef struct Freqs {

    double freq\_t2;

    double freq\_t3;

} Freqs;

 // Funtion Prototypes

void task1(void \*pvParameters);

void task2(void \*pvParameters);

void task3(void \*pvParameters);

void task4(void \*pvParameters);

void task5(void \*pvParameters);

void task6(void \*pvParameters);

void task7(void \*pvParameters);

 #endif  // \_\_FREERTOS\_H\_\_

// Convert period in us to frequency in Hz

#define periodToFreq\_us(T) (1 / (T / 1000000))

 // Convert freeRTOS ticks to real time in ms

#define waitTask(t) (vTaskDelay(t / portTICK\_PERIOD\_MS))

 #if CONFIG\_FREERTOS\_UNICORE

#define ARDUINO\_RUNNING\_CORE 0

#else

#define ARDUINO\_RUNNING\_CORE 1

#endif

 // Data structure & semaphore for storing task 2 & 3 frequencies

Freqs freqs;

SemaphoreHandle\_t freqSem;

 QueueHandle\_t btnQueue;

 uint16 anIn[NUM\_PARAMS];  // Array for storing previous analogue measurements

uint8 currInd;            // Current index to overwrite in anIn[]

 void setup() {

  Serial.begin(BAUD\_RATE);

   // Define pin inputs/outputs

  pinMode(T1\_PIN, OUTPUT);

  pinMode(T2\_PIN, INPUT);

  pinMode(T3\_PIN, INPUT);

  pinMode(T4\_ANIN\_PIN, INPUT);

  pinMode(T4\_LED\_PIN, OUTPUT);

  pinMode(T6\_PIN, INPUT);

  pinMode(T7\_PIN, OUTPUT);

   // Initialise task 4 analogue input array with 0's

  currInd = 0;

  for (uint8 i = 0; i < NUM\_PARAMS; i++) {

    anIn[i] = 0;

  }

   // Initialise task 2 & 3 frequencies to 0

  freqs.freq\_t2 = 0;

  freqs.freq\_t3 = 0;

   freqSem = xSemaphoreCreateMutex();

  btnQueue = xQueueCreate(1, sizeof(uint8));

   xTaskCreate(task1,"task1", 512,(void\*) 1, 3, NULL);

   xTaskCreate(task2,"task2",512,(void\*) 1, 2, NULL);

  xTaskCreate(task3,"task3",512,(void\*) 1,3, NULL);

   xTaskCreate(task4,"task4",512,(void\*) 1, 2, NULL);

   xTaskCreate(task5,"task5",512,(void\*) 1,1, NULL);

   xTaskCreate(task6, "task6",512,(void\*) 1,1, NULL);

   xTaskCreate(task7,"task7",512,(void\*) 1,1, NULL);

}

 // Period = 4ms / Rate = 250Hz

void task1(void \*pvParameters) {

  (void) pvParameters;

   for (;;) {

    // Generate waveform

    digitalWrite(T1\_PIN, HIGH);

    delayMicroseconds(200);

    digitalWrite(T1\_PIN, LOW);

    delayMicroseconds(50);

    digitalWrite(T1\_PIN, HIGH);

    delayMicroseconds(30);

    digitalWrite(T1\_PIN, LOW);

     waitTask(TASK1\_P);

  }

}

// Period = 20ms / Rate = 50Hz

void task2(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    // Measure period when signal is HIGH

    // Multiply by 2 because 50% duty cycle

    double period = (double) pulseIn(T2\_PIN, HIGH) \* 2;

    double freqT2 = periodToFreq\_us(period); // Convert to frequency using T = 1/f

     if(xSemaphoreTake(freqSem, portMAX\_DELAY) == pdTRUE) {

      freqs.freq\_t2 = freqT2;

      xSemaphoreGive(freqSem);

    }

    waitTask(TASK2\_P);

  }

}

// Period = 8ms / Rate = 125Hz

void task3(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    // Measure period when signal is HIGH

    // Multiply by 2 because 50% duty cycle

    double period = (double) pulseIn(T3\_PIN, HIGH) \* 2;

    double freqT3 = periodToFreq\_us(period); // Convert to frequency using T = 1/f

    if(xSemaphoreTake(freqSem, portMAX\_DELAY) == pdTRUE) {

      freqs.freq\_t3 = freqT3;

      xSemaphoreGive(freqSem);

    }

    waitTask(TASK3\_P);

  }

}

// Period = 20ms / Rate = 50Hz

void task4(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    // Read analogue signal and increment index for next reading

    // Analogue signal converted to 12 bit integer

    anIn[currInd] = analogRead(T4\_ANIN\_PIN);

    currInd = (currInd + 1) % NUM\_PARAMS;

    // Sum array and divide to get average

    double filtAnIn = 0;

    for (uint8 i = 0; i < NUM\_PARAMS; i++) {

      filtAnIn += anIn[i];

    }

    filtAnIn /= NUM\_PARAMS;

    // Turn on LED if average is above half maximium of 12 bit integer (4096 / 2 = THRESH)

    digitalWrite(T4\_LED\_PIN, (filtAnIn > TASK4\_THRESH));

    waitTask(TASK4\_P);

  }

}

// Period = 100ms / Rate = 10Hz

void task5(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    if(xSemaphoreTake(freqSem, portMAX\_DELAY) == pdTRUE) {

      // Map frequencies from 333Hz & 500Hz - 1000Hz to 0 - 99

      // Constrain to 0 - 99 as map() only creates gradient

      int normFreqT2 = map(freqs.freq\_t2, TASK2\_MINFREQ, TASK2\_MAXFREQ, TASK5\_MIN, TASK5\_MAX);

      normFreqT2 = constrain(normFreqT2, TASK5\_MIN, TASK5\_MAX);

      int normFreqT3 = map(freqs.freq\_t3, TASK3\_MINFREQ, TASK3\_MAXFREQ, TASK5\_MIN, TASK5\_MAX);

      normFreqT3 = constrain(normFreqT3, TASK5\_MIN, TASK5\_MAX);

      // Print to serial monitor

      Serial.print(normFreqT2);

      Serial.print(",");

      Serial.println(normFreqT3);

      xSemaphoreGive(freqSem);

    }

    waitTask(TASK5\_P);

  }

}

// Period = 10ms / Rate = 100Hz

void task6(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    uint8 btn = digitalRead(T6\_PIN);

    xQueueSend(btnQueue, &btn, 10);

    waitTask(TASK6\_P);

  }

}

// Period = 8ms / Rate = 125Hz

void task7(void \*pvParameters) {

  (void) pvParameters;

  for (;;) {

    uint8 btn = 0;

    if(xQueueReceive(btnQueue, &btn, 10) == pdPASS) {

      digitalWrite(T7\_PIN, btn);

    }

     waitTask(TASK7\_P);

  }

}

 void loop() {}