Heriot-Watt University Dubai Campus
[B31DG] Embedded Software 2023

REPORT

ASSIGNMENT NO. 2

INTRODUCTION

The code comprises of five distinct tasks, each serving a specific purpose:

Task 1: This task is responsible for blinking an LED at a specified frequency. It achieves this by utilizing the digital Write() function to set the LED pin either HIGH or LOW and the delay Microseconds() function to control the timing of the blinking.

Task 2: This task involves measuring the frequency of a signal on an input pin using an interrupt-driven approach. The pulse In() function is used to measure the pulse width of the signal. The frequency is then calculated based on the pulse width and the time interval between measurements.

Task 3: This task measures the frequency of a signal on an input pin using a polling approach. It employs the pulse In() function to measure the pulse width of the signal and calculates the frequency based on the pulse width and the time interval between measurements.

Task 4: This task is responsible for reading an analog voltage on an input pin and turning on an LED if the voltage is above a certain threshold. It makes use of the analog Read() function to read the voltage and the digital Write() function to set the LED pin either HIGH or LOW.

Task 5: This task is responsible for measuring the frequency of tasks 2 and 3 and logging the values to the serial monitor. It makes use of the millis() function to measure the time interval between measurements and calculates the frequency based on the time interval. Additionally, the code uses a monitoring library (B31DGMonitor.h) to measure the execution time of each task. The monitor.jobStarted() and monitor.jobEnded() functions are called at the beginning and end of each task, respectively. Regenerate response

1.ESP32

The ESP32 is a chip that seamlessly integrates both Wi-Fi and Bluetooth technologies, operating at a frequency of 2.4 GHz. Using TSMC's low power 40 nm technology, it has been designed to achieve optimal power and RF performance, making it highly reliable and versatile in a broad range of applications and power scenarios. With 34 digital pins that are similar to those found in Arduino, the ESP32 facilitates the addition of various components, such as sensors, LEDs, and more, to projects. The ESP32 WROOM module, in particular, boasts 25 GPIO pins, all of which are input pins. Some of these input pins come with input pull-up, while others do not have an internal pull-up.

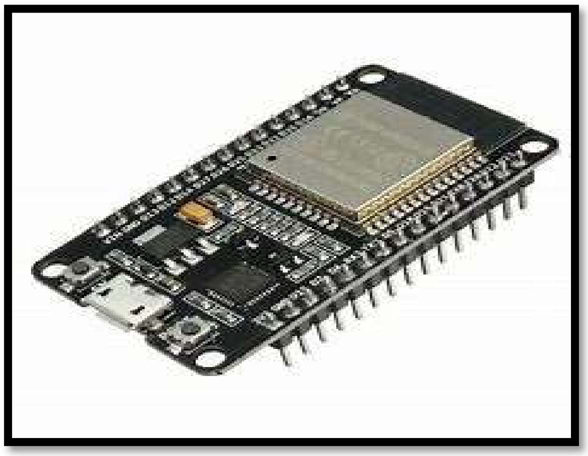


Fig 1.1: ESP32

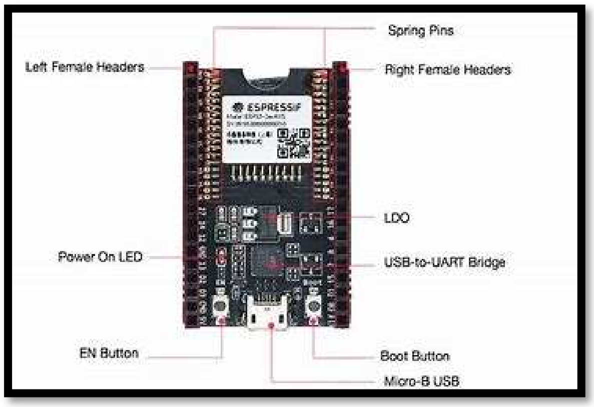


Fig 1.2: ESP32 DIAGRAM

2. Potentiometer

Fig 2.1: Potentiometer

Potentiometers are commonly used in electronic circuits for a variety of purposes, including volume and tone control in audio equipment, and as variable resistors in sensor circuits. The sliding contact, also known as the wiper, can be moved along the resistor element by turning a knob or shaft. This changes the resistance of the potentiometer, which in turn affects the voltage division ratio and the resulting output voltage. Potentiometers come in a range of sizes and values and can be linear or logarithmic in their response to changes in position. Linear potentiometers have a uniform change in resistance for each unit of displacement, while logarithmic (also known as "audio taper") potentiometers have a non-uniform response that matches the way human ears perceive changes in loudness.

3.Design of Cyclic Executive

A cyclic executive is a scheduling algorithm that is often used in real-time systems to execute a set of periodic tasks. The cyclic executive is a simple algorithm that does not require a real-time operating system. It is often used in embedded systems where the resources are limited. The cyclic executive schedules tasks based on their periods. The tasks are executed in a cyclic manner, where each task is executed once during its specified period. The cyclic executive assumes that the execution time of each task is known and constant. The tasks are treated as hard real-time tasks, meaning they must complete before their hard deadline, which is the end of their period. The cyclic executive works by dividing time into equal time slices, where each time slice is equal to the period of the task with the shortest period. During each time slice, the cyclic executive checks if any task is ready to be executed. If a task is ready to be executed, it is executed for its full duration. If a task is not ready to be executed, the cyclic executive waits for the next time slice. The cyclic executive ensures that each task is executed once during its specified period. However, it does not guarantee that the tasks will meet their hard deadlines. If the execution time of a task exceeds its period, the task will miss its deadline, which can lead to a failure in the system. Overall, the cyclic executive is a simple and efficient scheduling algorithm that is often used in embedded systems. However, it is not suitable for all real-time systems, particularly those with complex requirements or where meeting hard deadlines is critical.

CIRCUIT DIAGRAM

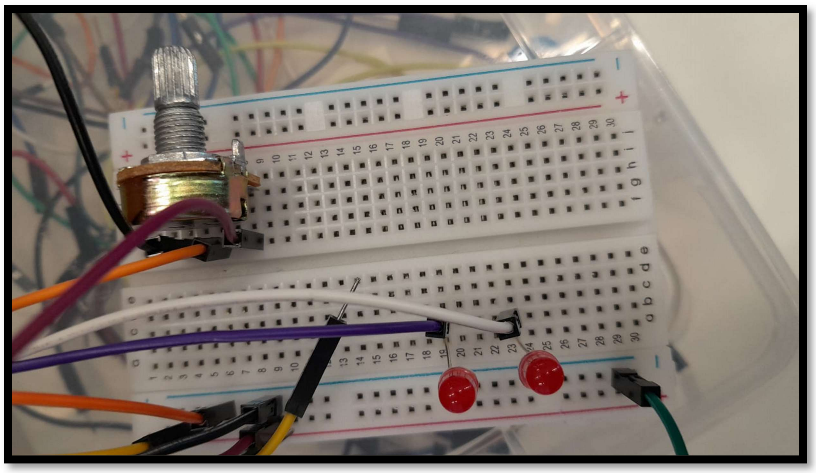


Fig 1.1: Simulation of circuit diagram

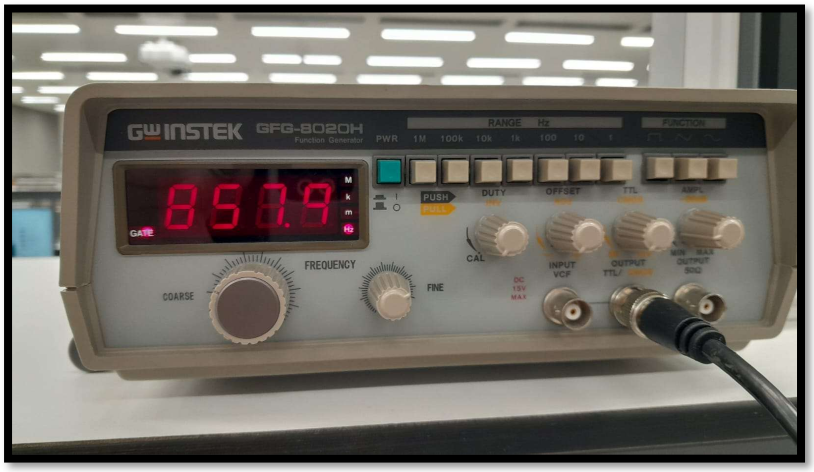


Fig 1.2: Function generator

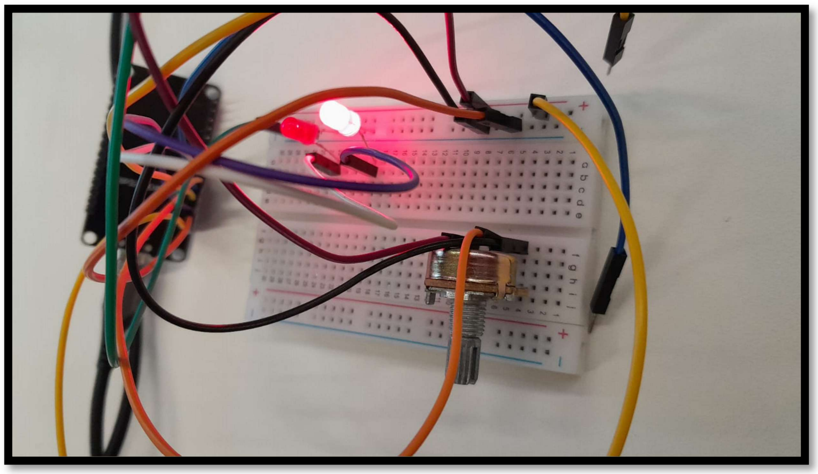
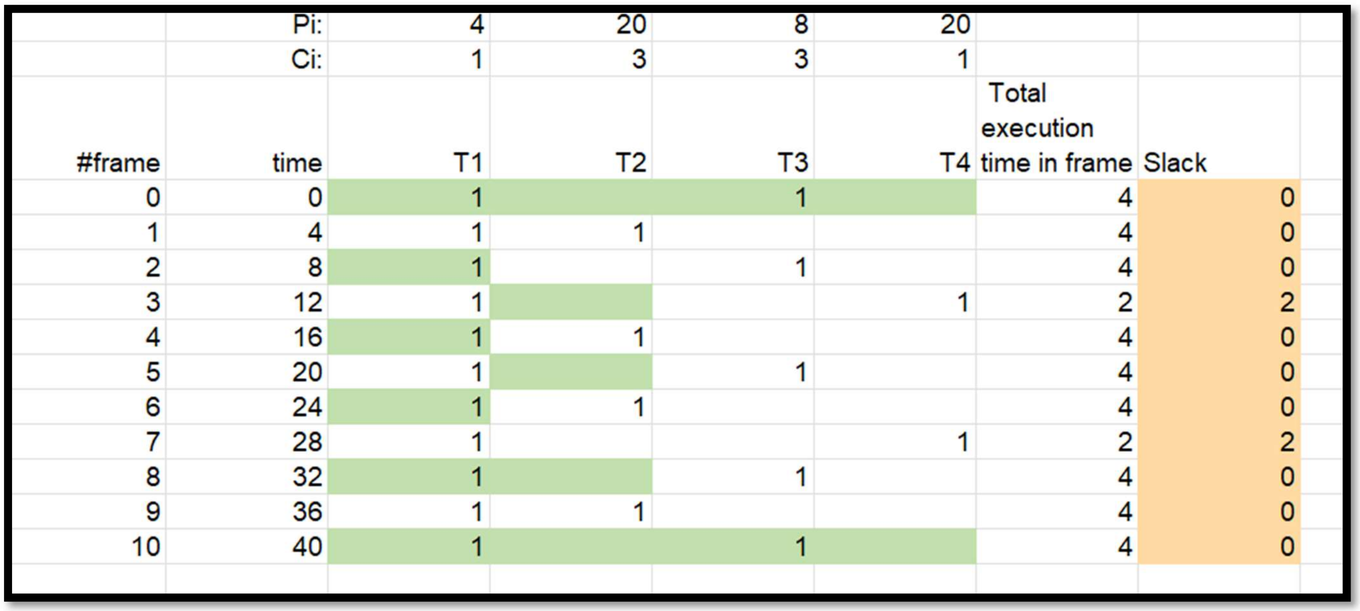
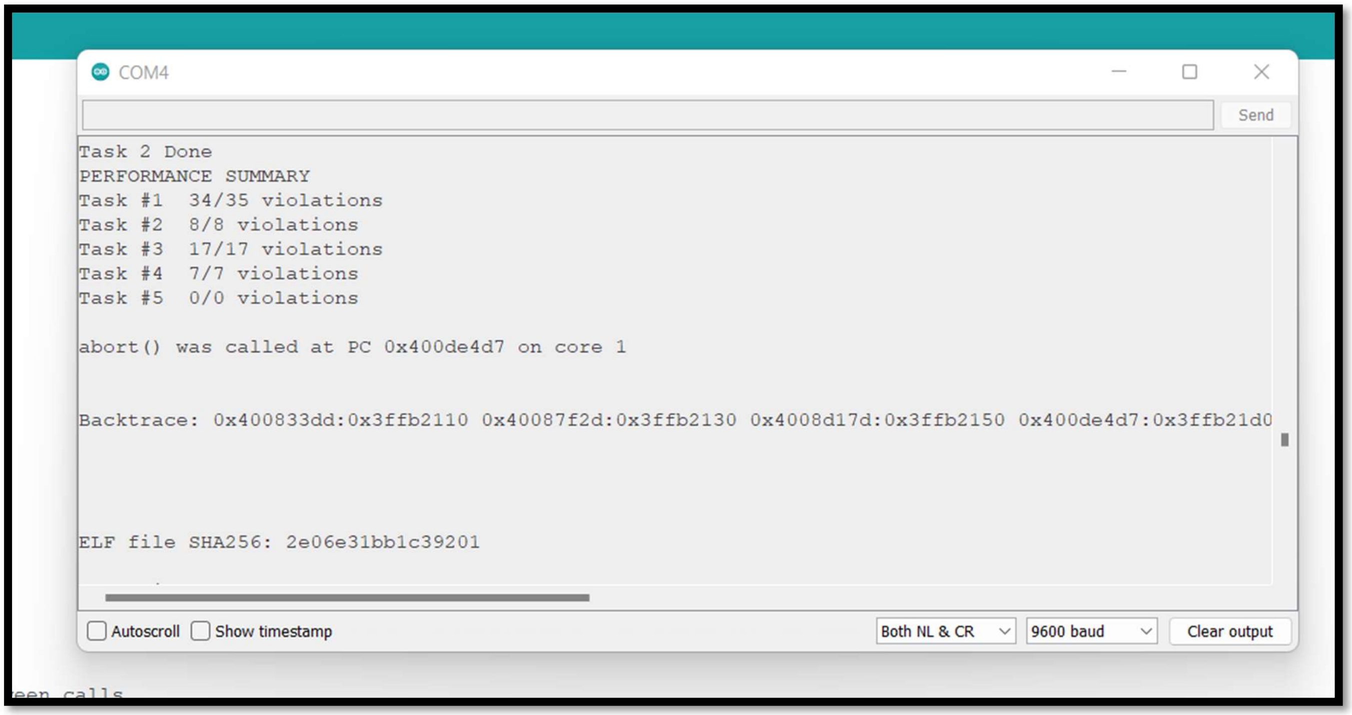


Fig1.2: Simulation of circuit diagram blink light



	Pi:	4	20	8	20		
	Ci:	1	3	3	1		
#frame	time	T1	T2	T3	T4	Total execution time in frame	Slack
0	0	1		1		4	0
1	4	1	1			4	0
2	8	1		1		4	0
3	12	1			1	2	2
4	16	1	1			4	0
5	20	1		1		4	0
6	24	1	1			4	0
7	28	1			1	2	2
8	32	1		1		4	0
9	36	1	1			4	0
10	40	1		1		4	0

Fig: Excel sheet



COM4

Send

```
Task 2 Done
PERFORMANCE SUMMARY
Task #1  34/35 violations
Task #2  8/8 violations
Task #3  17/17 violations
Task #4  7/7 violations
Task #5  0/0 violations

abort() was called at PC 0x400de4d7 on core 1


Backtrace: 0x400833dd:0x3ffb2110 0x40087f2d:0x3ffb2130 0x4008d17d:0x3ffb2150 0x400de4d7:0x3ffb21d0




ELF file SHA256: 2e06e31bb1c39201
```

Fig: Violations [COM4]

Code description:

This is an Arduino program that implements a cyclic executive for a real-time system.

The program consists of four tasks: Task1, Task2, Task3, and Task4, each of which has a specific job to do. The cyclic executive is implemented by dividing time into fixed time slots, and each task is assigned a specific slot in each frame.

Task1 is the highest priority task, with a time slot of 0.9 ms. It blinks an LED attached to pin 19 in a specific pattern, with a period of 4 ms. Task2 and Task3 both have a time slot of 4 ms and are assigned slots in alternate frames. Task2 measures the frequency of a signal generator attached to pin 12, while Task3 measures the frequency of another signal generator attached to pin 14. Task4 has a time slot of 4 ms and reads the value of a potentiometer attached to pin 27, and if the value is outside a specific range, it blinks an LED attached to pin 26.

The main function of the program is the FRAME function, which increments a slot counter and runs the appropriate task based on the current slot. The loop function calls the FRAME function repeatedly, ensuring that the cyclic executive runs continuously.

The program uses the B31DGMonitor library to monitor the execution of the tasks and provide feedback on their performance. The library provides a startMonitoring function to initialize the monitoring system and a jobStarted function to indicate the start of a task.

Similarly, the library provides a job-end function to indicate the end of a task. Overall, this program demonstrates how a cyclic executive can be used to implement a real-time system with multiple tasks of different priorities and execution times. It also highlights the importance of monitoring the system's performance to ensure that the tasks are meeting their deadlines and the system is operating correctly.

<u>Code:</u>

```
#include <B31DGMonitor.h>

B31DGCyclicExecutiveMonitor monitor;

// Author:GUNJAN DOD


#define Duration of FRAME  4     // 4ms

int L1_LED=19; //output port for LED of task 1

int T2_Freq=12;//input port from signal generator to count task-2Freq_1

int T3_Freq=14;//input port from signal geneerator to count task-3Freq_1

int T4_POT=27;//input port from potentiometer to show analogFreq_1

int LED_error=26;//output port to blink the led for error from potentiometer


unsigned long Timer_FRAME = 0;//Initializing Timer_FRAME

unsigned long Timer_Counter = 0;//Initializing Timer_Counter



void setup(void)

{

  monitor.startMonitoring();

  Serial.begin(9600);

  while(!Serial);

  Serial.println("Ready");

  pinMode(L1_LED, OUTPUT); // set pin 2 as output for Task_1

  pinMode(T2_Freq, INPUT); // set pin 2 as input for Task_2

  pinMode(T3_Freq, INPUT); // set pin 2 as input for Task_3

  pinMode(T4_POT, INPUT); // set pin 2 as input for Task_4
  pinMode(LED_error, OUTPUT); //Led pin output for Task_4
  // Initialize readings array with 0's



}




  // Increase FRAME counter and reset it after 10 FRAMEs



void FRAME() {


    static unsigned int slot = 0; // use static variable to retain value between calls
    slot = (slot + 1) % 10; // increment and wrap slot index

   switch (slot) {
     case 0: JOB_TASK_1();JOB_TASK_3();break;
     case 1: JOB_TASK_1(); JobTask2();break;
     case 2: JOB_TASK_1();JOB_TASK_3();break;
     case 3: JOB_TASK_1();JobTask4();break;
     case 4: JOB_TASK_1();JOB_TASK_3();break;
     case 5: JOB_TASK_1();JobTask2();break;
     case 6: JOB_TASK_1();JOB_TASK_3();break;
     case 7: JOB_TASK_1();JobTask4();break;
     case 8: JOB_TASK_1();JOB_TASK_3();break; //JobTask5();break;
     case 9: JOB_TASK_1();
}
}
```

```
void loop(void) // Single time slot function of the Cyclic Executive (repeating)
{
  /*
  unsigned long bT = micros();
    JobTask4();

  unsigned long timeItTook = micros()-bT;
  Serial.print("Duration SerialOutput Job = ");
  Serial.print(timeItTook);
  exit(0);
  */
FRAME();// TO-DO: wait the next FRAME

}

// Task 1, takes 0.9ms
void JOB_TASK_1(void)
{
  monitor.jobStarted(1);
  Serial.println("Task 1 Started");
  digitalWrite(L1_LED, HIGH); // set pin 2 high for 200us
  delayMicroseconds(200);
  digitalWrite(L1_LED, LOW); // set pin 2 low for 50us
  delayMicroseconds(50);
  digitalWrite(L1_LED, HIGH); // set pin 2 high for 30us
  delayMicroseconds(30);
  digitalWrite(L1_LED, LOW); // set pin 2 low for remaining period
  //delayMicroseconds(1720); // wait for 4ms minus the time spent in the loop
  monitor.jobEnded(1);
  Serial.println("Task 1 Done");
}

// Task 2, takes 4ms
void JobTask2(void)
{
   monitor.jobStarted(2);
   Serial.println("Task 2 Started");
 //  #define SAMPLES 10 // number of samples to take
int count = 0;
 // for (int i = 0; i < SAMPLES; i++)
  //{
    count += pulseIn(T2_Freq, HIGH); // count the pulse width of the input signal which is high
 // }
  count = count*2;//Pulse width*2 to calculate positive and negetive pulse as whole waveform
  float Freq_1 = 1000000.0 / (count ); //SAMPLES); // calculateFreq_1 in Hz
 Freq_1 = constrain(Freq_1, 333, 1000); // boundFreq_1 between 333 and 1000 Hz
  int scaled_Freq_1 = map(Freq_1, 333, 1000, 0, 99); // scaleFreq_1 between 0 and 99 for
  Serial.println("Freq_1:"); // outputFreq_1 value to serial port
  Serial.println(Freq_1); // outputFreq_1 value to serial port
  //delayMicroseconds(800); // wait for 20ms minus the time spent in the loop
  Serial.println("Task 2 Done");
  monitor.jobEnded(2);
}

// Task 3, takes 4ms
void JOB_TASK_3(void)
{
  monitor.jobStarted(3);
  Serial.println("Task 3 Started");
  //#define SAMPLES 8 // number of samples to take for pulse
  int count2 = 0;
  //for (int i = 0; i < SAMPLES; i++) {
    //count2 += pulseIn(T3_Freq, HIGH); // count the pulse width of the input signal that is high
  //}
 count2 = count2*2; //Pulse width*2 to calculate positive and negetive pulse as whole waveform
  float Freq_12 = 1000000.0 / (count2); // SAMPLES); // calculateFreq_1 in Hz
 Freq_12 = constrain(Freq_12, 500, 1000); // boundFreq_1 between 500 and 1000 Hz
  int scaled_Freq_12 = map(Freq_12, 500, 1000, 0, 99); // scaleFreq_1 between 0 and 99
  Serial.println("Freq_1_2:"); // outputFreq_1 value to serial port
  Serial.println(Freq_12); // outputFreq_1 value to serial port
  //delayMicroseconds(920); // wait for 8ms minus the time spent in the loop
  monitor.jobEnded(3);
  Serial.println("Task 3 Done");
}



// Task 4, takes 4ms
void JobTask4(void)
{
  monitor.jobStarted(4);
  Serial.println("Task 4 Started");
  const int maxAnalogIn = 1023;
  const int numReadings = 4;
  int readings[numReadings];
  int index = 0;
  int total = 0;
  int filteredValue = 0;
```

```
for (int i = 0; i < numReadings; i++)
{
readings[i] = 0;
}

// Read  the analog input value
int analogValue = analogRead(T4_POT);
// Subtract the oldest reading from the total
total -= readings[index];
// Add the new reading to the total
total += analogValue;
// Store the new reading in the readings array
readings[index] = analogValue;
// Increment the index
index++;
// Wrap the index if it exceeds the number of readings
if (index >= numReadings)
 {
  index = 0;
}
// Compute the filtered value as the average of the readings
filteredValue = total / numReadings;
// If the filtered value is greater than half of the maximum range, turn on the LED
if (filteredValue > maxAnalogIn / 2) {
  digitalWrite(LED_error, HIGH);
  Serial.println("error led HIGH");

} else {
  digitalWrite(LED_error, LOW);
  Serial.println("error led LOW");
 }
 // Send the filtered value to the serial port
 Serial.println(filteredValue);
 // Delay for 20ms
 //delay(20);
 monitor.jobEnded(4);
 Serial.println("Task 4 Done");//End of Task 4
}

 // Task 5, takes 4ms
void JobTask5(void)
{
   //Task 5
 Serial.println("Task 5 started");//Start of Task 5
 int task2Freq = 0;
 int task3Freq = 0;
   // count theFreq_1 of Task 2 signal
 task2Freq = pulseIn(T2_Freq, HIGH, 20000) == 0 ? 0 : 1000000 / pulseIn(T2_Freq, HIGH, 20000);
 // Scale and bound theFreq_1 between to 0-99
 task2Freq = map(T2_Freq, 333, 1000, 0, 99);
 // count theFreq_1 of Task 3 signal
 task3Freq = pulseIn(T3_Freq, HIGH, 8000) == 0 ? 0 : 1000000 / pulseIn(T2_Freq, HIGH, 8000);
 // Scale and bound theFreq_1 value between 0-99
 task3Freq = map(T3_Freq, 500, 1000, 0, 99);
 // Send theFreq_1 values to the serial port
 Serial.println(task2Freq);//To printFreq_1 of given waveform of Task2
 Serial.println(task3Freq);//To printFreq_1 of given waveform of Task3
 Serial.println("Task 5 Completed");// End of Task 5
}
```

GitHub Link :