**COMSATS University**

**Islamabad**



**Lab Report # 10**

**Real Time Embedded Systems**

**(EEE-446)**

|  |
| --- |
| **Implementation of RMS and EDF Scheduling algorithms using freeRTOS.** |

**Submitted By:**

**Arwa Aamir (FA16-EEE-002)**

**Submitted To:**

**Dr. Ahsen Malik**

# Lab # 10

## Implementation of RMS and EDF Scheduling algorithms using freeRTOS.

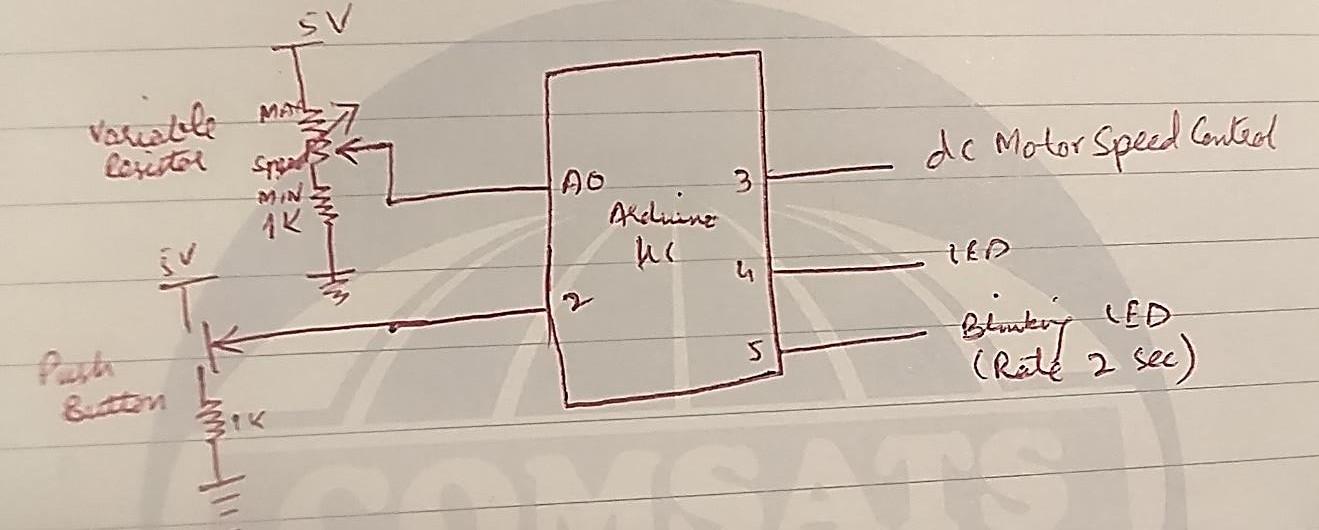
### Objectives

* Implementation of Rate Monotonic Scheduling (RMS) Algorithm
* Analysis of RMS
* Implementation of Earliest Deadline First (EDF)

### Tools

* Arduino
* Proteus ISIS
* freeRTOS library

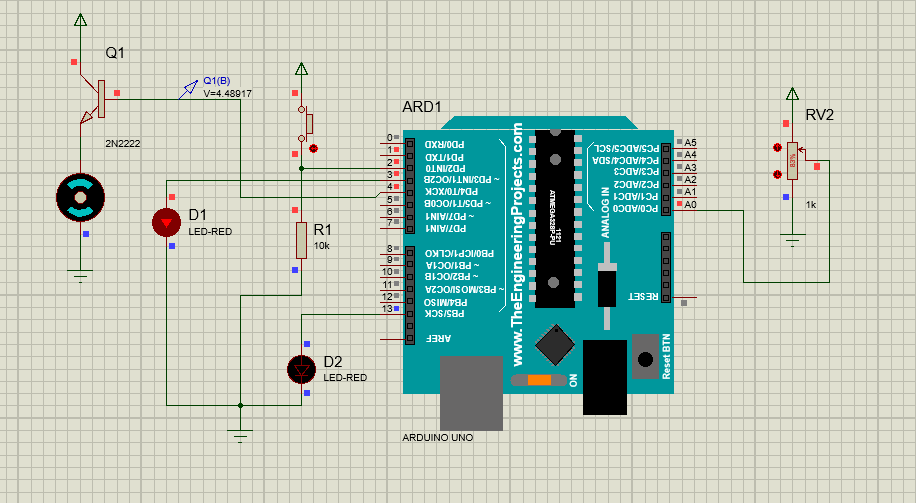
### In-Lab Task 1:

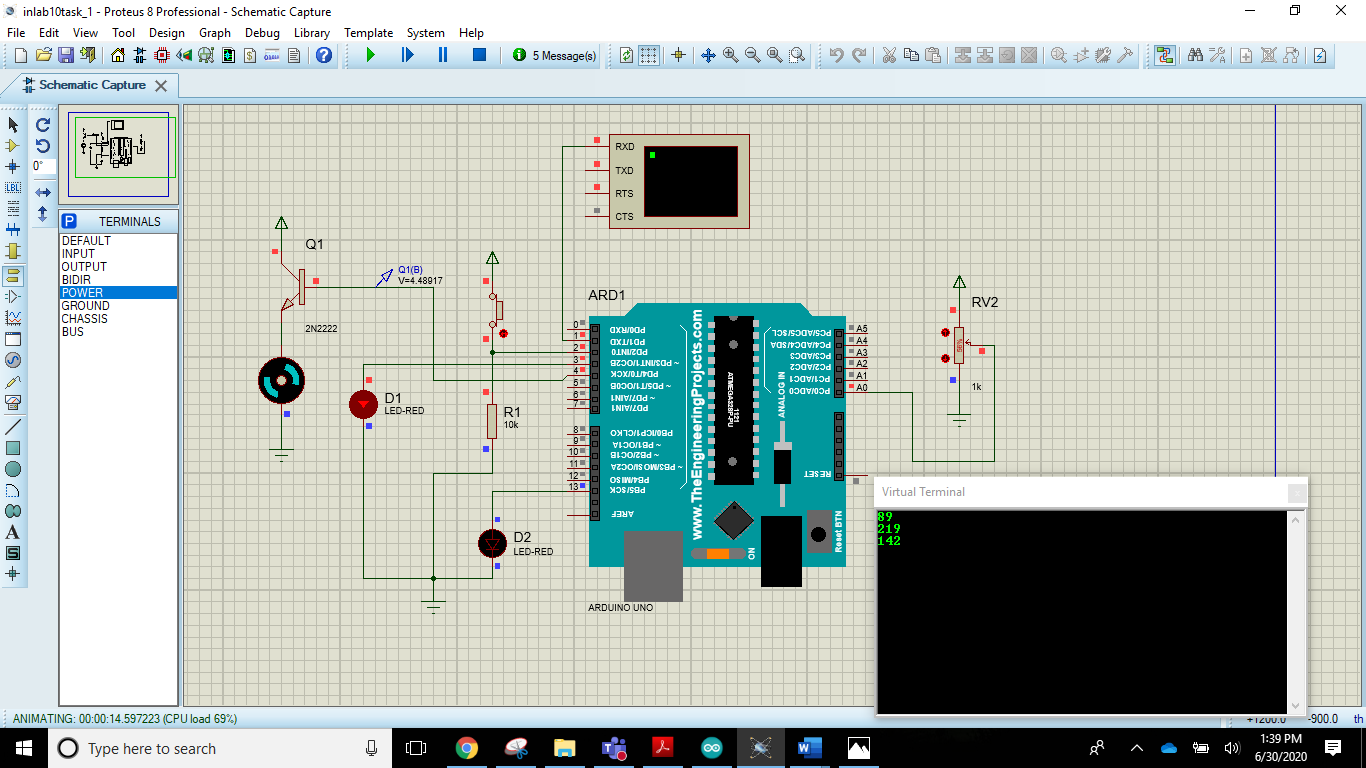


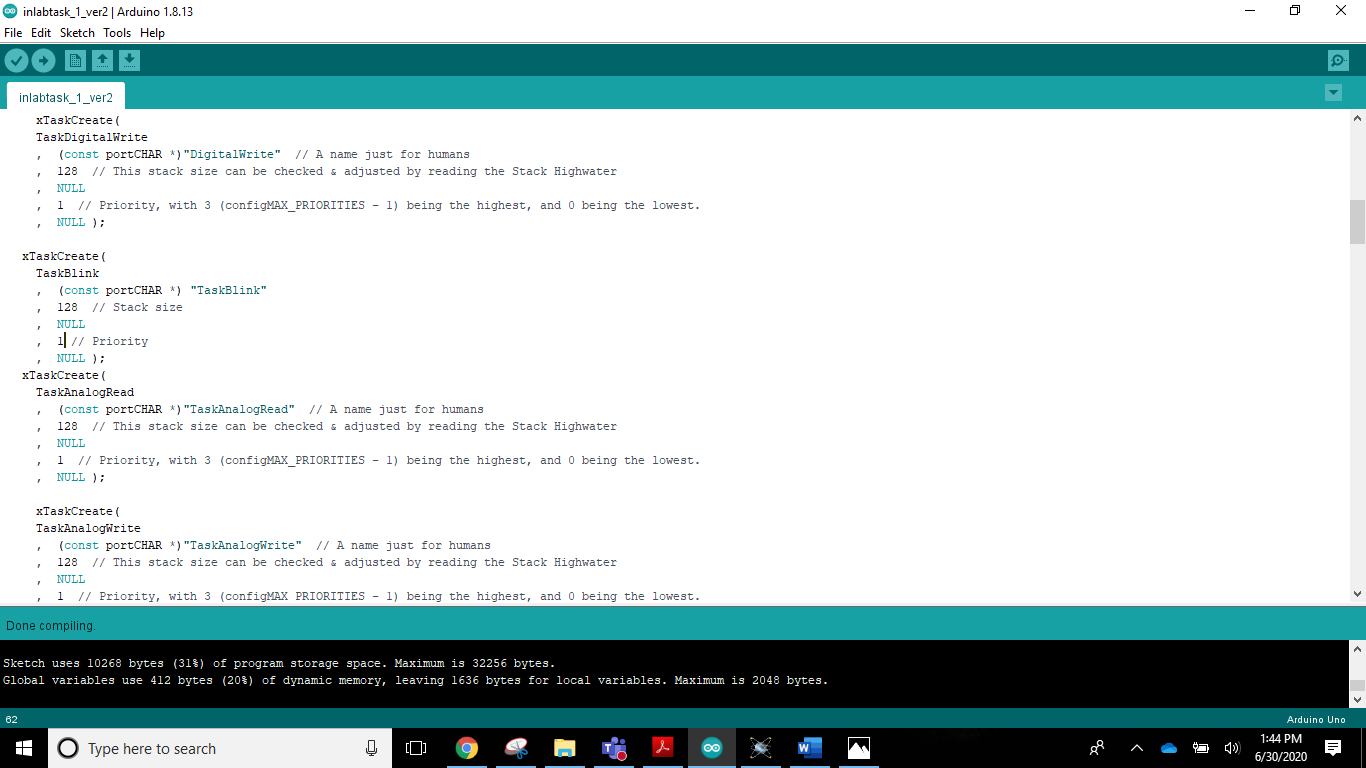
Please see and analyze the above task and complete it using RMS using freeRTOS library in Arduino with following specs controlled simultaneously by various tasks.

1. Variable resistor is controlling the speed of the DC Motor.
2. Push button is controlling the LED turn ON or OFF.
3. And one LED is consistently blinking after two seconds.

**Using 1 semaphore and Round Robin (Equal Priorities):**



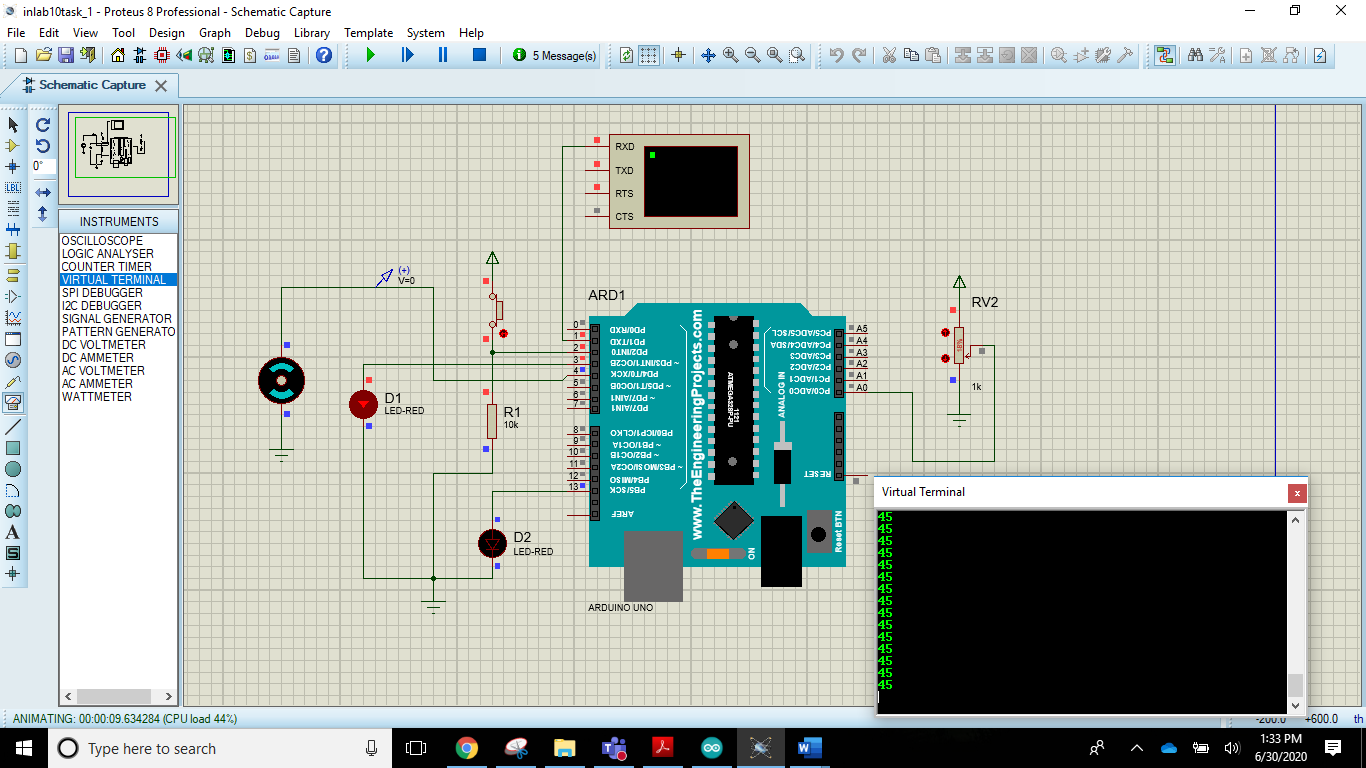


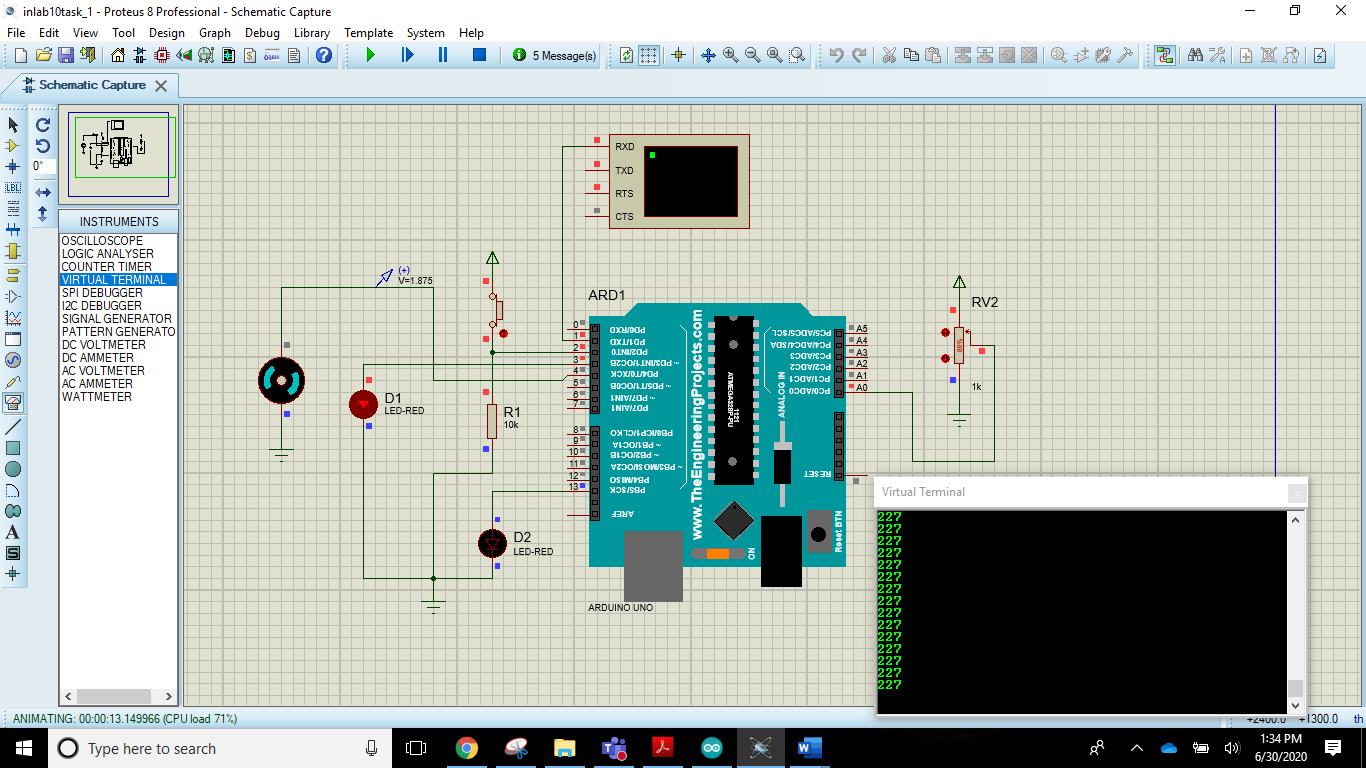


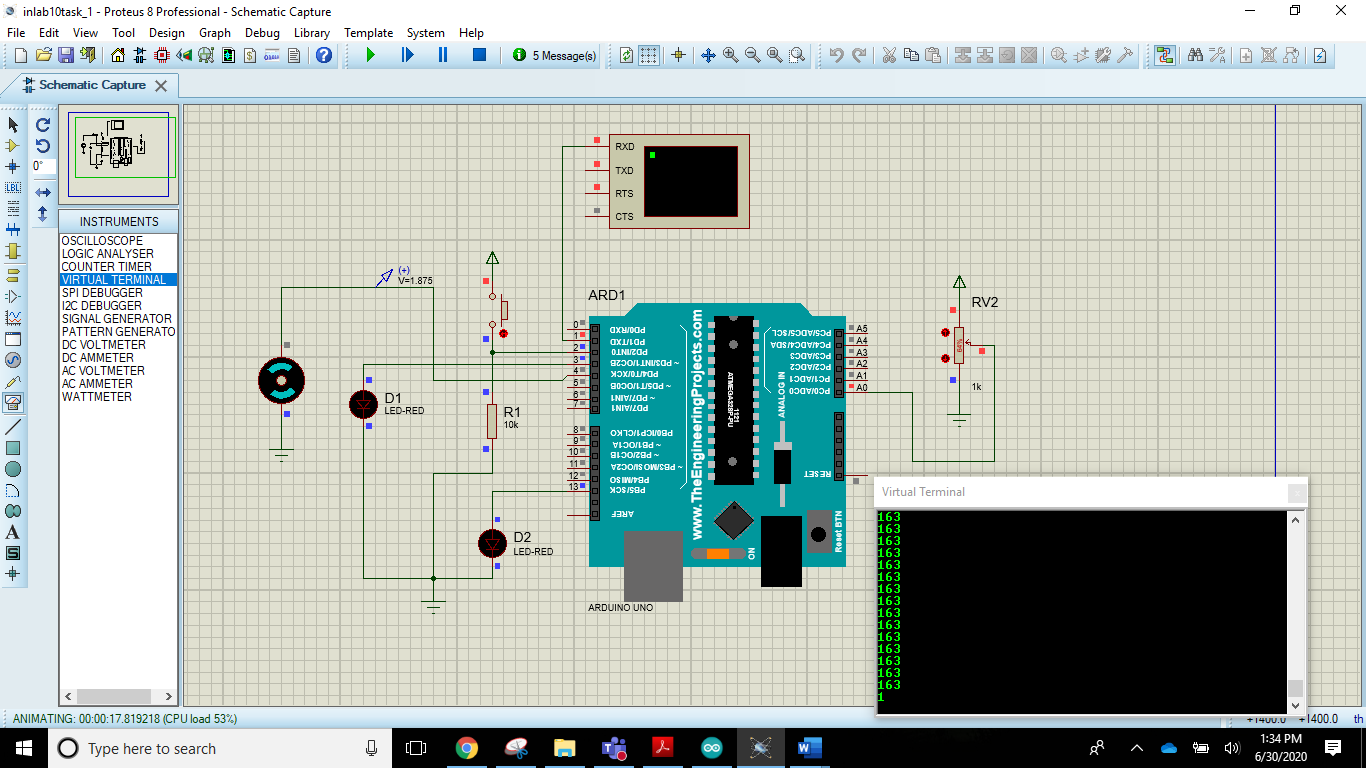
**Analysis:**

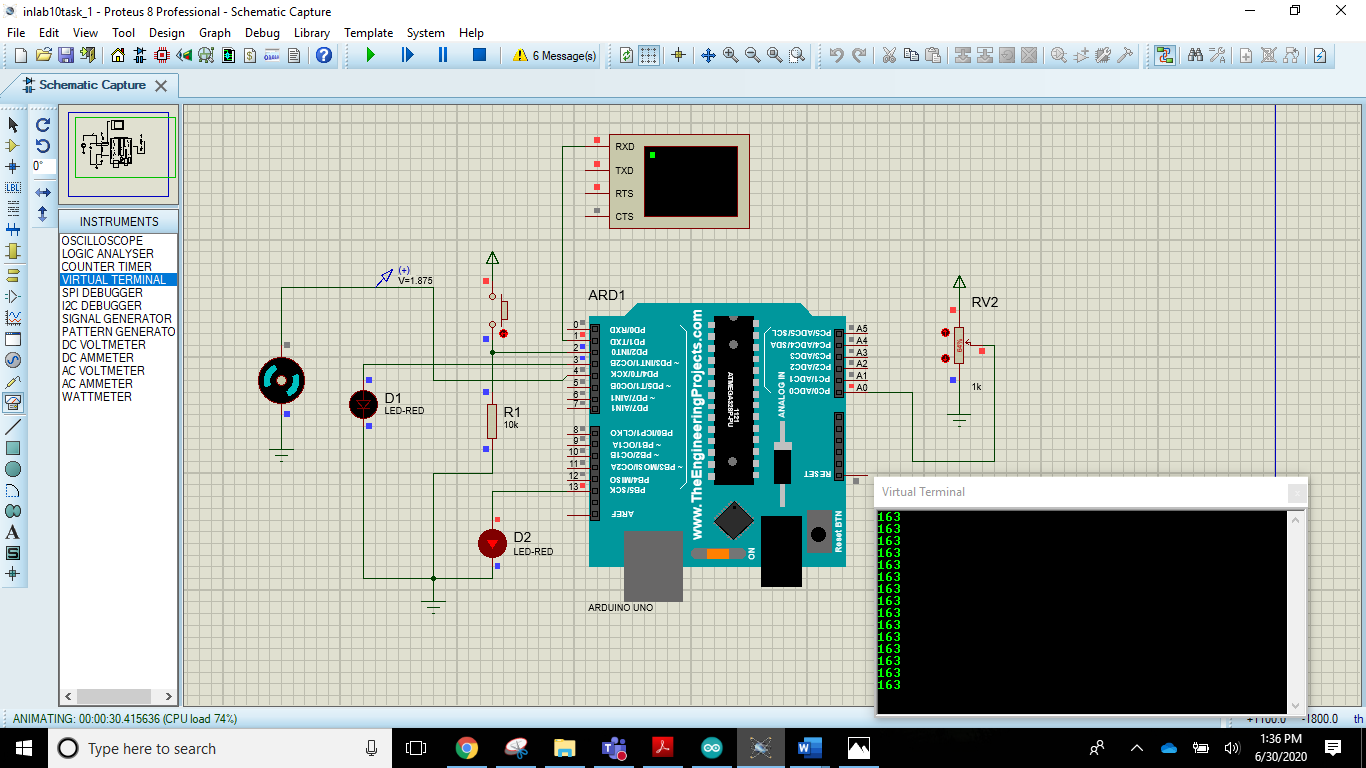
|  |
| --- |
| Speed of motor does changes but very slowly as we see at the virtual terminal we vary the pot value. All the other tasks are running but with a lot of delay.  The system Is overloaded and the scheduling is poor. |

**Using 1 semaphore and RMS Scheduling:**









Arduino IDE Code:

|  |
| --- |
| #include <Arduino\_FreeRTOS.h>  #include <semphr.h> // add the FreeRTOS functions for Semaphores (or Flags).  int buttonState;  unsigned int per;  // Declare a mutex Semaphore Handle which we will use to manage the Serial Port.  // It will be used to ensure only only one Task is accessing this resource at any time.  SemaphoreHandle\_t xSerialSemaphore;  // define two Tasks for DigitalRead & AnalogRead  void TaskDigitalRead( void \*pvParameters );  void TaskBlink( void \*pvParameters );  void TaskDigitalWrite( void \*pvParameters );  void TaskAnalogRead( void \*pvParameters );  void TaskAnalogWrite( void \*pvParameters );  // the setup function runs once when you press reset or power the board  void setup() {  // initialize serial communication at 9600 bits per second:  Serial.begin(9600);  pinMode(3,OUTPUT);  pinMode(A0,INPUT);  pinMode(4,OUTPUT);    while (!Serial) {  ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4 based boards.  }  // Semaphores are useful to stop a Task proceeding, where it should be paused to wait,  // because it is sharing a resource, such as the Serial port.  // Semaphores should only be used whilst the scheduler is running, but we can set it up here.  if ( xSerialSemaphore == NULL ) // Check to confirm that the Serial Semaphore has not already been created.  {  xSerialSemaphore = xSemaphoreCreateMutex(); // Create a mutex semaphore we will use to manage the Serial Port  if ( ( xSerialSemaphore ) != NULL )  xSemaphoreGive( ( xSerialSemaphore ) ); // Make the Serial Port available for use, by "Giving" the Semaphore.  }  // Now set up two Tasks to run independently.  xTaskCreate(  TaskDigitalRead  , (const portCHAR \*)"DigitalRead" // A name just for humans  , 128 // This stack size can be checked & adjusted by reading the Stack Highwater  , NULL  , 1 // Priority, with 3 (configMAX\_PRIORITIES - 1) being the highest, and 0 being the lowest.  , NULL );  xTaskCreate(  TaskDigitalWrite  , (const portCHAR \*)"DigitalRead" // A name just for humans  , 128 // This stack size can be checked & adjusted by reading the Stack Highwater  , NULL  , 1 // Priority, with 3 (configMAX\_PRIORITIES - 1) being the highest, and 0 being the lowest.  , NULL );    xTaskCreate(  TaskBlink  , (const portCHAR \*) "AnalogRead"  , 128 // Stack size  , NULL  , 1 // Priority  , NULL );  xTaskCreate(  TaskAnalogRead  , (const portCHAR \*)"DigitalRead" // A name just for humans  , 128 // This stack size can be checked & adjusted by reading the Stack Highwater  , NULL  , 1 // Priority, with 3 (configMAX\_PRIORITIES - 1) being the highest, and 0 being the lowest.  , NULL );  xTaskCreate(  TaskAnalogWrite  , (const portCHAR \*)"DigitalRead" // A name just for humans  , 128 // This stack size can be checked & adjusted by reading the Stack Highwater  , NULL  , 1 // Priority, with 3 (configMAX\_PRIORITIES - 1) being the highest, and 0 being the lowest.  , NULL );  // Now the Task scheduler, which takes over control of scheduling individual Tasks, is automatically started.  }  void loop()  {  // Empty. Things are done in Tasks.  }  /\*--------------------------------------------------\*/  /\*---------------------- Tasks ---------------------\*/  /\*--------------------------------------------------\*/  void TaskDigitalRead( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.  {  /\*  DigitalReadSerial  Reads a digital input on pin 2, prints the result to the serial monitor  This example code is in the public domain.  \*/  // digital pin 2 has a pushbutton attached to it. Give it a name:  uint8\_t pushButton = 2;  // make the pushbutton's pin an input:  pinMode(pushButton, INPUT);  for (;;) // A Task shall never return or exit.  {  // read the input pin:    // See if we can obtain or "Take" the Serial Semaphore.  // If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.  if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 5 ) == pdTRUE )  {  // We were able to obtain or "Take" the semaphore and can now access the shared resource.  // We want to have the Serial Port for us alone, as it takes some time to print,  // so we don't want it getting stolen during the middle of a conversion.  // print out the state of the button:  Serial.println(buttonState);  buttonState = digitalRead(pushButton);  xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.  }  vTaskDelay(1); // one tick delay (15ms) in between reads for stability  }  }  void TaskBlink(void \*pvParameters) // This is a task.  {  (void) pvParameters;  /\*  Blink  Turns on an LED on for one second, then off for one second, repeatedly.  Most Arduinos have an on-board LED you can control. On the UNO, LEONARDO, MEGA, and ZERO  it is attached to digital pin 13, on MKR1000 on pin 6. LED\_BUILTIN takes care  of use the correct LED pin whatever is the board used.    The MICRO does not have a LED\_BUILTIN available. For the MICRO board please substitute  the LED\_BUILTIN definition with either LED\_BUILTIN\_RX or LED\_BUILTIN\_TX.  e.g. pinMode(LED\_BUILTIN\_RX, OUTPUT); etc.    If you want to know what pin the on-board LED is connected to on your Arduino model, check  the Technical Specs of your board at https://www.arduino.cc/en/Main/Products    This example code is in the public domain.  modified 8 May 2014  by Scott Fitzgerald    modified 2 Sep 2016  by Arturo Guadalupi  \*/  // initialize digital LED\_BUILTIN on pin 13 as an output.  pinMode(LED\_BUILTIN, OUTPUT);  volatile int i=0;  for (;;) // A Task shall never return or exit.  {  digitalWrite(LED\_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  for (i=0; i<30000; i++);  digitalWrite(LED\_BUILTIN, LOW); // turn the LED off by making the voltage LOW  for (i=0; i<30000; i++);  }  }  void TaskDigitalWrite( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.  {  (void) pvParameters;  /\*  DigitalReadSerial  Reads a digital input on pin 2, prints the result to the serial monitor  This example code is in the public domain.  \*/  // digital pin 2 has a pushbutton attached to it. Give it a name:  pinMode(3,OUTPUT);  // make the pushbutton's pin an input:    for (;;) // A Task shall never return or exit.  {  // read the input pin:    // See if we can obtain or "Take" the Serial Semaphore.  // If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.  if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 5 ) == pdTRUE )  {  // We were able to obtain or "Take" the semaphore and can now access the shared resource.  // We want to have the Serial Port for us alone, as it takes some time to print,  // so we don't want it getting stolen during the middle of a conversion.  // print out the state of the button:    if(buttonState==HIGH)  {  digitalWrite(3,HIGH);  }  else  {  digitalWrite(3,LOW);  }  xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.  }        vTaskDelay(1); // one tick delay (15ms) in between reads for stability  }  }  void TaskAnalogRead( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.  {  for (;;)  {  // read the input on analog pin 0:    // See if we can obtain or "Take" the Serial Semaphore.  // If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.  if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 5 ) == pdTRUE )  {  // We were able to obtain or "Take" the semaphore and can now access the shared resource.  // We want to have the Serial Port for us alone, as it takes some time to print,  // so we don't want it getting stolen during the middle of a conversion.  // print out the value you read:    int sensorValue = analogRead(A0);  per=map(sensorValue,0,1023,0,255);  xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.  }  vTaskDelay(1); // one tick delay (15ms) in between reads for stability  }  }  void TaskAnalogWrite( void \*pvParameters \_\_attribute\_\_((unused)) ) // This is a Task.  {  for (;;)  {  // read the input on analog pin 0:    // See if we can obtain or "Take" the Serial Semaphore.  // If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free.  if ( xSemaphoreTake( xSerialSemaphore, ( TickType\_t ) 5 ) == pdTRUE )  {  // We were able to obtain or "Take" the semaphore and can now access the shared resource.  // We want to have the Serial Port for us alone, as it takes some time to print,  // so we don't want it getting stolen during the middle of a conversion.  // print out the value you read:    analogWrite(4,per);  xSemaphoreGive( xSerialSemaphore ); // Now free or "Give" the Serial Port for others.  }  vTaskDelay(1); // one tick delay (15ms) in between reads for stability  }  } |

### Analysis:

|  |
| --- |
| Speed of motor changes as soon as we change the potentiometer value, all the other tasks are also running without any delay.  All the tasks are running smoothly. |

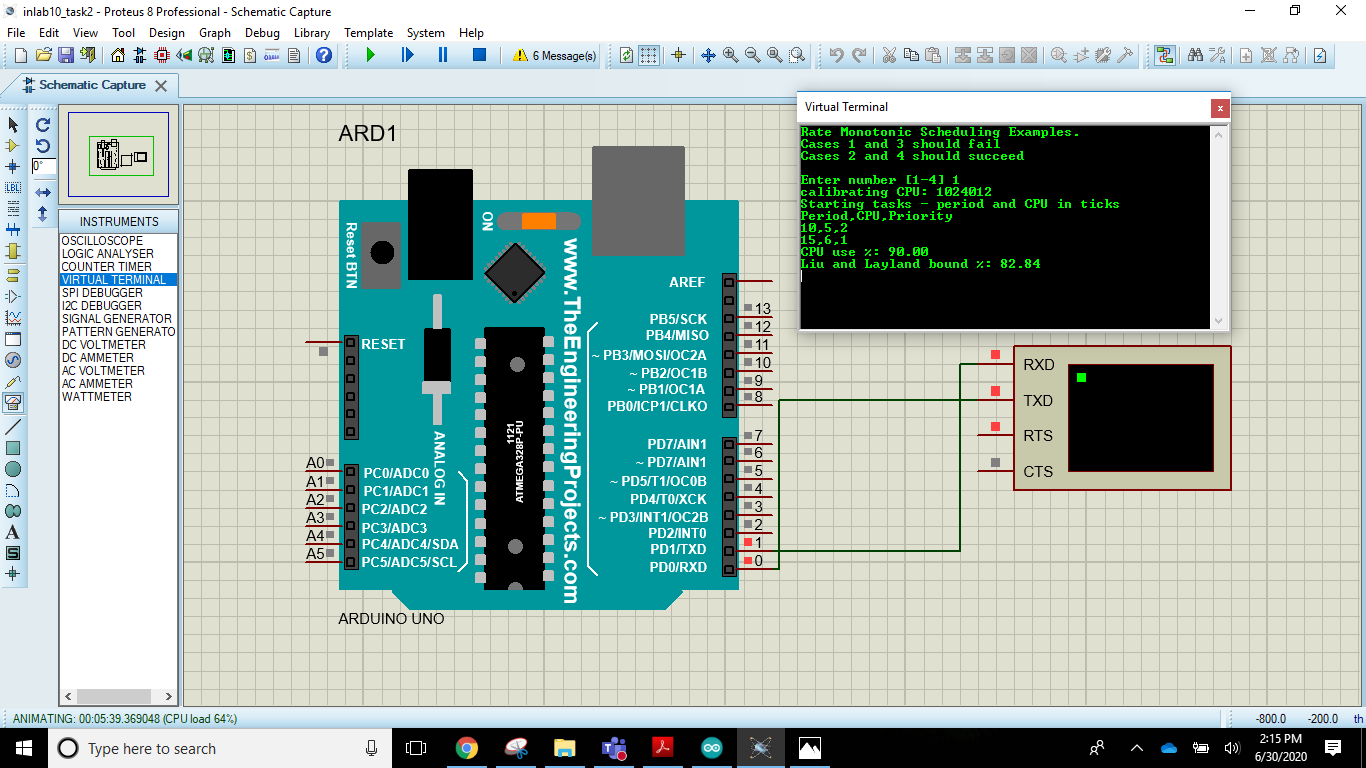
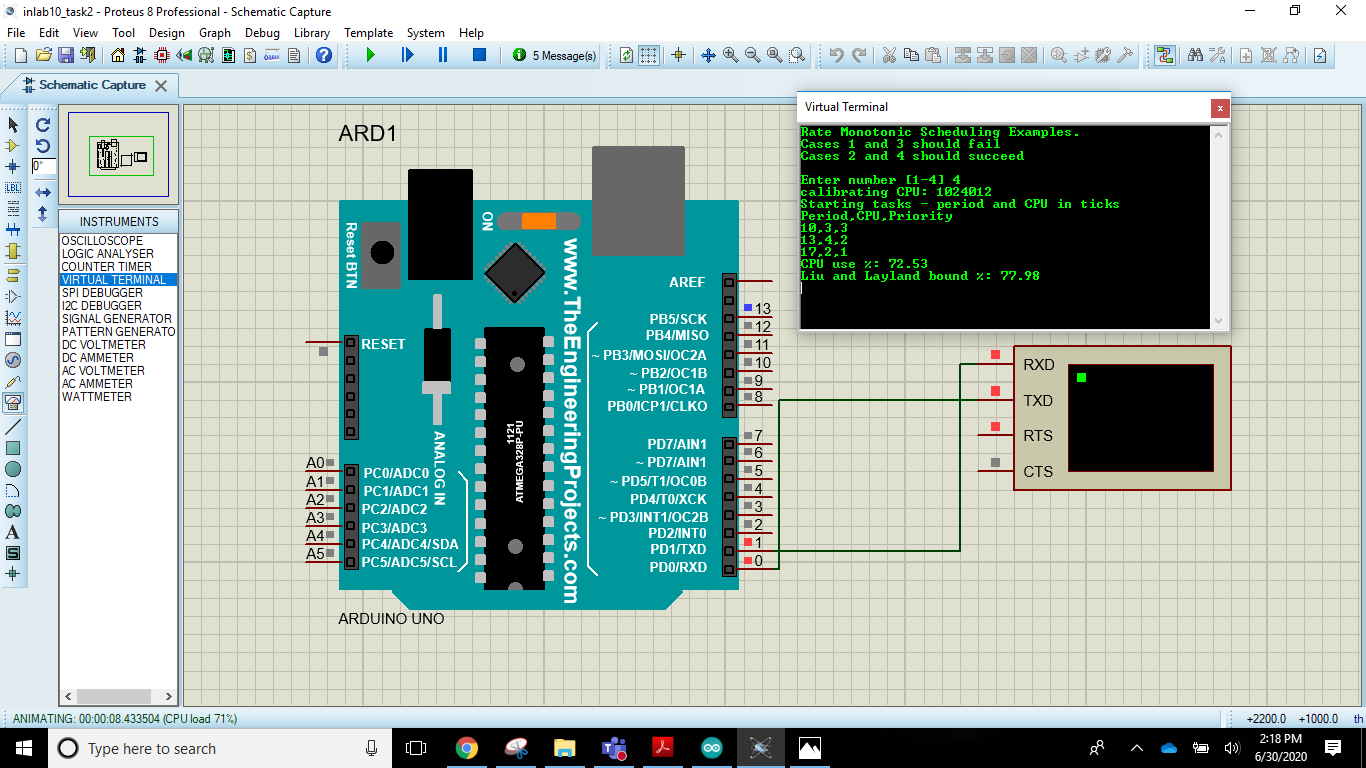
## Conclusion (Task 1):

|  |
| --- |
| When we have a lot of tasks to be scheduled, it is better to use RMS scheduling than Round Robin Scheduling. This makes efficient use of resources and tasks can be completed within their time period without any delay in execution time.  In Round Robin technique, if we have a lot of tasks, the scheduler/kernel takes up a lot of delay after each task to decide which task should be executed next. The clock interrupt is too much which causes poor system performance. |

### In Lab Task 2:

Please have a look at the following code, run in Arduino IDE and simulate in proteus to comment about what it actually do and discuss with your instructor.

## Proteus Schematic:



## Arduino IDE Code:

|  |
| --- |
| // Illustration of Rate Monotonic Scheduling from Liu and Layland paper  //  // Rate Monotonic Scheduling for a set of repeating tasks gives higher  // priority to a task with a smaller period.  //  // Theorem Liu and Layland 1973. Given a preemptive, fixed priority scheduler  // and a finite set of repeating tasks T = {T1; T2; ...; Tn} with associated  // periods {p1; p2 ...; pn} and no precedence constraints, if any priority  // assignment yields a feasible schedule, then the rate monotonic  // priority assignment yields a feasible schedule.  //  // Liu and Layland also derived a bound on CPU utilization that guarantees  // there will be a feasible Rate Monotonic Schedule when a set of n tasks  // have CPU utilization less than the bound.  //  // The Liu Layland bound = 100\*n\*(2^(1/n) - 1) in percent. For large n  // this approaches ln(2) or 69.3%. The extra CPU time can be used by  // lower priority tasks that do not have hard deadlines.  //  // Note that it may be possible to run a given set of tasks with higher CPU  // utilization, depending on task parameters. The Liu Layland bound works  // for every set of tasks independent of task parameters.  //  #include <Arduino\_FreeRTOS.h>  #include <semphr.h>  //------------------------------------------------------------------------------  struct task\_t {  uint16\_t period;  uint16\_t cpu;  uint16\_t priority;  };  task\_t tasks1[] = {{10, 5, 2}, {15, 6, 1}};  task\_t tasks2[] = {{10, 5, 2}, {15, 4, 1}};  task\_t tasks3[] = {{10, 3, 3}, {13, 4, 2}, {17, 4, 1}};  task\_t tasks4[] = {{10, 3, 3}, {13, 4, 2}, {17, 2, 1}};  task\_t\* taskList[] = {tasks1, tasks2, tasks3, tasks4};  int taskCount[] = {2, 2, 3, 3};  //------------------------------------------------------------------------------  // override IDE definition to prevent errors  void printTask(task\_t\* task);  void done(const char\* msg, task\_t\* task, TickType\_t now);  //------------------------------------------------------------------------------  // Liu Layland bound = 100\*n\*(2^(1/n) - 1) in percent  float LiuLayland[] = {100, 82.84271247, 77.97631497, 75.682846, 74.3491775};  //------------------------------------------------------------------------------  #ifdef \_\_AVR\_\_  const unsigned int CAL\_GUESS = 3000;  const float TICK\_USEC = 1024;  #else // \_\_AVR\_\_  const unsigned int CAL\_GUESS = 17000;  const float TICK\_USEC = 1000;  #endif // \_\_AVR\_\_  // dummy CPU utilization functions  static unsigned int cal = CAL\_GUESS;  void burnCPU(uint16\_t ticks) {  while (ticks--) {  for (unsigned int i = 0; i < cal; i++) {  asm("nop");  }  }  }  void calibrate() {  uint32\_t t = micros();  burnCPU(1000);  t = micros() - t;  cal = (TICK\_USEC\*1000\*cal)/t;  }  //------------------------------------------------------------------------------  // print helpers  void printTask(task\_t\* task) {  Serial.print(task->period);  Serial.write(',');  Serial.print(task->cpu);  Serial.write(',');  Serial.println(task->priority);  }  void done(const char\* msg, task\_t\* task, TickType\_t now) {  vTaskSuspendAll();  Serial.println(msg);  Serial.print("Tick: ");  Serial.println(now);  Serial.print("Task: ");  printTask(task);  while(1);  }  //------------------------------------------------------------------------------  // start tasks at 1000 ticks  TickType\_t startTime = 1000;  // test runs for 3000 ticks  TickType\_t finishTime = 4000;  // task code  void task(void\* arg) {  uint16\_t period = ((task\_t\*)arg)->period;  uint16\_t cpu = ((task\_t\*)arg)->cpu;  // simulate last wake time  TickType\_t lastWakeTime = startTime - period;  while (xTaskGetTickCount() < lastWakeTime) vTaskDelay(1);  while (1) {  vTaskDelayUntil(&lastWakeTime, period);  burnCPU(cpu);  // check of failure or success  TickType\_t now = xTaskGetTickCount();  if (now >= finishTime) {  done("Success", (task\_t\*)arg, now);  }  if (now >= (lastWakeTime + period)) {  done("Missed Deadline", (task\_t\*)arg, now);  }  }  }  //------------------------------------------------------------------------------  void setup() {  float cpuUse = 0; // total cpu utilization for set of tasks  int c; // Serial input  int n; // number of tasks to run  task\_t\* tasks; // list of tasks to run  portBASE\_TYPE s; // task create status  Serial.begin(9600);  while(!Serial) {}  Serial.println("Rate Monotonic Scheduling Examples.");  Serial.println("Cases 1 and 3 should fail");  Serial.println("Cases 2 and 4 should succeed");  Serial.println();  // get input  while (1) {  while (Serial.read() >= 0) {}  Serial.print("Enter number [1-4] ");  while ((c = Serial.read()) < 0) {}  Serial.println((char)c);  if (c < '1' || c > '4') {  Serial.println("Invalid input");  continue;  }  c -= '1';  tasks = taskList[c];  n = taskCount[c];  break;  }  Serial.print("calibrating CPU: ");  // insure no interrupts from Serial  Serial.flush();  delay(100);  calibrate();  uint32\_t t = micros();  burnCPU(1000);  Serial.println(micros() -t);  Serial.println("Starting tasks - period and CPU in ticks");  Serial.println("Period,CPU,Priority");  for (int i = 0; i < n; i++) {  printTask(&tasks[i]);  cpuUse += tasks[i].cpu/(float)tasks[i].period;  s = xTaskCreate(task, NULL, 200, (void\*)&tasks[i], tasks[i].priority, NULL);  if (s != pdPASS) {  Serial.println("task create failed");  while(1);  }  }  Serial.print("CPU use %: ");  Serial.println(cpuUse\*100);  Serial.print("Liu and Layland bound %: ");  Serial.println(LiuLayland[n - 1]);  // start tasks  vTaskStartScheduler();  Serial.println("Scheduler failed");  while(1);  }  //------------------------------------------------------------------------------  void loop() {  // Not used - idle loop has a very small, configMINIMAL\_STACK\_SIZE, stack  // loop must never block  } |

### POST LAB TASK:

Take any experimental setup to demonstrate Earliest Deadline First (EDF) scheduling algorithm implement, simulate, and comment on results.

**PROTEUS SIMULATIONS**

**ARDUINO IDE CODE:**

|  |
| --- |
|  |

# Critical Analysis/Conclusion:

|  |
| --- |
|  |

