COMSATS University Islamabad

Lab Report # 09

Real Time Embedded Systems (EEE-446)

Introduction to Free RTOS.

Submitted By:

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

Submitted To:

Dr. Ahsen Malik

Lab # 09

Introduction to Free RTOS.

Objectives

- Introduce a real-time operating system((RTOS)
- Learn what tasks are and how to create them
- Get to know task states and priorities
- Implement inter-task communication using semaphores

Tools

- Arduino
- Proteus ISIS
- freeRTOS library

In-Lab Task 1:

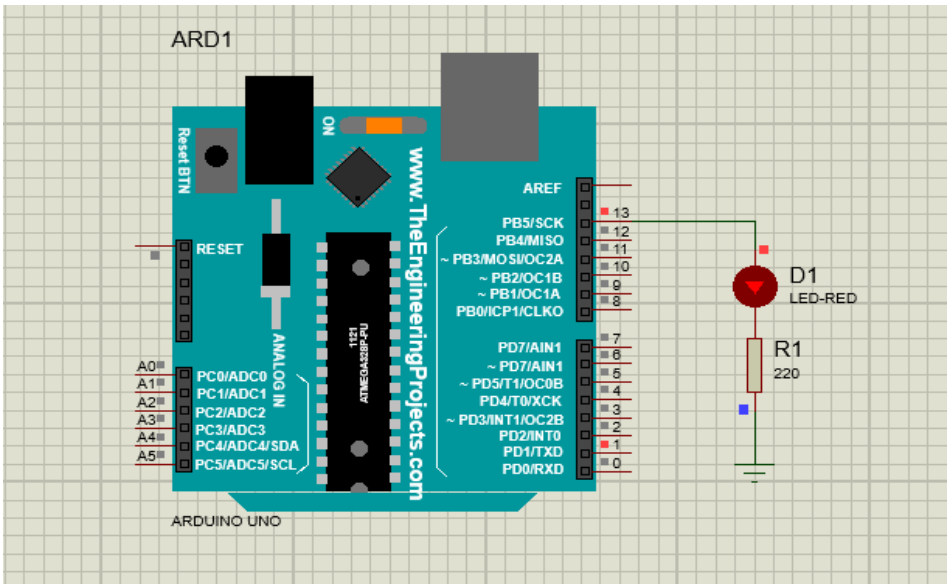
"blinkenlights" in Arduino FreeRTOS

ARDUINO IDE CODE:

```
#include <Arduino_FreeRTOS.h>

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );
//void TaskAnalogRead( 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);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO,
YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  (const portCHAR *)"Blink"   // A name just for humans
```

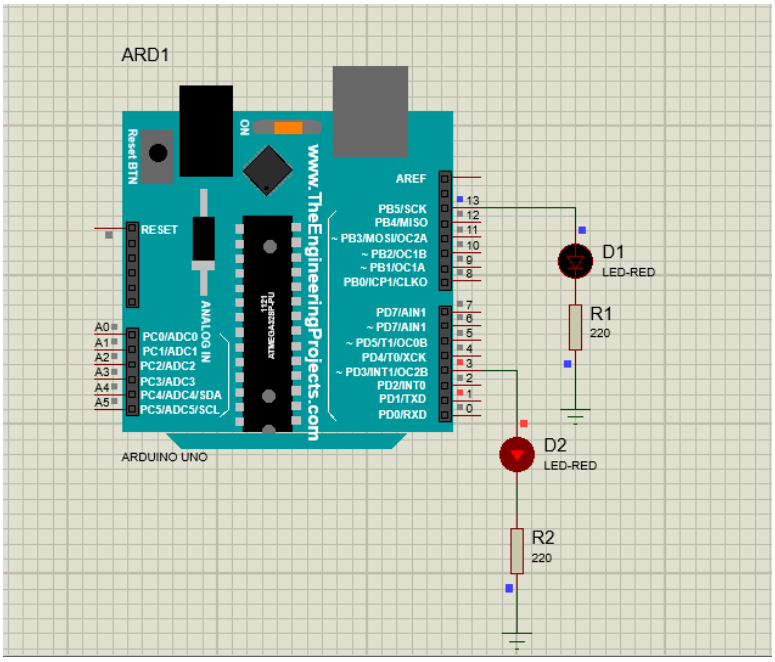
```
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the
lowest.
    ,  NULL );
}

void loop()
{
  // Empty. Things are done in Tasks.
}

void TaskBlink(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  // 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++);
  }
}
```

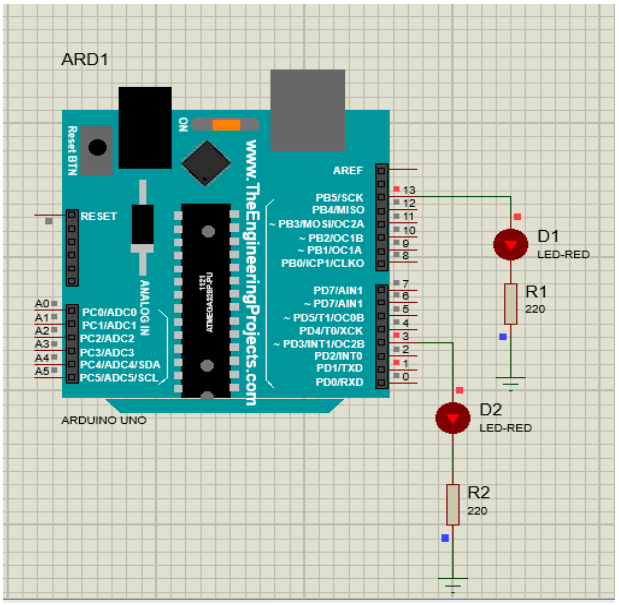
PROTEUS SIMULATIONS:

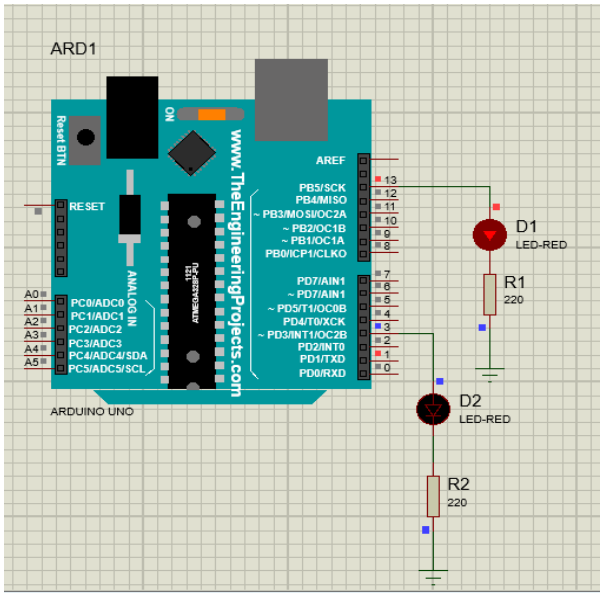


In Lab Task 2:

Multiple tasks in Arduino FreeRTOS
PROTEUS SIMULATIONS:







ARDUINO IDE CODE:

```
#include <Arduino_FreeRTOS.h>

// define two tasks for Blink & AnalogRead
void TaskBlink( void *pvParameters );
void TaskBlink1( void *pvParameters );
```

```
// the setup function runs once when you press reset or power the board
void setup() {

  pinMode(3,OUTPUT);

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4
based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  (const portCHAR *)"Blink"   // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskBlink1
    ,  (const portCHAR *) "Blink1"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

  // Now the task scheduler, which takes over control of scheduling individual tasks, is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

void TaskBlink(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
    vTaskDelay( 2000 / portTICK_PERIOD_MS ); // wait for one second
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( 2000 / portTICK_PERIOD_MS ); // wait for one second
  }
}

void TaskBlink1(void *pvParameters)  // This is a task.
```

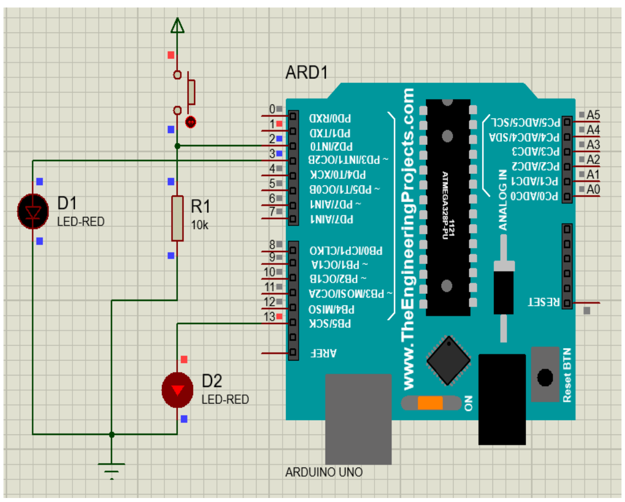
```
{
  (void) pvParameters;

  for (;;) // A Task shall never return or exit.
  {
    digitalWrite(3, HIGH);   // turn the LED on (HIGH is the voltage level)
    vTaskDelay( 5000 / portTICK_PERIOD_MS ); // wait for one second
    digitalWrite(3, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( 5000 / portTICK_PERIOD_MS ); // wait for one second
  }
}
```

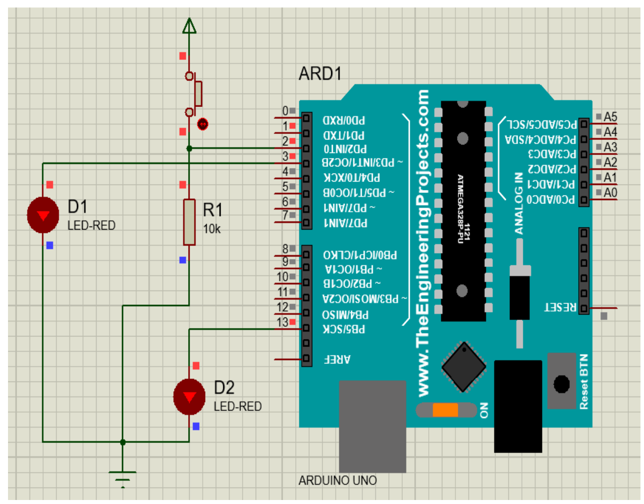
POST LAB TASK:

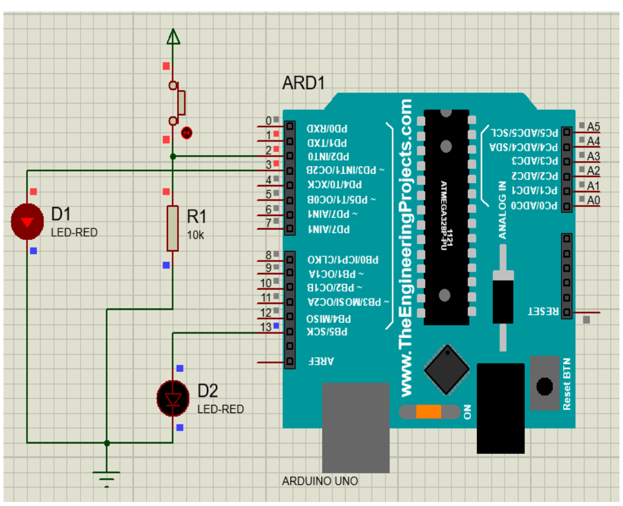
Implement the above scenario given in semaphore.

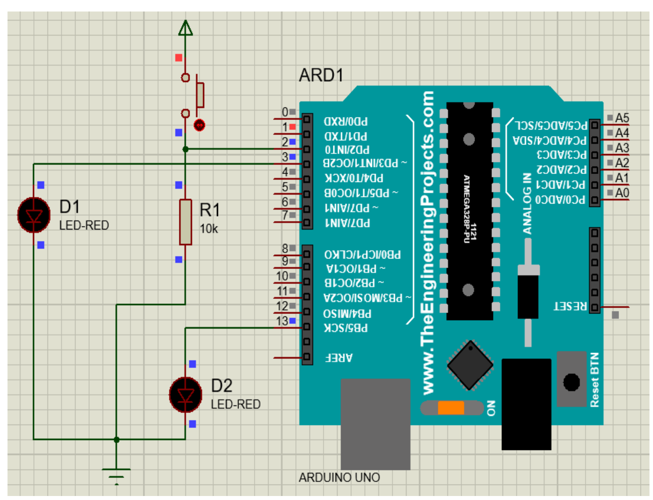
Please take the three tasks running in parallel, Button Input, LED Turn on when button is HIGH and blinking LED with delay of 2 seconds. Here the shared resource that you manage using semaphore is the button state.

PROTEUS SIMULATIONS









ARDUINO IDE CODE:

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>  // add the FreeRTOS functions for Semaphores (or Flags).
int buttonState;
SemaphoreHandle_t xSerialSemaphore;

// define two Tasks for DigitalRead & AnalogRead
void TaskDigitalRead( void *pvParameters );
void TaskBlink( void *pvParameters );
void TaskDigitalWrite( 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);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4
based boards.
  }

  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 );

}

void loop()
{
  // Empty. Things are done in Tasks.
}
void TaskDigitalRead( void *pvParameters __attribute__((unused)) )  // This is a Task.
{

  uint8_t pushButton = 2;

  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);

  for (;;) // A Task shall never return or exit.
  {

    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      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;

  // 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;
  pinMode(3,OUTPUT);

  for (;;) // A Task shall never return or exit.
  {

    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      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
  }
}
```

Critical Analysis/Conclusion:

In this lab we have discussed and implemented the advantages if freeRTOS in scheduling tasks that need to be executed in parallel using a single processor. This enables us to utilize efficiently all resources and saves us the cost of getting a new processor. All this is done at the cost of very minor delays which are mostly negligible.

We also saw several scheduling algorithms such as Round Robin and RMS. Round Robin Algorithm gives equal priority to all the tasks and all the task are executed in parallel. To apply Round Robin scheduling we used the Semaphore Library. Whereas, RMS gives priority to the tasks that has the longest period and executes tasks according to their priority in their respective execution times.

<table>
<tr><th colspan="5">Lab Assessment</th></tr>
<tr><td colspan="3">Pre Lab</td><td>/1</td><td rowspan="5">/10</td></tr>
<tr><td colspan="3">In Lab</td><td>/5</td></tr>
<tr><td rowspan="3">Post Lab</td><td>Data Analysis</td><td>/4</td><td rowspan="3">/4</td></tr>
<tr><td>Data Presentation</td><td>/4</td></tr>
<tr><td>Writing Style</td><td>/4</td></tr>
<tr><td colspan="5">Instructor Signature and Comments</td></tr>
</table>