Lab # 10

Implementation of RMS and EDF scheduling algorithm using freeRTOS

Objectives

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

Tools

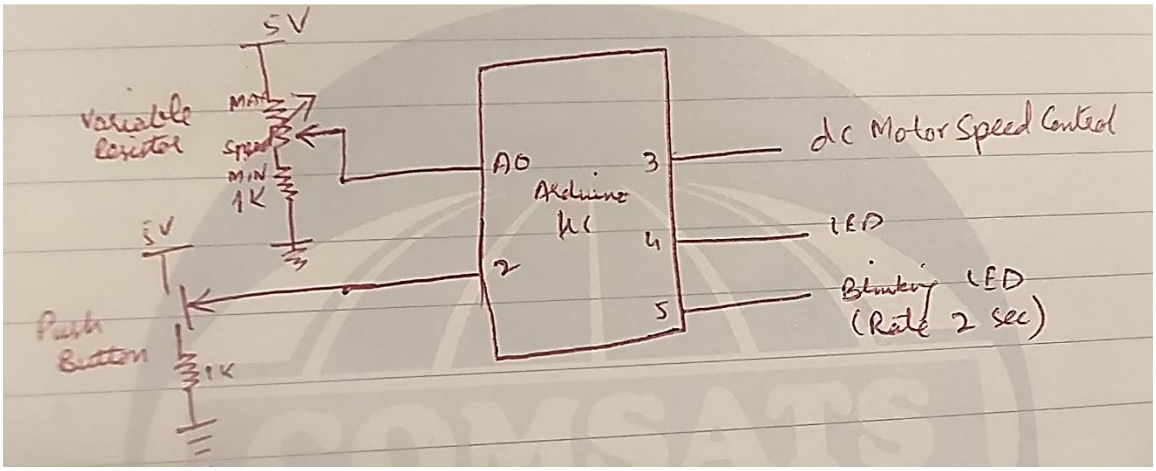
- Arduino
- Proteus ISIS
- freeRTOS library

Pre Lab

Please read the theoretical background of RTOS.

Please complete the previous lab post lab task before starting this lab.

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.

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.

```
// 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 1:
Take any experimental setup to demonstrate Earliest Deadline First (EDF) scheduling algorithm implement, simulate and comment on results.

Critical Analysis / Conclusion

(By Student about Learning from the Lab)

Lab Assessment				
Pre Lab			**/1**	**/10**
In Lab			**/5**	
Post Lab	**Data Analysis**	**/4**	**/4**	
	Data Presentation	**/4**		
	Writing Style	**/4**		
Instructor Signature and Comments				