# Lab 5 - Advanced Embedded Systems - Spring 2014

**Objective:**
1.FreeRTOS installation & operation on the Renesas RX63N

**General Information**
1 Advanced Book chapter on FreeRTOS was read.
2. The respective package was downloaded from the directed site
(As mentioned in the references)
3. The required tasks were created, scheduled & tested.

**General Steps:**
1.The API for the RX63N is downloaded & studied.
2.The API was simulated using 'Blinky' session.
3.Two tasks were created & scheduled to blink the LEDs at intervals 1sec & 10 sec respectively.
4. Then task was created to read the temperature from the on chip ADC. Before reading the internal temperature the ADC initializations were implemented. This is the producer task which continuously reads data i.e. the temperature value.
5. Then the task to display this value on the LCD was created and there priorities were decided. Thus is the consumer task. But an easier way to acknowledge the changing of temperature continuously & at 5 second interval was to insert break points & observe the varying values.
6.All these tasks were assigned to the scheduler.

**Detailed Steps:**
1.The FreeRTOS package is available for multiple processor cores .The relevant API for the RX63N is down loaded .The inbuilt task of prvQueueReceiveTask() & prvQueueSendTask() were studied for task creation, priority allotting & scheduling.

2.The following tasks were created

```
/*
 * The tasks as defined at the top of this file.
 */
static void LED1s(void *pvParameters );
static void LED10s(void *pvParameters );

static void get_temp(void *pvParameters );
static void write_temp(void *pvParameters );
```

```
//LED1s
                    xTaskCreate( LED1s, "LED1s_name",configMINIMAL_STACK_SIZE,NULL,
tskIDLE_PRIORITY + 1,NULL);
                    //LED10s
                    xTaskCreate( LED10s, "LED10s_name",configMINIMAL_STACK_SIZE,NULL,
tskIDLE_PRIORITY + 1,NULL);
                  //Task get temp
                    xTaskCreate( get_temp, "read temp int",configMINIMAL_STACK_SIZE,NULL,
tskIDLE_PRIORITY + 1,NULL);
                    //task write_temp to LCD
                    xTaskCreate( write_temp, "write temp int",configMINIMAL_STACK_SIZE,NULL,
tskIDLE_PRIORITY + 1,NULL);
```

3.Semaphore declaration.

```
                  vSemaphoreCreateBinary( semaphore_gaurd_tp);
```

4. Task 1 for toggling even leds

```
 static void LED1s( void *pvParameters )
{
                    for( ;; )
          {
                    vTaskDelay(delay_LED1s ); // Delay for toggling
                    LED4 = ~LED4;
                    LED6 = ~LED6;
                    LED8 = ~LED8;
                    LED10 = ~LED10;
                    LED12 = ~LED12;
                    LED14 = ~LED14;
          }
}
```

5. Task 2 for toggling  odd leds.

```
static void LED10s( void *pvParameters )
{
          for( ;; )
          {
                    vTaskDelay( delay_LED10s ); //Delay for toogling
                    LED5 = ~LED5;
                    LED7 = ~LED7;
                    LED9 = ~LED9;
                    LED11 = ~LED11;
                    LED13 = ~LED13;
                    LED15 = ~LED15;
```

```
        }
}
```

6. Task to get temperature i.e. reading internal temperature.

```
static void get_temp(void *pvParameters )
{
        int tan,may;
        int temp_int = 0;

        double voltage_sense_int = 0;
        TimerHandle_t xTimer;

        #ifdef PLATFORM_BOARD_RDKRX63N
        SYSTEM.PRCR.WORD = 0xA50B;
        #endif

        MSTP(S12AD) = 0;
        MSTP(TEMPS) = 0;

#ifdef PLATFORM_BOARD_RDKRX63N
        SYSTEM.PRCR.WORD = 0xA500;
#endif

        while(1){

        //ADC Initialization

        S12AD.ADEXICR.BIT.TSS = 1;
        S12AD.ADSSTR23.WORD = 0x1414 ; //Ad sampling state register
        TEMPS.TSCR.BIT.TSEN = 1; // start temp sensor
        for(may = 0; may<150; may++)   //Stabilization delay
        {
        }
        TEMPS.TSCR.BIT.TSOE = 1;

        S12AD.ADCSR.BIT.ADST = 1;  //start conversion
        for( tan = 0; tan<150; tan++)
        {
        }
        TEMPS.TSCR.BIT.TSEN = 0;
        TEMPS.TSCR.BIT.TSOE = 0;

        temp_int = (int) S12AD.ADTSDR;
        voltage_sense_int = (temp_int * 3.3) / 4096;
```

```
        if(semaphore_gaurd_tp != NULL) //Semaphore Check
        {
                if(xSemaphoreTake(semaphore_gaurd_tp, 0) == pdPASS)
// Check Semaphore value
                { //Atomic Section

                        Temp_value = ((voltage_sense_int - 1.26) / 0.0041) + 25;

                }
                xSemaphoreGive(semaphore_gaurd_tp); //Release Semaphore
        }

        }
}
```

7. The task to write/update temperature every 5seconds.

```
static void write_temp(void *pvParameters )
{
        float Temp_updated = 0;

        while(1){
                vTaskDelay( delay_temperature );
                if(semaphore_gaurd_tp != NULL)
                {
                if(xSemaphoreTake(semaphore_gaurd_tp, 0) == pdPASS)
//Acquire Semaphore Access
                {       //Atomic Section
                        Temp_updated = Temp_value;

                }
                xSemaphoreGive(semaphore_gaurd_tp);

                }
        }
}
```

**Learned Concepts:**

1. The task creation & scheduling depending upon their rate/delays.
2.The sharing of data can lead to producer-consumer problem.
3.The use of semaphore helps to resolve the shared data problem .In this case the shared data being the temperature read & the one updated.
4.The installation & operation of FreeRTOS on RX63N .