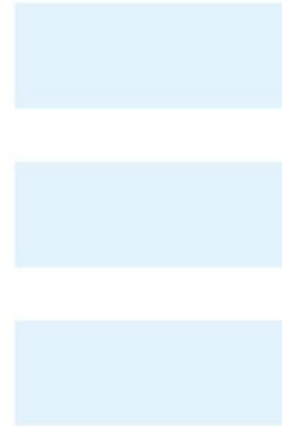
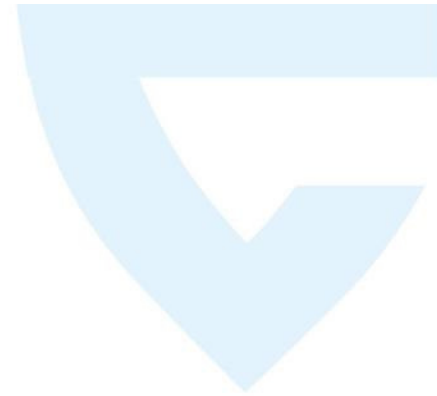


SoRTES Lab - Session 3

Ashok Thangarajan, Stefanos Peros, Emekcan Aras

DistrINet



Session 3 Overview

- › Introduction to RTOS
- › Introduction to FreeRTOS with examples using Arduino FreeRTOS library
- › Power Modes in ATmega32U4

What is an RTOS

- › Used in real-time and embedded systems
- › Provides multi-tasking capability in real-time environment
- › Guarantees temporal requirements

FreeRTOS

- › Small RTOS designed for microcontrollers
- › Provides basic functionality
 - ›› Real-time scheduling
 - ›› Inter-task communication
 - ›› Timing and synchronization primitives

FreeRTOS in Arduino

- › Can be included like any other library
- › Include “Arduino_FreeRTOS.h”
- › Follow the link:

<https://create.arduino.cc/projecthub/feilipu/using-freertos-multi-tasking-in-arduino-ebc3cc>

FreeRTOS Task

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode, //Task fn
                        const char * const pcName, //Human readable name
                        configSTACK_DEPTH_TYPE usStackDepth, //Stack size
                        void *pvParameters, //Thread function parameters
                        UBaseType_t uxPriority, //Task priority
                        TaskHandle_t *pxCreatedTask ); //Task handle
```

FreeRTOS Task

- › Priority can be between 0 to (`configMAX_PRIORITIES - 1`)
- › 0 lowest, (`configMAX_PRIORITIES - 1`) highest priority
- › `configMAX_PRIORITIES` -> defined in `FreeRTOSConfig.h`

FreeRTOS Task

```
void setup() {  
    xTaskCreate(TaskBlink, "Blink", 128, NULL, 0, NULL );  
}  
  
void loop() {}  
  
void TaskBlink(void *pvParameters)  
{  
    (void) pvParameters;  
    pinMode(LED_BUILTIN, OUTPUT);  
    for (;;)   
    {  
        digitalWrite(LED_BUILTIN, HIGH);  
        vTaskDelay( 250 / portTICK_PERIOD_MS );  
        digitalWrite(LED_BUILTIN, LOW);  
        vTaskDelay( 250 / portTICK_PERIOD_MS );  
    }  
}
```


FreeRTOS Task

- › Idle task – automatically created
- › Can be used for low power mode
- › Hook available for application
- › Should enable configUSE_IDLE_HOOK from FreeRTOSConfig.h
- › void vApplicationIdleHook(void); to write your code

FreeRTOS synchronization

- › Binary and counting semaphores, mutex, task notify and events

`SemaphoreHandle_t xSemaphoreCreateBinary(void);`

- › Created in empty state, must be given first before taken
- › Semaphore Give:

`xSemaphoreGive(SemaphoreHandle_t xSemaphore);`

- › Handle is obtained while creating the semaphore

FreeRTOS synchronization

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>
SemaphoreHandle_t SemaphoreHndl;

void setup()
{
    Serial.begin(38400);

    SemaphoreHndl = xSemaphoreCreateBinary();
    if ( ( SemaphoreHndl ) != NULL )
        xSemaphoreGive( ( SemaphoreHndl ) );
}
```

FreeRTOS synchronization

- › Semaphore Take:

```
xSemaphoreTake( SemaphoreHandle_t xSemaphore, TickType_t  
xTicksToWait );
```

- › Time in ticks to wait for semaphore to become available
- › If INCLUDE_vTaskSuspend is set to 1 in “FreeRTOSConfig.h”
portMAX_DELAY can be used to wait indefinitely
- › Semaphore give and take cannot be used from ISR, separate API defined
for that, and the semantics vary

FreeRTOS synchronization

```
static void TaskAnalogue(void *pvParameters)
{
    for (;;)
    {
        // See if we can obtain the Semaphore.
        if ( xSemaphoreTake( SemaphoreHndl, ( TickType_t ) 5 ) == pdTRUE )
        {
            // We were able to obtain the semaphore and can now access the shared resource.
            // Do what you need to do with the shared resource

            xSemaphoreGive( SemaphoreHndl ); //Give it back when you are done!!!
        }
        vTaskDelayUntil( 1000 / portTICK_PERIOD_MS );
    }
}
```

FreeRTOS synchronization

- › Counting semaphore operated similarly

```
SemaphoreHandle_t xSemaphoreCreateCounting(  
    UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);
```

- › Can be used for counting or resource management
- › Operates on same principle as binary semaphore

FreeRTOS IPC

- › Queues, message buffers and stream buffers
- › Queues

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,  
UBaseType_t uxItemSize );
```

- › Handle for reference, queue length and size of each item

FreeRTOS IPC

```
#include <Arduino_FreeRTOS.h>
// Include queue support
#include <queue.h>

/*
 * Declaring a global variable of type QueueHandle_t
 */
QueueHandle_t integerQueue;

void setup() {

    integerQueue = xQueueCreate(10, sizeof(int));

    if (integerQueue != NULL) {
        // Create task that consumes the queue if it was created.
        xTaskCreate(TaskSerial, "Serial", 128, NULL, 2, NULL);

        // Create task that publish data in the queue if it was created.
        xTaskCreate(TaskAnalogRead, "AnalogRead", 128, NULL, 1, NULL);
    }
}
```


FreeRTOS IPC

```
BaseType_t xQueueSend( QueueHandle_t xQueue, const void  
* pvItemToQueue, TickType_t xTicksToWait );
```

- › Queue handle reference, Item buffer and ticks to wait if the queue is full, when send is called.

FreeRTOS IPC

```
void TaskAnalogRead(void *pvParameters)
{
    (void) pvParameters;

    for (;;)
    {
        // Read the input on analog pin 0:
        int sensorValue = analogRead(A0);
        xQueueSend(integerQueue, &sensorValue, portMAX_DELAY);
        // One tick delay (15ms) in between reads for stability
        vTaskDelay(1);
    }
}
```

FreeRTOS IPC

`BaseType_t xQueueReceive(QueueHandle_t xQueue, void *pvBuffer, TickType_t xTicksToWait);`

- › Queue handle reference, Item buffer and time in ticks the task should wait in blocked state
- › Both `xQueueReceive` and `xQueueSend` should not be used from ISR, different API's for that
- › Message can be copied without removing from queue

FreeRTOS IPC

```
void TaskSerial(void * pvParameters) {
    (void) pvParameters;
    // Init Arduino serial
    Serial.begin(9600);

    while (!Serial) {
        vTaskDelay(1);
    }
    int valueFromQueue = 0;
    for (;;)
    {
        if (xQueueReceive(integerQueue, &valueFromQueue, portMAX_DELAY) == pdPASS) {
            Serial.println(valueFromQueue);
        }
    }
}
```

Power Management

Power Management

- › IoT devices need to run for years in battery
- › Careful management of energy is required
- › A typical IoT device operation:
 - ›› Wake Up -> do the operation (as fast as possible) -> Sleep
- › You select which sleep mode it needs to go to

Power modes

Sleep Mode	Active Clock Domains				Oscillators	Wake-up Sources								
	clk _{CPU}	clk _{FLASH}	clk _{I/O}	clk _{ADC}		Main Clock Source Enabled	INT6, INT3:0 and Pin Change	TWI Address Match	SPM/ EEPROM Ready	ADC	WDT Interrupt	Other I/O	USB Synchronous Interrupts	USB Asynchronous Interrupts ⁽³⁾
Idle			X	X		X	X	X	X	X	X	X	X	X
ADCNRM				X		X	X ⁽²⁾	X	X	X	X		X	X
Power-down							X ⁽²⁾	X			X			X
Power-save							X ⁽²⁾	X			X			X
Standby ⁽¹⁾						X	X ⁽²⁾	X			X			X
Extended Standby						X	X ⁽²⁾	X			X			X

- Notes:
1. Only recommended with external crystal or resonator selected as clock source.
 2. For INT6, only level interrupt.
 3. Asynchronous USB interrupts are VBUSTI and WAKEUPI.

Power Mode Selection

- › Depends on application
 - ›› Depends on response time required
- › Depends on your board
 - ›› Eg: You can always go to deep sleep (power down) and return if there is an external RTC and response times are not strict

Requirement in project

- › Two modes should be used
- › Based on use case
 - ›› Self wake up
 - ›› Wake up with external trigger

DistrinNet

Thank you!

<https://distrinet.cs.kuleuven.be/>