Lecture 18: FreeRTOS examples

Vikram Iyer

Adapted from material by Blake Hannaford

Announcements

- Lab 4 Part 1 (FreeRTOS setup + project idea) due Friday
- Lab 4 FFT + project due at end of finals week
- Please fill out the course evaluation, this really helps us!
  Link (closes 6/4 at 11:59pm):
  https://uw.iasystem.org/survey/274849
- Return lab kits by end of finals week to EE store or in lab

Creating a task

```
void vTaskCode( void * pvParameters ) { // Actual task definition
    /* The parameter value is expected to be 1 as 1 is passed in the pvParameters value in the
       call to xTaskCreate() below. configASSERT( ( ( uint32_t ) pvParameters ) == 1 );
    for( ;; ) {
        /* Task code goes here like in a loop() function. */
    }
}
```

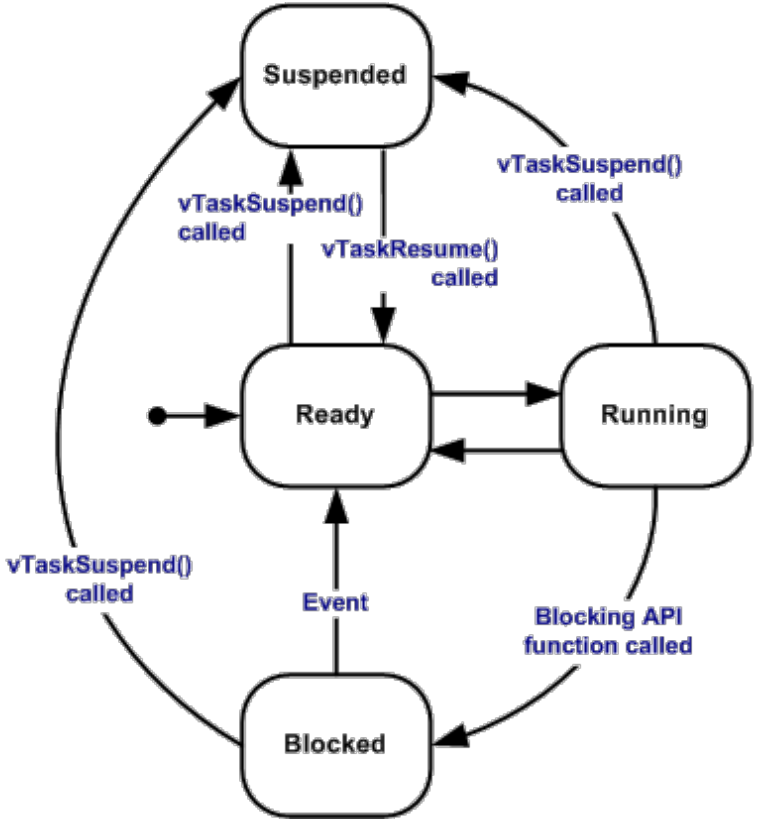
```
void setup() {
    BaseType_t xReturned;
    TaskHandle_t xHandle = NULL;
    xReturned = xTaskCreate(          /* Create the task, storing the handle. */
                    vTaskCode,        /* Function that implements the task. */
                    "NAME",           /* Text name for the task. */
                    STACK_SIZE,       /* Stack size in words, not bytes. */
                    ( void * ) 1,     /* Parameter passed into the task. */
                    tskIDLE_PRIORITY,/* Priority at which the task is created. */
                    &xHandle );       /* Used to pass out the created task's handle. */

    if( xReturned == pdPASS ){
        /* The task was created.  Use the task's handle to delete the task. */
        vTaskDelete( xHandle );
    }
}
```

Creating a task

```
void TaskBlink(void *pvParameters)
    for (;;) {
        digitalWrite(LED_PIN, HIGH); // turn the LED on (HIGH is the voltage level)
        vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second
        digitalWrite(LED_PIN, LOW); // turn the LED off by making the voltage LOW
        vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second
    }
}
```

```
void TaskBlink( void *pvParameters );

void setup() {
    xTaskCreate(
        TaskBlink,
        "Blink”, // A name just for humans
        128,     // This stack size in words (16b)
        NULL,    // Parameters
        2,       // Priority, with 3
                 //(configMAX_PRIORITIES - 1)
                 // being the highest, and 0
                 // being the lowest.
    , NULL );
}
```

FreeRTOS task states



Running- When a task is actually executing it is said to be in the Running state. It is currently utilising the processor. There can only be one task in the Running state at any given time.

Ready- Ready tasks are those that are able to execute not currently executing because another task is using CPU

Blocked- A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event (e.g. vTaskDelay(), waiting for queue. Tasks have a 'timeout' period, after which the task will be unblocked

Suspended- These tasks do not run. Tasks only enter or exit the Suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.

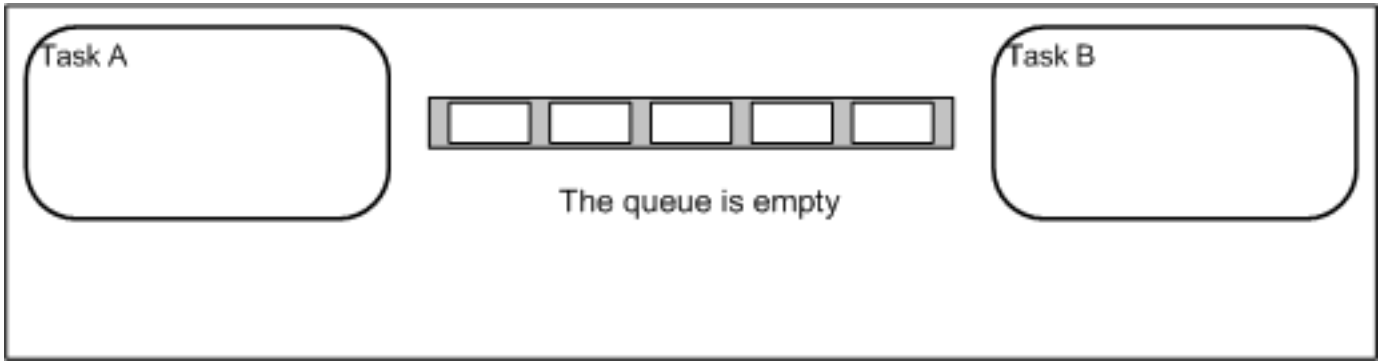
FreeRTOS timing and task control

```
vTaskDelay( 500 / portTICK_PERIOD_MS ); // wait for one second

TickType_t tick = xTaskGetTickCount();

vTaskSuspend( TaskHandle_t xTaskToSuspend );

vTaskResume( xHandle );
```

```
…
// Suspend ourselves.
vTaskSuspend( NULL );

/* We cannot get here unless
another task calls vTaskResume
with our handle as the
parameter.*/
…
```

```
// Resume the suspended task
vTaskResume( xHandle );
```

Sending data between tasks using Queues




```
// Set up the Queue
QueueHandle_t queue;

void setup(){
  queue = xQueueCreate(
      2,
      sizeof(int)
  );
}
```

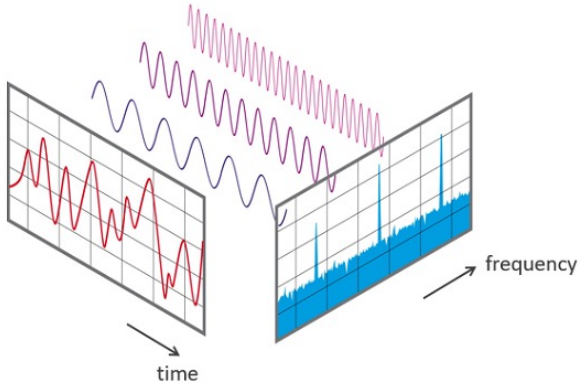
```
// Task A adds to the Queue
void taskA(){
  xQueueSendToBack(
    queue,
    &data_to_send,
    1);
}
```

```
// Task A adds to the Queue
void taskA(){
  xQueueReceive(
    queue,
    &received_data,
    0);
}
```

Fourier Transform

A mathematical operation that converts data into a form that describes the frequencies present

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$$




```
//Number of signal cycles that the sampling will read
double cycles = (((samples-1) * signalFrequency) / samplingFrequency);
double vReal[N_SAMPLES];
double vImag[N_SAMPLES];

for (uint16_t i = 0; i < samples; i++) {
 // sine wave
 vReal[i] = int8_t((amplitude * (sin((i * (twoPi * cycles)) / samples))) / 2.0);
 //Imaginary part must be zeroed in case of looping to avoid wrong calculations and overflows
 vImag[i] = 0.0;
}

arduinoFFT fft = arduinoFFT(); /* Create FFT object */

fft.Compute(vReal, vImag, samples, N_SAMPLES, FFT_FORWARD);
```