Yufei Lin

Note

Jun 11th 2019

ASTRI Project

How to Use FreeRTOS Execute Concurrent Programming in Arduino

I. Set Up FreeRTOS

1. Download Arduino IDE (Version 1.6.8 or Later)
2. Download FreeRTOS libraries from Github or use Arduino IDE
3. If download from Github, import FreeRTOS library into IDE, by first decompress the file and copy the file into ”../arduino-1.x.x/libraries”
4. Open and upload an example file under FreeRTOS library and then verify the output

II. Use FreeRTOS to control Arduino

In a basic FreeRTOS file, it includes the following:

1. File imports
2. Define tasks
3. Setup function
4. Loop function
5. Task functions

II.1. File Imports

This section includes all the necessary packages we need for one FreeRTOS file. It must include ”Arduino.FreeRTOS.h”.

Therefore, this section looks like the following:

```
#include <Arduino_FreeRTOS.h>
//#include <task.h>...
```

II.2. Define tasks

This section defines the task functions that we will need for developing future tasks. All functions are defined as the same way as it is defined in any other c file, include a return type, function name, and parameters.

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

II.3. Setup Function

In setup function, we need to define the following elements of this program:

1. Initialization of communication rate

2. A loop for port to connect

3. Initialization of tasks

II.3.1. Task Initialization

Generally we use xTaskCreate() function to create a not static task in order to adjust memory assigned to each task in RAM. Also, in migration to FreeRTOS, we use a memory management method to allocate memory in RAM and protect concurrent tasks.
A xTaskCreate() method usually takes in five parameters:

Parameter	Explanation
pvTaskCode	Pointer to the task entry function (just the name of the function that implements the task, see the example below). Tasks are normally implemented as an infinite loop, and must never attempt to return or exit from their implementing function. Tasks can however delete themselves.
pcName	A descriptive name for the task. This is mainly used to facilitate debugging, but can also be used to obtain a task handle.The maximum length of a task's name is set using the configMAX_TASK_NAME_LEN parameter in FreeRTOSConfig.h.
usStackDepth	The number of words (not bytes!) to allocate for use as the task's stack. For example, if the stack is 16-bits wide and usStackDepth is 100, then 200 bytes will be allocated for use as the task's stack. As another example, if the stack is 32-bits wide and usStackDepth is 400 then 1600 bytes will be allocated for use as the task's stack. The stack depth multiplied by the stack width must not exceed the maximum value that can be contained in a variable of type size_t.
pvParameters	A value that will passed into the created task as the task's parameter. If pvParameters is set to the address of a variable then the variable must still exist when the created task executes - so it is not valid to pass the address of a stack variable.
uxPriority	The priority at which the created task will execute. Systems that include MPU support can optionally create a task in a privileged (system) mode by setting bit portPRIVILEGE_BIT in uxPrriority. For example, to create a privileged task at priority 2 set uxPriority to (2 ‖ portPRIVILEGE_BIT).

Usually, xTaskCreate() function would be written in the following form:

```
xTaskCreate(
    TaskAnalogRead
    ,  (const portCHAR *) "AnalogRead"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );
```

II.4. Loop Function We left the loop function blank, because we are already initializing different tasks and running different tasks on different loops.

II.5. Tasks

For each task, we would have the normal set up as a normal Arduino task. For each setup function of the original Arduino task, we break down the function and take the parameters and methods out. And we would still have a loop embedded in the task, usually written in an infinite loop way. A demo task is shown as below:

```
void TaskSonar(void *pvParameters)
{
  (void) pvParameters;
  const int trigPin = 7;
  const int echoPin = 6;
  // defines variables
  long duration;
  int distance;

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication

  for (;;)
  {
    // Clears the trigPin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    // Sets the trigPin on HIGH state for 10 micro seconds
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Reads the echoPin, returns the sound wave travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
    // Calculating the distance
    distance= duration*0.034/2;
    // Prints the distance on the Serial Monitor
    Serial.print("Distance: ");
    Serial.println(distance);
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
```

III. Resources

1. https://www.freertos.org/a00125.html

2. https://www.youtube.com/watch?v=sjJkyBH_oks&t=214s

IV. Appendix

The finished code from section II is the following:

```
#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 );

  xTaskCreate(
    TaskAnalogRead
    ,  (const portCHAR *) "AnalogRead"
    ,  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.
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

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

/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, LEONARDO, MEGA, and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN takes care
  of use the correct LED pin whatever is the board used.

  The MICRO does not have a LED_BUILTIN available. For the MICRO board please substitute
  the LED_BUILTIN definition with either LED_BUILTIN_RX or LED_BUILTIN_TX.
  e.g. pinMode(LED_BUILTIN_RX, OUTPUT); etc.

  If you want to know what pin the on-board LED is connected to on your Arduino model, check
  the Technical Specs of your board  at https://www.arduino.cc/en/Main/Products

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald

  modified 2 Sep 2016
  by Arturo Guadalupi
*/

  // 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( 1000 / portTICK_PERIOD_MS ); // wait for one second
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
  }
}

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

/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Graphical representation is available using serial plotter (Tools > Serial Plotter menu)
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

  This example code is in the public domain.
*/

  for (;;)
  {
    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);
    // print out the value you read:
    Serial.println(sensorValue);
    vTaskDelay(1);  // one tick delay (15ms) in between reads for stability
  }
}
```