Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Real-Time Systems

Arduino-Eclipse-freeRTOS-Matlab environment configuration

Antonio Camacho Santiago
antonio.camacho.santiago@upc.edu

Eclipse

Install Eclipse Version: 2018-12 (4.10.0) C/C++ developers

Go to menu Help→Arduino Downloads Manager→Platforms→Add→Platforms and select Arduino AVR Boards and Arduino megaAVR Boards

Go to menu Help→Arduino Downloads Manager→Platforms→Add→Libraries and select SparkFun LSM9DS0 Breakout and FreeRTOS

Select File→New→Arduino Project

Copy ..\Arduino\libraries\FreeRTOS\src to working project

Rename *.c to *.cpp

Eclipse

Copy the BlinkAnalogRead example into the main project file.

Add #include <Arduino.h> on top of this file

Compile (Hammer button)

Create a new Launch target→Arduino. Click Next. Name the target as arduino_mega, select the COM port, Board type: Arduino/Genuino Mega or Mega 2560, Programmer AVR ISR.

Download and run the code (Play button)

Open a RS232 terminal view by clicking in menu Window->Show View->Terminal. (Select the same baudrate as in Serial.begin(xxxx))

Note: To import an existing project go to File→Inport→General→Existing Projects into Workspace→Select root directory and check Option “Copy projects into Workspace”

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In port.cpp:

```
//#define    portSCHEDULER_ISR                WDT_vect
//STR
#define    portSCHEDULER_ISR                TIMER1_COMPA_vect
//STR
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In port.cpp:

```
void vPortEndScheduler( void )
{
    /* It is unlikely that the AVR port will get stopped.  If required simply
    disable the tick interrupt here. */

//    wdt_disable();    // disable Watchdog Timer
    //STR
//disable timer1
    cli();
    TCCR1B = 0;
    //TIMSK1 |= (0 << OCIE1A); // deactivate timer's interrupt.
    TCCR1B &= ~(1<< CS12);  // turn off the clock altogether
    TCCR1B &= ~(1<< CS11);
    TCCR1B &= ~(1<< CS10);
    TIMSK1 &= ~(1 << OCIE1A); // turn off the timer interrupt
    //STR
}
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In port.cpp:

```
void prvSetupTimerInterrupt( void )
{
    //reset watchdog
    //wdt_reset();

    //set up WDT Interrupt (rather than the WDT Reset).
    //wdt_interrupt_enable( portUSE_WDTO );

//STR
// TIMER 1 for interrupt frequency 100 Hz:
cli(); // stop interrupts
TCCR1A = 0; // set entire TCCR1A register to 0
TCCR1B = 0; // same for TCCR1B
TCNT1  = 0; // initialize counter value to 0
// set compare match register for 100 Hz increments
OCR1A = 19999; // = 16000000 / (8 * 100) - 1 (must be <65536)
// turn on CTC mode
TCCR1B |= (1 << WGM12);
// Set CS12, CS11 and CS10 bits for 8 prescaler
TCCR1B |= (0 << CS12) | (1 << CS11) | (0 << CS10);
// enable timer compare interrupt
TIMSK1 |= (1 << OCIE1A);
sei(); // allow interrupts
//STR
}
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In port.cpp:

```
#if configUSE_PREEMPTION == 1

    /*
     * Tick ISR for preemptive scheduler.  We can use a naked attribute as
     * the context is saved at the start of vPortYieldFromTick().  The tick
     * count is incremented after the context is saved.
     *
     * use ISR_NOBLOCK where there is an important timer running, that should preempt the scheduler.
     *
     */
//     ISR(portSCHEDULER_ISR, ISR_NAKED) __attribute__ ((hot, flatten));
//STR
//Important!!! remove ISR_NAKED as follows:
ISR(portSCHEDULER_ISR) __attribute__ ((hot, flatten));
//  ISR(portSCHEDULER_ISR, ISR_NAKED ISR_NOBLOCK) __attribute__ ((hot, flatten));
//STR
    ISR(portSCHEDULER_ISR)
    {
        vPortYieldFromTick();
        //STR
        //Important!!! remove line below:
        //__asm__ __volatile__ ( "reti" );
        //STR
    }
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In port.cpp:

```
#else
    /*
     * Tick ISR for the cooperative scheduler.  All this does is increment the
     * tick count.  We don't need to switch context, this can only be done by
     * manual calls to taskYIELD();
     *
     * use ISR_NOBLOCK where there is an important timer running, that should preempt the scheduler.
     */
    ISR(portSCHEDULER_ISR) __attribute__ ((hot, flatten));
//  ISR(portSCHEDULER_ISR, ISR_NOBLOCK) __attribute__ ((hot, flatten));
    ISR(portSCHEDULER_ISR)
    {
        xTaskIncrementTick();
    }

#endif // configUSE_PREEMPTION
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In portmacro.h:

```
//STR
//#define portTICK_PERIOD_MS                ( (TickType_t) _BV(
portUSE_WDTO + 4 ) )    // Inaccurately assuming 128 kHz Watchdog
Timer.
#define portTICK_PERIOD_MS                  ( (TickType_t) 10)
//STR
```

Modify freeRTOS to trigger interrupts each 10ms based on Timer1 instead of Watchdog

In FreeRTOSVariant.h:

```
//STR
//#define configTICK_RATE_HZ      ( (TickType_t)( (uint32_t)128000 >> (portUSE_WDTO + 11) ) )  // 2^11 = 2048 WDT scaler for 128kHz Timer
#define configTICK_RATE_HZ      ( (TickType_t)( 100 )  //TODO: change based on registers instead of custom value
//STR
```

Add trace functionality as a HOOK to the context switich in

In ArduinoFreeRTOS.h:

```
//STR
extern void trace(void);
#ifndef traceTASK_SWITCHED_IN
    /* Called after a task has been selected to run.  pxCurrentTCB
holds a pointer
    to the task control block of the selected task. */
//    #define traceTASK_SWITCHED_IN()
#define traceTASK_SWITCHED_IN() trace()
#endif
//STR
```

Arduino code

Add handlers to tasks in order to be able to account for their state in trace().

Handlers should be global to make them available for trace

In main code.cpp:

```
//tasks handlers, required in last parameter of xTaskCreate when
accessing task info
TaskHandle_t Task1Handle;
TaskHandle_t Task2Handle;
TaskHandle_t Task3Handle;
TaskHandle_t Task4Handle;

// function prototypes
void Task1( void *pvParameters );
void Task2( void *pvParameters );
void Task3( void *pvParameters );
void Task4( void *pvParameters );
```

Arduino code

Create four tasks with their own handler as a parameter to the xTaskCreate

In main code.cpp:

```
xTaskCreate(
    Task1
    ,  (const portCHAR *)"Task1"   // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  3  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
    ,  &Task1Handle );
xTaskCreate(
    Task2
    ,  (const portCHAR *) "Task2"
    ,  128  // Stack size
    ,  NULL
    ,  2  // Priority
    ,  &Task2Handle );
```

```
xTaskCreate(
    Task3
    ,  (const portCHAR *)"Task3"   // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  1  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
    ,  &Task3Handle );
xTaskCreate(
    Task4
    ,  (const portCHAR *) "Task4"
    ,  128  // Stack size
    ,  NULL
    ,  0  // Priority
    ,  &Task4Handle );
```

Arduino code

Create circular buffers

In main code.cpp:

```
//circular buffer for debugging
#define BUFF_SIZE 500
float t[BUFF_SIZE] = {};
byte circ_buffer1[BUFF_SIZE] = {};
byte circ_buffer2[BUFF_SIZE] = {};
byte circ_buffer3[BUFF_SIZE] = {};
byte circ_buffer4[BUFF_SIZE] = {};
float debug_data1[BUFF_SIZE] = {};
unsigned int circ_buffer_counter = 0;

void trace(void)
{
  circ_buffer_counter++;
  if (circ_buffer_counter >= BUFF_SIZE)
  {
    circ_buffer_counter = 0;
  }

  t[circ_buffer_counter] = getTime();//sent time in milliseconds
  circ_buffer1[circ_buffer_counter] = eTaskGetState(Task1Handle);
  circ_buffer2[circ_buffer_counter] = eTaskGetState(Task2Handle);
  circ_buffer3[circ_buffer_counter] = eTaskGetState(Task3Handle);
  circ_buffer4[circ_buffer_counter] = eTaskGetState(Task4Handle);
  debug_data1[circ_buffer_counter]=2.7;
}
```

To remove undefined reference to `eTaskGetState‘ change define

In ArduinoFreeRTOS.h:

```
#ifndef INCLUDE_eTaskGetState
    //STR
    //#define INCLUDE_eTaskGetState 0
    #define INCLUDE_eTaskGetState 1
    //STR
#endif
```

Arduino code

Add getTime():

In main code.cpp:

```
//return time expired in milliseconds
float getTime(void)
{
  float t=(float)0.5e-3*((float)OCR1A*xTaskGetTickCount()+TCNT1));//Sent time in milliseconds!!!
  return t;
}
```

Arduino code

Add compute() to waste some time:

In main code.cpp:

```
//compute is only used to waste time without using delays
void compute(unsigned long milliseconds)
{
  unsigned int i = 0;
  unsigned int imax = 0;
  imax = milliseconds * 92;
  volatile float dummy = 1;
  for (i = 0; i < imax; i++)
  {
    dummy = dummy * dummy;
  }
}
```

Arduino code

Create a timer to stop the kernel and send data to Matlab

In main code.cpp:

```
#include "src/timers.h"
```

```
//timer handlers
TimerHandle_t xOneShotTimer;
BaseType_t xOneShotStarted;
```

```
 xOneShotTimer = xTimerCreate("OneShotTimer",
pdMS_TO_TICKS( 1000 ) , pdFALSE, 0,
OneShotTimerCallback );
  xOneShotStarted = xTimerStart( xOneShotTimer, 0 );
```

```
void OneShotTimerCallback( TimerHandle_t xTimer )
{
  TickType_t xTimeNow;
  xTimeNow = xTaskGetTickCount();
  //oneshottimer_count++;
  //stop the kernel...
  vTaskSuspend(Task1Handle);
  vTaskSuspend(Task2Handle);
  vTaskSuspend(Task3Handle);
  vTaskSuspend(Task4Handle);
  vTaskSuspendAll();

  //...and sent data to the host PC
  unsigned int i;
  for (i = 0; i < BUFF_SIZE; i++)
  {
    Serial.print((float)t[i]);
    Serial.write((uint8_t)circ_buffer1[i]);
    Serial.write((uint8_t)circ_buffer2[i]);
    Serial.write((uint8_t)circ_buffer3[i]);
    Serial.write((uint8_t)circ_buffer4[i]);
    Serial.print((float)debug_data1[i]);
    Serial.println();
  }
  delay(50);
}
```

Matlab

Create a Matlab script to get data from arduino

Launch the Matlab script rs232_r1.m