Use of Tracealyzer with XMC4500

Jinlong Shen[*]

jsdejsde@gmail.com

Abstract

Tracealyzer is a software solution to capture and visualize the behaviors of real-time operation system in microcontroller. In spite of reports of using Tracealyzer on a number of Cortex-M4 microcontrollers, there has been no report about practical use of Tracealyzer to trace FreeRTOS in XMC4500 microcontroller. This article presents such an example using the XMC4500 Relax Kit.

1. Introduction

XMc4500 is an Infineon microcontroller from the Cortex M4F family. FreeRTOS is a real-time operating system (RTOS) that is widely used in microcontrollers. Tracealyzer is an embedded software solution from Percepbio AB that captures the behaviors of RTOSs in the microcontroller and visualizes them on a computer. This article demonstrates how to configure and use FreeRTOS and Tracealyzer with the XMC4500 Relax Kit. The source code including this article is freely available at `https://svn.riouxsvn.com/Tracealyzerxmc4/trunk`

2. Precedure

2.1. System Setup

To use Tracealyzer, you need a host computer and an IDE (integrated development environment). DAVE4 is used as an IDE in this demonstration. In addition, the recorder library (Percepio AB) and the SEGGER_RTT library (SEGGER) are needed too. Tracealyzer can be acquired under free or commercial license from Percepio's website and the libraries can be found under the installation directory. XMC4500 and the computer are connected via a debug probe, for example, the J-Link debug probes made by SEGGER. In this case it wasn't required since the XMC4500 Relax Kit comes with an on-board J-Link debug probe. The system setup is shown in Figure 1.

[*]The author would like to thank Percepio AB and SEGGER Microcontroller GmbH & Co.

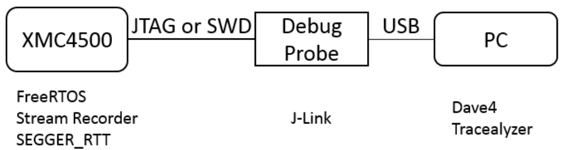
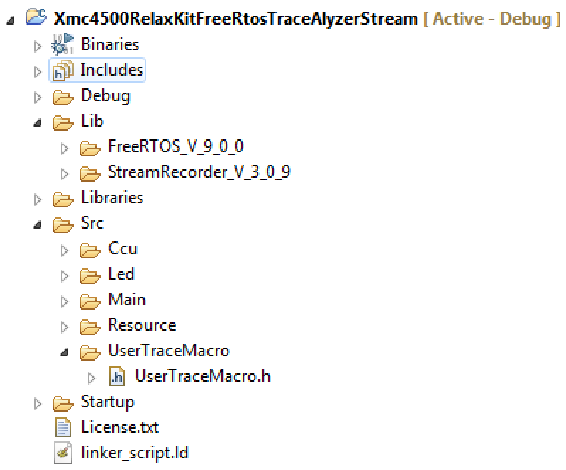


Figure 1: *System setup*

2.2. Project Directories

The project files are organized as shown in Figure 2. The system files are located in Libraries and Startup folders, FreeRTOS and stream recorder in the Lib, and the application in the Src. A few macros are defined in UserTraceMacro/UserTraceMacro.h.



```
Xmc4500RelaxKitFreeRtosTraceAlyzerStream [ Active - Debug ]
  Binaries
  Includes
  Debug
  Lib
    FreeRTOS_V_9_0_0
    StreamRecorder_V_3_0_9
  Libraries
  Src
    Ccu
    Led
    Main
    Resource
    UserTraceMacro
      UserTraceMacro.h
  Startup
  License.txt
  linker_script.ld
```

Figure 2: *Project directories*

2.3. Configuration of Tracealyzer and SEGGER_RTT

To use Tracealyzer, you need to configure FreeRTOS in FreeRTOSConfig.h, the recorder in trcConfig.h, _SEGGER_RTT in SEGGER_RTT.c as well as Tracealyzer on the computer.

In FreeRTOSConfig.h, the trace facility should be included as follows.

```
#define configUSE_TRACE_FACILITY 1
```

At the very end of the same file, choose to use Percepio's recorder as trace facility

```
#if (configUSE_TRACE_FACILITY==1)
#include "trcKernelPort.h"
#endif
```

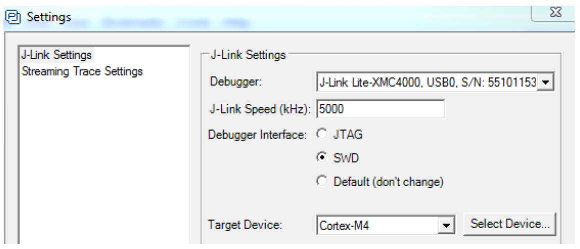
In trcConfig.h, specify the hardware (MCU), FreeRTOS version, the device header file as follows.

```
#define TRC_RECORDER_HARDWARE_PORT \
        TRC_PORT_ARM_Cortex_M

#include "XMC4500.h"

#define TRC_FREERTOS_VERSION \
         TRC_FREERTOS_VERSION_9_X
```

In SEGGER_RTT.c, in order for Tracealyzer PC-Application to find the trace data in the microcontroller, you need to find the definition of _SEGGER_RTT and place it in the RAM starting from 0x10000000 as follows. This is in fact the PSRAM region in XMC4500.

```
SEGGER_RTT_CB _SEGGER_RTT \
__attribute__((section("PSRAM_DATA")));
```

On the computer, configure the J-Link probe interface as in Figure 3 and the streaming trace setting as in Figure 4.

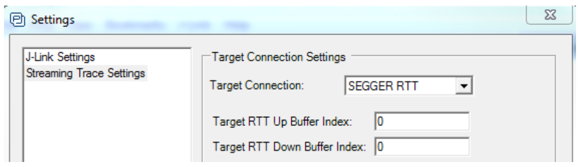


Settings

J-Link Settings
Streaming Trace Settings

J-Link Settings
Debugger: J-Link Lite-XMC4000, USB0, S/N: 55101153
J-Link Speed (kHz): 5000
Debugger Interface: JTAG [✓ SWD] Default (don't change)
Target Device: Cortex-M4
Select Device...

Figure 3: *J-Link settings*



Settings

J-Link Settings
Streaming Trace Settings

Target Connection Settings
Target Connection: SEGGER RTT
Target RTT Up Buffer Index: 0
Target RTT Down Buffer Index: 0

Figure 4: *Streaming trace setting*

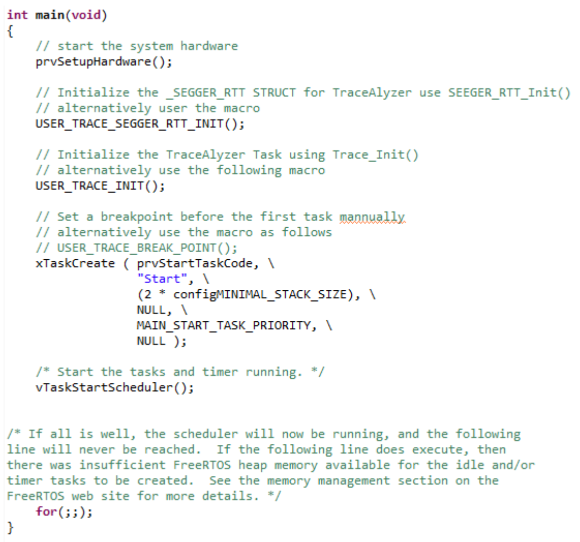
trcTCPIP.h, trcTCPIP.c and trcTCPIPConfig.h are excluded from the build.

2.4. Initialize Tracealyzer in the main()

The main() is shown in Figure 5. At the very beginning inside the main, a few lines of code are needed to initialize SEGGER_RTT and to create a Tracealyzer thread. For this purpose, one can call SEGGER_RTT_Init() and Trace_Init(), respectively. The author defines his own macros that do the same (Figure 5). Since XMC4500 has a specific deviation according to its errata sheet, the macro USER_TRACE_BREAK_POINT() is used to set the breakpoint immediately after Trace_Init(). Alternatively, the user can set the break point at the same place in a debug session. However, a breakpoint is necessary in this case due to this deviation. The macro USER_TRACE_START_DELAY_MS() may be called at the very beginning of the start thread ("Start", prvStartTaskCode) so that the initial events are also captured. These lines of code/macros do not enter into the software if configUSE_TRACE_FACILITY is undefined.

2.5. Embedded Application

The demo application is created completely inside the start thread ("Start", prvStartTaskCode). The start thread initializes the hardware, namely two LEDs and a timer. It then creates two threads ("LED1", "LED2", prvLedTaskCode) and another thread ("IsrCallback", prvIsrCbTaskCode) that



```c
int main(void)
{
    // start the system hardware
    prvSetupHardware();

    // Initialize the _SEGGER_RTT STRUCT for TraceAlyzer use SEEGER_RTT_Init()
    // alternatively user the macro
    USER_TRACE_SEGGER_RTT_INIT();

    // Initialize the TraceAlyzer Task using Trace_Init()
    // alternatively use the following macro
    USER_TRACE_INIT();

    // Set a breakpoint before the first task mannually
    // alternatively use the macro as follows
    // USER_TRACE_BREAK_POINT();
    xTaskCreate ( prvStartTaskCode, \
                  "Start", \
                  (2 * configMINIMAL_STACK_SIZE), \
                  NULL, \
                  MAIN_START_TASK_PRIORITY, \
                  NULL );

    /* Start the tasks and timer running. */
    vTaskStartScheduler();


/* If all is well, the scheduler will now be running, and the following
line will never be reached.  If the following line does execute, then
there was insufficient FreeRTOS heap memory available for the idle and/or
timer tasks to be created.  See the memory management section on the
FreeRTOS web site for more details. */
    for(;;);
}
```

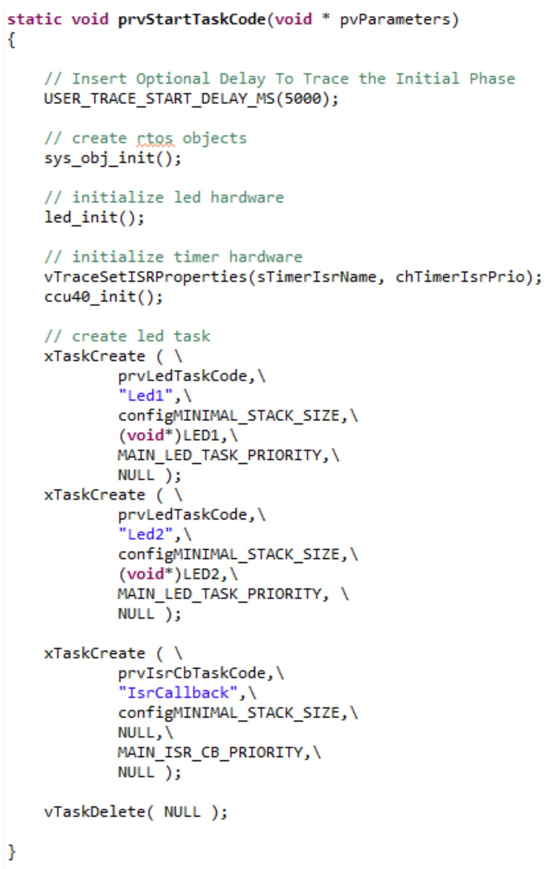
Figure 5: *The main function*

serves the Timer0 interrupt. The embedded application is shown in Figure 6.

2.6. Sequence of Steps to Start the Tracealyzer

The steps of operating the Tracealyzer and the debugger are listed as follows.

- Compile and download the application to XMC4500 using DAVE4
- Set a breakpoint after Trace_Init() or USER_TRACE_INIT()
- Run the application to the breakpint and pause there
- Open Tracealyzer
- Do the configuration on the computer
- Click File/Connect to Target System
- Step the embedded application over the breakpoint in DAVE4
- Keep the debug window alive throughout the whole period of trace
- Click start recording in Tracealyzer
- If needed, stop Recording or start Recording again as long as the debugger session is alive
- Save and view trace data



```c
static void prvStartTaskCode(void * pvParameters)
{

    // Insert Optional Delay To Trace the Initial Phase
    USER_TRACE_START_DELAY_MS(5000);

    // create rtos objects
    sys_obj_init();

    // initialize led hardware
    led_init();

    // initialize timer hardware
    vTraceSetISRProperties(sTimerIsrName, chTimerIsrPrio);
    ccu40_init();

    // create led task
    xTaskCreate ( \
            prvLedTaskCode,\
            "Led1",\
            configMINIMAL_STACK_SIZE,\
            (void*)LED1,\
            MAIN_LED_TASK_PRIORITY,\
            NULL );
    xTaskCreate ( \
            prvLedTaskCode,\
            "Led2",\
            configMINIMAL_STACK_SIZE,\
            (void*)LED2,\
            MAIN_LED_TASK_PRIORITY, \
            NULL );

    xTaskCreate ( \
            prvIsrCbTaskCode,\
            "IsrCallback",\
            configMINIMAL_STACK_SIZE,\
            NULL,\
            MAIN_ISR_CB_PRIORITY,\
            NULL );

    vTaskDelete( NULL );

}
```

Figure 6: *Embedded application in the start hhread*

3. Result

A screenshot of the recording activity is shown in Figure 7.

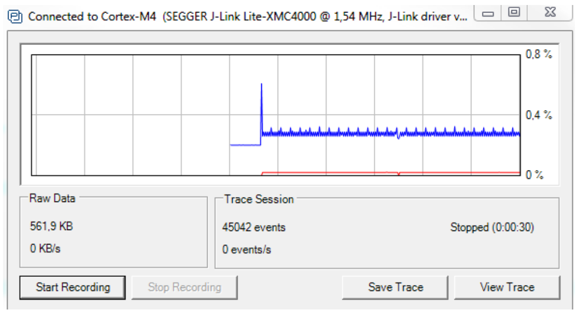


Figure 7: *Recording FreeRTOS events*

The recorded events can be saved and viewed using the "Save Trace" and "View Trace"

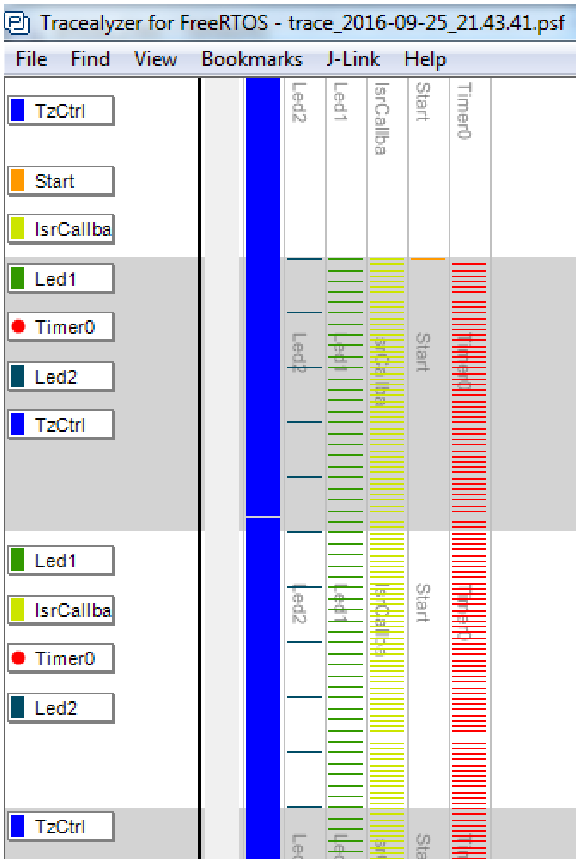


Figure 8: *Viewing FreeRTOS events*

buttons. The four threads and one interrupt are shown in (Figure 8). Note that TzCtrl is the Tracealyzer thread.

4. Summary

In this article we looked at a demonstration to trace FreeRTOS in XMC4500. Some "tricks" are used to make it possible. First of all, the _SEEGER_RTT structure is properly placed in the memory. Secondly, a debugger session is started and maintained before running Tracealyzer. Thirdly, a breakpoint is inserted after calling Trace_Init(). Fourthly, a period of delay is optionally inserted at the beginning of an embedded application. Last of all, a few macros are defined such that the relevant lines of code do not enter into the embedded application if configUSE_TRACE_FACILITY is undefined.