Implementation Guide for TraceSniffer: Frontend

Carl-Jonas Mair
Hochschule Darmstadt

October 6, 2018

Contents

1 Introduction

The Tracer is the code of the TraceSniffer which sits on the hardware, capturing information produced by the FreeRTOS-System. Since we wanted to be as independent from the hardware as we can be, its on the user to implement the communication between the Tracer and the GUI. This allows basically every system which can run FreeRTOS to run the Tracer, ranging from the ATMEGA328P to the ARMv7.
The porting of the Tracer designed as compact as it can be. This will guide you step by step threw the porting.

2 Implementation

1. Download the Tracer from github and save it in your project repository. Include the Tracer-folder into your paths. Now open sniffPort.h, where we will implement all the functionalities we need behind the # define xxx() statements.

2. The most important function is sendByteOverInterface(sendByte), which connects the Tracer with the interface. Preferably you implement a FIFO (see selfFIFO.h) which buffers bytes (given by sendByte) and an interface-interrupt (currently only tested with UART as interface), which empties this FIFO. sendByteOverInterface then writes into the FIFO and enables the interrupt, the interrupt disables himself if the FIFO is empty. A baud-rate of 1000000 is advised, because much data is produced and has to be send.

3. To fully use the TraceSniffer you have to implement the interface in both ways. enableReceiveInterrupt() should do what the name suggests. You can enable the interrupt manually in your main before you start the Tracer, but here you can be sure that the interrupt is activated in time.

4. sendReceivedByteToSniffTask(receivedbyte) ensures the communication from the interface to the Tracer. You dont have to change the definition, but you have to place this function into your receiver-interrupt and feed the received byte as argument at the receivedbyte placeholder.

5. Now only readOutSysTimerHigh() and readOutSysTimerLow() is left to implement. The function should return the timer counter value for the timer used as SysTick-Timer. This allows us to have a better time resolution than our SysTick. If your timer counts to less than 2^{16}, you can return the lowest two bytes of your timer. Otherwise you have to use the highest and second highest byte the timercounter counts up to. Due to this function we have a max resolution of $1/(2^{16}*1000)$seconds = 15ns which beats the 1ms by far.

   TIP: If you just want to have a quick look into your system, only the sendByteOverInterface() is mandatory and in the SysTimer reads you should return 0. But be aware that some features like the ObjectList and the Filter might not work then.

6. Before you can start a measurement, we also have to adapt some configurations for your system. Have a look at the sniffConfig.h:

   Behind every config you can see a comment which tells you what the value means. Most important are the SIZE_OF_X_FIFO defines. If you have a system with enough memory, values (from Stream to Error) from 2000-3000/20-30/100-200 are recommended. If you are experiencing missing packets, increase the Stream-FIFO size further. On systems with less memory, you have to experiment a little bit. Try also to decrease the MAX_LENGTH_OF_OBJECTNAME and MAX_NUMBER _OF_OBJECTS in this case.

   WARNING: Dont change the SEND_PREAMBLE define unless you know how to change it on the pc-side too!

7. Now to activate the Tracer in your code, add # include "sniffImport.h" at the end of your FreeRTOSConfig.h and initialiseTraceSniffer(); in the initialisation code in your main BEFORE any FreeRTOS-Object, like queues and taks, are initialised.

   You are basically set and done. If you have questions, look at the examples or ask us over github.

   Happy Sniffing!

References

[CJM] *TraceSniffer Protocol V1.0*, Carl-Jonas Mair