TraceRecorder Integration Quick Start Guide

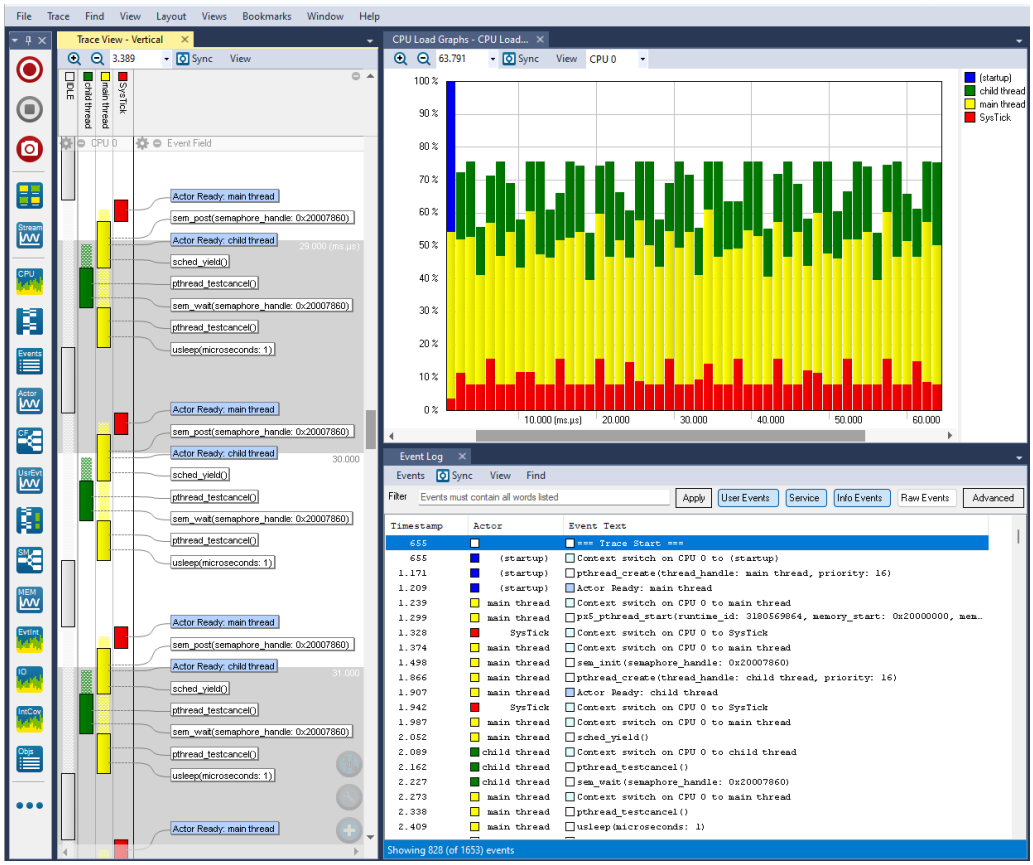
Version 1.0.0

Published November 8, 2023

Overview

The purpose of this document is to explain how to integrate your PX5 RTOS application with Percepio's TraceRecorder library so you can create traces that can be viewed in Percepio's Tracealyzer application.

Functionally, the PX5 RTOS contains trace hooks that are configured to call the TraceRecorder library for a variety of events, including context-switches, API calls, etc. TraceRecorder records these events into a buffer on the target, which is then exported and displayed by Tracealyzer:



The application can also record its own custom user events, which is described more in [Tracing Custom User Events].

Integration Steps

Step 1. Download and install Tracealyzer from https://percepio.com/download/. In the registration form, make sure to select PX5 RTOS as the Target OS. You will receive an email, usually within one minute, with an evaluation license key and the download link. If you do not receive your email, check your spam folder, and then contact support@percepio.com.

Step 2. Clone the PX5-RTOS_TraceRecorder library from GitHub: https://github.com/PX5-RTOS/PX5-RTOS_TraceRecorderSource.

Step 3. Include all C source files from the cloned PX5-RTOS_TraceRecorderSource repository into your existing PX5 RTOS project; there are two folders that include C source files: *PX5-RTOS_TraceRecorderSource* and *PX5-RTOS_TraceRecorderSource/streamports/RingBuffer*.

Step 4. Add the following directories to your compiler's include path:

1. *PX5-RTOS_TraceRecorderSource/config*
2. *PX5-RTOS_TraceRecorderSource/include*
3. *PX5-RTOS_TraceRecorderSource/streamports/RingBuffer/config*
4. *PX5-RTOS_TraceRecorderSource/streamports/RingBuffer/include*

Step 5. In ***PX5-RTOS_TraceRecorderSource/config/trcConfig.h*** set *TRC_CFG_HARDWARE_PORT* to the architecture used. For Cortex-M3/M4/M7 devices, use *TRC_HARDWARE_PORT_ARM_Cortex_M*. For other devices, all available hardware ports can be found at the bottom of ***PX5-RTOS_TraceRecorderSource/include/trcDefines.h***.

Step 6. In ***PX5-RTOS_TraceRecorderSource/config/trcConfig.h***, for some hardware ports an *#include* of the processor's header file needs to be done by replacing the line *#error "Trace Recorder: Please include your processor's header file here and remove this line."* If the error line is removed and your project compiles the header file isn't needed.

Step 7. In ***PX5-RTOS_TraceRecorderSource/config/trcKernelPortConfig.h*** set *TRC_CFG_CPU_CLOCK_HZ* to the frequency used by the timer used for time stamping in the hardware port.

Step 8. In ***px5_user_config.h***, add `#include "trcRecorder.h"`.

Step 9. Define *PX5_THREAD_ENTER_EXIT_NOTIFY_ENABLE* in your assembler's preprocessor settings.

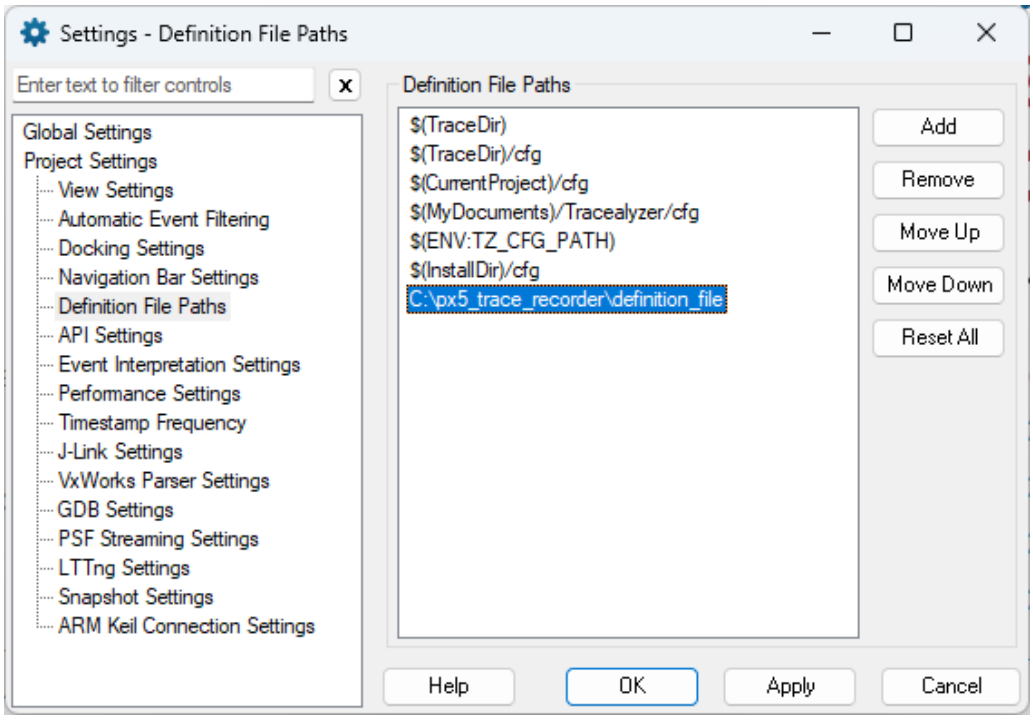
Step 10. In main after *platform_setup* is called (or any time after the timestamp source is setup) and before *px5_pthread_start* is called, add a call to *xTraceEnable* with *TRC_START* as the argument. For example:

```
int main()
{

       /* Call the platform setup function.  */
       platform_setup();

       /* Enable and start tracing.  */
       xTraceEnable(TRC_START);

       /* Start the PX5 RTOS.  */
       px5_pthread_start(...);

       . . .
}
```

Step 11. Build and then run the program for a few seconds to allow the internal trace buffer to fill up with events.

Step 12. Pause the program and export the internal trace buffer to a .hex or .bin file. How the export is done depends on the IDE. The Tracealyzer User Manual provides examples for various IDEs under the “Making snapshots” section - please refer to it.

Step 13. Open Tracealyzer and go to “File -> Settings -> Project Settings -> Definition File Paths” and add the *PX5-RTOS_TraceRecorderSource/definition_file* directory to the list of paths.



Step 14. Open the trace file from the last step in Tracealyzer; you should see something like the image at the start of this guide.

Tracing Custom User Events

Custom user events can be recorded by first registering a User Event Channel– which is used to group related events – and then calling *xTracePrintF* with it. *xTracePrintF* is similar to the standard *printf()* function, however it doesn't support all specifiers. Following is an example usage:

```
/* Declare a string handle for the user event channel.  */
TraceStringHandle_t my_channel;

/* Register the user event channel string.  */
xTraceStringRegister("MyChannel", &my_channel);

/* Record a user event.  */
xTracePrintF(my_channel, "myValue: %d", myValue);
```

In Tracealyzer, user events appear as yellow labels in the trace view.

Tracing ISRs

To trace ISRs, first declare an ISR trace handle:

```
TraceISRHandle_t ISRTraceHandle;
```

Next, call *xTraceISRRegister* to specify the name and priority of the interrupt; this should be done after *xTraceEnable* but before *px5_pthread_start*:

```
#define INTERRUPT_PRIORITY 4

int main()
{

       . . .

       /* Enable and start tracing.  */
       xTraceEnable(TRC_START);

       /* Register the SysTick ISR.  */
       xTraceISRRegister("Interrupt       Name",       INTERRUPT_PRIORITY,
&ISRTraceHandle);

       /* Start the PX5 RTOS.  */
       px5_pthread_start(...);

       . . .
}
```

Next, add calls to *xTraceISRBegin* and *xTraceISREnd*; for the latter, pass PX5_TRACE_CONTEXT_SWITCH_PENDING_CHECK as the argument:

```
void InterruptHandler(void)
{

       /* Record the beginning of interrupt handler.  */
       xTraceISRBegin(ISRTraceHandle);

       . . .

       /* Record end of interrupt handler.  */
       xTraceISREnd(PX5_TRACE_CONTEXT_SWITCH_PENDING_CHECK);
}
```

Enhance • Simplify • Unite

11440 West Bernardo Court • Suite 300
San Diego, CA 92127, USA

Phone: +1 (858) 753-1715
Website: px5rtos.com