Using Tracealyzer with Cypress PSoC devices

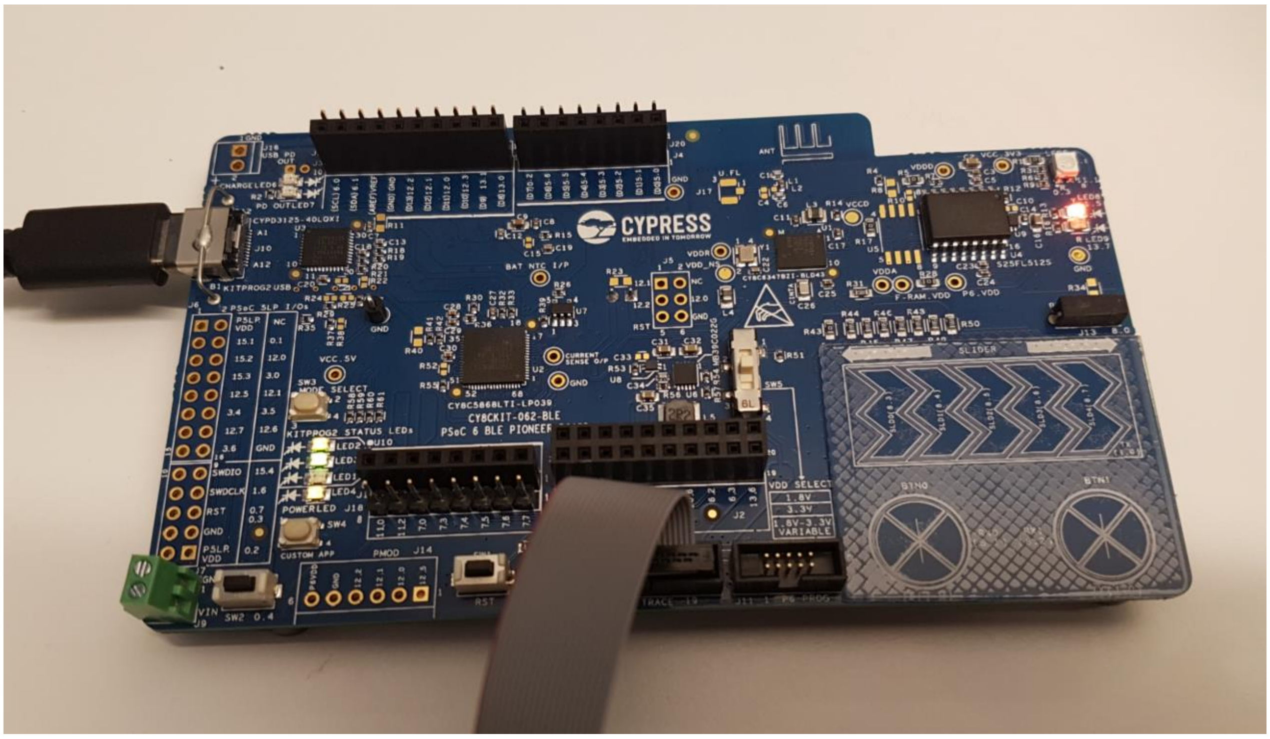
Application Note PA019, 2018-02-14

Introduction

Percepio Tracealyzer lets you see what is going on inside RTOS-based applications during runtime. This document gives an introduction of how to use Tracealyzer with Cypress PSoC MCUs and Cypress PSoC Creator, using a SEGGER J-Link for the data transfer.

The example setup is for the PSoC 6 BLE Pioneer Kit and FreeRTOS, but this is applicable also for PSoC 4 and PSoC 5 devices. All you need is a J-Link and a matching JTAG/SWD debug port on your board.

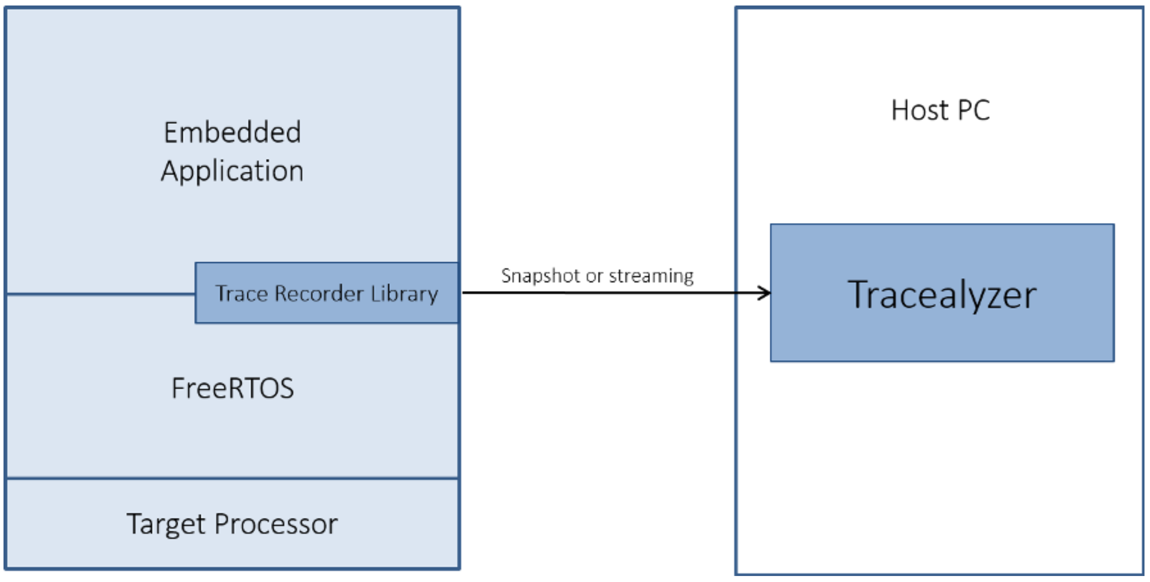
This application note can also be applied for Micrium µC/OS-III and SafeRTOS with only minor changes in how you integrate the trace recorder library into the respective RTOS kernel, as described by the Tracealyzer documentation for each RTOS. Tracealyzer also supports ThreadX, Arm Keil RTX5 and other RTOSes.



This document also describes how to use Tracealyzer to visualize data from the profiler unit in PSoC 6 devices. This unit, named the "Energy Profiler", is more powerful than it sounds, as it monitors the general activity of a variety of PSoC 6 hardware functions. For instance, you can see when your DMA channels are active and when your BLE radio is used. With Tracealyzer, you can see this data in parallel to your software tasks, ISRs, and anything else you choose to log. This way, you get better means to debug your code, and to optimize performance and energy usage of your PSoC application.

Recording Data for Percepio Tracealyzer

Percepio Tracealyzer is intended for 32-bit MCUs running a real-time operating system, such as a PSoC 6 MCU with FreeRTOS. Tracealyzer relies on a trace recorder library (the recorder), that automatically records events from the RTOS kernel, such as context-switching and RTOS API calls. You may also trace additional “User Events” from your application code, similar to a classic “printf” call but much faster. This allows for logging any software event or data of interest, such as the PSoC 6 MCU profiler data, as explained later.



The recorded trace data is visualized by the Tracealyzer host application. The data transfer from the target-side recorder to the host-side Tracealyzer application can be performed in two ways:

- Snapshot mode - keeps the trace data in a target-side RAM buffer, allowing for saving "snapshots" of the trace data at any point. When the buffer is full, you can choose to either stop the tracing or start overwriting older data.
- Streaming mode – transfers the data continuously to the host PC. This way, you can record as long as you have disk space on your PC. Because the amount of data produced by RTOS tracing is moderate, you may record traces spanning several days, or even weeks with a big hard drive.

Getting Started - Step 1: Integrating the recorder with a FreeRTOS project

Tracealyzer supports SEGGER J-Link natively, both for Snapshot and Streaming mode. To integrate the recorder in a FreeRTOS project and configure it for streaming mode, follow these general steps.

1. Download the recorder library from http://percepio.com/docs/FreeRTOS/TraceRecorder.zip
2. Add the three source files to your PSoC creator project (trcKernelPort.c, trcSnapshotRecorder.c and trcStreamingRecorder.c)
3. Copy all header files from the following subdirectories into a suitable project folder; i.e., where your other header files are located. (Or, add these to your include path in PSoC Creator.)
   3.1. TraceRecorder/config
   3.2. TraceRecorder/include
   3.3. TraceRecorder/streamports/Jlink_RTT
4. Make the following changes in *trcConfig.h*
   4.1. Replace the "#error" line in the beginning with #include "project.h".
   4.2. #define TRC_CFG_HARDWARE_PORT TRC_HARDWARE_PORT_ARM_Cortex_M
   4.3. #define TRC_CFG_RECORDER_MODE TRC_RECORDER_MODE_STREAMING
   4.4. Set TRC_CFG_FREERTOS_VERSION to match your FreeRTOS version.
5. Make the following changes to *FreeRTOSConfig.h*
   5.1. #define configUSE_TRACE_FACILITY 1
   5.2. In the end (after the above define), add this line: #include "trcRecorder.h"
6. In your *main* function, call vTraceEnable(TRC_INIT) in the beginning. This must be called before any FreeRTOS calls are made.

After following these six steps, the recorder should now be operational and configured for J-Link streaming.

Getting Started - Step 2: Connecting the SEGGER J-Link

On the Cypress PSoC 6 BLE Pioneer kit, you connect your J-Link to connector P6, using a SEGGER J-Link 9-Pin Cortex-M Adapter. Cypress PSoC Creator does not, however, support debugging via third party debug probes such as SEGGER J-Link (as of v4.2). Moreover, it seems you cannot have a J-Link connected during a PSoC Creator debug session, as the Cypress KitProg debug interface is disabled when you plug in the J-Link. Luckily, there is a simple workaround:

- Program and launch the PSoC Creator project with the J-Link disconnected. (You can have the debug cable connected; just unplug the USB cable from the J-Link device.)
- Connect the J-Link after the project is up and running. (PSoC Creator will then give an error message that the connection has been lost, which is expected.)

You can now connect with Tracealyzer and start recording traces, as explained in Step 3. To switch back to the PSoC Creator debugger, just unplug the J-Link again and launch a new debug session in PSoC Creator. The KitProg connection should now be restored.

Getting Started - Step 3: Setting up the Tracealyzer application

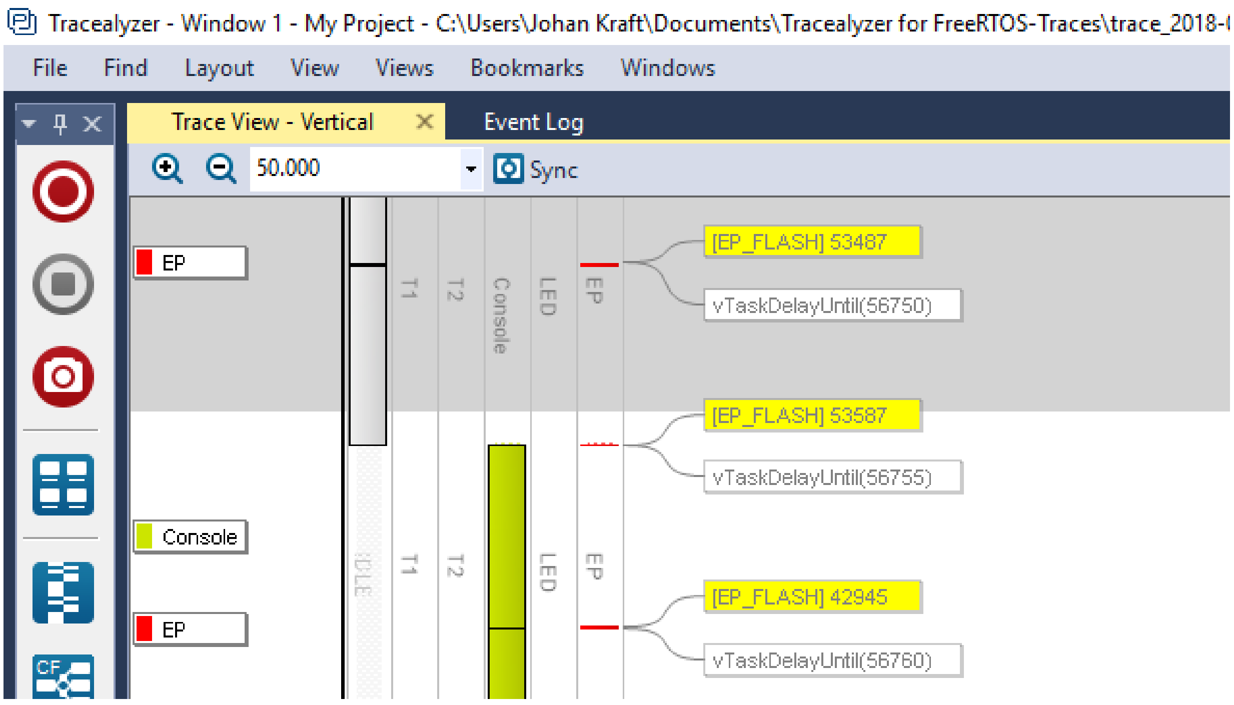
To view the data and control the recording, you need the Tracealyzer application, so the next step is to download and install Tracealyzer from https://percepio.com/downloadform. Make sure you select FreeRTOS when asked about the target operating system.

This is a commercial product, so you will need a license key to record your own traces. To begin with, request an evaluation license by selecting “Evaluate” on the welcome screen. Complete the registration form and the license key will be emailed within minutes. Select “Enter License Key” and then “Online Activation”. Enter the emailed license key.

Next, you need to configure Tracealyzer to receive the trace data from the J-Link. In the Tracealyzer settings (File->Settings). Make sure you have the following setup:

- “J-Link Settings”, make sure to set your target device (for the PSoC 6 BLE Pioneer kit, this is CY8C6xx7_CM4). If your device is not listed, update your J-Link drivers. See below.
- “Streaming Trace” - Select SEGGER RTT as target connection. The other settings there can remain at their default values.

Assuming you use Tracealyzer v4.0 or later, just click the red “Record” button in the left-side navigation bar. To stop the recording, click the black “Stop” button just under it. The views are updated live while recording, but additional features are available after the recording has been stopped.

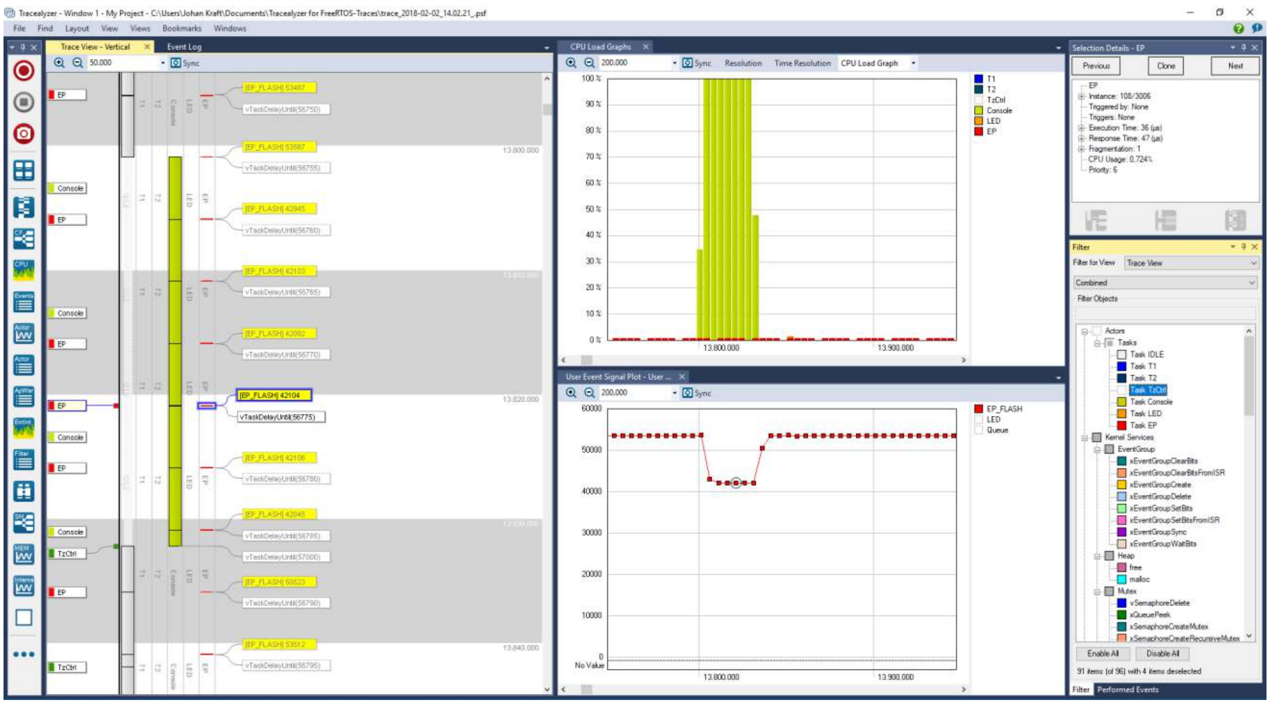


Visualizing data from the PSoC 6 Energy Profiler

The “Energy Profiler” is a monitoring unit in PSoC 6 MCUs, that can be used for energy optimization, but also for general debugging, validation and performance optimization. This is really an “activity monitor” that reports the activity of up to eight hardware functions at the same time, from 27 available sources, such as the two Arm cores, USB, serial communication blocks, and BLE radio.

The meaning of the raw data from the profiler depends on each monitored hardware function. For instance, for the Arm cores, the value reported is the active cycle count, while for USB it is the number of AHB transfers. The profiler data can be used to calculate an energy usage estimate by multiplying the raw values with appropriate coefficients/weights and adding up the weighted values. Moreover, the profiler data also shows the general activity of these hardware functions, which is very valuable for system-level debugging, validation, and optimization.

A code example is provided in Appendix A, where the profiler data is stored as “User Events” for display in Tracealyzer as shown below. Samples of the profiler data are shown as yellow labels in the vertical trace view on the left, together with the FreeRTOS task execution. The profiler data is also shown in the lower right view, on a horizontal timeline, and the upper right view shows CPU load of the FreeRTOS tasks.



The periodic dips in the flash reads (EP_FLASH) obviously are caused by the Console task (the bright green one). The reason is that this task writes to the debug console, which uses polling to check whether the UART transmitter is ready. Because the instructions in the polling loop are cached, fewer instructions are read from flash during that time.

Troubleshooting

Device not listed - updating the J-Link Drivers

If your device is not listed in the J-Link Settings, you need to update the J-Link driver (the list is provided by this driver). For PSoC 6 MCUs, we recommend using J-Link drivers of version 6.30 or later.

Download and install the latest J-Link Software and Documentation Pack from the SEGGER website (https://www.segger.com/). Do this AFTER you have installed Tracealyzer to ensure J-Link drivers in the Tracealyzer directory are also updated.

Technical Support

Percepio Tracealyzer is provided by Percepio AB, based in Västerås, Sweden. For technical support or other questions concerning Tracealyzer, feel free to contact us at support@percepio.com, or by phone +46 21 14 62 10.

Learning More

To learn more about the capabilities of Tracealyzer, please refer to the “Getting Started” page at https://percepio.com/gettingstarted/, that offers introduction videos, articles and other material.

For more detailed information about how to use Tracealyzer for a specific RTOS, begin by visiting https://percepio.com/tracealyzer/ to ensure your RTOS it is supported by Tracealyzer. Then download Tracealyzer (specify the correct RTOS when asked) and refer to the provided user manual (“Help” menu).

See also the PSoC-specific blog posts about Tracealyzer at iotexpert.com, https://iotexpert.com/?s=tracealyzer

Appendix A - Code Example

Use the function Cy_Profile_ConfigureCounter to define what to measure (CPUSS_MONITOR_FLASH, defined in psoc63_config.h). These functions are documented in cy_profile.h and full documentation is found in C:\Program Files (x86)\Cypress\PDL\3.0.1\doc\pdl_api_reference_manual.html.

```
#include "project.h"

#define WEIGHT_NORM  1ul /* Coefficient, use 1 to get raw value */

/* pointer to profile counter configuration */
cy_stc_profile_ctr_ptr_t flash_ctr = NULL;

void my_ProfileInit(void)
{
    Cy_Profile_Init();

    Cy_Profile_ClearConfiguration();

    /* Initialize counter for Flash (read count) */
    flash_ctr = Cy_Profile_ConfigureCounter(
        CPUSS_MONITOR_FLASH,
        CY_PROFILE_EVENT,
        CY_PROFILE_CLK_HF,
        WEIGHT_NORM);

    if (Cy_Profile_EnableCounter(flash_ctrs) != CY_PROFILE_SUCCESS)
    {
        /* Profiler setup failed - insert error handling here*/
    }

    /* Start the Profiler */
    Cy_Profile_StartProfiling();
}
```

To read the data and then display the results in Percepio Tracealyzer, you may use the following code example. Call this function whenever you like to sample the profiler value; e.g., in a periodic RTOS task.

```
traceString tzUserEventChn = NULL;

void mySampleProfiler(void)
{
    uint64_t sample;
    int delta;
    static uint64_t previous_sample = 0;

    if (tzUserEventChn == NULL)
        tzUserEventChn = xTraceRegisterString("EP_FLASH");

    Cy_Profile_GetRawCount(flash_ctr, &sample);
    delta = sample - previous_sample;
    vTracePrintF(tzUserEventChn,"%d", delta);
    previous_sample = sample;
}
```

The actual reading of the profiler data is done by calling Cy_Profile_GetRawCount, with the flash_ctr handle returned by Cy_Profile_ConfigureCounter. Because the profiler counter is free running in this case, we calculate the difference since the last sample, representing the amount of activity over this period.

The result is stored using vTracePrintF, producing a “User Event” consisting of a format string and associated data, displayed in Tracealyzer as shown in the previous screenshot.