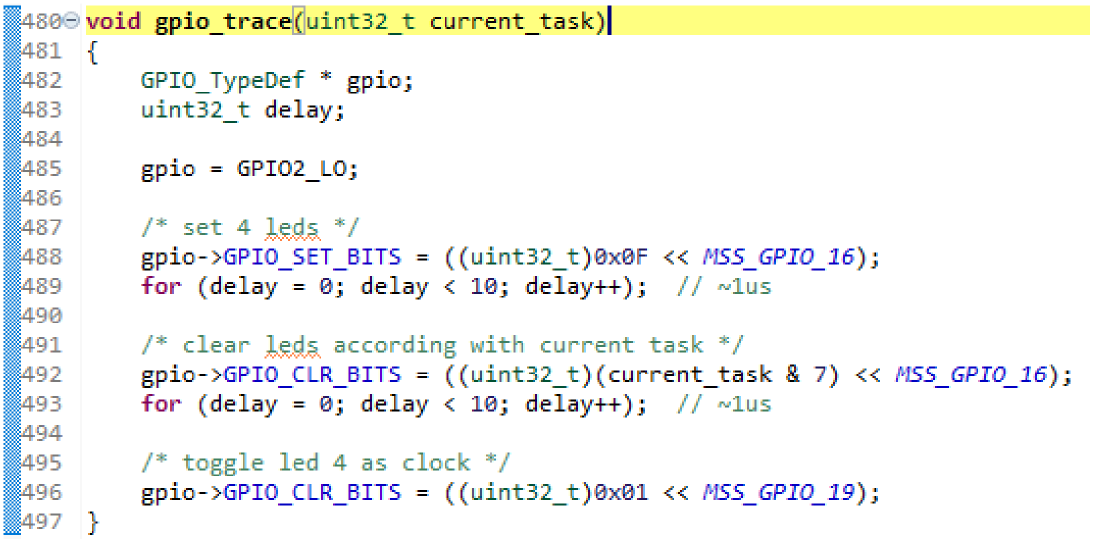
Utilizing TRACE32 Mixed-Signal Probe to Trace Task Switches in Real-Time Operating Systems (RTOS)

2024-12-13 - Comments (0) - Trace

Today, nearly every application runs on an operating system. To effectively analyze the timing behavior of the OS, it is crucial to trace the currently running task. Unfortunately, the tracing infrastructure of the core architecture sometimes lacks both data tracing and context ID tracing. In other cases, the available on-chip trace is so small that a thorough analysis of timing behavior is not possible.

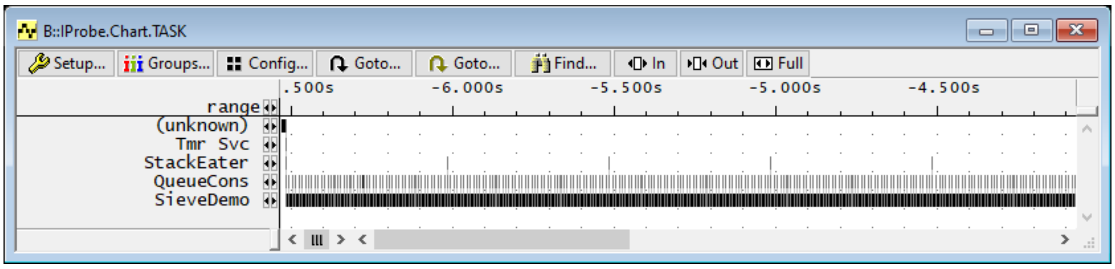
For use cases like these, where long-term task runtime analysis is required, you can leverage GPIO signals available on the target board along with a TRACE32 Mixed-Signal Probe. For example, a solution has been developed for the PolarFire SoC (RISC-V, 64-bit) running FreeRTOS on a single U54 hart. While the PolarFire SoC offers on-chip and off-chip program flow trace, it provides neither data trace nor context ID tracing. FPGA-based SoCs typically have many GPIOs, which makes implementing such a solution easier.

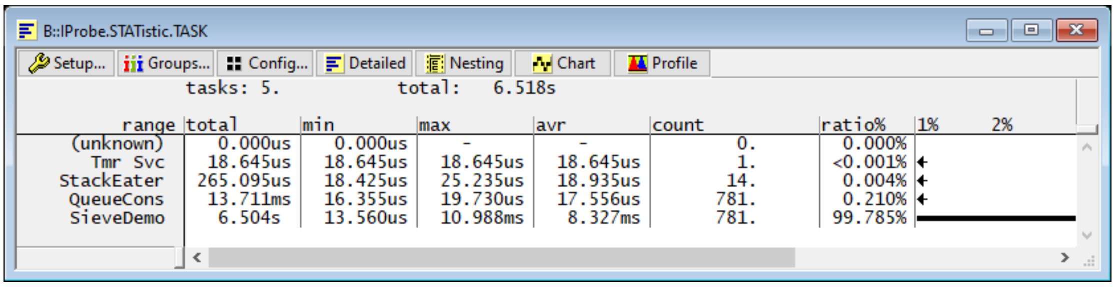
The basic idea is to associate certain GPIO switches with the task switch operations of an RTOS. In this case, the FreeRTOS task switch hook function is utilized to add code that writes the task number to specific GPIOs.



```
void gpio_trace(uint32_t current_task)
{
    GPIO_TypeDef * gpio;
    uint32_t delay;

    gpio = GPIO2_LO;

    /* set 4 leds */
    gpio->GPIO_SET_BITS = ((uint32_t)0x0F << MSS_GPIO_16);
    for (delay = 0; delay < 10; delay++);  // ~1us

    /* clear leds according with current task */
    gpio->GPIO_CLR_BITS = ((uint32_t)(current_task & 7) << MSS_GPIO_16);
    for (delay = 0; delay < 10; delay++);  // ~1us

    /* toggle led 4 as clock */
    gpio->GPIO_CLR_BITS = ((uint32_t)0x01 << MSS_GPIO_19);
}
```

The GPIO signals can be recorded using a Lauterbach logic analyzer (Mixed-Signal Probe). A `DisConfig` command in TRACE32 allows the recorded GPIO signals to be mapped to the task number. To understand the following screenshots, it's important to know that the signals recorded by the Mixed-Signal Probe are displayed in TRACE32 using the TRACE32 `CIProbe` or `IProbe` main command groups. The commands `IProbe.Chart.TASK` and `IProbe.STATistic.TASK` enable now the desired long-term task runtime analysis.



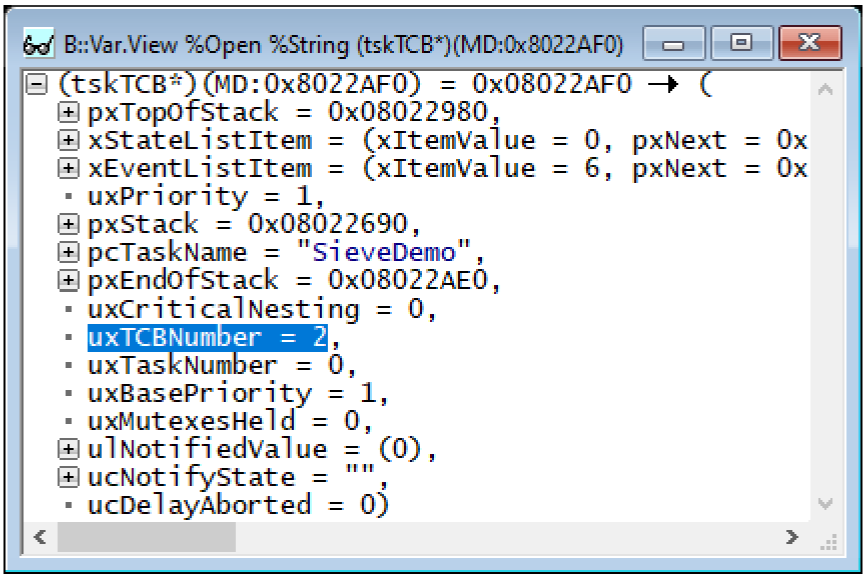


```
          tasks: 5.          total:    6.518s

     range  total      min        max        avr        count        ratio%  1%      2%
 (unknown)     0.000us    0.000us          -          -          0.     0.000%
   Tmr Svc    18.645us   18.645us   18.645us   18.645us          1.    <0.001%
StackEater   265.095us   18.425us   25.235us   18.935us         14.     0.004%
 QueueCons    13.711ms   16.355us   19.730us   17.556us        781.     0.210%
 SieveDemo     6.504s    13.560us   10.988ms    8.327ms        781.    99.785%
```

Now, for those interested in a deeper understanding of the solution, here are some additional details.

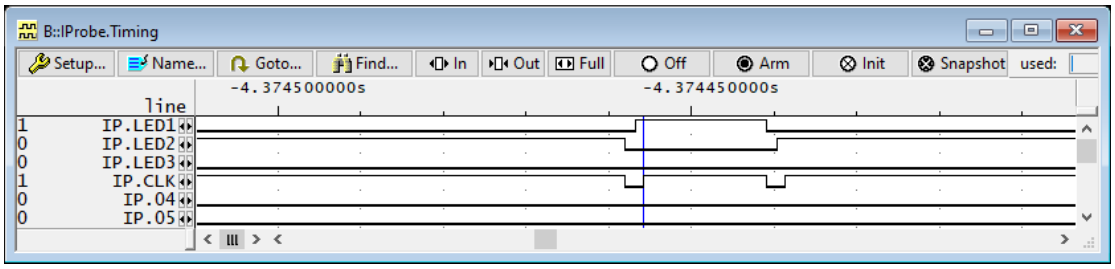
FreeRTOS provides a "task num" for each created task. It's a progressive number starting from 1.

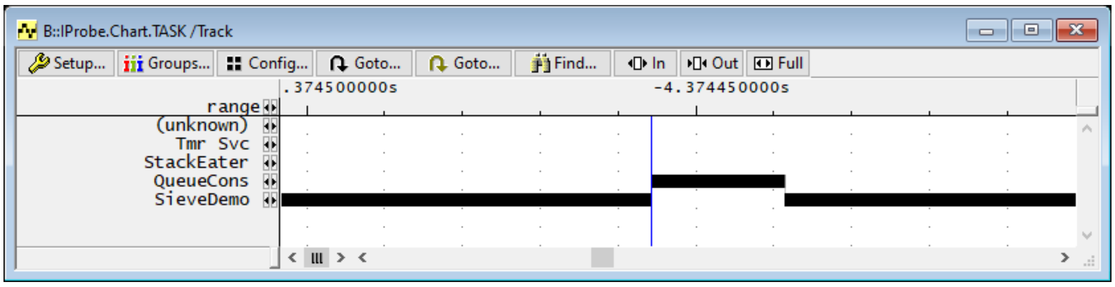
```
    #if ( configUSE_TRACE_FACILITY == 1 )
        UBaseType_t uxTCBNumber;  /*< Stores a number that increments each time a TCB is created.
                                   *  It allows debuggers to determine when a task has been deleted
                                   *  and then recreated. */
        UBaseType_t uxTaskNumber; /*< Stores a number specifically for use by third party trace code. */
    #endif
```



```
(tskTCB*)(MD:0x8022AF0) = 0x08022AF0 → (
  pxTopOfStack = 0x08022980,
  xStateListItem = (xItemValue = 0, pxNext = 0x
  xEventListItem = (xItemValue = 6, pxNext = 0x
  uxPriority = 1,
  pxStack = 0x08022690,
  pcTaskName = "SieveDemo",
  pxEndOfStack = 0x08022AE0,
  uxCriticalNesting = 0,
  uxTCBNumber = 2,
  uxTaskNumber = 0,
  uxBasePriority = 1,
  uxMutexesHeld = 0,
  ulNotifiedValue = (0),
  ucNotifyState = "",
  ucDelayAborted = 0)
```

The "task num" is encoded on several GPIOs during each task switch. In this implementation, three data GPIOs are used (allowing for up to seven available tasks) along with one clock GPIO to sample valid data on the Mixed-Signal Probe. This configuration effectively functions like a bus trace with a single clock line.

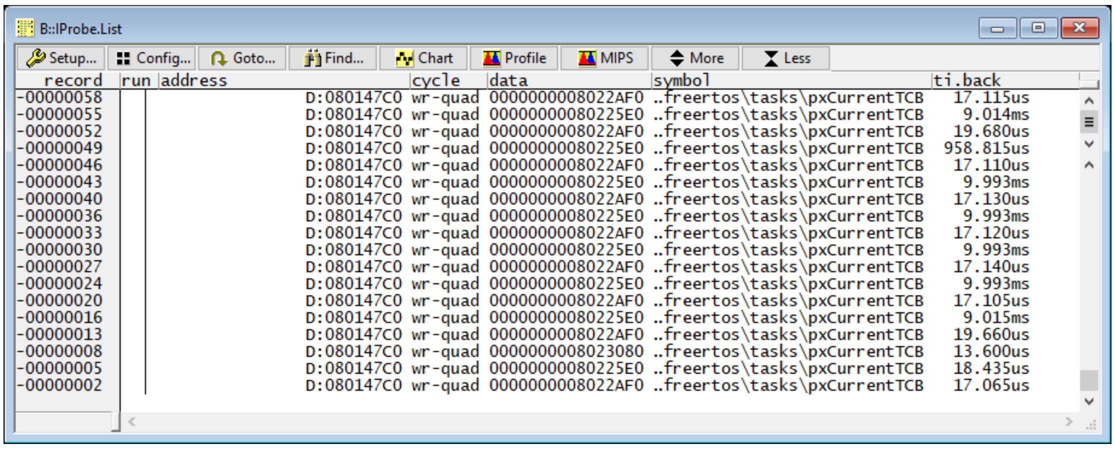




Several `CIProbe/IProbe.DisConfig.CYcle` commands are configured to detect write cycles, along with the appropriate 'Strobe', 'Address', and 'Data' fields:

- The Address is the task number
- The values from the GPIOs are used as the Strobe.
- The Data is encoded to retrieve the original FreeRTOS task number based on the "task num" encoded on the GPIOs.

```
PRIVATE &task_magic
&task_magic=TASK.CONFIG(magic)

; TASK2 is SieveDemo
NAME.Word W.TASK2 IP.04 LOW LOW LOW HIGH HIGH HIGH HIGH LOW HIGH LOW HIGH LOW HIGH LOW LOW LOW
HIGH LOW LOW LOW LOW LOW LOW LOW LOW LOW HIGH LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW
LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW LOW

PRIVATE &strobetask2
&strobetask2="Strobe IP.LED1 LOW Strobe IP.LED2 HIGH Strobe IP.LED3 LOW"
IProbe.DisConfig.CYcle "wr-quad,2" Write Strobe IP.CLK Rising &strobetask2 Address LOW AddressBase
&task_magic Data W.TASK2
```

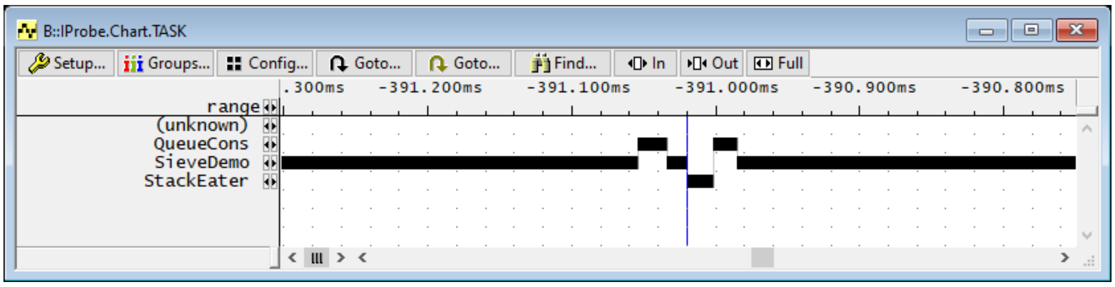


```
 record   run address                   cycle   data             symbol                        ti.back
-00000058             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.115us
-00000055             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.014ms
-00000052             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  19.680us
-00000049             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB 958.815us
-00000046             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.110us
-00000043             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.993ms
-00000040             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.130us
-00000036             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.993ms
-00000033             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.120us
-00000030             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.993ms
-00000027             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.140us
-00000024             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.993ms
-00000020             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.105us
-00000016             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB   9.015ms
-00000013             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  19.660us
-00000008             D:080147C0 wr-quad 0000000008023080 ..freertos\tasks\pxCurrentTCB  13.600us
-00000005             D:080147C0 wr-quad 00000000080225E0 ..freertos\tasks\pxCurrentTCB  18.435us
-00000002             D:080147C0 wr-quad 0000000008022AF0 ..freertos\tasks\pxCurrentTCB  17.065us
```

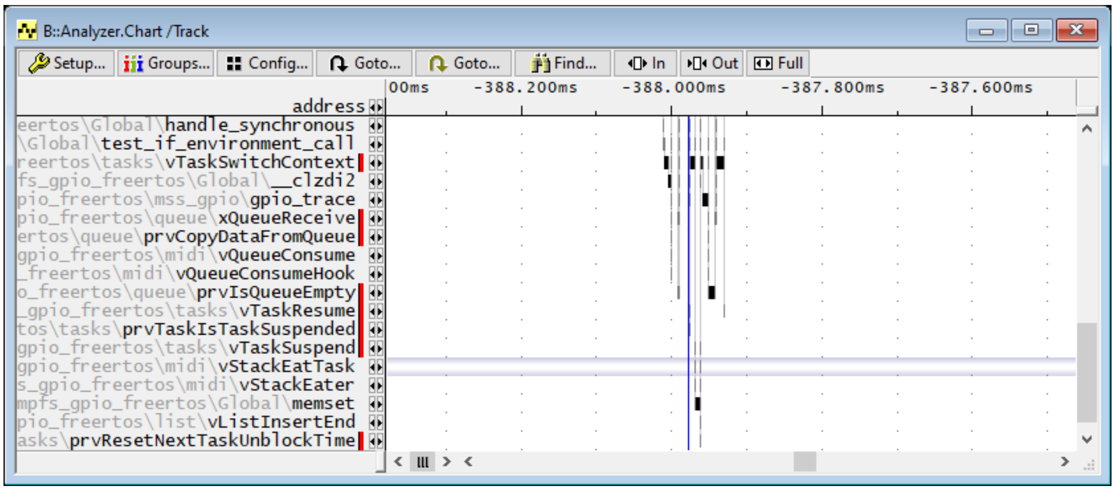
Please note that more than one `DisConfig` command is required—specifically, one command for each task. Each `DisConfig` command uses a distinct strobe to identify a specific task.

The information obtained from `CIProbe/IProbe.List` allows the standard FreeRTOS awareness to generate the `CIProbe/IProbe.Chart.TASK` view. This approach is designed for a single-core RTOS.

Additionally, the PolarFire SoC generates a program flow trace. Since both the PowerTrace (Analyzer), which records the program flow, and the Mixed-Signal Probe (IProbe/CIProbe), which captures the GPIOs, tag their trace records with the same - across all TRACE32 trace tools - timestamp, these recordings can be easily synchronized. The `Analyzer.Chart` Window below shows which functions are executed when the change from task SieveDemo to task StackEater occurred (blue cursor in both windows).



```
                 .300ms   -391.200ms   -391.100ms   -391.000ms   -390.900ms   -390.800ms
       range
   (unknown)
   QueueCons
  SieveDemo
 StackEater
```



```
                                          00ms   -388.200ms   -388.000ms   -387.800ms   -387.600ms
                                  address
eertos\Global\handle_synchronous
\Global\test_if_environment_call
reertos\tasks\vTaskSwitchContext
fs_gpio_freertos\Global\__clzdi2
pio_freertos\mss_gpio\gpio_trace
pio_freertos\queue\xQueueReceive
ertos\queue\prvCopyDataFromQueue
gpio_freertos\midi\vQueueConsume
_freertos\midi\vQueueConsumeHook
o_freertos\queue\prvIsQueueEmpty
_gpio_freertos\tasks\vTaskResume
tos\tasks\prvTaskIsTaskSuspended
gpio_freertos\tasks\vTaskSuspend
gpio_freertos\midi\vStackEatTask
s_gpio_freertos\midi\vStackEater
mpfs_gpio_freertos\Global\memset
pio_freertos\list\vListInsertEnd
asks\prvResetNextTaskUnblockTime
```