Painless Multithreading:

How to Verify RTOS Best Practices in Runtime

Dr. Johan Kraft

Embedded Online Conference

percepio®

www.embeddedonlineconference.com

AGENDA

1. Real-time operating systems
2. Runtime monitoring and visual trace diagnostics
3. Best Practices - Example 1
4. Best Practices - Example 2
5. Best Practices - Example 3
6. Summary

© EmbeddedOnlineConference.com All rights reserved

THE SPEAKER

Dr. Johan Kraft

CEO, CTO and founder, Percepio AB

Focus: visual trace diagnostics for simplified embedded software development

Original developer of Percepio's first product for visual trace diagnostics, Tracealyzer, and the founder of the company. Background in applied academic research in collaboration with industry, focused on embedded software timing analysis, and embedded software development at ABB Robotics. PhD in computer science.

© EmbeddedOnlineConference.com All rights reserved

REAL-TIME OPERATING SYSTEMS (RTOS)

A software platform for your embedded application code

Provides multithreading

- Tasks – Separate execution threads (sub programs)
- Supporting services – Semaphores, Queues, Timers, etc.

Allows for modularizing your system into multiple smaller programs

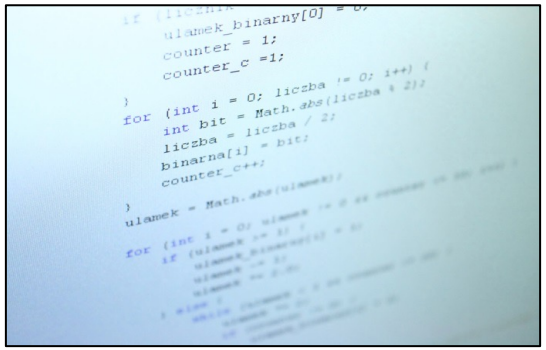
- With a single responsibility each, e.g. for USB, Display, TCP/IP, Bluetooth, …
- Easier to understand and maintain the code.
- More efficient code – if done correctly.

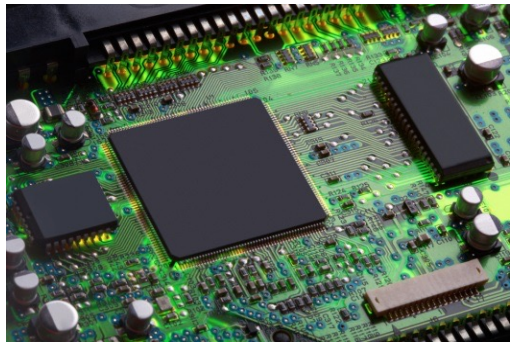
Multithreading bring new challenges in software design, debugging and testing.

- Performance? Reliability? Testability?
- Important to follow best practices in RTOS application design

© EmbeddedOnlineConference.com All rights reserved

THE SOURCE CODE IS NOT THE FULL PICTURE



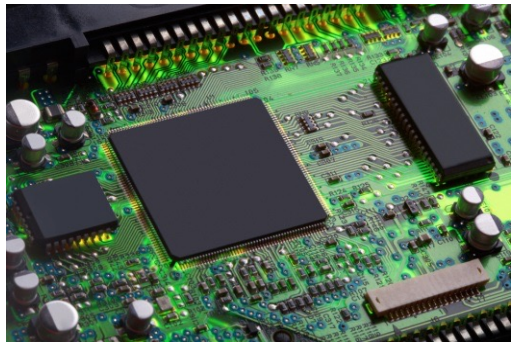


For multithreaded systems, the runtime behavior is affected by interference between tasks and timing variations.

Emergent properties, not visible in the source code, only in runtime!

© EmbeddedOnlineConference.com All rights reserved

RUNTIME MONITORING



```
184140        000100fc -> write r31
184141  100a8   IncrementCounterBy1  add          %r0,
184141        0000002e -> write r0
184142  100ac   IncrementCounterBy1+0x04  extb
184142        0000002e -> write r0
184143  100b0   IncrementCounterBy1+0x08  j_s
184144  100fc   main+0x30       stb          %r0,[%r1]
184144             2e -> write mem [0x011000]
184145  10100   main+0x34       ldb          %r0,[%r2]
184145             5a <- read  mem [0x011001] -> writ
184146        0000005a -> write r0
184147  10104   main+0x38       bl_s         IncrementC
184147        00010106 -> write r31
184148  100b4   IncrementCounterBy2  add          %r0,
184148        0000005c -> write r0
184149  100b8   IncrementCounterBy2+0x04  extb
184149        0000005c -> write r0
184150  100bc   IncrementCounterBy2+0x08  j_s
```

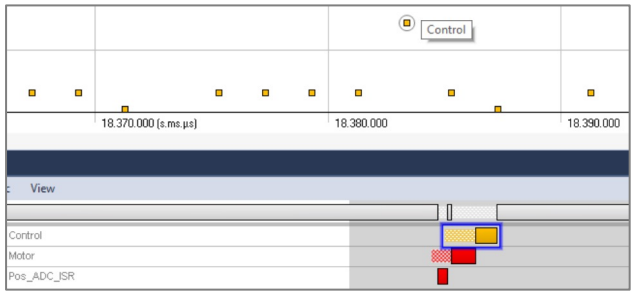
```
9.330]  xQueueReceive(CtrlDataQueue, 100) retu
0.253]  OS Ticks: 8109
1.253]  OS Ticks: 8110
1.270]  Context switch on CPU 0 to Pos_ADC_ISR
1.281]  xQueueSendFromISR(CtrlDataQueue)
1.290]  Context switch on CPU 0 to Control
1.868]  xQueueSend(MotorQueue)
1.878]  Actor Ready: Motor
1.889]  Context switch on CPU 0 to Motor
1.900]  xQueueReceive(MotorQueue, 10) returns
1.934]  xQueueReceive(MotorQueue, 10) blocks
1.954]  Context switch on CPU 0 to Control
1.965]  xQueueReceive(CtrlCmdQueue, 0) timeout
1.977]  xQueueReceive(CtrlDataQueue, 100)
1.990]  xQueueReceive(CtrlCmdQueue, 0) timeout
```

```
vc] Starting key provisioning...
vc] Write root certificate...
vc] Write device private key...
Svc] Write device certificate...
Svc] Key provisioning done...
Svc] Starting WiFi...
r Svc] WiFi module initialized.
S-MAIN] WiFi connected to AP AndroidAP.
WS-MAIN] Attempt to Get IP.
WS-MAIN] IP Address acquired 192.168.0.51
WS-LED] [Shadow 0] MQTT: Creation of dedicated MQ
WS-LED] Sending command to MQTT task.
QTT] Received message 10000 from queue.
QTT] Looked up a7sw0r7rvpirn.iot.us-east-1.amazona
MQTT] MQTT Connect was accepted. Connection estab
MQTT] Notifying task.
AWS-LED] Command sent to MQTT task passed.
```

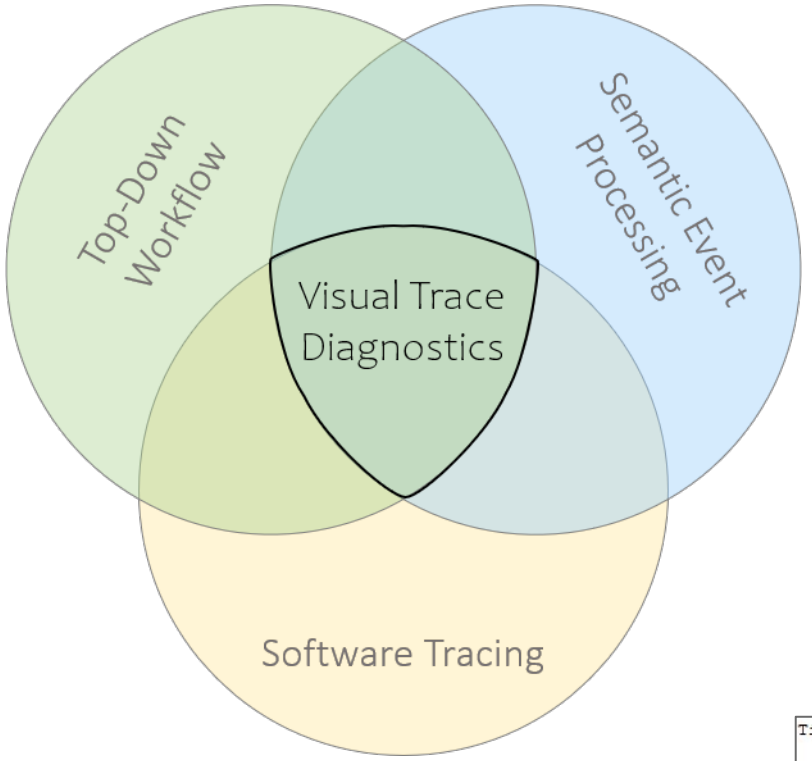
	Instruction Trace	Software Event Trace	Application Logging
Producer	Processor core	Software	Software
Abstraction Level	Low	Medium to High	High
Overhead	None	Low	Medium to High
Special HW needed	Yes	No	No

© EmbeddedOnlineConference.com All rights reserved

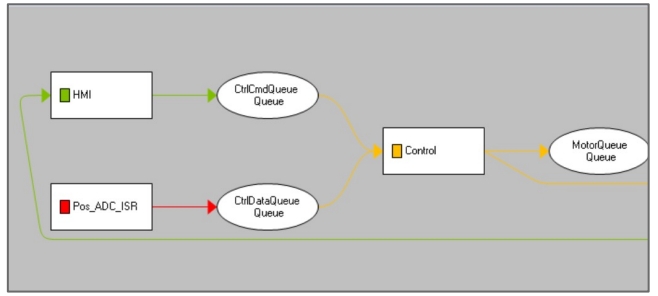
VISUAL TRACE DIAGNOSTICS



Visualization allowing for drill-down from overviews to details



Data collection

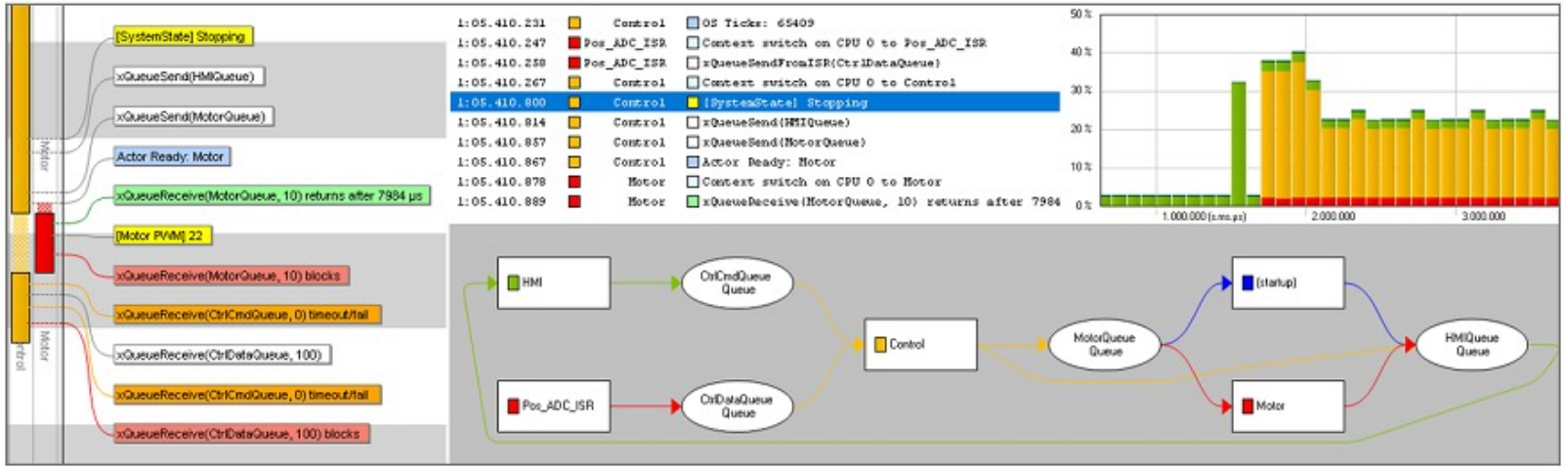


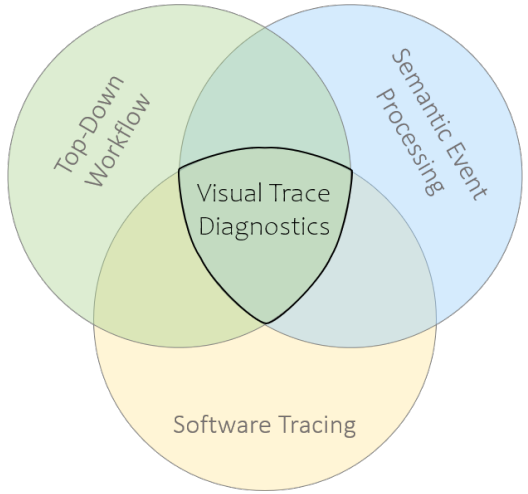
Data processed into a meaningful model connecting related events and objects

Timestamp	Actor	Event Text
1.305	Control	xQueueReceive(CtrlDataQueue, 100) blocks
1.325	HMI	Context switch on CPU 0 to HMI
1.333	HMI	vTaskDelayUntil(500)
1.349	TzCtrl	Context switch on CPU 0 to TzCtrl
1.426	TzCtrl	Unused Stack for TzCtrl: 81
1.433	TzCtrl	vTaskDelay(20)
1.448	IDLE	Context switch on CPU 0 to IDLE

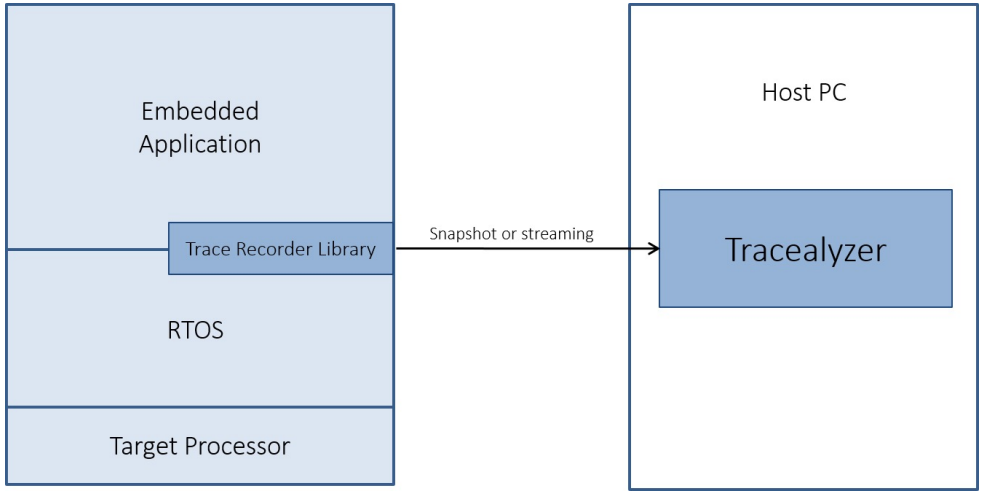
© EmbeddedOnlineConference.com All rights reserved

PERCEPIO TRACEALYZER





- Debugging on System Level
- Measuring Software Metrics
- Analyzing Software Design



© EmbeddedOnlineConference.com All rights reserved

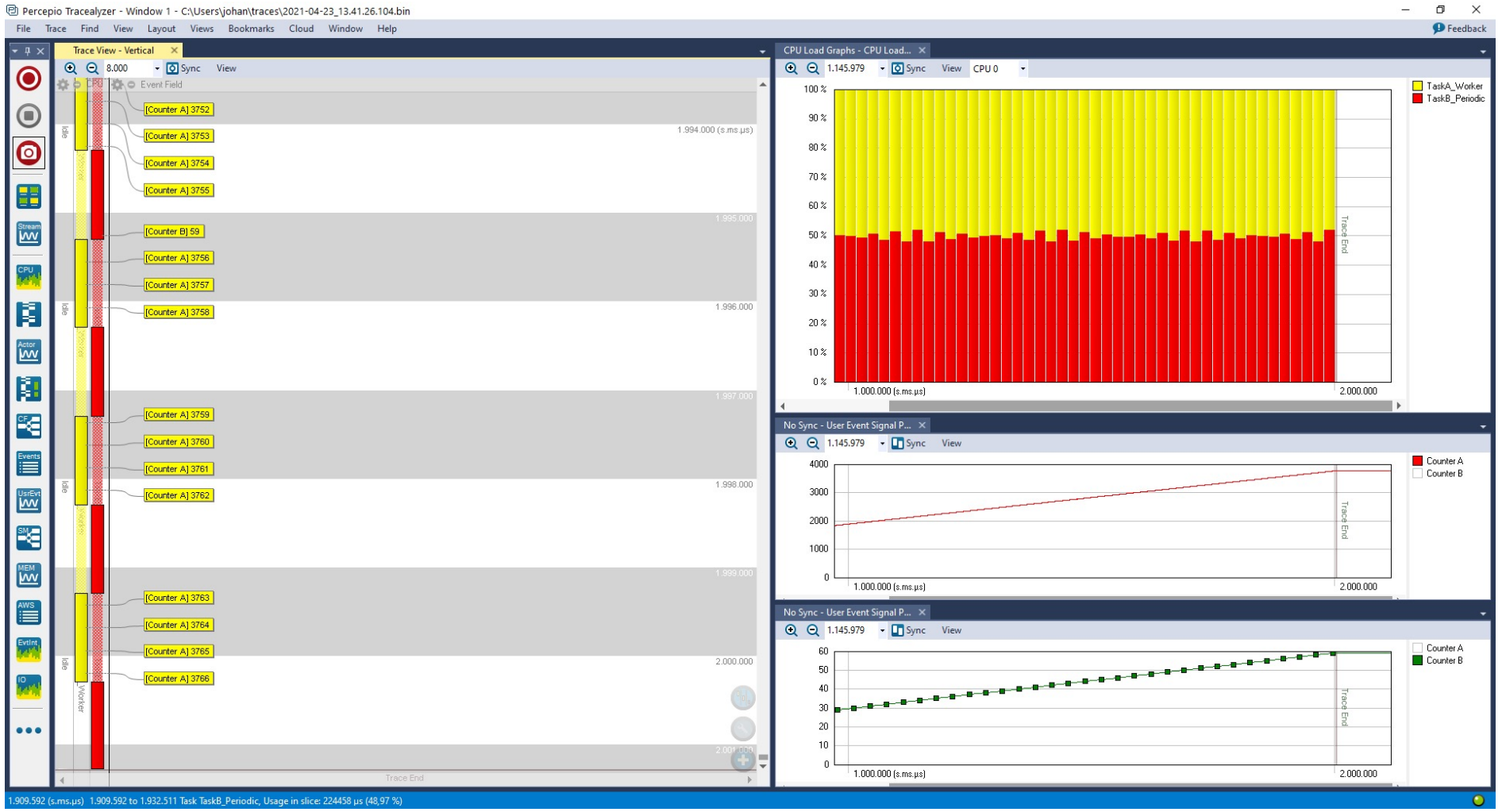
EXAMPLE 1: BASIC RTOS TASKS

```c
void InitDesign(void)
{
    /* User event channel, used for logging CounterA */
    chA = xTraceRegisterString("Counter A");

    /* User event channel, used for logging CounterB */
    chB = xTraceRegisterString("Counter B");

    xTaskCreate(TaskA_Worker, "TaskA_Worker", 1000, NULL, 1, NULL);

    xTaskCreate(TaskB_Periodic, "TaskB_Periodic", 1000, NULL, 1, NULL);
}


void TaskA_Worker(void *argument)
{
  for(;;)
  {
      DoSomethingA(); /* Incrementing and logging Counter A */
  }
}

void TaskB_Periodic(void *argument)
{
  for(;;)
  {
      if (CheckSomething())
      {
          DoSomethingB(); /* Incrementing and logging Counter B */
      }
      HAL_Delay(10);
  }
}
```

© EmbeddedOnlineConference.com All rights reserved

THE RESULTING TRACE

TaskB uses 50% of the processor time
Where is the 10 ms delay?



Counter: 3766
("amount of work")

© EmbeddedOnlineConference.com All rights reserved

THE EXPLAINATION

```
void TaskB_Periodic(void *argument)
{
  for(;;)
  {
      if (CheckSomething())
      {
          DoSomethingB(); /* Incrementing and logging Counter B */
      }
      HAL_Delay(10);
  }
}
```

Busy waiting! Doesn't suspend the RTOS task.

© EmbeddedOnlineConference.com All rights reserved

A BETTER SOLUTION

```c
void InitDesign(void)
{
    /* User event channel, used for logging CounterA */
    chA = xTraceRegisterString("Counter A");

    /* User event channel, used for logging CounterB */
    chB = xTraceRegisterString("Counter B");

    xTaskCreate(TaskA_Worker, "TaskA_Worker", 1000, NULL, 1, NULL);

    xTaskCreate(TaskB_Periodic, "TaskB_Periodic", 1000, NULL, 2, NULL); /* HIGHER PRIORITY */
}

void TaskA_Worker(void *argument)
{
  for(;;)
  {
      DoSomethingA();
  }
}

void TaskB_Periodic(void *argument)
{
  for(;;)
  {
      if (CheckSomething())
      {
          DoSomethingB();
      }
      vTaskDelay(10); /* NO BUSY WAIT */
  }
}
```

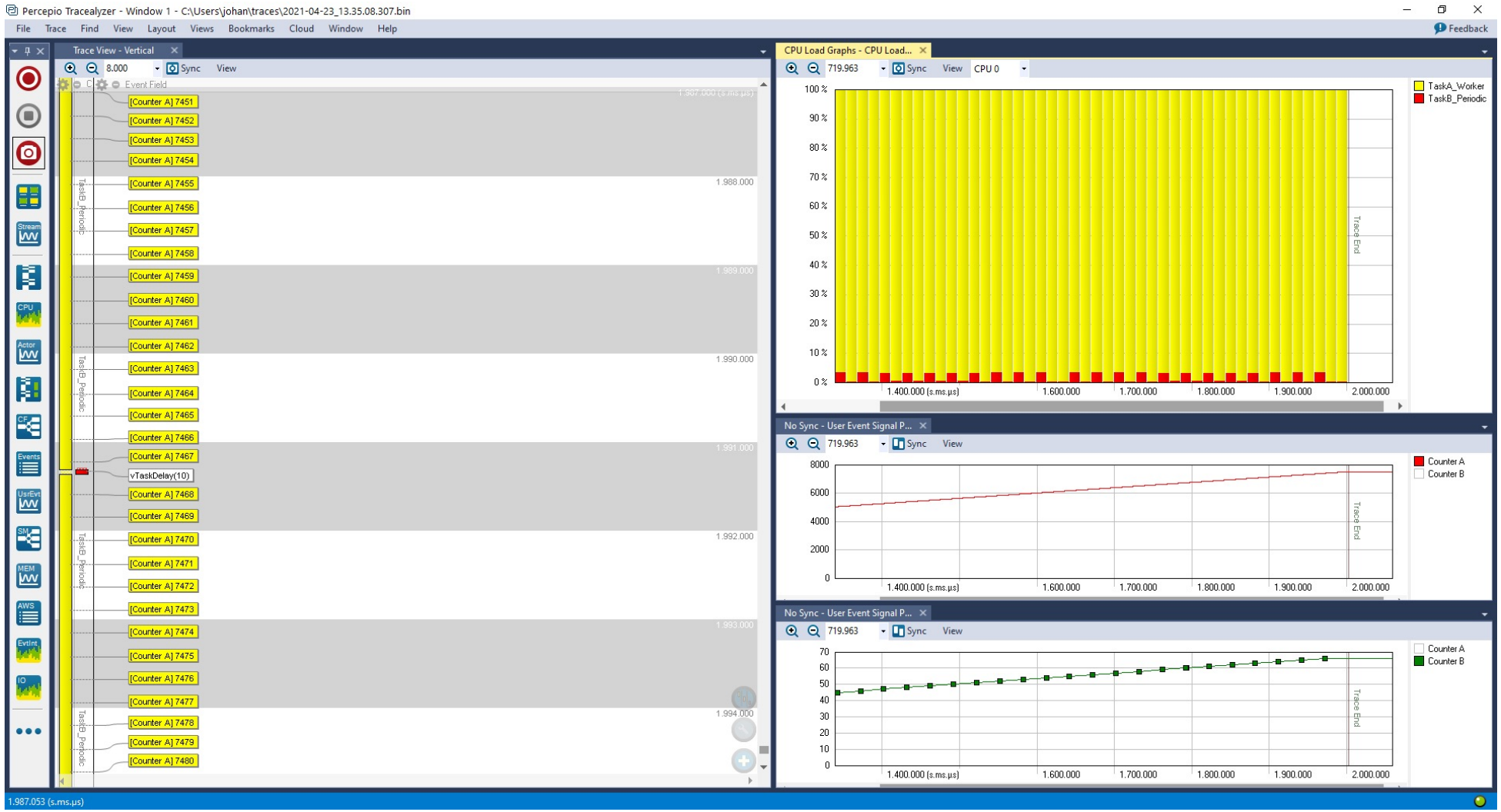
TaskB has highest priority -> may preeempt TaskA.
Runs directly when activated, after the delay.

TaskB suspends itself, allowing TaskA to run.

© EmbeddedOnlineConference.com All rights reserved

THE NEW RESULT

TaskA gets nearly 100% of the processor time, runs twice as fast
TaskB runs on higher priority, every 10 ms, preempting TaskA



Counter: 7497
(instead of 3766)

© EmbeddedOnlineConference.com All rights reserved

SUMMARY, EXAMPLE 1

What did we learn?

- Avoid busy waiting. Prevents other tasks from running.
- Use an RTOS function to suspend the tasks after each job.
- Use scheduling priorities to allow urgent tasks to respond quickly.

© EmbeddedOnlineConference.com All rights reserved

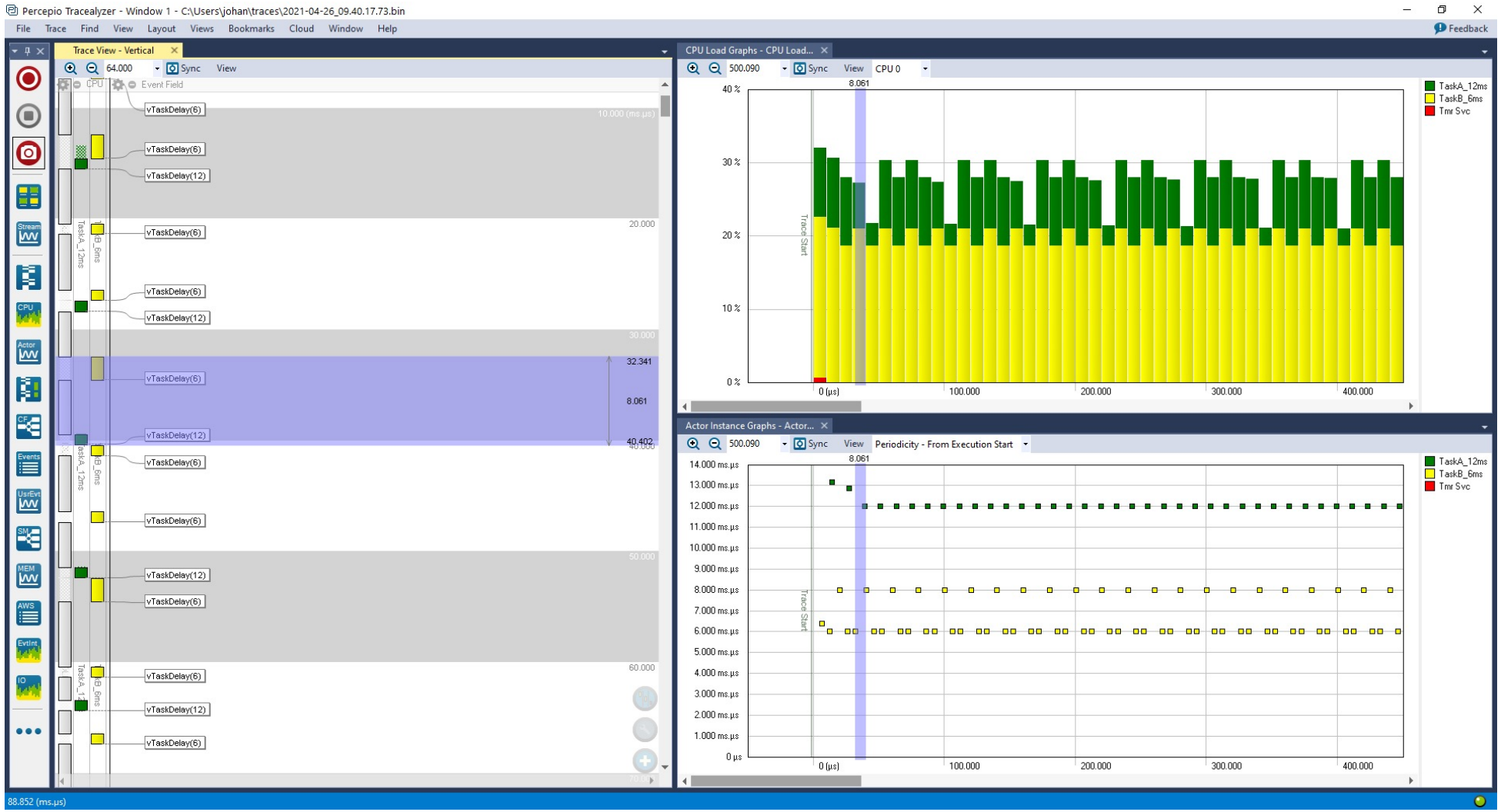
EXAMPLE 2: TWO PERIODIC TASKS

```
void InitDesign(void)
{
    int status;

    status = xTaskCreate(TaskA_12ms, "TaskA_12ms", 1000, NULL, 1, NULL);
    if (status != pdPASS)
    {
        for(;;);
    }

    status = xTaskCreate(TaskB_6ms, "TaskB_6ms", 1000, NULL, 2, NULL);
    if (status != pdPASS)
    {
        for(;;);
    }
}

void TaskA_12ms(void *argument)
{
  for(;;)
  {
      DoSomethingA();
      vTaskDelay(12);
  }
}

void TaskB_6ms(void *argument)
{
  for(;;)
  {
      DoSomethingB();

      if (CheckSomething())
      {
          DoSomethingC();
      }

      vTaskDelay(6);
  }
}
```

© EmbeddedOnlineConference.com All rights reserved

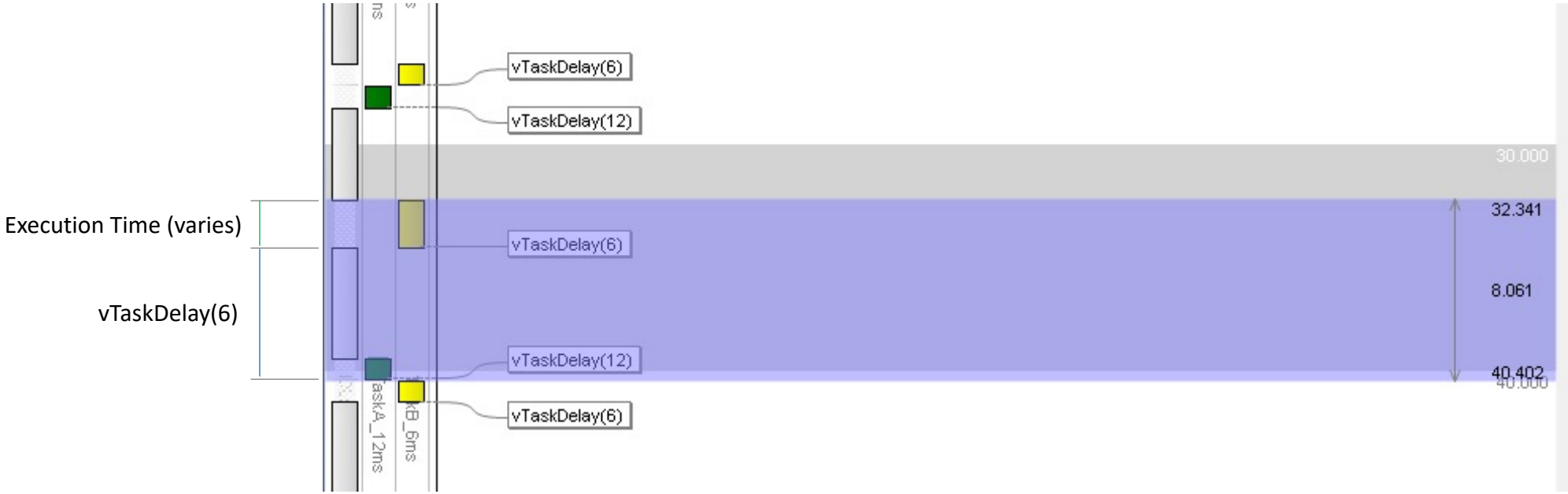
BUT NOT REALLY PERIODIC…

TaskB periodicity varies between 6-8 ms



© EmbeddedOnlineConference.com All rights reserved

WHAT'S THE PROBLEM?



vTaskDelay is relative to the current time, when called.
Does not account for the time elapsed since the last activation, such as execution time.

© EmbeddedOnlineConference.com All rights reserved

SOLUTION: YET ANOTHER DELAY FUNCTION?

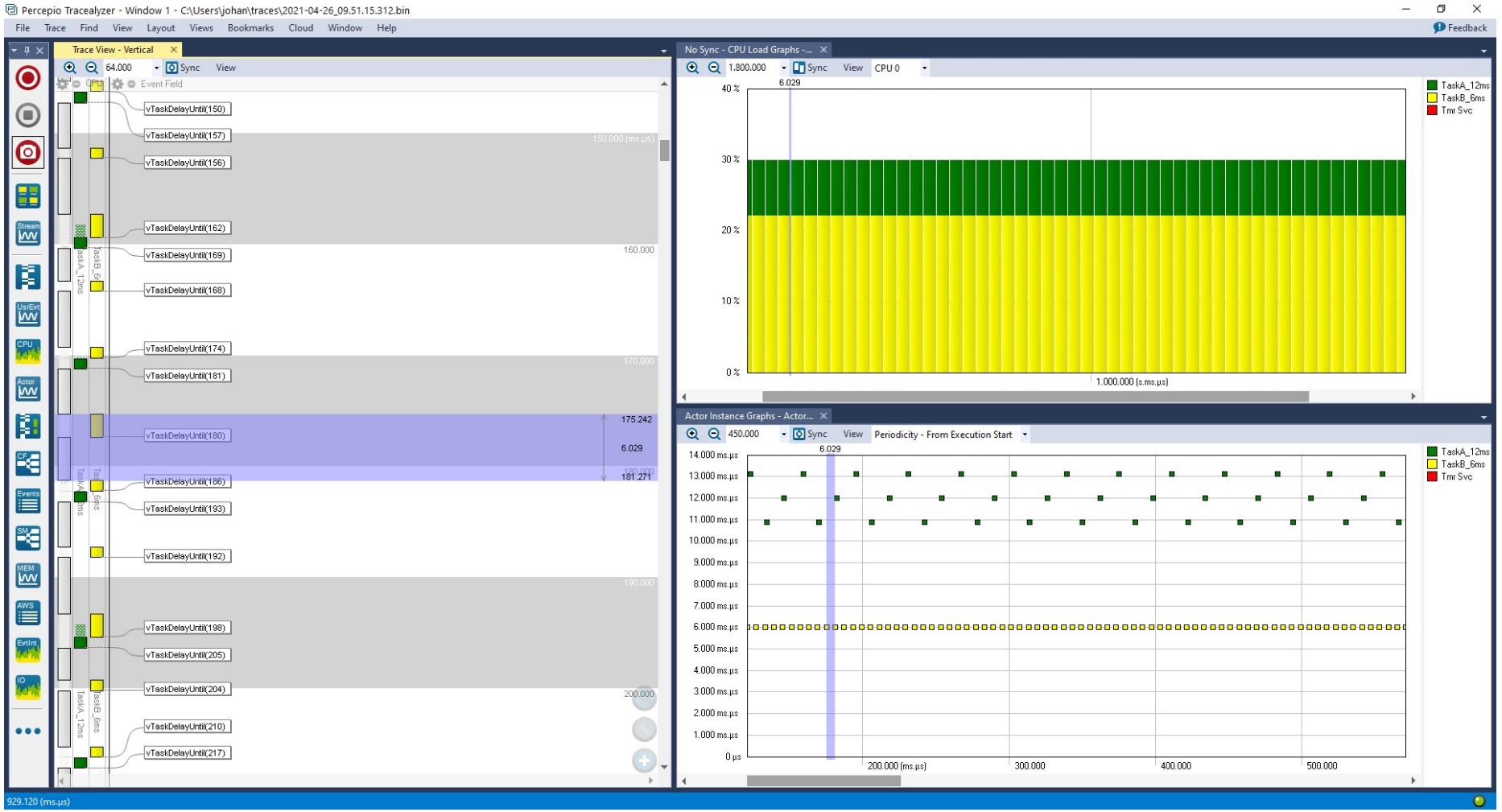
```
void TaskA_12ms(void *argument)
{
    TickType_t xLastWakeTime = xTaskGetTickCount();

    for(;;)
    {
        DoSomethingA();
        vTaskDelayUntil(&xLastWakeTime, 12); // Using vTaskDelayUntil instead of vTaskDelay
    }
}

void TaskB_6ms(void *argument)
{
  TickType_t xLastWakeTime = xTaskGetTickCount();

  for(;;)
  {
      DoSomethingB();

      if (CheckSomething())
      {
          DoSomethingC();
      }

      vTaskDelayUntil(&xLastWakeTime, 6); // Using vTaskDelayUntil instead of vTaskDelay
  }
}
```

vTaskDelayUntil is relative to the last activation, regardless of execution time.

© EmbeddedOnlineConference.com All rights reserved

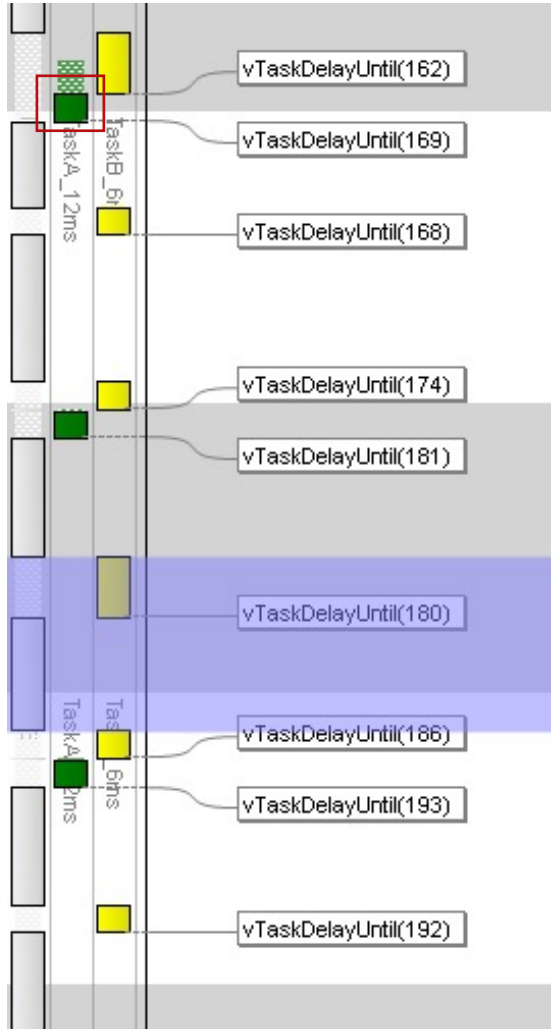
EFFECT

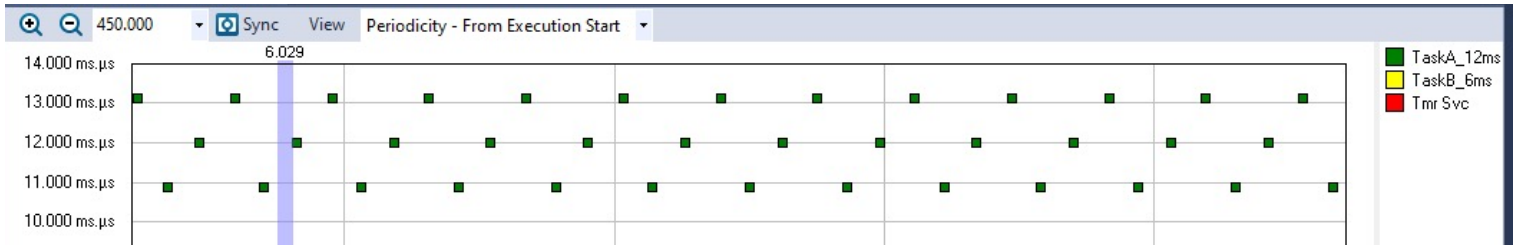
TaskB now perfectly periodic at 6 ms
But TaskA varies between 11-13 ms…



© EmbeddedOnlineConference.com All rights reserved

WHY?





By using vTaskDelayUntil (instead of vTaskDelay) we synchronize the task execution. Both tasks are always activated every 6 or 12 ms.
Good!

But TaskB has variations in the execution time since higher scheduling priority, the timing variation also affects TaskA.
Bad!

Local timing variations should not propagate to other tasks!

© EmbeddedOnlineConference.com All rights reserved

SOFTWARE TIMING VARIATIONS

Why important? Tasks are independent, right?

Tasks may need to access global resources, e.g. data structures or HW interfaces.

The order of global events DOES matter.

Variations in access patterns may cause race conditions and sporadic errors.

There are best practices for protecting shared resources (e.g. mutexes)
but **BUGS DON'T FOLLOW THE RULES!**

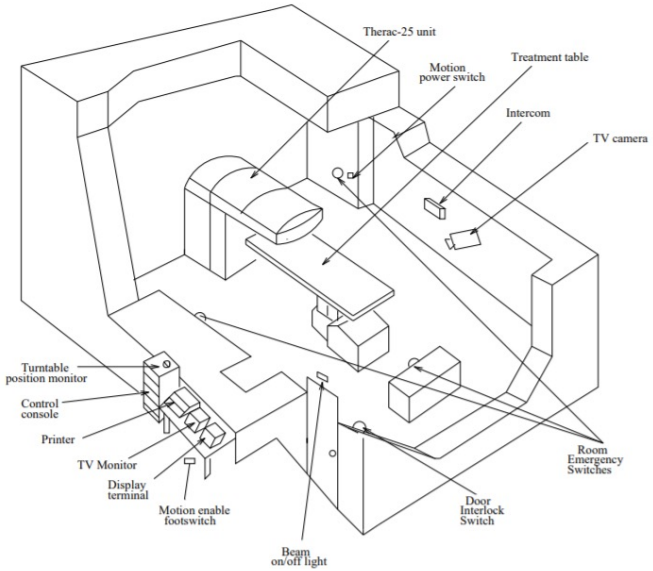
© EmbeddedOnlineConference.com All rights reserved

VARIABILTY VS. RELIABILITY

Latent bugs may always exist…

May depend on execution patterns, depending on context-switch timing, depending on execution time variations…

Less variability ->
fewer patterns ->
better testability ->
improved reliability



Therac-25 radiation therapy machine worked fine most of the time, but killed 4 people due to latent bugs

© EmbeddedOnlineConference.com All rights reserved

FIXING THE PROBLEM

We want to avoid timing variations in high-priority tasks, since propagating to lower priority tasks.

Perhaps move the conditional code in TaskB to a separate low-priority task?

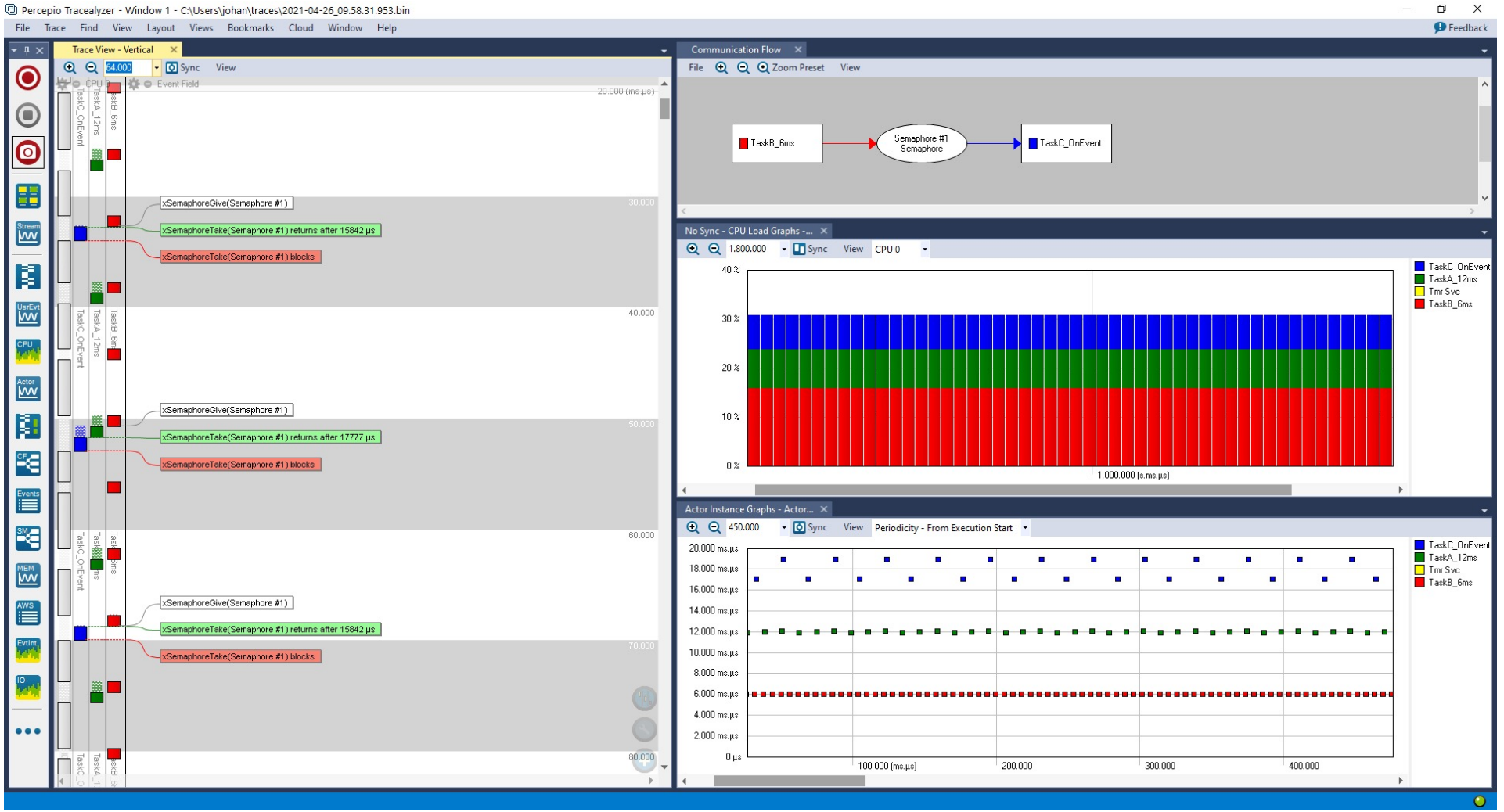
TaskB triggers TaskC using a semaphore.

TaskC runs only when A and B are dormant.

```
void TaskA_12ms(void *argument)
{
    TickType_t xLastWakeTime = xTaskGetTickCount();

    for(;;)
    {
        DoSomethingA();
        vTaskDelayUntil(&xLastWakeTime, 12);
    }
}

void TaskB_6ms(void *argument)
{
  TickType_t xLastWakeTime = xTaskGetTickCount();

  for(;;)
  {
      DoSomethingB();

      if (CheckSomething())
      {
         xSemaphoreGive(semaphore); // Triggering TaskC_OnEvent
      }

      vTaskDelayUntil(&xLastWakeTime, 6);
  }
}

// Runs DoSomethingC in a separate task, on lower scheduling priority
void TaskC_OnEvent(void *argument)
{
    for(;;)
    {
        if( xSemaphoreTake( semaphore, portMAX_DELAY ) == pdTRUE )
        {
            DoSomethingC();
        }
    }
}
```

© EmbeddedOnlineConference.com All rights reserved

FINAL RESULT



© EmbeddedOnlineConference.com All rights reserved

SUMMARY, EXAMPLE 2

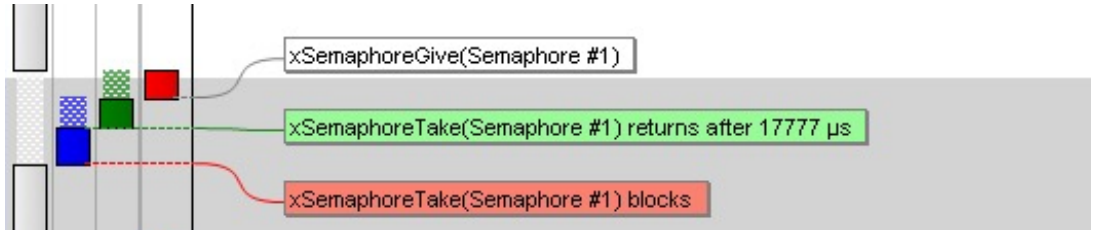
What did we learn?

- Two types of RTOS delay functions – vTaskDelay vs vTaskDelayUntil
- Avoid timing variations, especially in high-priority tasks
- Separating jobs on multiple tasks, with suitable scheduling priorities.

© EmbeddedOnlineConference.com All rights reserved

EXAMPLE 3: SEMAPHORES

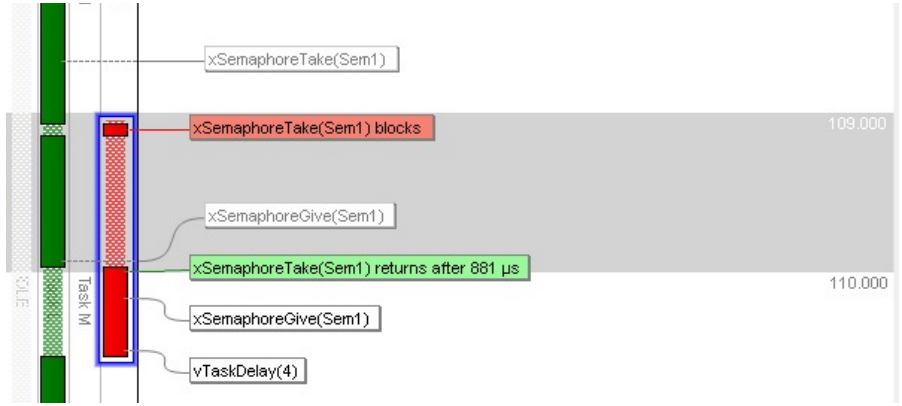
Used as “signals” between tasks




```
      if (CheckSomething())
      {
         xSemaphoreGive(semaphore); // Triggering TaskC_OnEvent
      }

      vTaskDelayUntil(&xLastWakeTime, 6);
  }
}

// Runs DoSomethingC in a separate task, on lower scheduling priority
void TaskC_OnEvent(void *argument)
{
    for(;;)
    {
        if( xSemaphoreTake( semaphore, portMAX_DELAY ) == pdTRUE )
        {
            DoSomethingC();
```

For mutual exclusion?

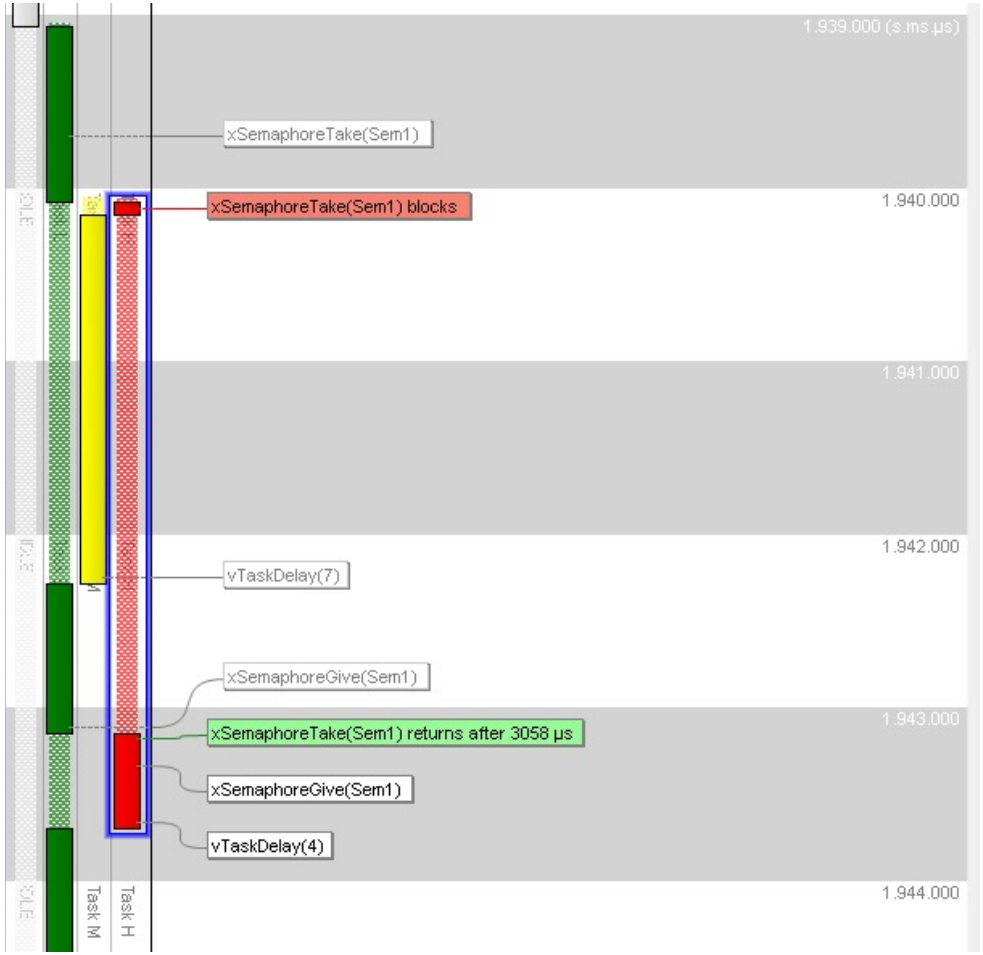
To protect critical sections…




```
for( ;; )
{
    if( xSemaphoreTake( Sem1, 100 ) == pdTRUE )
    {

        /* Taken the semaphore, entered the critical section */
        DoSomethingA();

        xSemaphoreGive(Sem1);
    }
    else
    {
        /* Timeout - error handling needed */
```

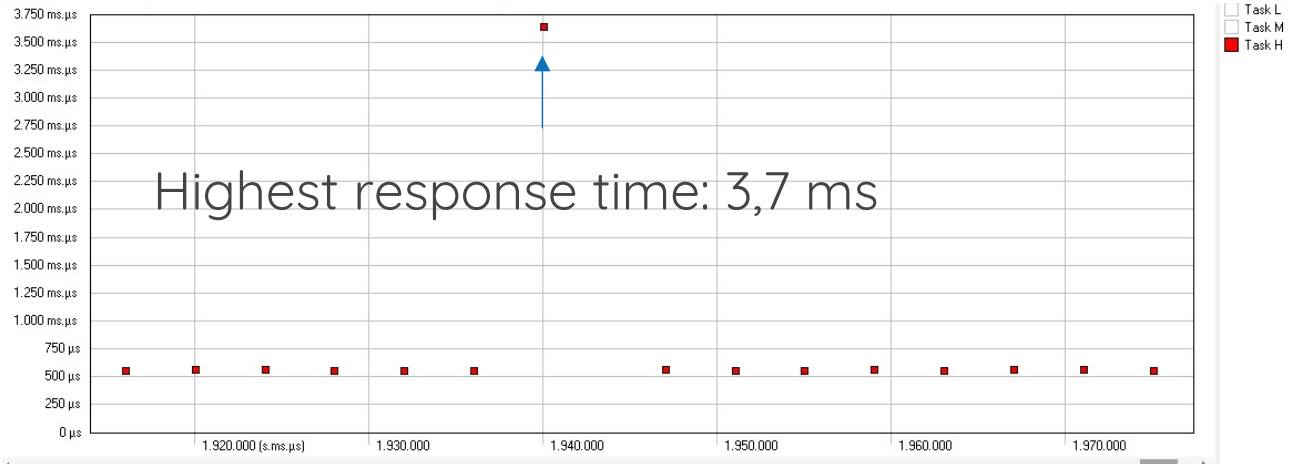
© EmbeddedOnlineConference.com All rights reserved

SEMAPHORES MAY CAUSE PRIORITY INVERSION



High-priority task delayed by lower-priority task

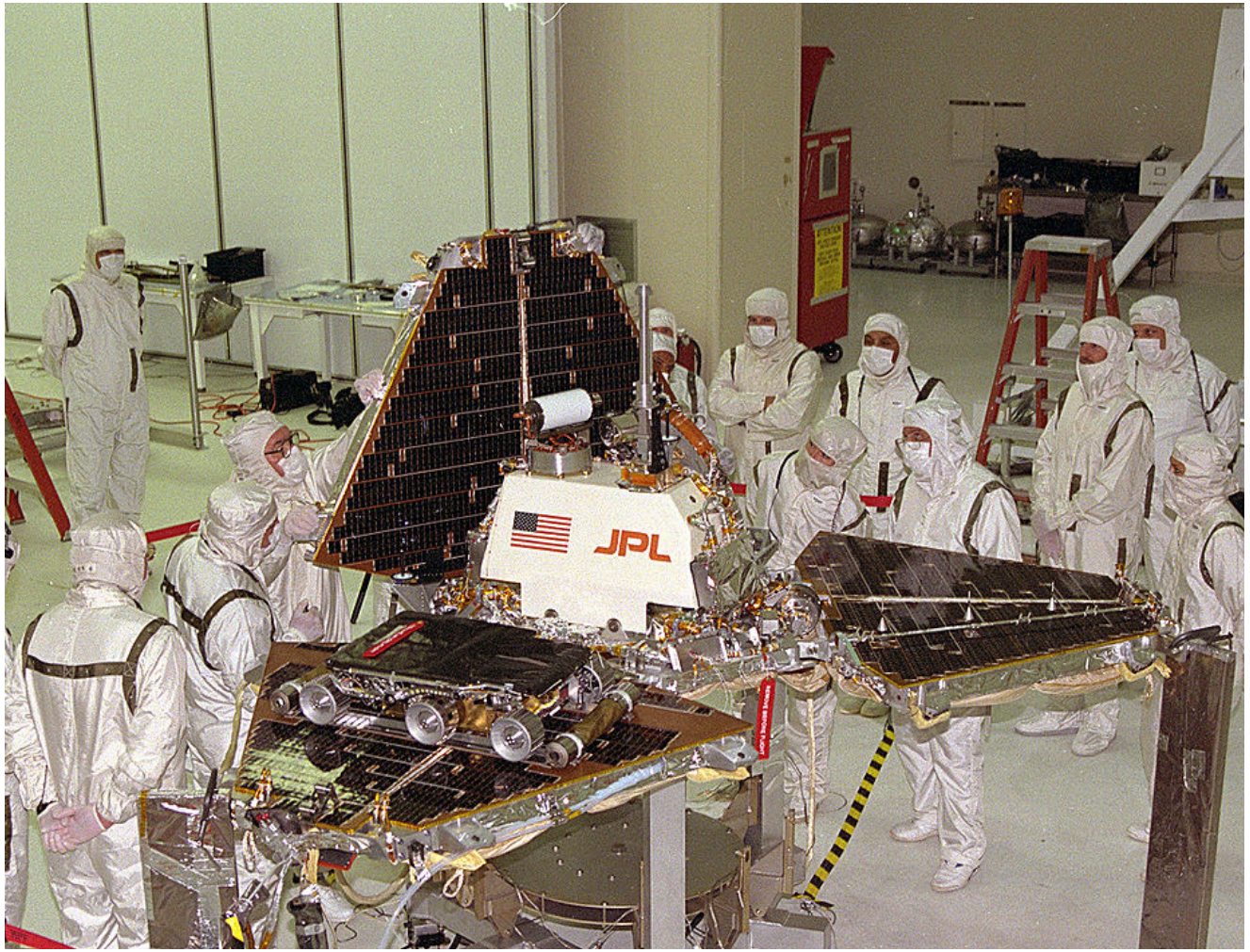
- TaskL takes the semaphore.
- TaskH tries to take it but is blocked. Waiting...
- TaskM preempts TaskL, delaying TaskL and TaskH



© EmbeddedOnlineConference.com All rights reserved

THE PATHFINDER CASE

Priority inversion caused system resets in space



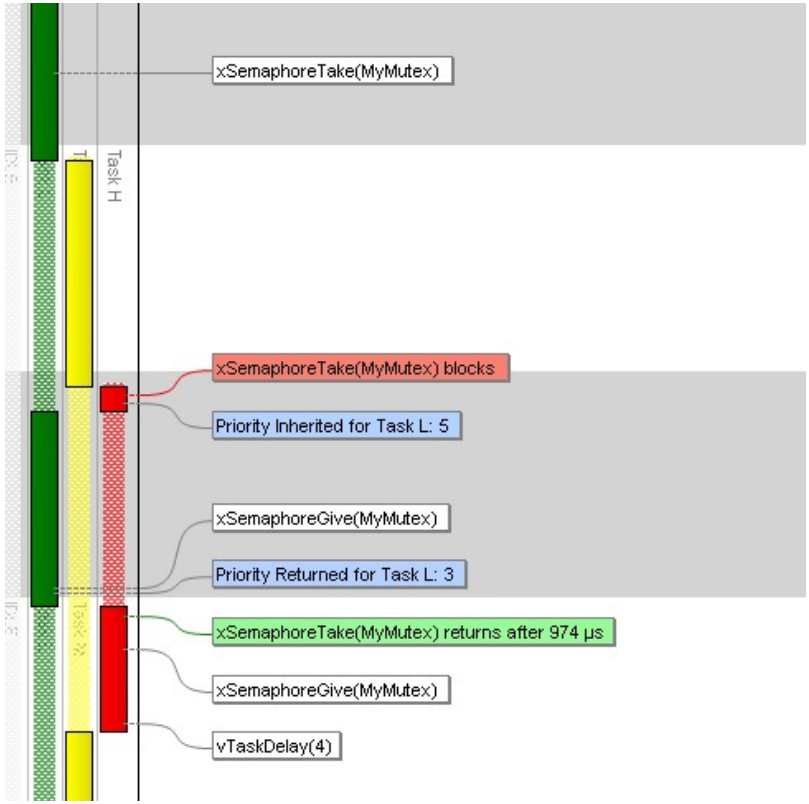
Was debugged using software tracing

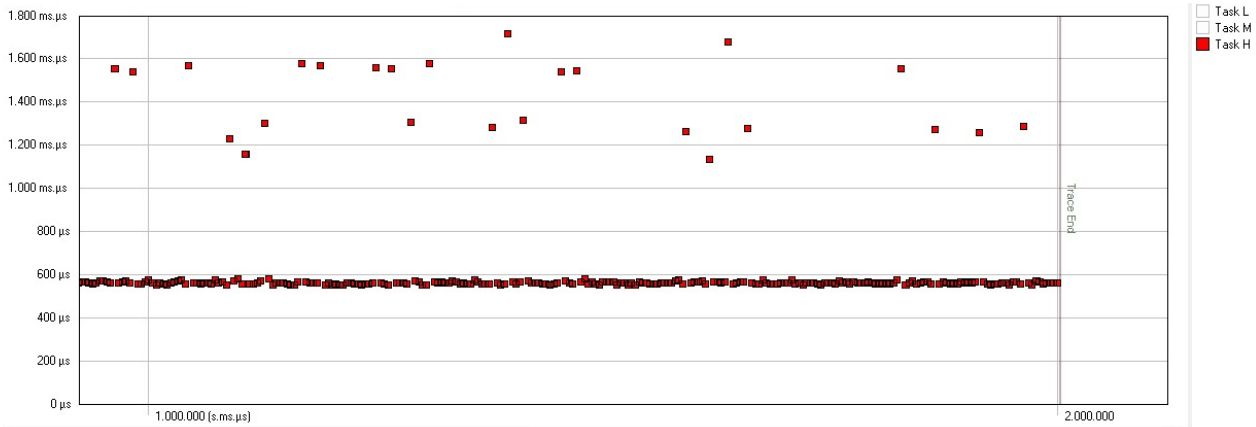
Solution:
Use *priority inheritance*.

© EmbeddedOnlineConference.com All rights reserved

USING MUTEXES INSTEAD

With priority inheritance - avoids priority inversion





Highest response time now below 2 ms

TaskL inherits priority of any tasks waiting for the mutex (TaskH).
Therefore, TaskM can't preempt TaskL and thereby delay TaskH.

© EmbeddedOnlineConference.com All rights reserved

CODE CHANGE

```
void InitDesign(void)
{
    TaskHandle_t objectHandle = NULL;

    vSemaphoreCreateBinary(Sem1);

    if( Sem1 == NULL )
    {
        for (;;);
```

```
void InitDesign(void)
{
    TaskHandle_t objectHandle = NULL;

    Sem1 = xSemaphoreCreateMutex();

    if( Sem1 == NULL )
    {
        for (;;);
```

Creating a (binary) semaphore in FreeRTOS

Changed to mutex instead

Priority inheritance enabled by default.

© EmbeddedOnlineConference.com All rights reserved

SUMMARY, EXAMPLE 3

What did we learn?

- Semaphores are used as signals between tasks
- Using plain semaphores for mutual exclusion may cause priority inversions
- Use mutexes for mutual exclusion, with priority inheritance

© EmbeddedOnlineConference.com All rights reserved

CONCLUSION

- RTOSes provide multithreading. Important to follow best practices!
- Runtime monitoring is required to get the full picture of multithreaded software
- Visual trace diagnostics allows for verifying best practices in runtime
  - Suspend tasks properly after each job (avoid busy waiting)
  - Use the right delay function for periodic execution
  - Use higher scheduling priorities for urgent tasks
  - Minimize timing variations (for better testability and reliability)
  - Place code with timing variations in low priority tasks, if possible
  - Protect critical sections with mutexes, not plain semaphores.
  - …

© EmbeddedOnlineConference.com All rights reserved

LEARNING MORE

Check out our articles at https://percepio.com/rtos-debug-portal

Get a Tracealyzer evaluation at https://percepio.com/download

Contact me at johan.kraft@percepio.com

© EmbeddedOnlineConference.com All rights reserved

THANK YOU

Embedded
Online
Conference

www.embeddedonlineconference.com

Embedded Online Conference

www.embeddedonlineconference.com