FreeRTOS Task Notifications

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel
  by Richard Barry
  - Chapter 9: Task Notifications

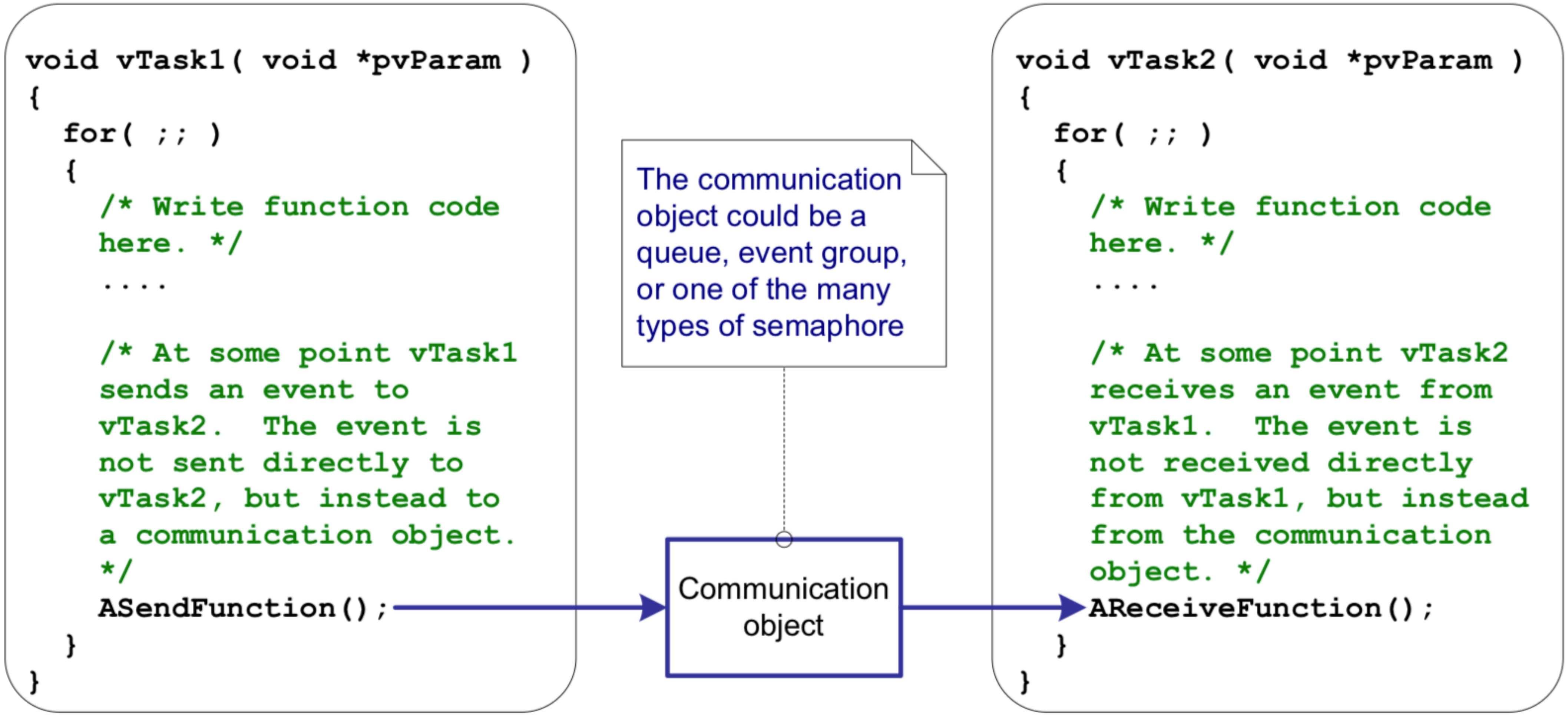
Topics

- 9.1 Intro and Scope
- 9.2 Task Notifications: Benefits and Limitations
- 9.3 Using Task Notifications

9.1 Intro and Scope

- FreeRTOS apps structured as set of independent tasks
- These autonomous tasks may need to communicate with each other
- Previous communications methods required a communication object
    - Queues, Semaphores, Event Groups
- When communication objects are used, the event data is sent via the object - not directly to the task

Task Communication with Communication Objects



```
void vTask1( void *pvParam )
{
  for( ;; )
  {
    /* Write function code
    here. */
    ....

    /* At some point vTask1
    sends an event to
    vTask2.  The event is
    not sent directly to
    vTask2, but instead to
    a communication object.
    */
    ASendFunction();
  }
}
```

The communication object could be a queue, event group, or one of the many types of semaphore

```
void vTask2( void *pvParam )
{
  for( ;; )
  {
    /* Write function code
    here. */
    ....

    /* At some point vTask2
    receives an event from
    vTask1.  The event is
    not received directly
    from vTask1, but instead
    from the communication
    object. */
    AReceiveFunction();
  }
}
```

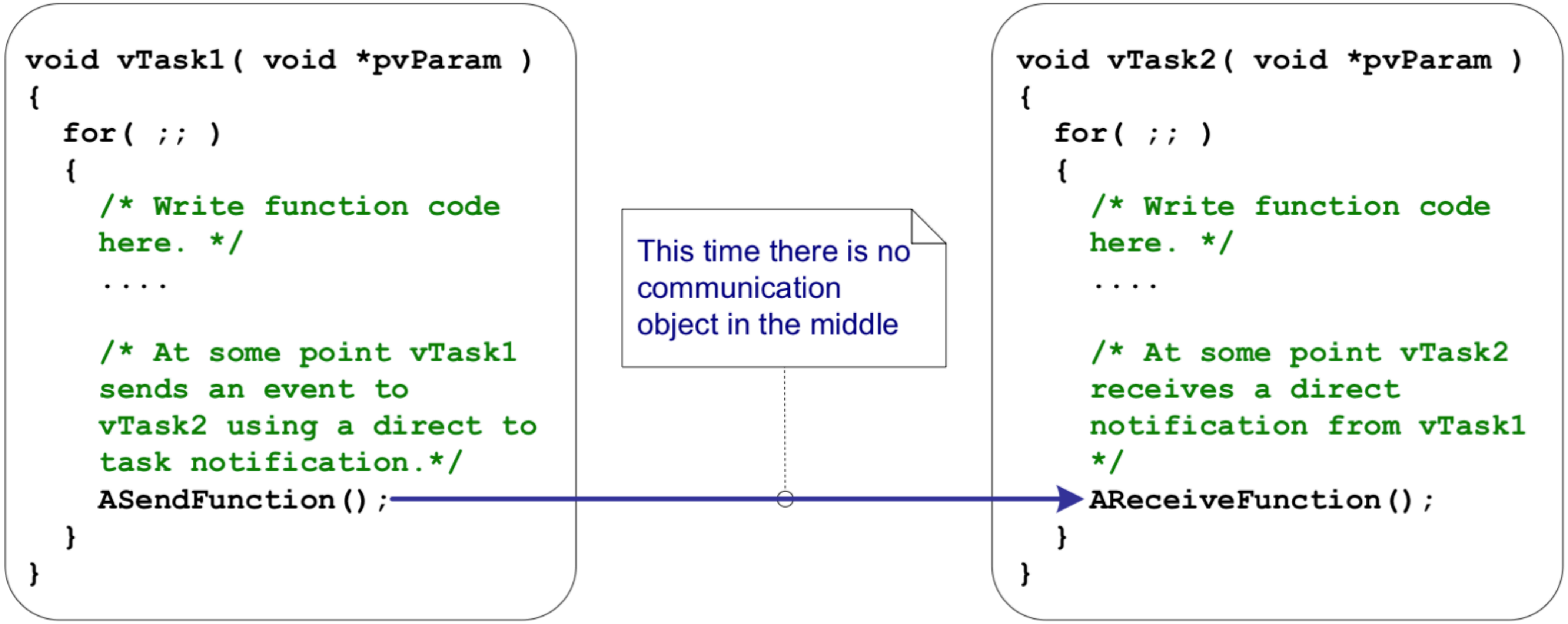
Task Notifications
Direct to Task Communication

- Task Notifications allow tasks to:
  - #1. Interact with other tasks directly
  - #2. To synchronize with ISRs
- Task Notifications do not use a communication object
- Task Notification feature is optional
  - **configUSE_TASK_NOTIFICATIONS** must be set to 1

When configUSE_TASK_NOTIFICATIONS is set to 1

- Each task has:
  - Notification State
    - Pending
      - When task receives a notification, state is set to Pending
    - Not Pending
      - When task reads notification, state set to Not Pending
  - Notification Value
    - uint32_t

Task Notifications Go Direct to the Task



```
void vTask1( void *pvParam )
{
  for( ;; )
  {
    /* Write function code
    here. */
    ....

    /* At some point vTask1
    sends an event to
    vTask2 using a direct to
    task notification.*/
    ASendFunction();
  }
}
```

This time there is no communication object in the middle

```
void vTask2( void *pvParam )
{
  for( ;; )
  {
    /* Write function code
    here. */
    ....

    /* At some point vTask2
    receives a direct
    notification from vTask1
    */
    AReceiveFunction();
  }
}
```

9.2 Task Notifications: Benefits

- Faster Performance
  - Task Notification is significantly faster than using a queue, semaphore, or event group
- Less RAM
  - Task Notifications uses less RAM than queue, semaphore, or event group

Task Notifications Limitations - Part 1

- Cannot send an event or data **to** an ISR
    - But OK to use to send event **from** an ISR

Task Notifications Limitations - Part 2

- Cannot enable more than one receiving task - task notification always sent to a specific task
    - However in many cases there is only one receiving task anyway

Task Notifications Limitations - Part 3

- Cannot buffer multiple data items - Task Notification is “single”
    - If you need buffering then use Queues

Task Notifications Limitations - Part 4

- Cannot broadcast to more than one task - Task Notification is “single”
    - If you need to broadcast the use Event Groups

Task Notifications Limitations - Part 5

- Cannot wait in the blocked state for a send to complete
  - Unlike communication objects that can auto block the sender unil the communication object has space

Using Task Notifications

- Can often be used in place of
  - Queue
  - Semaphore: Binary and Counting
  - Event Group

Task Notification APIs
Option 1 - Simpler

- xTaskNotify**Give**()
  - Send a task notification
- UlTaskNotify**Take**()
  - Receive a task notification

Task Notification APIs
Option 2 - Full Featured

- xTask**Notify**()
    - Send a task notification
- xTaskNotify**Wait**()
    - Receive a task notification

xTaskNotifyGive()

- BaseType_t
  xTaskNotifyGive(
      TaskHandle_t xTaskToNotify)
- Notes
  - Sends a notification directly to the task
  - Increments the receiving task’s notification value
  - Sets the receiving task’s notification state to Pending
  - Nice alternative to using binary or counting semaphore
  - pdPass is the only return value

vTaskNotifyGiveFromISR()

- BaseType_t
  vTaskNotifyGiveFromISR(
      TaskHandle_t xTaskToNotify,
      BaseType_t *pxHigherPriorityTaskWoken)

- Notes
  - Works same as xTaskNotifyGive() but with additional parameter: pxHigherPriorityTaskWoken
  - Set pxHigherPriorityTaskWoken to pdFALSE before calling
    - More info on next slide

pxHigherPriorityTaskWoken

- If calling vTaskNotifyGiveFromISR() causes a task to leave the blocked state…
    - …and the unblocked task has a higher priority than the task currently executing…
        - …then pxHigherPriorityTaskWoken will be set to pdTRUE
- Important: Be sure to set pxHigherPriorityTaskWoken to pdFALSE before calling the function

ulTaskNotifyTake()

- uint32_t
  ulTaskNotifyTake(
  BaseType_t xClearCountOnExit,
  TickType_t xTicksToWait)

Using Task Notification In Place of Semaphore

- In previous lesson we saw this code demo:
  - Used a binary semaphore to unblock a task from within an ISR
    - Result is task sync with ISR
- In this example: Use task notification in place of binary semaphore

Code Demo - Part 1

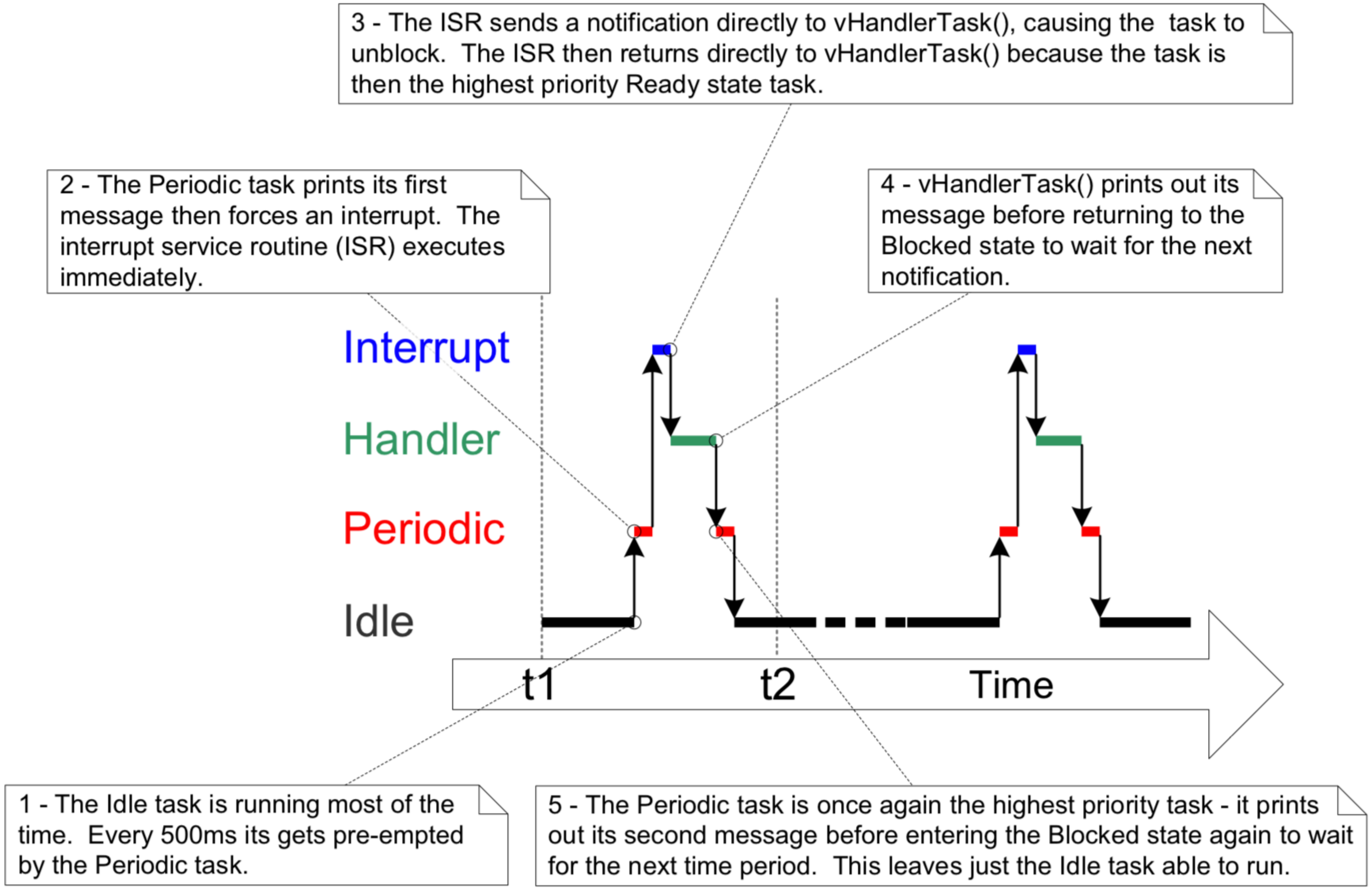
- const TickType_t xInterruptFrequency = pdMS_TO_TICKS(500UL)

-
  ```
  static void vHandlerTask(void *pvPararms) {
      const TickType_t xMaxExpectedBlockTime =
          xInterruptFrequency + pdMS_TO_TICKS(10);

      uint32_t ulEventsToProcess;

      for ( ;; ) {
          ulEventsToProcess = ulTaskNotifyTake(pdTRUE, xMaxExpectedBlockTime);
          if (ulEventsToProcess != 0) {
               while (ulEventsToProcess > 0) {
                    vPrintString(“Handler task - Processing Event\r\n”);
                     ulEventsToProcess—;
                }
          }
          else {
                …
          }
      }
  }
  ```

Code Demo - Part 2

-
```
static uint32_t ulExampleInterruptHandler(void) {
    BaseType_t xHigherPriorityTaskWoken;

    xHigherPriorityTaskWoken = pdFALSE;

    vTaskNotifyGiveFromISR(xHandlerTask,
        &xHigherrPriorityTaskWoken);

    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

}
```

Block Diagram



3 - The ISR sends a notification directly to vHandlerTask(), causing the task to unblock. The ISR then returns directly to vHandlerTask() because the task is then the highest priority Ready state task.

2 - The Periodic task prints its first message then forces an interrupt. The interrupt service routine (ISR) executes immediately.

4 - vHandlerTask() prints out its message before returning to the Blocked state to wait for the next notification.

1 - The Idle task is running most of the time. Every 500ms its gets pre-empted by the Periodic task.

5 - The Periodic task is once again the highest priority task - it prints out its second message before entering the Blocked state again to wait for the next time period. This leaves just the Idle task able to run.

xTaskNotify()
xTaskNotifyFromISR()

- xTaskNotify() is a more capable version of xTaskNotifyGive()
    - Can update the receiving task's notification value in any of the following ways
        - Increment receiving task's notification value (hence same as xTaskNotifyGive())
        - Set one or more bits in the receiving task's notification value
        - Write a completely new number into the receiving task's notification value, but only if the receiving task has read its notification value since it was last updated
        - Write a complete new number into the receiving task's notification value, even if receiving task has not read value - sometimes called a mailbox

xTaskNotify()
xTaskNotifyFromISR()

-
  ```
  BaseType_t
  xTaskNotify(
      TaskHandle_t xTaskToNotify,
      uint32_t ulValue,
      eNotifyAction eAction)
  ```

-
  ```
  BaseType_t
  xTaskNotifyFromISR(
      TaskHandle_t xTaskToNotify,
      uint32_t ulValue,
      eNotifyAction eAction, //See next slide
      BaseType *pxHigherPriorityTaskWoken)
  ```

eNotifyAction Values

- eNoAction
  - Faster and lighter alternative to binary semaphore
- eSetBits
  - Faster and lighter alternative to event group
- eIncrement
  - Faster and lighter alternative to binary and counting semaphores
- eSetValueWithoutOverwrite
  - Will return pdFAIL if receiving task has not read previous value
- eSetValueWithOverwrite
  - Used like a mailbox

xTaskNotifyWait()

- BaseType_t
  xTaskNotifyWait(
      uint32_t ulBitsToClearOnEntry,
      uint32_t ulBitsToClearOnExit,
      uint32_t *pulNotificationValue,
      TickType_t xTicksToWait)

Summary

- 9.1 Intro and Scope
- 9.2 Task Notifications: Benefits and Limitations
- 9.3 Using Task Notifications