Contents

- Event Groups
- Task Notifications

Event Groups

Barry, R. (2016). *Mastering the freertos real time kernel. Pre-release 161204 Edition*. Real Time Engineers Ltd.

Chapter 8

Event driven design

Real-time embedded systems have to take actions in response to events

Events must be communicated to tasks, for example with semaphores and queues

Properties of this type of communication are:

- Allows a task to wait in the Blocked state for a **single event** to occur
- Unblock a **single task** (the one with the highest priority) when the event occurs

Event driven design

Event groups are another FreeRTOS feature

Properties of event groups are:

- Allows a task to wait in the Blocked state for **a combination of one of more events** to occur.
- Unblock **all the tasks** that were waiting for the same event, or combination of events, when the event occurs

Event driven design

This makes event groups useful for :

- synchronizing multiple tasks
- broadcasting events
- allowing a task to wait in the Blocked state for any one of a set of events to occur
- allowing a task to wait in the Blocked state for multiple actions to complete
- possibly reduce RAM usage, because they may replace many binary semaphores

Characteristics of an Event Group

An event ‘flag’ or ‘bit’ is a Boolean (1 or 0) stored in a single bit

- 0: event has not occurred
- 1: event has occurred

An event ‘group’ is a set of event flags: EventBits_t

An application designer assigns the meaning to individual bits

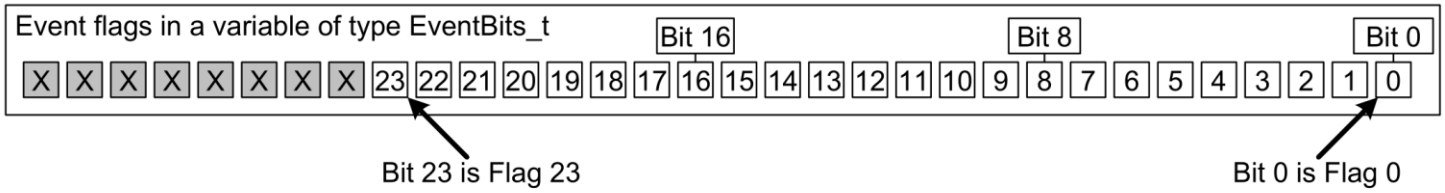


Figure 71 Event flag to bit number mapping in a variable of type EventBits_t

Characteristics of an Event Group

The number of event bits in an event group is dependent on configUSE_16_BIT_TICKS

- If configUSE_16_BIT_TICKS is 1, then each event group contains **8** usable event bits
- If configUSE_16_BIT_TICKS is 0, then each event group contains **24** usable event bits

Event Management Using Event Groups

Event groups are referenced using variables of type EventGroupHandle_t.

```
EventGroupHandle_t xEventGroupCreate( void );
```

Listing 132. The xEventGroupCreate() API function prototype

If NULL is returned, then the event group cannot be created.

A non-NULL value being returned indicates that the event group has been created successfully. The returned value is the handle to the created event group.

Event Management Using Event Groups

Notify one or more tasks that the events represented by the bit, or bits, being set has occurred.

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
                                const EventBits_t uxBitsToSet );
```

Listing 133. The xEventGroupSetBits() API function prototype

The handle of the event group in which bits are being set.

Event Management Using Event Groups

Notify one or more tasks that the events represented by the bit, or bits, being set has occurred.

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
                                const EventBits_t uxBitsToSet );
```

Listing 133. The xEventGroupSetBits() API function prototype

A bit mask that specifies the event bit, or event bits, to set to 1 in the event group. The value of the event group is updated by **bitwise ORing** the event group's existing value with the value passed in uxBitsToSet.

Event Management Using Event Groups

Notify one or more tasks that the events represented by the bit, or bits, being set has occurred.

```
EventBits_t xEventGroupSetBits( EventGroupHandle_t xEventGroup,
                                const EventBits_t uxBitsToSet );
```

Listing 133. The xEventGroupSetBits() API function prototype

The value of the event group at the time the call to xEventGroupSetBits() returned.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set.

The **unblock condition** is specified by two variables

- *uxBitsToWaitFor* specifies which event bits in the event group to test
- *xWaitForAllBits* specifies if a single of those bits must be set (bitwise OR test), or if all those bits must be set (bitwise AND test)

A task will not enter the Blocked state if its unblock condition is met at the time xEventGroupWaitBits() is called.

Event Management Using Event Groups

Example unblock conditions

Event group value	uxBitsToWaitFor	xWaitForAllBits	Resultant Behavior
0b00000000	0b00000101	pdFALSE	The calling task will enter the Blocked state because neither of bit 0 or bit 2 are set.
0b00000100	0b00000101	pdTRUE	The calling task will enter the Blocked state because bit 0 and bit 2 are not both set
0b00000100	0b00000101	pdFALSE	The calling task will not enter the Blocked state because bit 2 is already set
0b00000111	0b00000101	pdTRUE	The calling task will not enter the Blocked state because both bits are already set

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

The handle of the event group in which bits are being set.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

A bit mask that specifies the event bit, or event bits, to test in the event group.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

If xWaitForAllBits is set to **pdFALSE**, then a task will leave the Blocked state when **any** of the bits specified by uxBitsToWaitFor become set.

If xWaitForAllBits is set to **pdTRUE**, then a task will leave the Blocked state when **all** of the bits specified by uxBitsToWaitFor become set.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

If the calling task's unblock condition has been met, and xClearOnExit is set to pdTRUE, then the event bits specified by uxBitsToWaitFor will be cleared back to 0 in the event group.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

The maximum amount of time the task should remain in the Blocked state to wait for its unblock condition to be met.

Event Management Using Event Groups

Read the value of an event group, and optionally wait in the Blocked state for one or more event bits in the event group to become set, if the event bits are not already set.

```
EventBits_t xEventGroupWaitBits( const EventGroupHandle_t xEventGroup,
                                 const EventBits_t uxBitsToWaitFor,
                                 const BaseType_t xClearOnExit,
                                 const BaseType_t xWaitForAllBits,
                                 TickType_t xTicksToWait );
```

Listing 135. The xEventGroupWaitBits() API function prototype

The value of the event group at the time the calling task's unblock condition was met or the value at the time the block time expired.

Event Groups Example

```
/*-----------------------------------------------------------------------------*/
// Local variables
/*-----------------------------------------------------------------------------*/
static EventGroupHandle_t xEventGroup;

const EventBits_t xEventBits[3] =
{
    0b00000000000000000000000000000001, // Is set by vTask1
    0b00000000000000000000000000000010, // Is set by vTask2
    0b00000000000000000000000000000100, // Is set by vTask3
};

const TickType_t xDelaysMs[3] =
{
    pdMS_TO_TICKS(1000),  // Delay for vTask1
    pdMS_TO_TICKS(5000),  // Delay for vTask2
    pdMS_TO_TICKS(10000), // Delay for vTask3
};
```

Event Groups Example

```
int main(void)
{
    rgb_init();
    xSerialPortInit(921600, 128);

    vSerialPutString("\r\nFRDM-KL25Z FreeRTOS demo Week 6 - Example 01\r\n");
    vSerialPutString("By Hugo Arends\r\n\r\n");

    xTaskCreate(vTask,        "vTask1",       configMINIMAL_STACK_SIZE, (void *)1, 3, NULL );
    xTaskCreate(vTask,        "vTask2",       configMINIMAL_STACK_SIZE, (void *)2, 2, NULL );
    xTaskCreate(vTask,        "vTask3",       configMINIMAL_STACK_SIZE, (void *)3, 1, NULL );
    xTaskCreate(vEventReader, "vEventReader", configMINIMAL_STACK_SIZE, NULL,      4, NULL );

    // Create the event group
    xEventGroup = xEventGroupCreate();

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well then main() will never reach here as the scheduler will
    now be running the tasks. If main() does reach here then it is likely that
    there was insufficient heap memory available for the idle task to be created.
    Chapter 2 provides more information on heap memory management. */
    for( ;; );
}
```

Event Groups Example

```
static void vTask(void *parameters)
{
    char str[48];

    // Set task specific values.
    int32_t n = (int32_t)parameters;
    TickType_t xLastWakeTime = xTaskGetTickCount();
    TickType_t xDelayMs = xDelaysMs[n-1];

    sprintf(str, "[% 7d - Task %d     ] Created\r\n", xLastWakeTime, n);
    vSerialPutString(str);

    // As per most tasks, this task is implemented in an infinite loop.
    for( ;; )
    {
        // Wait
        vTaskDelayUntil(&xLastWakeTime, xDelayMs);

        // Print info
        sprintf(str, "[% 7d - Task %d     ] Set bit %d\r\n", xLastWakeTime, n, (n-1));
        vSerialPutString(str);

        // Set the bit in the event group
        xEventGroupSetBits(xEventGroup, xEventBits[n-1]);
    }
}
```

Event Groups Example

```
static void vEventReader(void *parameters)
{
    char str[48];

    EventBits_t xEventGroupValue;
    const EventBits_t xBitsToWaitFor = xEventBits[0] | xEventBits[1] | xEventBits[2];

    const BaseType_t xWaitForAllBits = pdTRUE; // pdFALSE

    sprintf(str, "[% 7d - EventReader] Created\r\n", xTaskGetTickCount());
    vSerialPutString(str);
```

Event Groups Example

```
    // As per most tasks, this task is implemented in an infinite loop.
    for( ;; )
    {
        // Block to wait for event bits to become set within the event group.
        xEventGroupValue = xEventGroupWaitBits(xEventGroup,
                                               xBitsToWaitFor,
                                               pdTRUE,
                                               xWaitForAllBits,
                                               portMAX_DELAY);

        // Prepare the string
        sprintf(str, "[% 7d - EventReader] EventBits: 000..0", xTaskGetTickCount());

        // Add the value of the last three bits
        strcat(str, ((xEventGroupValue & xEventBits[2]) != 0) ? "1" : "0");
        strcat(str, ((xEventGroupValue & xEventBits[1]) != 0) ? "1" : "0");
        strcat(str, ((xEventGroupValue & xEventBits[0]) != 0) ? "1" : "0");

        // Finalize string and print it
        strcat(str, "\r\n");
        vSerialPutString(str);
    }
}
```

Event Groups Example

xWaitForAllBits = pdFALSE;

```
[      0 - EventReader] Created
[      1 - Task 1      ] Created
[      2 - Task 2      ] Created
[      3 - Task 3      ] Created
[   1001 - Task 1      ] Set bit 0
[   1002 - EventReader] EventBits: 000..0001
[   2001 - Task 1      ] Set bit 0
[   2002 - EventReader] EventBits: 000..0001
[   3001 - Task 1      ] Set bit 0
[   3002 - EventReader] EventBits: 000..0001
[   4001 - Task 1      ] Set bit 0
[   4002 - EventReader] EventBits: 000..0001
[   5001 - Task 1      ] Set bit 0
[   5002 - EventReader] EventBits: 000..0001
[   5002 - Task 2      ] Set bit 1
[   5004 - EventReader] EventBits: 000..0010
[   6001 - Task 1      ] Set bit 0
[   6002 - EventReader] EventBits: 000..0001
[   7001 - Task 1      ] Set bit 0
[   7002 - EventReader] EventBits: 000..0001
[   8001 - Task 1      ] Set bit 0
[   8002 - EventReader] EventBits: 000..0001
[   9001 - Task 1      ] Set bit 0
[   9002 - EventReader] EventBits: 000..0001
[  10001 - Task 1      ] Set bit 0
[  10002 - EventReader] EventBits: 000..0001
[  10002 - Task 2      ] Set bit 1
[  10004 - EventReader] EventBits: 000..0010
[  10003 - Task 3      ] Set bit 2
[  10006 - EventReader] EventBits: 000..0100
[  11001 - Task 1      ] Set bit 0
[  11002 - EventReader] EventBits: 000..0001
```

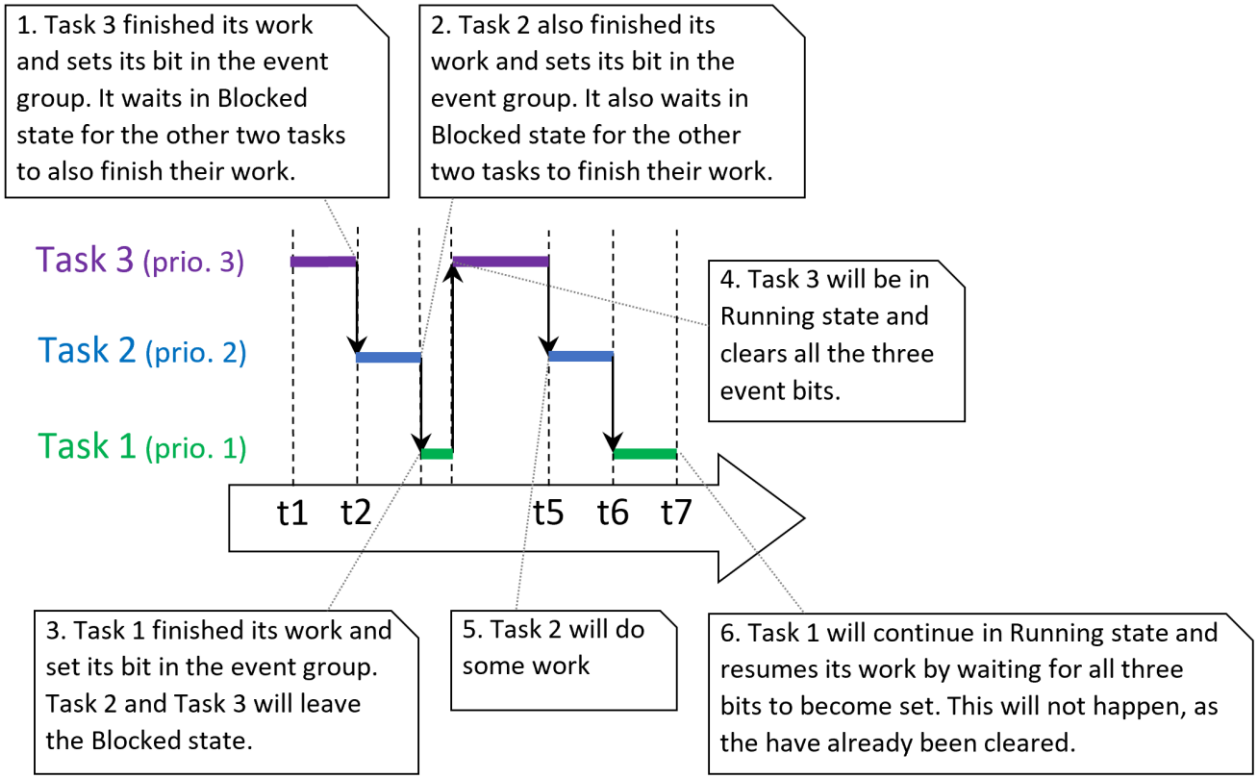
xWaitForAllBits = pdTRUE;

```
[      0 - EventReader] Created
[      1 - Task 1      ] Created
[      2 - Task 2      ] Created
[      3 - Task 3      ] Created
[   1001 - Task 1      ] Set bit 0
[   2001 - Task 1      ] Set bit 0
[   3001 - Task 1      ] Set bit 0
[   4001 - Task 1      ] Set bit 0
[   5001 - Task 1      ] Set bit 0
[   5002 - Task 2      ] Set bit 1
[   6001 - Task 1      ] Set bit 0
[   7001 - Task 1      ] Set bit 0
[   8001 - Task 1      ] Set bit 0
[   9001 - Task 1      ] Set bit 0
[  10001 - Task 1      ] Set bit 0
[  10002 - Task 2      ] Set bit 1
[  10003 - Task 3      ] Set bit 2
[  10004 - EventReader] EventBits: 000..0111
[  11001 - Task 1      ] Set bit 0
[  12001 - Task 1      ] Set bit 0
```

Task Synchronization Using an Event Group

Goal: to synchronize three tasks with event groups

Problem example: the bits in the event group are already cleared!



Task Synchronization Using an Event Group

xEventGroupSync() is a special function for task synchronisation with event groups

- It allows a task to set one or more event bits in an event group
- It then waits for a combination of event bits to become set in the same event group **as a single uninterruptable operation**.

```
EventBits_t xEventGroupSync( EventGroupHandle_t xEventGroup,
                             const EventBits_t uxBitsToSet,
                             const EventBits_t uxBitsToWaitFor,
                             TickType_t xTicksToWait );
```

Listing 142. The xEventGroupSync() API function prototype

Summary

- Practical uses for event groups.
- The advantages and disadvantages of event groups relative to other FreeRTOS features.
- How to set bits in an event group.
- How to wait in the Blocked state for bits to become set in an event group.
- How to use an event group to synchronize a set of tasks.

Task Notifications

Barry, R. (2016). *Mastering the freertos real time kernel. Pre-release 161204 Edition*. Real Time Engineers Ltd.

Chapter 9

FreeRTOS communication

Set of independent tasks that:

- Communicate with each other
- Collectively implement system functionality

So far we have used **communication objects**

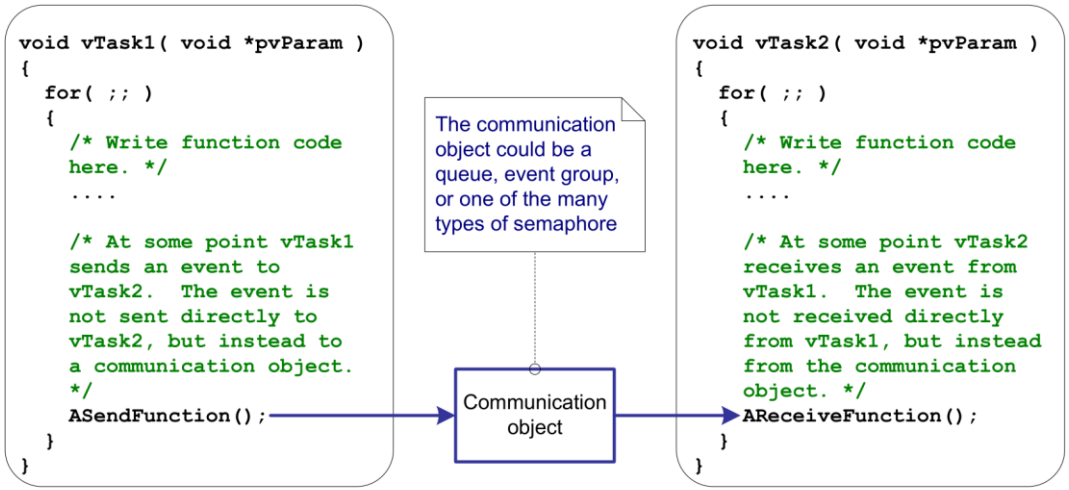


Figure 76 A communication object being used to send an event from one task to another

FreeRTOS communication

We can use task notifications instead, to send an event directly

Enabled by setting configUSE_TASK_NOTIFICATIONS to 1

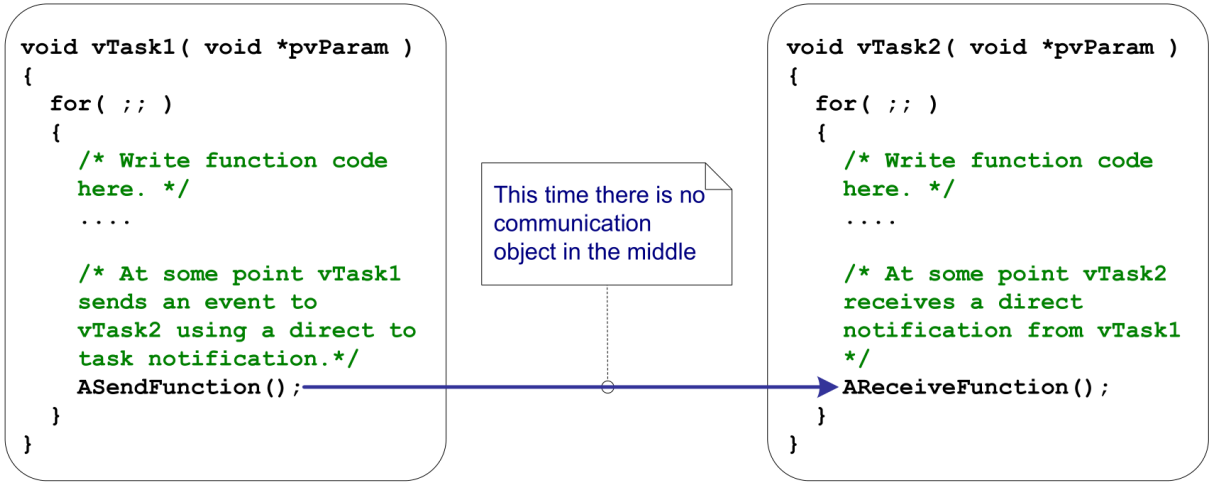


Figure 77 A task notification used to send an event directly from one task to another

Task notification characteristics

Each task has a **Notification State**:

- *Pending*
  When a task receives a notification, its notification state is set to pending.
- *Not-Pending*
  When a task reads its notification value, its notification state is set to not-pending.

Each task has a **Notification Value**:

- A 32-bit unsigned integer

A task can wait in the Blocked state, with an optional time out, for its notification state to become pending.

Benefits

When compared to a queue, semaphore or event group to perform an equivalent operation

- Performance - significantly faster
- RAM usage - significantly less RAM

Limitations

Scenarios in which a task notification cannot be used

- Sending an event or data to an ISR
- Buffering multiple data items
- Broadcasting to more than one task
- Waiting in the blocked state for a send to complete

  If a task attempts to send a task notification to a task that already has a notification pending, then it is not possible for the sending task to wait in the Blocked state for the receiving task to reset its notification state.

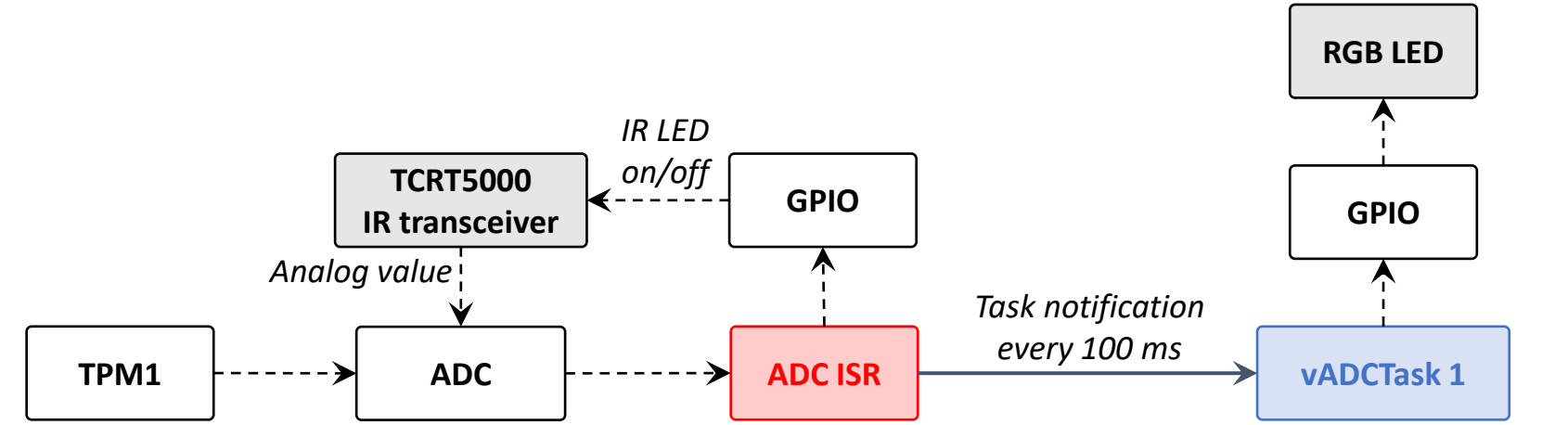
Using Task Notifications

API functions overview

	Lightest weight and fastest alternative to a binary or counting semaphore	*Lighter weight and faster alternative to a binary or counting semaphore, or event group, or queue of length one*
Sending	**xTaskNotifyGive(…)** **xTaskNotifyGiveFromISR(…)** • Sets notification state to pending • Notification value: +1	**xTaskNotify(…)** **xTaskNotifyFromISR(…)** • Sets notification state to pending • Notification value: +1 *or* set bits *or* a completely new number
Receiving	**ulTaskNotifyTake(…)** • Task optionally waits in Blocked state for notification state to become pending • Notification value: -1 *or* cleared to 0	**xTaskNotifyWait(…)** • Task optionally waits in Blocked state for notification state to become pending • Notification value: bits to clear on entry *and* bits to clear on exit

Task Notifications Example

See Week 6 – Example 03



Configured to trigger an ADC conversion every 50 ms

Is executed every 50 ms and reads the ADC conversion

The task is notified after having combined two conversions: one with the IR LED off and one with the IR LED on. The notification value is the difference of these two conversions.

Waits for a notification to process the data. If an object is in range, the red LED is on, otherwise the green LED

Summary

- A task's notification state and notification value.
- How and when a task notification can be used in place of a communication object, such as a semaphore.
- The advantages of using a task notification in place of a communication object.