5.6 Synchronization Mechanisms: A Comparative Guide

FreeRTOS provides three primary mechanisms for safely managing shared resources between tasks and ISRs.

Critical Sections: The "Big Hammer" for Protection

A **critical section** is a section of code that is guaranteed to run to completion without being preempted by an interrupt or another task. It is the most direct and forceful way to prevent a race condition.

How it Works: It works by temporarily disabling all interrupts system-wide.

- In a task, you wrap the critical code with `taskENTER_CRITICAL()` and `taskEXIT_CRITICAL()`.
- In an ISR, you use `portENTER_CRITICAL_ISR()` and `portEXIT_CRITICAL_ISR()`.

Example:

```
volatile int counter = 0;

void IRAM_ATTR onTimer() {
    portENTER_CRITICAL_ISR(&timerMux);
    counter++; // This is now safe
    portEXIT_CRITICAL_ISR(&timerMux);
}

void printValues(void * parameter) {
    while (true) {
        taskENTER_CRITICAL();
        counter--; // This is now safe
        Serial.println(counter);
        taskEXIT_CRITICAL();
        vTaskDelay(pdMS_TO_TICKS(2000));
```

```
    }
}
```

- **When to Use:** Only for protecting very short, fast operations on shared variables where other mechanisms are too slow or complex.
- **Risk:** While a critical section is active, all interrupts are disabled. This can severely impact the real-time responsiveness of the system. **Keep critical sections as short as humanly possible.**

Semaphores: The Best Tool for Pure Signaling

A **semaphore** is a signaling mechanism. It does not transfer data. It is used to signal that an event has occurred or to control access to a resource. For ISR-to-task communication, a **binary semaphore** is typically used.

How it Works: Think of it as a flag.

1. A task tries to "take" the semaphore using `xSemaphoreTake()`. If the semaphore is not available, the task enters the Blocked state, consuming no CPU time.
2. An ISR, responding to a hardware event, "gives" the semaphore using `xSemaphoreGiveFromISR()`.
3. Giving the semaphore unblocks the waiting task, moving it to the Ready state. The scheduler will then run the task when it is its turn.

Example:

```
SemaphoreHandle_t binSemaphore = NULL;

void IRAM_ATTR onTimer() {
    xSemaphoreGiveFromISR(binSemaphore, NULL);
}

void processValues(void * parameter) {
    while (true) {
        // Wait here until the ISR gives the semaphore
        if (xSemaphoreTake(binSemaphore, portMAX_DELAY) == pdTRUE) {
            // The event occurred. Process the data.
            Serial.println("Processing data now...");
        }
    }
}
```

- **When to Use:** When an ISR needs to notify a task that an event has happened (e.g., "ADC conversion is complete, the data is ready to be read"). It's a pure synchronization primitive.

Queues: The Best Tool for Transferring Data

A **queue** is the most powerful and often the best mechanism for ISR-to-task communication. It provides a thread-safe First-In, First-Out (FIFO) buffer to not only signal an event but to also safely transfer data from the ISR to the task.

How it Works:

1. A task waits to "receive" from a queue using `xQueueReceive()`. If the queue is empty, the task enters the Blocked state.
2. An ISR generates some data (e.g., reads a sensor). It then "sends" this data to the queue using `xQueueSendFromISR()`. This action copies the data into the queue's buffer.
3. The act of sending data to the queue unblocks the waiting task. The task then receives the copy of the data from the queue for safe processing.

This approach is superior because the ISR and the task never access the same variable directly. They only interact via the RTOS-managed queue, which eliminates race conditions by design.

Example:

```
QueueHandle_t sensorQueue = NULL;

void IRAM_ATTR onTimer() {
    // Read sensor and package data into a struct
    sensorData_t data;
    data.adcValue1 = analogRead(34);

    // Send a COPY of the data to the queue
    xQueueSendFromISR(sensorQueue, &data, NULL);
}

void processValues(void * parameter) {
    sensorData_t receivedData;
    while (true) {
        // Wait here until data arrives in the queue
        if (xQueueReceive(sensorQueue, &receivedData, portMAX_DELAY) == pdTRUE) {
            // Safely process the received data
            Serial.println(receivedData.adcValue1);
```

```
        }
    }
}
```

- **When to Use:** Whenever an ISR needs to pass data to a task for processing. This is the preferred method in almost all data-generating ISR scenarios.

Revision #1
Created 2025-08-28 13:02:48 UTC by GI
Updated 2025-08-28 13:05:08 UTC by GI