Code Sample

This example demonstrates a simple FreeRTOS queue communication between two tasks (`Task1` and `Task2`) running on an ESP32. Here's how it works:

1. `Task1`: This task generates a random integer between 0 and 100, dynamically allocates memory for it using `pvPortMalloc()`, and attempts to send the pointer to the integer to a queue (`xQueue`). If the queue is full and the message cannot be sent, it prints an error message and frees the allocated memory. After each send attempt, the task delays for 1 second.
2. `Task2`: This task waits to receive data from the queue. When a message is received, it prints the received value and then frees the memory used for the integer.
3. `setup()`: Initializes the serial communication, creates the queue, and starts the two tasks (`Task1` and `Task2`) pinned to core 1. If the queue creation fails, it prints an error message.
4. `loop()`: The main Arduino loop remains empty since the tasks are running independently of it.

Here’s the full code:

```
QueueHandle_t xQueue;

void Task1(void *pvParameters) {
  int *p;
  while (1) {
    // Dynamically allocate memory for an integer
    p = (int *)pvPortMalloc(sizeof(int));
    *p = random(0, 100); // Generate a random number between 0 and 100

    // Attempt to send the pointer to the queue, wait indefinitely if needed
    if (xQueueSend(xQueue, &p, portMAX_DELAY) != pdPASS) {
      Serial.println("Failed to post to queue");
      vPortFree(p); // Free memory if message could not be sent to the queue
    }

    vTaskDelay(1000 / portTICK_PERIOD_MS); // Delay for 1 second
  }
}

void Task2(void *pvParameters) {
```

```
  int *p;
  while (1) {
    // Wait to receive a pointer from the queue, wait indefinitely if needed
    if (xQueueReceive(xQueue, &p, portMAX_DELAY)) {
      Serial.print("Received: ");
      Serial.println(*p); // Print the received value
      vPortFree(p); // Free memory after processing
    }
  }
}

void setup() {
  Serial.begin(115200);

  // Create a queue capable of holding 10 integer pointers
  xQueue = xQueueCreate(10, sizeof(int *));

  if (xQueue == NULL) {
    Serial.println("Error creating the queue");
  }

  // Create two tasks pinned to core 1
  xTaskCreatePinnedToCore(Task1, "Task1", 10000, NULL, 1, NULL, 1);
  xTaskCreatePinnedToCore(Task2, "Task2", 10000, NULL, 1, NULL, 1);

  vTaskDelete(NULL); // Delete the setup task to free memory
}

void loop() {
  // Empty loop since tasks are handled in FreeRTOS tasks
}
```

Key Points:

- **Dynamic Memory Allocation**: Each task dynamically allocates memory for the integer it sends, and the receiving task is responsible for freeing that memory after use.
- **Queue**: A queue is created to hold pointers to integers. Both tasks communicate through this queue.
- **Core Assignment**: The tasks are pinned to core 1 for performance reasons, but this can be changed based on requirements.

Revision #1
Created 1 October 2024 04:55:23 by AJ
Updated 1 October 2024 04:56:27 by AJ