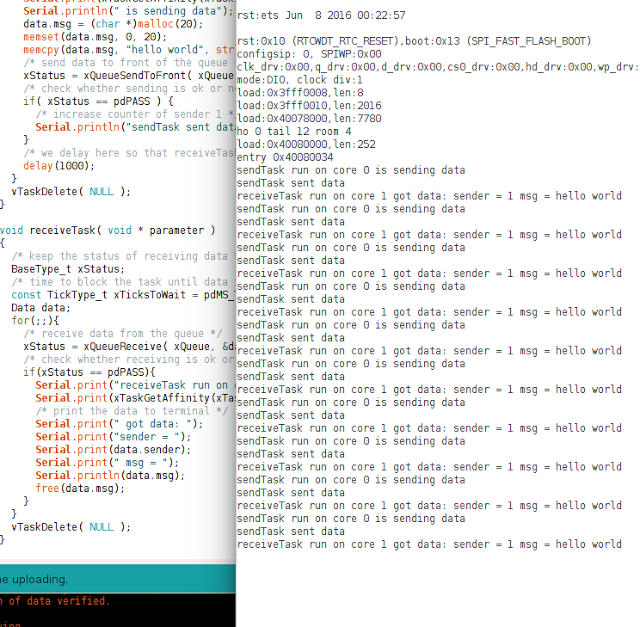
# **[Demo 25: How to configure ESP32 Dual core - Multicore in Arduino ESP32](http://www.iotsharing.com/2017/07/how-to-configure-esp32-multicore-arduino-esp32.html)**

**1.Introduction**  
- ESP32 is a big improvement of ESP8266 (after taking time to play with it, I see that it is faster, more stable than ESP8266). One of special features of ESP32 is that it support dual core. This demo will show you how to configure ESP32 Multicore using Arduino ESP32.  
**2. Demo**  
- We will re-use [Queue](http://www.iotsharing.com/2017/06/arduino-esp32-freertos-how-to-use-message-queue.html" \t "http://www.iotsharing.com/2017/07/_blank) tutorial. In this demo, we create 2 tasks, 1 task called "sendTask" and 1 task called "receiveTask". The "sendTask" is pinned on core 0. The "receiveTask" is pinned on core 1. The "sendTask" will send data to the "receiveTask" every second through [Queue.](http://www.iotsharing.com/2017/06/arduino-esp32-freertos-how-to-use-message-queue.html" \t "http://www.iotsharing.com/2017/07/_blank)  
- In order to pin the task to a specific core we will use the FreeRTOS API function **xTaskCreatePinnedToCore**instead of using**xTaskCreate**. The API **xTaskCreatePinnedToCore**has the last argument is the core that the task will be pinned to. Beside that we can use the API function **xTaskGetAffinity**to know which core the task was pinned to. This function has one argument. That is the task handler which is created using **xTaskCreatePinnedToCore**.

[ESP32 GIO12 - BUTTON - GND]

|  |
| --- |
| /\* structure that hold data\*/  typedef struct{  int sender;  char \*msg;  }Data;  /\* this variable hold queue handle \*/  xQueueHandle xQueue;  TaskHandle\_t xTask1;  TaskHandle\_t xTask2;  void setup() {  Serial.begin(112500);  /\* create the queue which size can contains 5 elements of Data \*/  xQueue = xQueueCreate(5, sizeof(Data));  xTaskCreatePinnedToCore(  sendTask, /\* Task function. \*/  "sendTask", /\* name of task. \*/  10000, /\* Stack size of task \*/  NULL, /\* parameter of the task \*/  1, /\* priority of the task \*/  &xTask1, /\* Task handle to keep track of created task \*/  0); /\* pin task to core 0 \*/  xTaskCreatePinnedToCore(  receiveTask, /\* Task function. \*/  "receiveTask", /\* name of task. \*/  10000, /\* Stack size of task \*/  NULL, /\* parameter of the task \*/  1, /\* priority of the task \*/  &xTask2, /\* Task handle to keep track of created task \*/  1); /\* pin task to core 1 \*/  }  void loop() {  }  void sendTask( void \* parameter )  {  /\* keep the status of sending data \*/  BaseType\_t xStatus;  /\* time to block the task until the queue has free space \*/  const TickType\_t xTicksToWait = pdMS\_TO\_TICKS(100);  /\* create data to send \*/  Data data;  /\* sender 1 has id is 1 \*/  data.sender = 1;  for(;;){  Serial.print("sendTask run on core ");  /\* get the core that the task was pinned to \*/  Serial.print(xTaskGetAffinity(xTask1));  Serial.println(" is sending data");  data.msg = (char \*)malloc(20);  memset(data.msg, 0, 20);  memcpy(data.msg, "hello world", strlen("hello world"));  /\* send data to front of the queue \*/  xStatus = xQueueSendToFront( xQueue, &data, xTicksToWait );  /\* check whether sending is ok or not \*/  if( xStatus == pdPASS ) {  /\* increase counter of sender 1 \*/  Serial.println("sendTask sent data");  }  /\* we delay here so that receiveTask has chance to receive data \*/  delay(1000);  }  vTaskDelete( NULL );  }  void receiveTask( void \* parameter )  {  /\* keep the status of receiving data \*/  BaseType\_t xStatus;  /\* time to block the task until data is available \*/  const TickType\_t xTicksToWait = pdMS\_TO\_TICKS(100);  Data data;  for(;;){  /\* receive data from the queue \*/  xStatus = xQueueReceive( xQueue, &data, xTicksToWait );  /\* check whether receiving is ok or not \*/  if(xStatus == pdPASS){  Serial.print("receiveTask run on core ");  /\* get the core that the task was pinned to \*/  Serial.print(xTaskGetAffinity(xTask2));  /\* print the data to terminal \*/  Serial.print(" got data: ");  Serial.print("sender = ");  Serial.print(data.sender);  Serial.print(" msg = ");  Serial.println(data.msg);  free(data.msg);  }  }  vTaskDelete( NULL );  } |

[](https://4.bp.blogspot.com/-863YZWlU9hw/WVZvqx3sJWI/AAAAAAAAEHE/8l_ZmcKkmwwAI5C2Rw2A1ZPSfHCtGbw1gCLcBGAs/s1600/esp32multicore.png)

**Figure: multicore on ESP32**