Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería



**SISTEMAS EMBEBIDOS**

**Práctica 3: UART**

**Docente: Evangelina Lara Camacho**

**Alumno:** Gómez Cárdenas Emmanuel Alberto

**Matricula:** 01261509

## Objetivo

El alumno se familiarizará con el uso del periférico UART usando el sistema embebido ESP32 DevKit v1 para desarrollar aplicaciones para sistemas basados en microcontrolador para aplicarlos en la resolución de problemas de cómputo, de una manera eficaz y responsable.

## Equipo

Computadora personal con conexión a internet.

## Teoría

### Describa el modo UART half-duplex del ESP32

El término dúplex es utilizado para definir un dispositivo capaz de mantener una comunicación bidireccional, enviando y recibiendo mensajes de forma simultánea.

### Dúplex (dúplex completo o full-duplex)

### Permiten canales de envío y recepción simultáneos.

### Semidúplex (half-duplex)

### Es una conexión en la que los datos fluyen en una u otra dirección, pero no las dos al mismo tiempo

### Simplex

### Únicamente permiten la transmisión en un sentido.

#### Half-duplex en el ESP32

El controlador UART del ESP32 no cuenta con un soporte integrado para la comunicación automática en half-duplex, sin embargo, este tipo de UART puede ser logrado mediante software (controlando el pin RTS manualmente).

Para poder utilizar UART en modo half-duplex se hace lo siguiente:

1. Configurar UART: No hay pasos específicos en comparación con full-duplex.
2. Establecer UART al modo RS485: Utilizando la función uart\_set\_mode() mandando UART\_MODE\_RS485\_HALF\_DUPLEX como argumento.
3. Configurar PIN: Conectar el pin RTS al pin ~RE/DE del controlador RS485 para habilitar la transmisión o la recepción.

# Desarrollo

Implemente en dos ESP32 ESP-IDF una pancarta ASCII haciendo uso de **UART y tareas.** La implementación debe ser eficiente en el uso de recursos de cómputo (procesador, memoria y periféricos).

Un ESP32 recibe del usuario una cadena de texto máximo 25 caracteres usando UART. El ESP32 envía por UART la cadena al otro ESP32. Este ESP23 convierte los caracteres a un formato de pancarta ASCII y los despliega en la terminal por medio de UART. Limite los caracteres válidos a las letras del alfabeto en mayúscula y minúscula, números, espacio y los caracteres especiales ‘!’, ‘.’, ‘+’ y ‘-’.

A pink squares with black text

Description automatically generated

La pancarta ASCII es de su elección, pero debe ser similar a las dos de los siguientes ejemplos.

A screenshot of a computer screen

Description automatically generated

#### UARTs en uso:

### UART0: Utilizado para la interfaz con la terminal.

### ESP32-1: Usa la terminal para recibir datos.

### ESP32-2: Usa la terminal para mostrar los datos recibidos mediante UART2.

### UART2: Utilizado para la interfaz con el otro ESP32.

### ESP32-1: Los datos obtenidos en el UART0 son enviados mediante el UART2 hacia el otro ESP32.

### ESP32-2: Recibe los datos del otro ESP32 y los manda al UART0.

#### Diferenciación entre ESP32

Debido a que el mismo programa es usado en ambos ESP32, he decidido utilizar el pin 23 para indicar si el ESP32 es un receiver o un transmitter.

### Transmitter: Configurado al conectar el pin 23 a tierra, este recibe los datos mediante la consola (UART0) y los transmite al otro ESP32 (UART2).

### Receiver: Configurado al conectar el pin 23 a 3V3, el LED interno (GPIO2) del ESP32 será encendido para indicar que es el ESP32 receptor. Este recibe los datos del otro ESP32 (UART2) y los enviará a la consola (UART0) en formato pancarta.

A screenshot of a computer

Description automatically generated

La evidencia del funcionamiento se encuentra en [Drive](https://drive.google.com/file/d/1etj5G7x3s9hQRSoa7jys3eBxPe9JElKa/view?usp=drive_link)

## Conclusiones y Comentarios

El uso del UART es muy importante ya que es muy fiable, fácil de usar además de ser versátil, gracias a esto es un protocolo fundamental en una amplia gama de aplicaciones, especialmente en sistemas embebidos.

## Dificultades en el Desarrollo

Al estar usando el mismo UART para la comunicación a consola, esto hace que, si está mal configurado, es imposible depurar mediante texto, ya que puede provocar conflictos.

Igualmente, al configurar ambos ESP32 para enviar/recibir con el UART2 y no recibir datos, se me complicó el descubrir si el problema era en el ESP32 que enviaba datos o el que recibía.

## Referencias

<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/index.html>

Código  
El código fuente puede ser encontrado en el [Repositorio de GitHub “Sistemas Embebidos”](https://github.com/AlbGmx/SistemasEmbebidos)

Al ser múltiples archivos, solo presentaré archivo principal “main.c”

#include "myUart.h"

#define *IS\_RECEIVER\_CONTROL\_PIN* GPIO\_NUM\_23

#define *SHOW\_MODE\_PIN* GPIO\_NUM\_2

static const char \*TAG = "P3 - UART";

*bool* isReceiver = *false*;

void *gpio\_setup*() {

*gpio\_config\_t* io\_conf;

   io\_conf.intr\_type = GPIO\_INTR\_DISABLE;

   io\_conf.mode = GPIO\_MODE\_INPUT\_OUTPUT;

   io\_conf.pin\_bit\_mask = 1ULL << *IS\_RECEIVER\_CONTROL\_PIN*

| 1ULL << *SHOW\_MODE\_PIN*;

   io\_conf.pull\_down\_en = GPIO\_PULLDOWN\_ENABLE;

   io\_conf.pull\_up\_en = GPIO\_PULLUP\_DISABLE;

*gpio\_config*(&io\_conf);

*ESP\_LOGI*(TAG, "GPIO setup complete");

}

void *app\_main*(void) {

*init\_UARTs*();

*gpio\_setup*();

   char \*data = (char \*)*malloc*(*BUFFER\_SIZE*);

*// Set mode*

   isReceiver = (*gpio\_get\_level*(*IS\_RECEIVER\_CONTROL\_PIN*) == 1);  
 *uint8\_t* uartReceiving = (isReceiver) ? *UART\_ESP* : *UART\_CONSOLE*;  
 *uint8\_t* uartTransmitting = (!isReceiver) ? *UART\_ESP* : *UART\_CONSOLE*;  
   *gpio\_set\_level*(*SHOW\_MODE\_PIN*, isReceiver);  
 *ESP\_LOGI*(TAG, "Receiving on UART%d, Transmitting on UART%d",   
 uartReceiving, uartTransmitting);

   while (*true*) {

      if (isReceiver) {

         int len = *uart\_read\_bytes*(uartReceiving, (*uint8\_t* \*)data,   
 *BUFFER\_SIZE*, 10 / *portTICK\_PERIOD\_MS*);

         if (len > 0) {

            data[len] = '\0';

*print\_ascii\_art*(uartTransmitting, data);

         }

      } else {

*put\_str*(uartReceiving, "Enter data: ");

         data = *get\_line*(uartReceiving);

*put\_str*(uartTransmitting, data);

      }

   }

*free*(data);

}