Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería

SISTEMAS EMBEBIDOS

Práctica 5: SPI

Docente: Evangelina Lara Camacho

Alumnos:

Gómez Cárdenas Emmanuel Alberto - 01261509

Lizarraga Soto Diego - 01244912

Objetivo

El alumno se familiariza con el uso del periférico SPI 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 SPI half-duplex del ESP32**

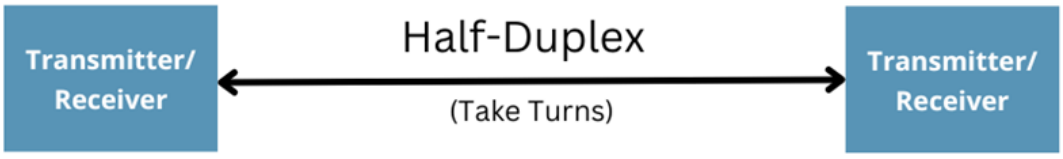
Definición: El SPI (Serial Peripheral Interface) half-duplex es un modo de comunicación en el que los datos pueden fluir en una sola dirección a la vez, es decir, el maestro puede enviar datos al esclavo o recibir datos del esclavo, pero no ambos simultáneamente.

1. Uso compartido del bus de datos: en lugar de requerir pines dedicados para MOSI (Master Out, Slave In) y MISO (Master In, Slave Out), el modo half-duplex permite compartir un solo pin para ambas funciones.
2. Alternancia de transmisión y recepción: el ESP32 alterna entre transmitir y recibir en el mismo pin, lo que se controla mediante el código.
3. Reducción de velocidad efectiva de comunicación: debido a la alternancia entre las operaciones, la velocidad efectiva de transmisión de datos puede ser menor que en el modo full-duplex, ya que el dispositivo necesita cambiar el sentido de los datos en el bus.
4. Compatibilidad con dispositivos SPI de un solo cable: algunos sensores o dispositivos SPI que usan un solo cable para comunicación de datos se benefician de este modo half-duplex.

Configuración de Hardware: SPI usa cuatro líneas principales:

- MOSI (Master Out Slave In): Datos del maestro al esclavo.
- MISO (Master In Slave Out): Datos del esclavo al maestro.
- SCK (Clock): Sincroniza la transferencia de datos.
- CS (Chip Select): Habilita a esclavos específicos.

ESP-IDF permite configurar el modo half-duplex SPI configurando la estructura spi_device_interface_config_t. Se activa el modo half-duplex estableciendo el campo flags en la configuración con el valor SPI DEVICE HALF DUPLEX.



Desarrollo

Implemente en un ESP 32 ESP-IDF una aplicación que descomprime un mensaje comprimido almacenado en un archivo de texto en una SD card, haciendo uso de **SPI y tareas.** La implementación debe ser eficiente en el uso de recursos de cómputo (procesador, memoria y periféricos).

El ESP32 está conectado por SPI a un adaptador de tarjetas micro SD card. El ESP32 recibe del usuario por UART el nombre de un archivo, buscá el archivo en la SD card, lee el contenido, lo descomprime e imprime en pantalla el texto resultante. Si el archivo no existe, despliega en pantalla “Archivo no encontrado”.

El mensaje comprimido consiste en lo siguiente:

- Consta únicamente de letras, números y corchetes.
- Cuándo hay un bloque de código dentro de los corchetes que consiste en un número y letras, significa que se tienen que repetir las letras la cantidad de veces indicada por el número.
- El mensaje puede tener varias capas de bloques.

Por ejemplo, el bloque [12AB] significa que se tiene que repetir 12 veces las letras AB.

- Puede basarse en el código “sd_card_example_main.c” sobre manejo de SD cards en el ESP32 ESP-IDF.

Ejemplo 1:

Mensaje en la SD Card: AB[3CD]

El programa imprime en pantalla: ABCDCDCD

Ejemplo 2:

Mensaje en la SD Card: AB[2C[2EF]G]

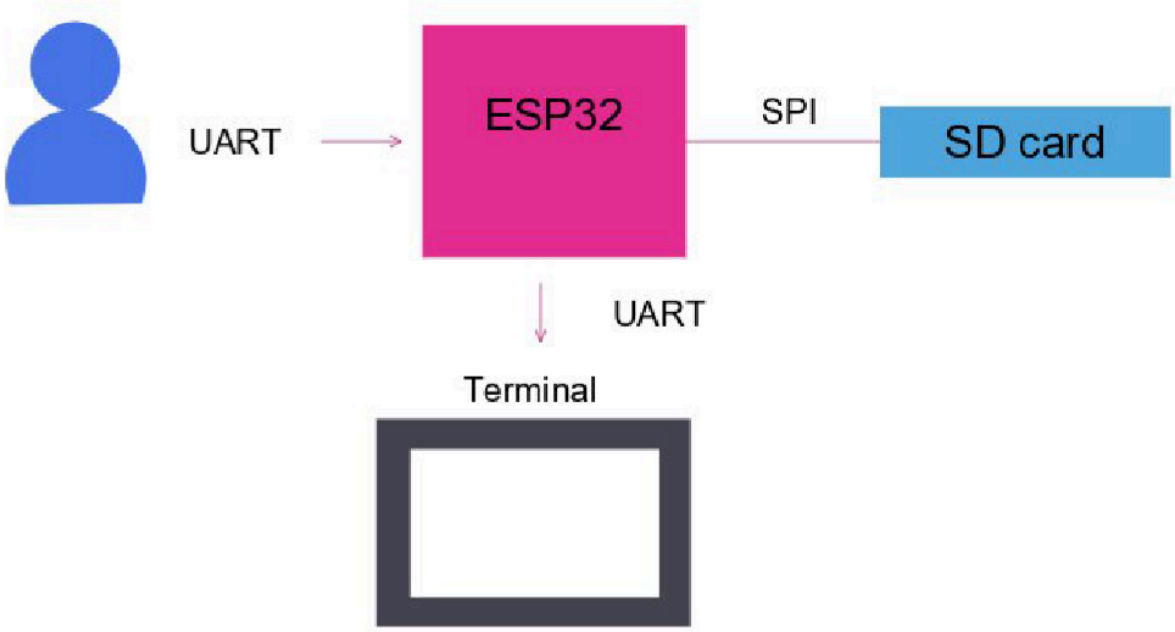
El programa imprime en pantalla: ABCEFEFGCEFEFG

Ejemplo 3:

Mensaje en la SD Card: IF[2A]LG[5M]D

El programa imprime en pantalla: IFAALGMMMMMD

Fig. 1. Diagrama a bloques.



Evidencia

Inicio

```
I (317) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (323) heap_init: At 4008F950 len 000106B0 (65 KiB): IRAM
I (331) spi_flash: detected chip: generic
I (334) spi_flash: flash io: dio
W (338) spi_flash: Detected size(4096k) larger than the size in the binary image header(2048k). Using the size in the binary image header.
I (352) main_taI (362) My Uart Library: UART setup complete
I (1362) gpio: GPIO[5]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (1362) sdspi_transaction: cmd=52, R1 response: command not supported
I (1402) sdspi_transaction: cmd=5, R1 response: command not supported
I (1422) sd_card: Filesystem montado
Name: GF8S5
Type: SDHC/SDXC
Speed: 5.00 MHz (limit: 5.00 MHz)
Size: 488960MB
CSD: ver=2, sector_size=512, capacity=1001390080 read_bl_len=9
SSR: bus_width=1
I (1432) sd_card:       Imprimiendo FileSystem: /sdcard
I (1432) sd_card:       ========================================
I (1442) sd_card:               Filename --> EJEMPLO4.TXT
I (1442) sd_card:               Filename --> EJEMPLO.TXT
I (1452) sd_card:               Filename --> EJEMPLO2.TXT
I (1452) sd_card:               Filename --> EJEMPLO3.TXT
I (1462) sd_card:               Filename --> EJEMPLO5.TXT
I (1462) sd_card:       ========================================
```

Ejemplo 1

```
Leer o escribir archivo? (r/w):
Introduce el nombre de archivo a leer: ejemplo.txt
I (44782) sd_card: Abriendo archivo /sdcard/ejemplo.txt
I (44792) sd_card: RAW values: 'AB[3CD]'

ABCDCDCD

I (44792) sd_card: Hacer alguna otra accion? (y/n):
```

Ejemplo 2

```
Leer o escribir archivo? (r/w):
Introduce el nombre de archivo a leer: ejemplo.txt
I (44782) sd_card: Abriendo archivo /sdcard/ejemplo.txt
I (44792) sd_card: RAW values: 'AB[3CD]'

ABCDCDCD

I (44792) sd_card: Hacer alguna otra accion? (y/n):
I (76812) sd_card:      Imprimiendo FileSystem: /sdcard
I (76812) sd_card:      ========================================
I (76812) sd_card:              Filename --> EJEMPLO4.TXT
I (76812) sd_card:              Filename --> EJEMPLO.TXT
I (76822) sd_card:              Filename --> EJEMPLO2.TXT
I (76822) sd_card:              Filename --> EJEMPLO3.TXT
I (76832) sd_card:              Filename --> EJEMPLO5.TXT
I (76832) sd_card:      ========================================


Leer o escribir archivo? (r/w):
Introduce el nombre de archivo a leer: ejemplo2.txt
I (84902) sd_card: Abriendo archivo /sdcard/ejemplo2.txt
I (84902) sd_card: RAW values: 'AB[2C[2EF]G]'

ABCEFEFGCEFEFG

I (84912) sd_card: Hacer alguna otra accion? (y/n):
```

Ejemplo 3

```
Leer o escribir archivo? (r/w):
Introduce el nombre de archivo a leer: ejemplo3.txt
I (106062) sd_card: Abriendo archivo /sdcard/ejemplo3.txt
I (106062) sd_card: RAW values: 'IF[2A]LG[5M]D'

IFAALGMMMMMD

I (106062) sd_card: Hacer alguna otra accion? (y/n):
```

Edge case testing

```
Leer o escribir archivo? (r/w):
Introduce el nombre de archivo a leer: ejemplo4.txt
I (134412) sd_card: Abriendo archivo /sdcard/ejemplo4.txt
I (134412) sd_card: RAW values: '[2ABC[3]D[2EF][H]IJ[]KL]'

ABCDEFEFHIJKLABCDEFEFHIJKL

I (134412) sd_card: Hacer alguna otra accion? (y/n):
```

Conclusiones y Comentarios.

Emmanuel Alberto Gómez Cárdenas: SPI es un protocolo de comunicación rápido y eficiente, la simplicidad con la que se maneja lo hace ideal para la transmisión de datos entre dispositivos a distancias cortas, como lo son los sensores, memorias, y demás.

Diego Lizarraga Soto: La práctica de comunicación SPI con una tarjeta SD permite transferir datos rápidamente entre un microcontrolador y el dispositivo de almacenamiento.Además, se destaca la importancia de comprender la configuración adecuada de pines y el manejo de comandos específicos para interactuar correctamente con la tarjeta SD.

Dificultades en el Desarrollo

La recursión es una herramienta que nos ayudó a resolver el problema de la descompresión, sin embargo, al ser muy sensible a caer en los problemas como el stack overflow o un uso de memoria ineficiente, nos tomó tiempo implementar la recursión correctamente.

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”]

```
#include <ctype.h>

#include <dirent.h>

#include <string.h>

#include <sys/stat.h>

#include <sys/unistd.h>


#include "driver/gpio.h"

#include "esp_vfs_fat.h"

#include "myUart.h"

#include "sdmmc_cmd.h"


#define EXAMPLE_MAX_CHAR_SIZE 64

#define MOUNT_POINT "/sdcard"


#define PIN_NUM_MISO 19

#define PIN_NUM_MOSI 23

#define PIN_NUM_CLK 18

#define PIN_NUM_CS 5


#define START_LOOP_CHAR '['

#define END_LOOP_CHAR ']'

#define END_LOOP_CHAR_TO_IGNORE 1


static const char *TAG = "sd_card";
```

```c
void list_files_in_directory(const char *path) {

  struct dirent *de;

  DIR *dr = opendir(path);



  if (dr == NULL) {

     ESP_LOGE(TAG, "Could not open directory: %s", path);

     return;

  }



  ESP_LOGI(TAG, "\tImprimiendo FileSystem: %s", path);

  ESP_LOGI(TAG, "\t=============================================");

  while ((de = readdir(dr)) != NULL) {

     ESP_LOGI(TAG, "\t\tFilename --> %s", de->d_name);

  }

  ESP_LOGI(TAG, "\t=============================================\n\n");

  closedir(dr);

}
```

```
static esp_err_t s_example_write_file(const char *path, char *data) {
  ESP_LOGI(TAG, "Abriendo archivo %s", path);
  FILE *f = fopen(path, "w");
  if (f == NULL) {
     ESP_LOGE(TAG, "Fallo abrir el archivo para escritura");
     return ESP_FAIL;
  }
  fprintf(f, "%s", data);  // Use "%s" to avoid formatting issues
  fclose(f);
  ESP_LOGI(TAG, "Archivo escrito");
  return ESP_OK;
}


int getLoopEndingChar(const char *str) {
  int auxIndex = 0;
  int openBrackets = 1;


  while (str[auxIndex] && openBrackets > 0) {
     char currentChar = str[auxIndex++];


     if (currentChar == START_LOOP_CHAR) {
        openBrackets++;
     } else if (currentChar == END_LOOP_CHAR) {
        openBrackets--;
     }
  }
  return openBrackets == 0 ? auxIndex : -1;
}
```

```
int8_t print_recursion_string(char *str) {

  int8_t auxIndex = getLoopEndingChar(str);

  if (auxIndex < 0) {

     ESP_LOGE(TAG, "Mismatched brackets.\n");

     return -1;

  }



  int counter = 0;

  int iterations = 0;



  while (isdigit((unsigned char)str[counter]) && counter < auxIndex) {

     iterations = iterations * 10 + (str[counter] - '0');

     counter++;

  }



  if (iterations == 0) iterations = 1;



  char *content_start = &str[counter];

  int content_length = auxIndex - counter - END_LOOP_CHAR_TO_IGNORE;
```

```
  for (int i = 0; i < iterations; ++i) {

     int j = 0;

     while (j < content_length) {

        if (content_start[j] == START_LOOP_CHAR) {

           j++;

           int8_t nestedIndex = print_recursion_string(&content_start[j]);

           if (nestedIndex < 0) return -1;

           j += nestedIndex;

        } else {

           putchar(content_start[j]);

           j++;

        }

     }

  }

  return auxIndex;

}
```

```c
void print_expanded_string(char *str) {

  while (*str) {

     if (*str == START_LOOP_CHAR) {

        int auxIndex = print_recursion_string(str + 1);

        if (auxIndex < 0) {

           ESP_LOGE(TAG, "Error\n");

           return;

        }

        str += auxIndex + 1;

     } else {

        putchar(*str++);

     }

  }

  putchar('\n');

}
```

```c
static esp_err_t s_example_read_file(const char *path) {
  ESP_LOGI(TAG, "Abriendo archivo %s", path);
  FILE *f = fopen(path, "r");
  if (f == NULL) {
     ESP_LOGE(TAG, "Fallo abrir el archivo para lectura");
     return ESP_FAIL;
  }
  char line[EXAMPLE_MAX_CHAR_SIZE];
  fgets(line, sizeof(line), f);
  fclose(f);
  // Strip newline
  char *pos = strchr(line, '\n');
  if (pos) *pos = '\0';
  ESP_LOGI(TAG, "RAW values: '%s'\n", line);
  print_expanded_string(line);
  printf("\n");
  return ESP_OK;
}
```

```
void app_main(void) {
  init_UARTs();
  // Give some time for the SD card to stabilize
  vTaskDelay(pdMS_TO_TICKS(1000));

  esp_vfs_fat_sdmmc_mount_config_t mount_config = {
      .format_if_mount_failed = true,
      .max_files = 5,
      .allocation_unit_size = 16 * 1024,
  };

  sdmmc_card_t *card;
  sdmmc_host_t host = SDSPI_HOST_DEFAULT();
  host.max_freq_khz = 5000;
  spi_bus_config_t bus_cfg = {
      .mosi_io_num = PIN_NUM_MOSI,
      .miso_io_num = PIN_NUM_MISO,
      .sclk_io_num = PIN_NUM_CLK,
      .quadwp_io_num = -1,
      .quadhd_io_num = -1,
      .max_transfer_sz = 4000,
  };
  ESP_ERROR_CHECK(spi_bus_initialize(host.slot, &bus_cfg, SDSPI_DEFAULT_DMA));
  sdspi_device_config_t slot_config = SDSPI_DEVICE_CONFIG_DEFAULT();
  slot_config.gpio_cs = PIN_NUM_CS;
  slot_config.host_id = host.slot;
  ESP_ERROR_CHECK(esp_vfs_fat_sdspi_mount(MOUNT_POINT, &host, &slot_config,
&mount_config, &card));
```

```
  ESP_LOGI(TAG, "Filesystem montado");

  sdmmc_card_print_info(stdout, card);

  char *aux = (char *)malloc(EXAMPLE_MAX_CHAR_SIZE);

  char *filename_path = (char *)malloc(EXAMPLE_MAX_CHAR_SIZE);

  if (!aux || !filename_path) {

     ESP_LOGE(TAG, "Memory allocation failed!");

  } else {

     while (true) {

        list_files_in_directory(MOUNT_POINT);



        put_str(UART_CONSOLE, "Leer o escribir archivo? (r/w): ");

        char action = get_char(UART_CONSOLE);



        if (action != 'r' && action != 'w') {

           ESP_LOGE(TAG, "Accion invalida");

           continue;

        }



        snprintf(aux, EXAMPLE_MAX_CHAR_SIZE, "\nIntroduce el nombre de archivo a %s:
", (action == 'r') ? "leer" : "ecribir");

        put_str(UART_CONSOLE, aux);

        char *data = get_line(UART_CONSOLE);

        if (data == NULL) {

           ESP_LOGE(TAG, "Error leyendo nombre de archivo!");

           continue;

        }

        snprintf(filename_path, EXAMPLE_MAX_CHAR_SIZE, "%s/%s", MOUNT_POINT, data);

        esp_err_t ret;
```

```
        if (action == 'r') {
            ret = s_example_read_file(filename_path);
            if (ret == ESP_FAIL) {
                ESP_LOGE(TAG, "Error leyendo archivo!");
            }
        } else {
            put_str(UART_CONSOLE, "\nIntroduce datos a escribir: ");
            data = get_line(UART_CONSOLE);
            if (data == NULL) {
                ESP_LOGE(TAG, "Error obteniendo data a escribir!");
                continue;
            }
            ESP_ERROR_CHECK(s_example_write_file(filename_path, data));
        }
        ESP_LOGI(TAG, "Hacer alguna otra accion? (y/n): ");
        if (get_char(UART_CONSOLE) != 'y') {
            break;
        }
    }
    // Free allocated memory
    free(filename_path);
    free(aux);


    esp_vfs_fat_sdcard_unmount(MOUNT_POINT, card);
    ESP_LOGW(TAG, "Tarjeta desmontada");
    spi_bus_free(host.slot);
  }
}
```