Universidad Autónoma de Baja California

Ingeniería en Computación

Facultad de Ciencias Químicas e Ingeniería

Microcontroladores

Práctica 4. Programación del uC del periférico de comunicación serie utilizando interrupciones.

Alumno: Zavala Román Irvin Eduardo

Docente: Castro Gonzales Ricardo

Grupo: 561

31/03/2022

Periodo 2022-1

Objetivo

Mediante esta práctica el alumno aprenderá el uso básico para inicializar y operar, bajo un esquema de interrupciones, el puerto serie del microcontrolador.

Material

- Computadora Personal y tarjeta de desarrollo del ESP32.

Teoría

Manejo del periférico de comunicación serie (UART) del microcontrolador ESP32

El UART es una característica de hardware que se encarga de la comunicación usando interfaces de comunicación serial asíncrona, permitiendo el intercambio de datos full duplex o half duplex entre dispositivos. El ESP32 tiene 3 controladores UART con set de registros idénticos, cada controlador es configurable independientemente de los otros.

El primer paso para poder utilizar los UART es configurar los parámetros de configuración, que puede ser con la función `uart_param_config()` y pasandola a la estructura `uart_config_t` o llamar funciones para configurar parametros especificos.

El segundo paso consiste en configurar los pines GPIO para conectar el otro dispositivo con la función `uart_set_pin()`.

```
// Set UART pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)

ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, 4, 5, 18, 19));
```

El último paso de configuración consiste en instalar drivers con la función `uart_driver_install()` con los parámetros de buffer Tx, Ex, tamaño de la cola y banderas para interrupciones.

```
// Setup UART buffered IO with event queue
```

```
const int uart_buffer_size = (1024 * 2);
QueueHandle_t uart_queue;
// Install UART driver using an event queue here
ESP_ERROR_CHECK(uart_driver_install(UART_NUM_2, uart_buffer_size, \
                                        uart_buffer_size, 10,
&uart_queue, 0));
```

Para comunicarse se realiza un proceso basado en que la comunicación se basa en una máquina de estados finita (FSM), para transmitir los pasos son: escribir datos en el buffer Tx, el FSM serializa los datos y por último FSM envía los datos. El proceso para recibir información es parecido pero al revés y el último paso recibe datos.

Las interrupciones se usan para estados específicos del UART o errores, se pueden habilitar o deshabilitar interrupciones con las funciones `uart_enable_intr_mask()` y `uart_disable_intr_mask()`, las interrupciones se dividen en detección de eventos, límite de espacio FIFO o tiempo de transmisión superado y detección de patrones.

Secuencias de escape ANSI

Un código ansi es una secuencia especial de caracteres que un terminal de texto, en vez de mostrar, interpreta como comandos y "ejecuta". La estructura empieza con ^[que representa el ESC en ASCII seguido del resto del código que son cadenas ASCII. Como son bastantes códigos se pueden ver todos en la siguiente página con su nombre, descripción y código:

https://www.csie.ntu.edu.tw/~r92094/c++/VT100.html

Instrucciones

1. Implementar las siguientes funciones:
   - `void uartInit(uint8_t com, uint16_t baudrate, uint8_t size, uint8_t parity, uint8_t stop)`
   - `void uartGets(uint8_t com, char *str)`
   - `void uartPuts(char *str)` → `void uartputs(uint8_t com, char *str)`
   - `void myItoa(uint16_t number, char* str, uint8_t base)`
   - `uint16_t myAtoi(char *str)`
   - `void uartSetColor(uint8_t com, uint8_t color)`

- `void uartGotoxy(uint8_t com, uint8_t x, uint8_t y)`

Es necesario interconectar la transmisión del UART1 hacia la recepción del UART2.

Desarrollo

La primera función modificada fue uartInit para que dependiendo del número de puerto que utilice se seleccione diferente pin.

```
if(com == 0)
    ESP_ERROR_CHECK(uart_set_pin(com, 1, 3,
                                 UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE));
if(com == 1)
    ESP_ERROR_CHECK(uart_set_pin(com, 10, 9,
                                 UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE));
if(com == 2)
    ESP_ERROR_CHECK(uart_set_pin(com, 17, 16,
                                 UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE));
```

La función más fácil de implementar fue uartPuts ya que simplemente se imprimía con uartPutchar los valores apuntador por str hasta encontrar el caracter nulo.

```
while(*str != '\0'){
    uartPutchar(com, *str);
    str++;
}
```

Las funciones uartSetColor y uartGotoxy se hicieron usando la función strcat para poder concatenar los parámetros de las secuencias de escape en un solo string y así escribirlos en el uart (no se usó otro método como sprintf ya que imprimía basura), para convertir enteros a string se usó la función myItoa que se explicará después. Para el color se usó la secuencia “x\1B[3Xm” y para el Gotoxy “\x1B[x,yf”.

```
// Gotoxy
char x_str[20];
char y_str[20];
myItoa(x,x_str, 10);
myItoa(y,y_str, 10);
char caGotoxy[50] = "\x1B[";
```

```
strcat(caGotoxy, x_str);
strcat(caGotoxy, ";");
strcat(caGotoxy, y_str);
strcat(caGotoxy, "f");
uart_write_bytes(com, caGotoxy, sizeof(caGotoxy));
```

```
//Setcolor
const char color_str[50];
myItoa(color,color_str, 10);
char setColor[50] = "\x1B[";
strcat(setColor, color_str);
strcat(setColor, "m");
uart_write_bytes(com, setColor, sizeof(setColor));
```

La función myItoa consta de 2 partes importantes, la parte de conversión del número a la base pedida y la parte en la que se recorren los valores obtenidos en orden inverso sumando el carácter ‘0’ para números menores que 10 y ‘7’ para mayores, esto para convertirlo a char. Al final se agrega un carácter nulo para indicar cuando acaba la cadena.

```
int base_n[16], i;
if(number == 0){*str = '0';str++;}
for(i = 0; number > 0; i++){
    base_n[i] = number % base;
    number /= base;
}
for(i = i - 1; i >= 0 ; i--) {
    if(base >= 11){
        if(base_n[i] < 10)
            *str = base_n[i] + '0';
        else
            *str = base_n[i] + '7';
    }
    else
        *str = base_n[i]+'0';
    str++;
}

if(*str != '\0')
    *str = '\0';
```

Para convertir de string a int de 16 bits con myAtoi también consta de 2 partes, se recorre la cadena guardando los caracteres - ‘0’ para convertirlos a int y se guardan en un arreglo, si se detecta un número diferente de la base decimal o más de 5 dígitos se detiene el proceso para evitar caracteres indeseados. La otra parte recorre el arreglo de enteros de manera inversa sumando los dígitos dependiendo de su posición (unidad, decena, centena, etc) y se guarda en una variable que será retornada.

```
int number[5], i, j = 0;
uint16_t n = 0;

for(i = 0; *str != '\0';i++){

    if(*str-'0' < 10 && *str-'0' >= 0){
        number[i] = *str - '0';
        str++;
    }
    else{
        str++;
        break;
    }
    if(i == 5) return 0;

}
for(i = i-1; i >= 0; i--){
    n += (int)pow(10, j)*number[i];
    j++;
}

return n;
```

La última función es uartGets, esta es la que requiere más cuidados ya que al ser la que interactúa directamente con el usuario los errores de esta son bastante evidentes. Consiste en un while que se detiene al detectar un enter, si no es enter y es un caracter válido que guarda en el arreglo y lo escribe en el uart, si es un backspace se usan las secuencias “\e[1D” y “\e[0K” para retroceder un lugar y borrar el último carácter, se decrementa el contador del índice del string y se coloca un carácter nulo para que al imprimirlo no coloque basura.

```
int i = 0;
while(1){
    char a = uartGetchar(com);
    if(a != 13){
        if(a == 127 || a == 8){
            uartPuts(com, "\e[1D");
```

```
            uartPuts(com, "\e[0K");
            i--;
            if(i < 0) i = 0;
            str[i] = '\0';
            uart_flush(com);
            uart_flush_input(com);
        }
        else{
            uartPutchar(com,a);
            str[i] = a;
            i++;
        }

    }
    else break;
}
str[i] = '\0';
```

Conclusiones

El UART es un protocolo muy importante manejado por una gran variedad de microcontroladores para poder mandar y recibir información entre diferentes dispositivos, aunque no es tan fácil, si se conoce como funciona el saber como comunicar microcontroladores puede ampliar la escala de los proyectos a depender de uno solo que se encargue de todo.

Bibliografía

Chuidiang (2007). *Codigos ANSI*. Consultado el 26 de marzo de 2022, desde http://www.chuidiang.org/clinux/ansi/ansi.php

CSIE(1997). *VT100 escape codes*. Consultado el 26 de marzo de 2022, desde https://www.csie.ntu.edu.tw/~r92094/c++/VT100.html

Universal Asynchronous Receiver/Transmitter (UART) - ESP32 - — ESP-IDF Programming Guide latest documentation. (s.f.). Technical Documents. Consultado el 26 de marzo de 2022, desde https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/uart.html