Universidad Autónoma de Baja California

Ingeniería en Computación

Facultad de Ciencias Químicas e Ingeniería

Microcontroladores

Práctica 2. Manejo de la sección de E/S del microcontrolador ESP32

Alumno: Zavala Román Irvin Eduardo

Docente: Castro Gonzales Ricardo

Grupo: 561

10/03/2022

Periodo 2022-1

Objetivo

Mediante esta práctica el alumno analizará la implementación de retardos por software, así como también se familiarizará con la configuración y uso de puertos.

Material

- Computadora Personal y tarjeta de desarrollo del ESP32.

Teoría

Técnicas de anti-rebote de botones

El rebote de un pulsador es el ruido que se obtiene por el botón al ser presionado, antes de estar en estado alto el pulsador crea muchas pulsaciones pequeñas consideradas como ruido y señales indeseables. Para esto existen soluciones por hardware y software, dependiendo de la aplicación una puede ser más efectiva que otra.

Antirrebote por hardware

Este método es deseable cuando el microcontrolador no tiene gran potencia de procesamiento y el usar los limitados recursos disponibles para mitigar los rebotes por software no es lo indicado. Los métodos más comunes es usar una red resistencia - capacitor (a veces con diodos o más resistencias), el uso de latch SR usado por IBM en la década de los 60 y otra opción es usar un dispositivo dedicado como el MX14490DWG.

Antirrebote por software

La manera más común para eliminar el rebote por software es que al obtener una pulsación esperar un tiempo determinado para obtener el estado definitivo del botón. Este tiempo de rebote varía en la literatura, ya que dependiendo de la fuente consultada pueden aparecer valores desde 1 ms hasta 6 ms, y en aplicaciones industriales se recomienda 20 ms.

El pseudocódigo de esta implementación quedaría como:

```
//Fuente
https://electronicprojects9.tumblr.com/post/141491477942/algor
itmo-antirrebote-o-debounce

if(tempo de revisar el botón){
     lee el estado del botón
     if(estado actual != estado anterior){
          incrementa contador
          if(contador >= 5){
               cambio del estado del dispositivo
               resetear contador
          }
     }
     else
          resetea contador
}
```

Otra opción sin contador y con un condensador en el pulsador:

```
//Fuente
http://arduparatodos.blogspot.com/2017/01/pulsadores-y-antirre
bote-con-arduino.html

if(pulsador == bajo){
          presionado = 1
          if(pulsador == 1 y presionado == 1){
               //Realizar acción
               presionado = 0
          }
}
```

Charlieplexing (aplicado para control de LEDs)

Esta es una técnica para controlar una matriz de LEDs con una cantidad reducida de pines I/O. El uso del charlieplexing permite controlar hasta N*(N-1) diodos led donde N es el número de pines disponibles. En la forma más simple, este método funciona con pares complementarios como en la figura 1:

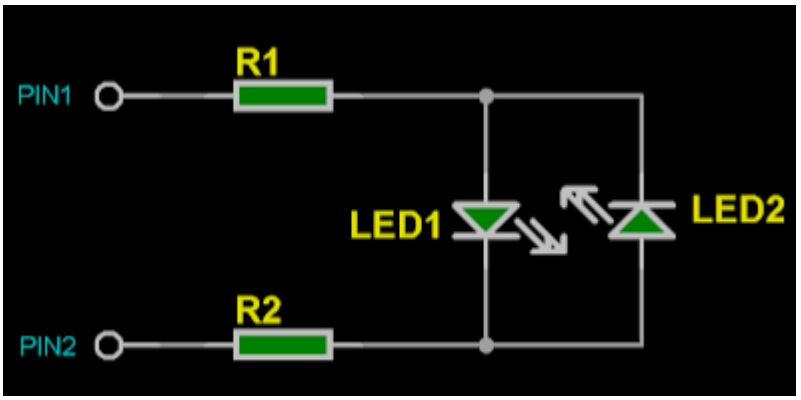

Figura 1. Charlieplexing mas simple

Fuente:
http://electrocirc.blogspot.com/2012/05/tecnica-charlieplexing-para-controlar.html

A partir de 3 pines, se debe desconectar (poner en alta impedancia) uno de los pines antes de energizar los otros 2, esto se resuelve con propiedades lógicas.

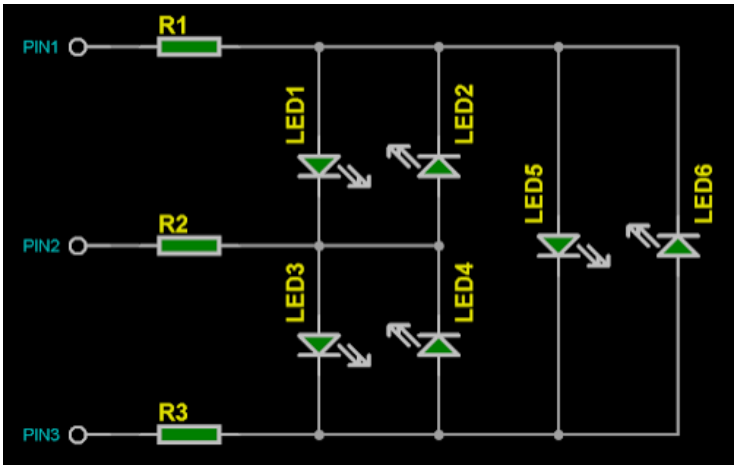

Figura 2. Charlieplexing con 3 pines

Fuente:
http://electrocirc.blogspot.com/2012/05/tecnica-charlieplexing-para-controlar.html

https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html

Sabiendo de antemano que es necesario realizar la configuración de las herramientas de desarrollo, puesto que PlatformIO ya lo encapsula.

Familiarizarse con la tarjeta de desarrollo que han adquirido:
https://docs.espressif.com/projects/esp-idf/en/stable/esp32/hw-reference/esp32/get-started-devkitc.html (Buscar el esquemático correcto si es que compraron uno diferente)
Leer sección de los periféricos de entrada y salida del SDK:
https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/gpio.html

Una vez realizada la lectura continuamos con otro ejemplo introductorio, el cual se encuentra dentro del folder /print example/. Dentro de este ejemplo podrán ver como se podría usar printf() al incluir (y el SDK se encarga de conectar la tubería). Para poder ver las impresiones, es necesario el uso de una terminal (que ya fue cubierta en la primera lectura). Así mismo, en este ejemplo también se muestra la configuración de un puerto de entrada al cual podemos conectar un botón y el LED de la tarjeta de desarrollo se encenderá mientras este presionado el botón.

Modificaciones a realizar: Ahora pasamos al folder de /game/, dentro de este ya encontraran la lógica de un pequeño juego de reacción de tiempo. Antes de empezar a modificar el código, van a ocupar hacer un cambio dentro de los archivos generados por PlatformIO. Abrir el archivo .pio\build\esp-32s\config\ sdkconfig.h, cambiar la línea: #define CONFIG_FREERTOS_HZ 100 por: #define CONFIG_FREERTOS_HZ 1000 Eso logrará que nuestros retardos puedan tener resolución de 1 milisegundo.

Funciones a implementar:

void initIO(void);

uint8_t checkBtn(void);

void updateLeds(uint8_t gameState)

Desarrollo

La primera parte de la práctica que consiste en probar el printf con PlatformIO se realizó en las primeras horas de laboratorio sin complicaciones.

La primera función que se pide modificar es initIO para iniciar los puertos que se van a utilizar, esta función fue la parte más sencilla de la práctica ya que en la clase se explicó cómo poner los puertos para poder operar correctamente.

```
static void initIO(void){
    gpio_reset_pin(LED_GPIO_12);
    gpio_reset_pin(LED_GPIO_13);
    gpio_reset_pin(LED_GPIO_14);
    gpio_reset_pin(LED_GPIO_27);
    gpio_reset_pin(BTN_GPIO);
    gpio_reset_pin(BTN_GPIO_5);

    /* Set LED GPIO as a push/pull output */
    gpio_set_direction(LED_GPIO_12, GPIO_MODE_INPUT);
    gpio_set_direction(LED_GPIO_13, GPIO_MODE_INPUT);
    gpio_set_direction(LED_GPIO_14, GPIO_MODE_INPUT);
    gpio_set_direction(LED_GPIO_27, GPIO_MODE_INPUT);
    gpio_set_direction(BTN_GPIO_5, GPIO_MODE_OUTPUT);

    /* Set LED GPIO as a push/pull output */
    gpio_set_direction(BTN_GPIO, GPIO_MODE_INPUT);
    gpio_pullup_en(BTN_GPIO);
}
```

La función checkBtn se complicó un poco, para poder crear diferencias de tiempos se crearon variables globales t1_boton,t2_boton y pulse para poder detectar la longitud del pulso, el uso de pulse es para cambiar si tomar el tiempo 1 o el 2 y calcular el tiempo desde que se detectó el inicio de la pulsación al final.

```
uint8_t checkBtn(void){
    printf("MILLIS %d\tT2-T1 = %d\n", _millis, t2_boton-t1_boton);
```

```
    if(!gpio_get_level(BTN_GPIO) && pulse == 0){
        t1_boton = _millis;
        pulse = 1;
    }
    if(pulse == 1 && !gpio_get_level(BTN_GPIO)){
        t2_boton = _millis;
        if(t2_boton-t1_boton >= 1000){
            t2_boton = 0;
            t1_boton = 0;
            return eBtnLongPressed;
        }

    }
    if(gpio_get_level(BTN_GPIO)) pulse = 0;
     if(pulse == 0 && t2_boton-t1_boton < 1000 && t2_boton-t1_boton > 30){
t2_boton = 0; t1_boton = 0; return eBtnShortPressed;}
    return eBtnUndefined;
}
```

La última función pedida es updateLeds y la que tomó más tiempo en esta práctica ya que al ser la que tiene más impacto en la interactividad se trato de tener el mejor resultado posible, aunque el código es largo, la estructura de todos los casos gameState es similar:

```
        //eYouLoose
        if(start_count == 0){
            start_count = 1;
            t1_led = _millis;
        }
        if(start_count == 1){
            t2_led = _millis;
            if(t2_led-t1_led <= 500){
                encenderled(1);delayMs(1);_millis++;
                encenderled(2);delayMs(1);_millis++;
                encenderled(3);delayMs(1);_millis++;
                encenderled(4);delayMs(1);_millis++;
            }
            else{
                if(t2_led-t1_led <= 1000){
                    encenderled(5);delayMs(1);_millis++;
                    encenderled(6);delayMs(1);_millis++;
                    encenderled(7);delayMs(1);_millis++;
                    encenderled(8);delayMs(1);_millis++;
                }
                else{
                    start_count = 0;
                }
            }
        }
```

Se puede observar el uso de una variable global llamada start_count para tomar el primer tiempo cuando se entró al estado y ahora en vez de usar t1_boton/t2_boton se usa t1_led/t2_led porque si se unifican en t1/t2 o algo por el estilo los tiempos se combinan y el resultado sería errático. La siguiente parte se puede observar es que hay if's para prender leds por los ms requeridos usando _milllis y no delays.

Aparte de estas funciones obligatorias se usaron otras que se implementaron por la abstracción realizada, la función encenderled(uint8_t led) permite encender un solo led recibido en un parámetro entero entre 1 y 8, esta se usa dentro de updateLeds para realizar los patrones especificados. Otra función extra incluida es setZeros() y lo que hace es poner los pines (si es que están en output) en nivel lógico 0, apagando todos los leds, esto se usa para interrumpir ciertas secuencias y evitar leds encendidos por más tiempo de lo que deberían.

Algunas secciones del código principal dentro del while fueron modificadas para acercarse un poco más al video dado como ejemplo como una modificación para que el pulso corto no cambie el estado eYouLoose o eYouWin sino que solo lo haga con pulsaciones largas. La última modificación importante fue el cambio de la variable countdown de usar la función esp_random al rand de C, delimitando el tiempo que tarda el juego a uno más corto y rápido.

Conclusiones

El manejar puertos es lo más básico para poder hacer que el microcontrolador se comunique con el mundo, por ahora se manejan leds pero en el futuro se podrán usar circuiteria externa o sensores un poco más complejas. El uso del charlieplexing es una técnica bastante interesante ya que sin complicar demasiado el circuito se puede ahorrar un recurso tan importante y limitado como lo son los pines del ESP32, conceptos aprendidos para prender los leds serán de gran utilidad en prácticas futuras ya que nunca esta de mas usar retroalimentación visual al usuario.

Bibliografía

Clive "Max" Maxfield. (2021). *Implementación del rebote del interruptor de hardware | DigiKey*. Digikey. Consultado el 24 de febrero de 2022, desde https://www.digikey.com/es/articles/how-to-implement-hardware-debounce-for-switches-and-relays

Ecuarobot. (2020). *Charlieplexing Arduino: control de 12 LED con 4 pines GPIO*. EcuaRobot. Consultado el 24 de febrero de 2022, desde https://ecuarobot.com/2020/05/30/charlieplexing-arduino-control-de-12-led-con-4-pines-gpio/

Electronic Projects — Algoritmo antirrebote o debounce. (2016). Electronic Projects. Consultado el 24 de febrero de 2022, desde https://electronicprojects9.tumblr.com/post/141491477942/algoritmo-antirrebote-o-debounce

Hernández, L. d. V. (n.d.). *Controla una matriz de LEDs con Arduino con Charlie Plexing*. Programar fácil. Consultado el 24 de febrero de 2022, desde https://programarfacil.com/blog/controlar-matriz-de-leds-con-arduino/

Técnica Charlieplexing para controlar Múltiples LEDs con pocos pines. (2012, May 18). Electrónica y circuitos. Consultado el 24 de febrero de 2022, desde http://electrocirc.blogspot.com/2012/05/tecnica-charlieplexing-para-controlar.html

Vega, E. (2017). *Pulsadores y antirrebote con Arduino*. Arduino para todos. Consultado el 24 de febrero de 2022, desde http://arduparatodos.blogspot.com/2017/01/pulsadores-y-antirrebote-con-arduino.html