UNIVERSIDAD NACIONAL DE COLOMBIA

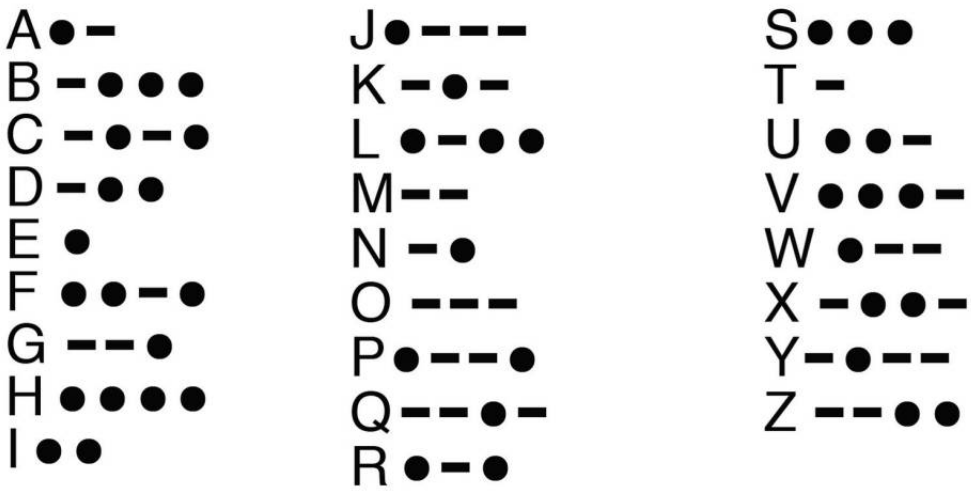
Octavo proyecto
Código Morse

Alejandro Ramírez Jaramillo

Universidad Nacional de Colombia sede Manizales Email: alramirezja@unal.edu.co

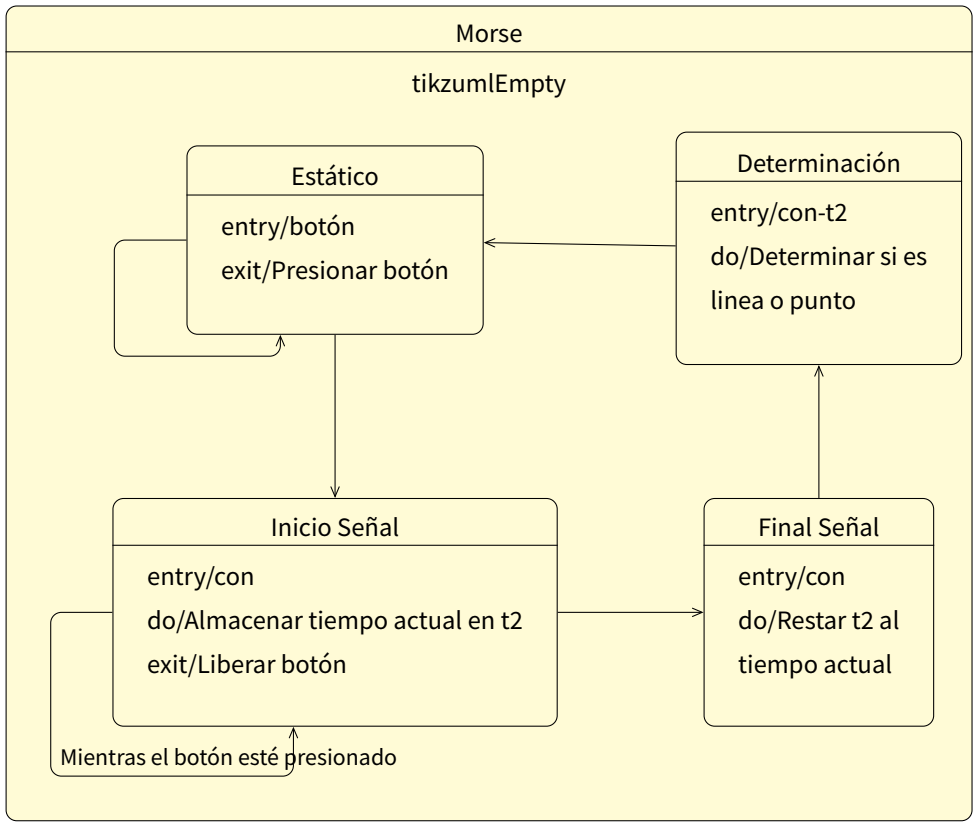
Problema

Se busca establecer comunicación entre un individuo y un microcontrolador, sin embargo solo se cuenta con un botón para esto. Como solución usted propone que le comunicación se haga a través de código morse, de manera que si el usuario oprime el botón por más de 5 segundos se interprete como una linea y si es por menos tiempo se interpretará como un punto. Prepare un programa que permita identificar si el usuario esta transmitiendo un punto o una línea y que guarde esta información.



1 Solución

1.1 Máquina de estados



- Estático: El microcontrolador espera la señal de entrada, el tiempo durante este estado no altera la interpretación de linea o punto.
- Inicio Señal: Al ser presionado el botón se ejecuta la interrupción y se almacena el tiempo actual extraído de la base de tiempo.
- Final Señal: Cuando el botón es liberado nuevamente se acciona la interrupción y se le resta el tiempo actual el medido en el estado anterior.
- Determinación: Teniendo en cuenta la diferencia de tiempo calculada en el estado anterior se determina si la señal correspondía a un punto o a una linea.

1.2 Código

Para el ejercicio se usará: El Timer, la interrupción externa y las GPIO. Hay tres secciones de código que darán forma a la maquina de estados:
En primer lugar el manejador del Timer, el cual esta configurado para ejecutarse cada 2 segundos (factor común de los tiempos entre cambios de estado) y aumentar en 1 el valor de un contador con el cual podremos llevar registro del tiempo. Para comenzar el código, se inicializan las variables y funciones que se van a usar.

1.2.1 Variables globales

```
int boton=1;
int con=0;
int t2=0;
volatile uint32_t i = 0;
volatile uint32_t code[10];
/**********************************************
* function declarations
**********************************************/
int main(void);
void GPIO_init(void);
void Ext_init(void);
void TIMER_init(void);
void morse(void);
```

- boton: Indica si el botón esta siendo presionado o no, 0 para "esta presionado" y 1 para "sin presionar".
- con: Base de tiempo del sistema, se usará determinar cuanto tiempo se ha estado oprimiendo el botón. Esta base de tiempo nunca debe ser alterada, para que pueda ser usada en distintas máquinas de estado.
- lock: Bandera que indica cuando se debe leer el valor del contador actual, se inicializa dentro de la función "morse". Permite diferenciar el estado Estático del estado Final Señal, pues en ambos casos el botón NO esta siendo oprimido.
- t2: Esta variable almacena el tiempo cuando el botón es presionado.
- i: Variable que va a indicar la posición para guardar las lineas y puntos según se determine.
- code[10]: Vector que almacenará los caracteres "." o "_" del estado Determinación en forma de código ASCII.
- GPIO_init(): Inicializa el pin 13 del puerto C que esta conectado al botón.
- Ext_init(): Inicializa la interrupción externa, para más información remitirse a la guía de interrupción externa.
- TIMER_init(): Inicializa el Timer que se va a usar como base de tiempo, contando los segundos y almacenando el tiempo en la variable con. Para más información remitirse a la guía de TIMER.
- morse(): Máquina de estados que cumplirá la función de medir los tiempos de la señal y determinar si corresponde a un punto una linea.

1.2.2 Base de tiempo (TIM2_IRQHandler)

El TIMER aumentará en 1 el contador "con" cada 0.1 segundos, esto se usará para conocer el tiempo de ejecución del programa con el cual se calculará el tiempo que el botón ha sido oprimido. Esta base de tiempo no debe ser alterada para poder usarla en múltiples maquinas de estado a la vez sin la necesidad de crear un nuevo contador para cada una.

```
void TIM2_IRQHandler(void)
{

    // clear interrupt status
    if (TIM2->DIER & 0x01) {
        if (TIM2->SR & 0x01) {
            TIM2->SR &= ~(1U << 0);
        }
    }
                con +=1;
}
```

1.2.3 Botón (EXTI15_10_IRQHandler)

Al presionar o liberar el botón se alterna el valor del la variable "boton" entre 0 y 1, respectivamente. Esta interrupción se ejecuta al detectar filo de subida o bajada.

```
void EXTI15_10_IRQHandler(void)
```

```
{
        //Check if the interrupt came from exti13
        if(EXTI->PR1 & (1 <<13)){
        //Cambio de valor para la variable que indica
        //cuando se presiona  y cuando se libera
                                boton ^=0x1;
                        // Clear pending bit
                EXTI->PR1 = 0x00002000;
        }
}
```

1.2.4 Main

```
int main(void)
{
        //Se inicializa las GPIO que se van a usar (Puerto A y C)
        GPIO_init();
        // Se configura la interrupci n externa
        Ext_init();
        //Inicializamos el TIMER que marcara el tiempo de la secuencia
        TIMER_init();
        morse();

        while(1)
        {

        }

        __asm__("NOP"); // Assembly inline can be used if needed
        return 0;
}
```

1.2.5 GPIO_init()

Se configura el pin 13 del puerto C como una entrada para la interrupción externa.

```
void GPIO_init(void){
        RCC->AHB2ENR |= 0x00000005;
        // Make GPIOD Pin13 as input pin (bits 27:26 in MODER register)
        GPIOC->MODER &= 0xFFFFFFFF;             // Clear bits 27, 26 for P13
        GPIOC->MODER &= 0xF3FFFFFF;             // Write 00 to bits 27, 26 for P13
}
```

1.2.6 Ext_init()

Configura la interrupción externa, escogiendo el pin 13 del puerto C como la entrada de la señal de interrupción y haciendo que detecte los filos de subida y bajada.

```
void Ext_init(void){
    // enable SYSCFG clock
        RCC->APB2ENR |= 0x1;
        // Writing a 0b0010 to pin13 location ties PC13 to EXT4
        SYSCFG->EXTICR[3] |= 0x20; // Write 0002 to map PC13 to EXTI4
        // Choose either rising edge trigger (RTSR1) or falling edge trigger (FTSR1)
        EXTI->RTSR1 |= 0x2000;   // Enable rising edge trigger on EXTI4
        EXTI->FTSR1 |= 0x2000;   // Enable falling edge trigger on EXTI4
        // Mask the used external interrupt numbers.
        EXTI->IMR1 |= 0x2000;    // Mask EXTI4
        // Set Priority for each interrupt request
        NVIC->IP[EXTI15_10_IRQn] = 0x10; // Priority level 1
        // enable EXT0 IRQ from NVIC
        NVIC_EnableIRQ(EXTI15_10_IRQn);
}
```

1.2.7 TIMER_init()

El Timer funcionará como una interrupción que se ejecutará cada 1 segundo, para esto se escoge el preescalador y el valor del tope (ARR). Este tiempo se almacenará en el contador "con".

```
void TIMER_init(void){
    // enable TIM2 clock (bit0)
    RCC->APB1ENR1 |= (1<<0);
    TIM2->PSC = 7999;
    TIM2->ARR = 40;
    // Update Interrupt Enable
    TIM2->DIER |= (1 << 0);
    NVIC_SetPriority(TIM2_IRQn, 2); // Priority level 2
    // enable TIM2 IRQ from NVIC
    NVIC_EnableIRQ(TIM2_IRQn);
    // Enable Timer 2 module (CEN, bit0)
    TIM2->CR1 |= (1 << 0);
}
```

1.2.8 Máquina de estados

```
 void morse(void){
//Lock es una bandera que permite
        int lock=0;
        while(1){
//Al presionarse el boton, se detecta un flanco y la interrupci n cambia las variables para
     entrar
//al primer if solo una vez
                if ((boton==0)&(lock==0)){ //Estado: Inicio se al
                        t2=con;
                        lock=1;
                }
//Cuando se libera el boton entra al segundo if
                if ((t2>0)&(boton==1)){ //Estado: Final Se al
// si el tiempo que el boton paso es menor a 5 segundos, es un punto.
                        //Estado: Determinacion
                        if (((con-t2)>10)&((con-t2)<30)){
                                code[i]='.';
                                i++;
                        }
// Si es mayor, es una linea.
                        else{
                                code[i]='_';
                                i++;
                        }
//Entonces se reinicia la variable de tiempo t2
                        t2=0;
                        lock=0;
                }
        }
 }
```

2 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ STM32l476rg datasheet.