INGENIERIA EN MICROCONTROLADORES

Maquinas de Estado

Introducción

MAQUINA DE ESTADOS

www.i-micro.com

© Ingeniería en Microcontroladores
Teléfono 044 55 11 29 55 05
E-mail: cursos@i-micro.com
elp@i-micro.com

Maquinas de Estado

Las máquinas de estado son una parte integral de la programación de software. Las máquinas de estado hacen al código más eficiente, más fácil de depurar y ayudan a organizar el flujo del programa.

Una Una Máquina de Estado Finita (FSM = Finite State Machine) está basada en la idea de que hay un número finito de estados para un sistema determinado. Por ejemplo, cuando una aplicación enciende o apaga un LED, existen dos estados; un estado es cuando el LED está encendido y el otro cuando está apagado.

Las máquinas de estado requieren una Variable de Estado (State Variable - SV). La variable de estado es un apuntador que mantiene un control del estado en que se encuentra el microcontrolador y dirige el flujo del programa al modulo de software correspondiente.

La variable de estado puede modificarse en los módulos (o estados) de software por si misma o por una función externa.

La primera ventaja de utilizar las máquinas de estado, es que promueve buenas técnicas de diseño de firmware. Cuando se comience a implementar una aplicación, piense sobre que estados son necesarios para que funcione la aplicación. Haga una lista de todas las piezas, o estados, de una aplicación y después explore como se relacionan entre sí. Esto ayudará a prevenir se desarrollen bucles en el código. Esta forma de pensar también lleva al desarrollo de una herramienta de ingeniería muy útil – el diagrama de flujo.

Las máquinas de estado tienen una característica muy importante. Siempre regresan a un punto en el código, en el cual se canaliza el flujo del programa por la variable de estado, al correspondiente modulo de software. Esto provee varias ventajas:
Primero, esta característica hace simple el llamado de tareas repetitivas.

Como por ejemplo, el refrescar el perro guardián (watchdog timer) de un microcontrolador, revisar la veces que se presiona un botón de entrada o comunicarse con una computadora que requiere una comunicación periódica, estos son ejemplos de tareas repetitivas.

Una alternativa al uso de máquinas de estado es usar el código de bucles o loops infinitos. Para que el código de bucles maneje tareas repetitivas, las funciones que manejan estas tareas deben estar distribuidas a través del código en cada uno de los bucles. **Esto no solo es altamente ineficiente sino que también es confuso para entenderlo**.

La figura 1 muestra un diagrama de bloques, el cual ejemplifica cómo luciría el código ejemplo si una máquina de estado no fuera utilizado. Comparado con la figura 2, que muestra un diagrama de bloque de un código basado en la máquina de estado, queda claro que usar una máquina de estado disminuye la longitud del código y la posibilidad de perder el llamado de una tarea repetitiva.

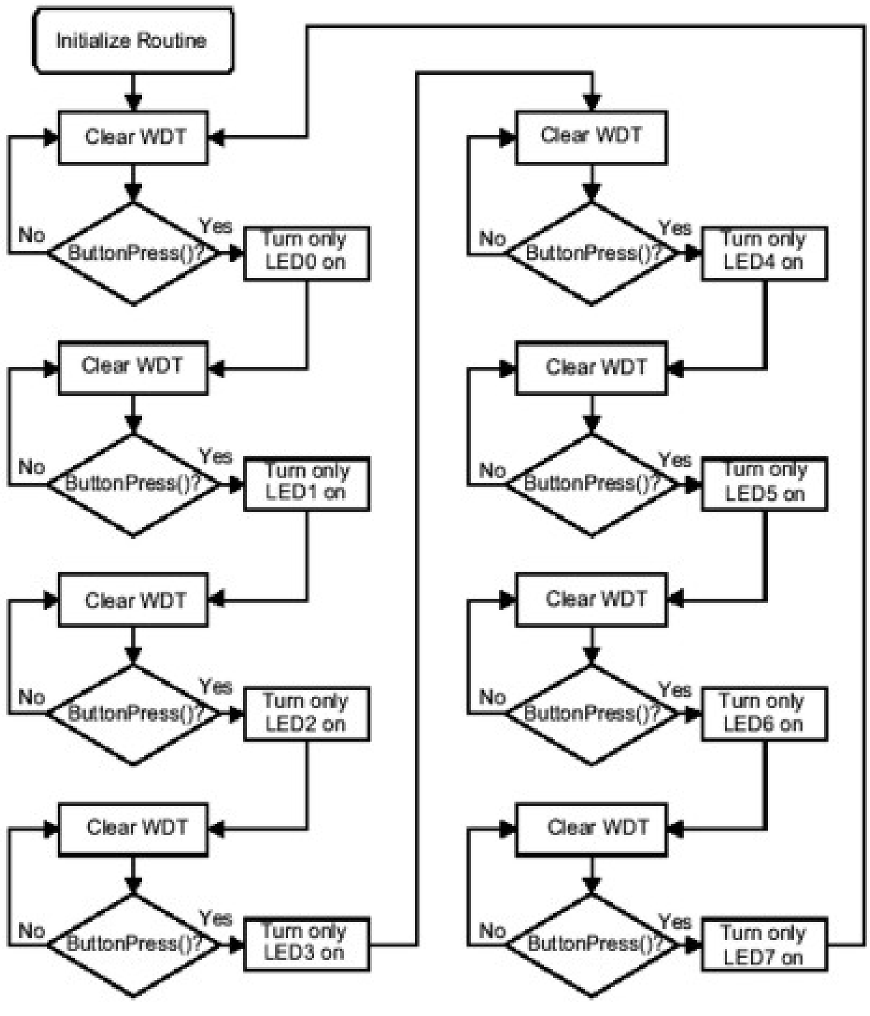

Figura 1: Código sin implementar la Maquina de Estados

El hecho de que el firmware basado en la máquina de estado siempre regrese al mismo punto en el código, hace al firmware más fácil de depurar.

Cuando se encuentra una falla (bug), se debe establecer un punto de ruptura en el punto de inicio. Entonces, pase a través del programa estado por estado hasta encontrar la falla.

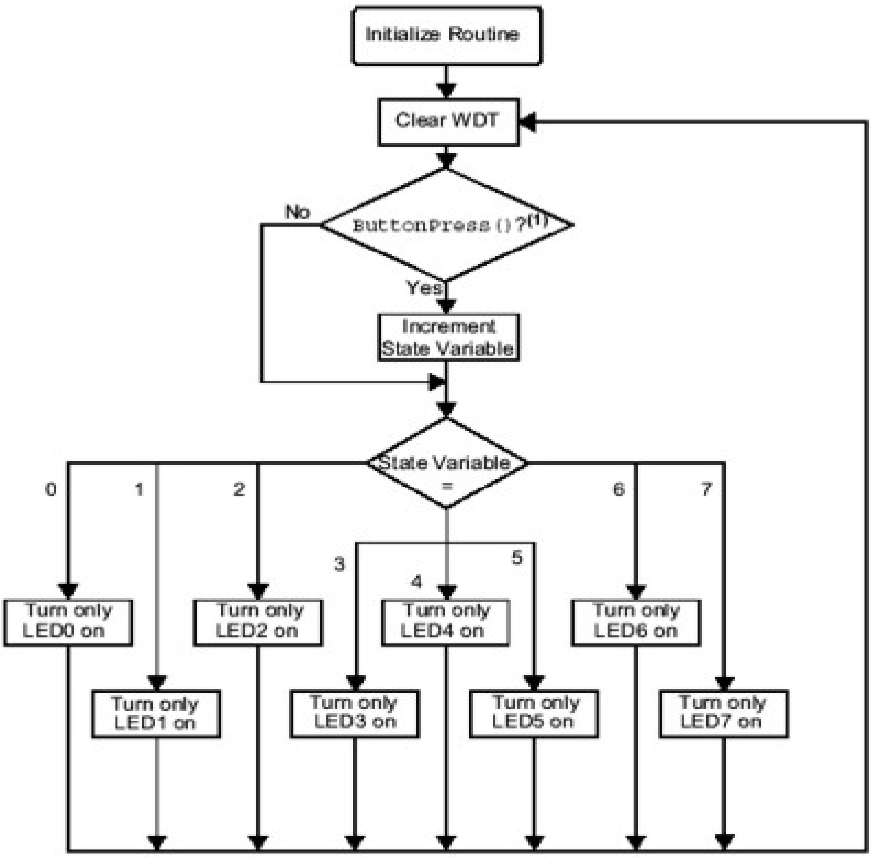

Figura 2: Código implementando la Maquina de Estados

Un ejemplo de una falla, sería si la variable de estado (SV) es modificada incorrectamente en uno de los estados. Si esto sucede, un estado incorrecto será llamado la próxima vez que el flujo del programa regrese al punto de inicio. De cualquier manera, al estar monitoreando la variable de estado , será más fácil ver cuando un cambio no intencional se haga a la SV y en cual estado la SV fue modificada incorrectamente.

Otro beneficio de las máquinas de estado es el firmware que incorpora las máquinas de estado naturalmente promueve un código modular.

Para finalizar, la utilización de un código Modular tiene sus beneficios como lo son:
1. Mejoras y características especiales pueden añadirse fácilmente al código en revisiones posteriores conforme un producto evoluciona.
2. Los Módulos pueden copiarse y pegarse en otras aplicaciones rápida y fácilmente.
3. Otros programadores podrán entender el código para así modificarlo si en un futuro así se necesitara.

Implementación:

Cuando se implementa el concepto de la maquina de estados, se debe de elaborar una lluvia de ideas de todos los estados que se necesitan para una determinada aplicación. Una vez hecho esto se debe identificar el primer estado. Acto seguido debemos responder la siguiente pregunta

¿Que condición se necesita para salir de este estado y que estado es el siguiente?

Dependiendo de lo que suceda en un estado en particular, la variable de estado se incrementa o decrementa con el objetivo de pasar o saltar al siguiente estado. Se sugiere la implementación de un diagrama de flujo.
Finalmente se debe de crear lo módulos de software de cada uno de los estados de acuerdo a nuestro diagrama de flujo.

Implementación de una maquina de estado en lenguaje C y Ensamblador

La implementación de una maquina de estados en lenguaje C, es muy fácil ya que se hace uso de la sentencia **Switch**, el siguiente código muestra como se aplica este concepto:

```
switch (STATE)
{
        case (State0):        // Encender  LED0
        break;
        case (State1):        // Encender  LED1
        break;
        case (State2);        // Encender  LED0
        break;
                                      // ... y así continuamos
        default:
              STATE = State0        //Si por alguna razón un estado
                                      //indefinido ocurre
 }
```

La creación de una maquina de estado en lenguaje ensamblador, es un poco mas difícil. Para este ejemplo utilizamos los **Microcontroladores PIC (MICROCHIP)**. cuya arquitectura hace que para incrementar la variable de estado debemos incrementar también el contador de Programa

Se debe tener cuidado cuando el contador de programa se incrementa, ya que al hacer esto podemos saltar a un lugar equivocado del programa. Por lo tanto es responsabilidad del programador verificar que cada salto sea el correcto.

Inmediatamente de la rutina de incremento, esta una lista de "goto's" que direccionan el flujo del programa hacia un estado. Este método es llamado **"goto computado**". El siguiente programa utiliza un **goto computado** para la implementación de la maquina de estado.

```
Iniciar
        clrf STATE ; limpio la variable de estado
. . .

Inicio
        call StateMachine
        . . .
        goto Inicio

StateMachine
        movlw high StateTable
        movwf PCLATH
        movf STATE, W
        andlw 03h
        addlw low StateTable
        btfsc STATUS, C
        incf PCLATH, F
        movwf PCL

StateTable
        goto State0
        goto State1
        goto State2
        goto State3

State0
. . .
        incf STATE, F              ;Ir al siguiente estado
return


State1
. . .   incf STATE, F              ;Ir al siguiente estado
return
```

Bibliografía

Este articulo fue extraído de la nota de aplicación de **DS40051B** de Microchip

Traducido por: **Yalautitla Miranda Flores**
Elaborado por : **Ing. Eric López Pérez**
elp@i-micro.com