UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERIA

Campus Aeropuerto

PRACTICA 2

Reporte

Equipo:

Ferrusca Pérez Juan Carlos

García Ramos Jesus Humberto

Saloma Hernández Jorge Adrián

Introducción

Normalmente, la velocidad de ejecución de un microcontrolador está determinada por un cristal externo. La placa Stellaris® EK- TM4C123GXL tiene un cristal de 16 MHz . La mayoría de los microcontroladores incluyen un bucle de bloqueo de fase (PLL) que permite que el software ajuste la velocidad de ejecución de la computadora. Por lo general, la elección de la frecuencia implica el equilibrio entre la velocidad de ejecución del software y la energía eléctrica. En otras palabras, ralentizar el reloj del bus requerirá menos energía para operar y generará menos calor. Acelerar el reloj del bus obviamente permite más cálculos por segundo, a costa de requerir más energía para operar y generar más calor.

En el desarrollo de esta practica, modificaremos la frecuencia de reloj con el objetivo de diseñar una maquina de estados finitos, representado en un semaforo.

Antecedentes

Una Máquina de Estado Finito *(Finite State Machine)*, llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de Estados y un número determinado de Transiciones entre dicho Estados.

Las Transiciones de un estado a otro se generan en respuesta a eventos de entrada externos e internos; a su vez estas transiciones y/o subsecuentes estados pueden generar otros eventos de salida. Esta dependencia de las acciones (respuesta) del sistema a los eventos de entrada hace que las Máquinas de Estado Finito (MEF) sean una herramienta adecuada para el diseño de Sistemas Reactivos y la Programación Conducida por Eventos (Event Driven Programming), cual es el caso de la mayoría de los sistemas embebidos basados en microcontroladores o microprocesadores.

Las MEF se describen gráficamente mediante los llamados Diagramas de Estado Finito (DEF), llamados también Diagramas de Transición de Estados.]

Sistemas Reactivos

Un Sistema Reactivo es aquel que interactúa constantemente con su medio ambiente, tiene la característica de ser conducido por eventos (event driven), la respuesta de tiempo es crítica y una vez que el sistema se activa permanece en ese estado de manera indefinida. En estos sistemas los

eventos llegan en tiempos impredecibles y el sistema debe tener la capacidad de responder de manera inmediata, en el orden de los milisegundos o microsegundos, sobre todo en sistemas donde la seguridad es crítica (ejemplo: un piloto automático en un avión o una máquina para soporte de vida en un hospital).

La gran mayoría de los sistemas embebidos (en base a microcontroladores o microprocesadores) corresponden a esta categoría, debido a que estos sistemas están típicamente conectados a varios tipos de sensores y transductores de entrada encargados de captar los estímulos del medio ambiente (temperatura, presión, luz, magnetismo, fuerza / peso, etc.), procesar dicha información y generar una respuesta del sistema hacia el medio ambiente a través de transductores de salida y actuadores.

Sistemas Transformacionales

A diferencia de los Sistemas Reactivos un Sistema Transformacional es aquel que recibe cierta información de entrada, realiza una cierta cantidad de cómputo, produce cierta información de salida y luego termina. No muchos sistemas embebidos caen en esta categoría; ejemplo más típicos son las aplicaciones para PC, como por ejemplo: Un procesador de texto.

Diagrama de Estado Finito o Diagrama de Transición de Estados

Un Diagrama de Estado Finito es un gráfico que representa los diferentes estados de una MEF y todas las transiciones posibles entre los estados.

Como ejemplo, consideremos un muy simplificado sistema de control de un ascensor:

Máquina de Estado Finito Para un Ascensor (Transductora)

Estados: El sistema está formado por tres estados: DETENIDO, YENDO_ARRIBA y YENDO_ABAJO. Los diferentes estados se los representa mediante bloques cuadrados (como en este caso) o círculos.

Transiciones: Las transiciones se las representa mediante flechas que indican la dirección de transición de un estado a otro.

Eventos: Los eventos para el sistema en este ejemplo son los siguientes:

- seleccion_piso: Es un evento externo que se genera toda vez que un usuario selecciona un piso o llama al ascensor desde otro piso.
- arribo_nuevo_piso: Es un evento interno que se genera cada vez que los sensores detectan que se ha arribado al nuevo piso seleccionado por el usuario.

Los eventos se anotan en el gráfico por encima de las flechas de transición.

Condiciones de Transición: Dos transiciones en este sistema de ejemplo tienen asociadas sus respectivas Condiciones de Transición. No todas las transiciones poseen Condiciones de Transición.

- piso_nuevo > piso_actual: Es la condición necesaria para que se produzca una transición del estado DETENIDO al estado YENDO_ARRIBA.
- piso_nuevo < piso_actual: Es la condición necesaria para que se produzca una transición del estado DETENIDO al estado YENDO_ABAJO.

Metodología

Objetivo:

Diseñe un controlador de semáforo para la intersección de dos calles de sentido único igualmente transitadas. El objetivo es maximizar el flujo de tráfico, minimizar el tiempo de espera en un semáforo en rojo y evitar accidentes.

Solución:

La intersección tiene dos caminos de sentido único con la misma cantidad de tráfico: Norte y Este, como se muestra en la Figura 1. Controlar el tráfico es un buen ejemplo porque todos sabemos lo que se supone que sucede en la intersección de dos calles transitadas de un solo sentido.

Comenzamos el diseño definiendo lo que constituye un estado. En este sistema, un estado describe qué camino tiene autoridad para cruzar la intersección. La idea básica, por supuesto, es evitar que los automóviles que van hacia el sur entren en la intersección al mismo tiempo que los automóviles que van hacia el oeste. En este sistema, el patrón de luces define qué camino tiene preferencia sobre el otro.

Dado que es necesario un patrón de salida a las luces para permanecer en un estado, resolveremos este sistema con una FSM de Moore. Contará con dos entradas (sensores de coche en carreteras Norte y Este) y seis salidas (una para cada semáforo del semáforo).

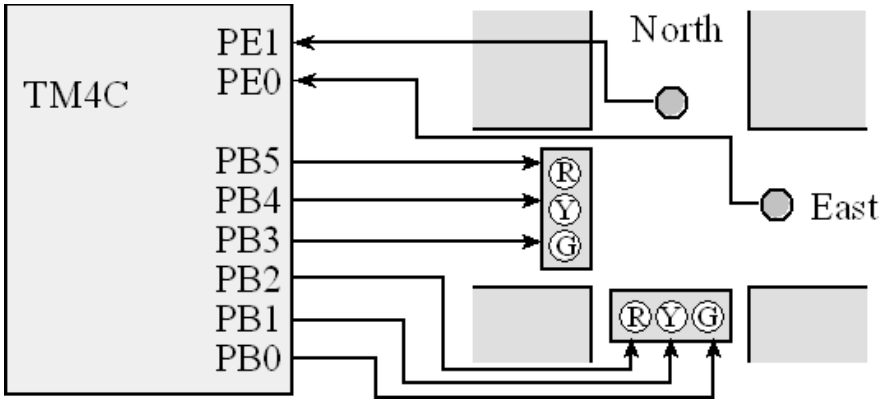

Figura 1: Interfaz de semáforo con dos sensores y 6 luces.

PE1=0, PE0=0 significa que no hay coches en ninguna de las carreteras

PE1=0, PE0=1 significa que hay coches en la carretera Este

PE1=1, PE0=0 significa que hay coches en la carretera del norte

PE1=1, PE0=1 significa que hay coches en ambas carreteras

El próximo paso en el diseño del FSM es crear algunos estados. Una vez más, se eligió la implementación de Moore porque el patrón de salida (qué luces están encendidas) define en qué estado nos encontramos. Cada estado recibe un nombre simbólico:

goN , PB5-0 = 100001

lo hace verde en el norte y rojo en el este

waitN , PB5-0 = 100010

lo hace amarillo en el norte y rojo en el este

goE , PB5-0 =

001100 lo hace rojo en el norte y verde en el este

waitE , PB5-0 = 010100

lo hace rojo en el norte y amarillo en el este

El patrón de salida para cada estado se dibuja dentro del círculo de estado. También se incluye el tiempo de espera de cada estado. El funcionamiento de la máquina estará dictado por las transiciones de estado dependientes de la entrada. Creamos reglas de decisión que definen qué hacer para cada entrada posible y para cada estado. Para este diseño podemos enumerar heurísticas que describen cómo debe funcionar el semáforo:

- Si no vienen autos, permanezca en un estado verde, pero cuál no importa.
- Para cambiar de verde a rojo, implemente una luz amarilla de exactamente 5 segundos.
- Las luces verdes durarán al menos 30 segundos.
- Si los automóviles solo vienen en una dirección, muévase y manténgase verde en esa dirección.
- Si vienen autos en ambas direcciones, recorra los cuatro estados.

Antes de dibujar el gráfico de estado, debemos decidir la secuencia de operaciones.

1. Inicialice los registros de dirección y temporizador
2. Especifique el estado inicial
3. Realizar controlador FSM
    a) Salida a semáforos, que depende del estado
    b) Retraso, que depende del estado
    c) Entrada de sensores
    d) Cambiar estados, que depende del estado y la entrada

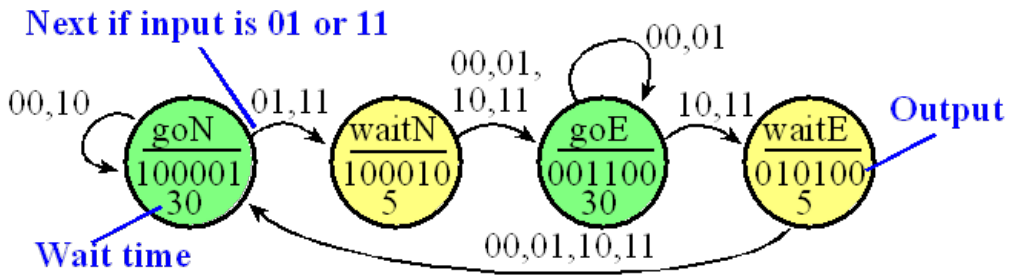

Figura 2. Forma gráfica de una FSM de Moore que implementa un semáforo

Una **tabla de transición de estado** tiene exactamente la misma información que el gráfico de transición de estado, pero en forma tabular. La primera columna especifica el número de estado, que numeraremos secuencialmente desde 0. Cada estado tiene un nombre descriptivo. La columna "Luces" define los patrones de salida para seis semáforos. La columna "Tiempo" es el tiempo de espera con esta salida. Las últimas cuatro columnas serán los siguientes estados para cada patrón de entrada posible.

número	*Nombre*	*Luces*	*Tiempo*	*En 0*	*En 1*	*En 2*	*En 3*
0	goN	100001	30	goN	waitN	goN	waitN
1	waitN	100010	5	goE	goE	goE	goE
2	goE	001100	30	goE	goE	waitE	waitE
3	waitE	010100	5	goN	goN	goN	goN

Tabla 1. Forma tabular de una FSM de Moore que implementa un semáforo.

El siguiente paso es mapear el gráfico FSM en una estructura de datos que se puede almacenar en ROM. El programa realizado usa una matriz de estructuras, donde cada estado es un elemento de la matriz y las transiciones de estado se definen como índices a otros nodos. Los cuatro parámetros **Next** definen las transiciones de estado dependientes de la entrada. Los tiempos de espera se definen en el software como números decimales con unidades de 10 ms, lo que da un rango de 10 ms a unos 10 minutos. Usar buenas etiquetas hace que el programa sea más fácil de entender, en otras palabras, **goN** es más descriptivo que **0** .

El programa principal comienza especificando los bits 1 y 0 del Puerto E para que sean entradas y los bits 5–0 del Puerto B para que sean salidas. El estado inicial se define como **goN** . El bucle principal de nuestro controlador primero emite el patrón de luz deseado a los seis LED, espera la cantidad de tiempo especificada, lee las entradas del sensor del Puerto E y luego cambia al siguiente estado dependiendo de los datos de entrada. La función **SysTick_Wait10ms** esperará 10ms veces el parámetro. El direccionamiento específico de bit facilitará el acceso fácil a los puertos B y E. **SENSOR** accede a PE1–PE0 y **LIGHT** accede a PB5–PB0.

```
#include "lib/include.h"

#define SENSOR  (*((volatile uint32_t *)0x4002400C))
#define LIGHT   (*((volatile uint32_t *)0x400050FC))
// Linked data structure
struct State {
```

```
  uint32_t Out;
  uint32_t Time;
  uint32_t Next[4];};
typedef const struct State State_t;
#define goN   0
#define waitN 1
#define goE   2
#define waitE 3
State_t FSM[4]={
 {0x21,300,{goN,waitN,goN,waitN}},
 {0x22, 50,{goE,goE,goE,goE}},
 {0x0C,300,{goE,goE,waitE,waitE}},
 {0x14, 50,{goN,goN,goN,goN}}};
uint32_t S;  // index to the current state
uint32_t Input;
int main(void){ volatile uint32_t delay;
  PLL_Init();       // 80 MHz, Program 10.1
  SysTick_Init();   // Program 10.2
  SYSCTL->RCGCGPIO |= 0x12;      // B E
  delay = SYSCTL->RCGCGPIO;      // no need to unlock
  GPIOE->DIR &= ~0x03;   // inputs on PE1-0
  GPIOE->DEN |= 0x03;    // enable digital on PE1-0
  GPIOB->DIR |= 0x3F;    // outputs on PB5-0
```

```
  GPIOB->DEN |= 0x3F;    // enable digital on PB5-0
  S = goN;
  while(1){
    LIGHT = FSM[S].Out;  // set lights
    SysTick_Wait10ms(FSM[S].Time);
    Input = SENSOR;      // read sensors
    S = FSM[S].Next[Input];
  }
}
```

Tabla 2. Código realizado

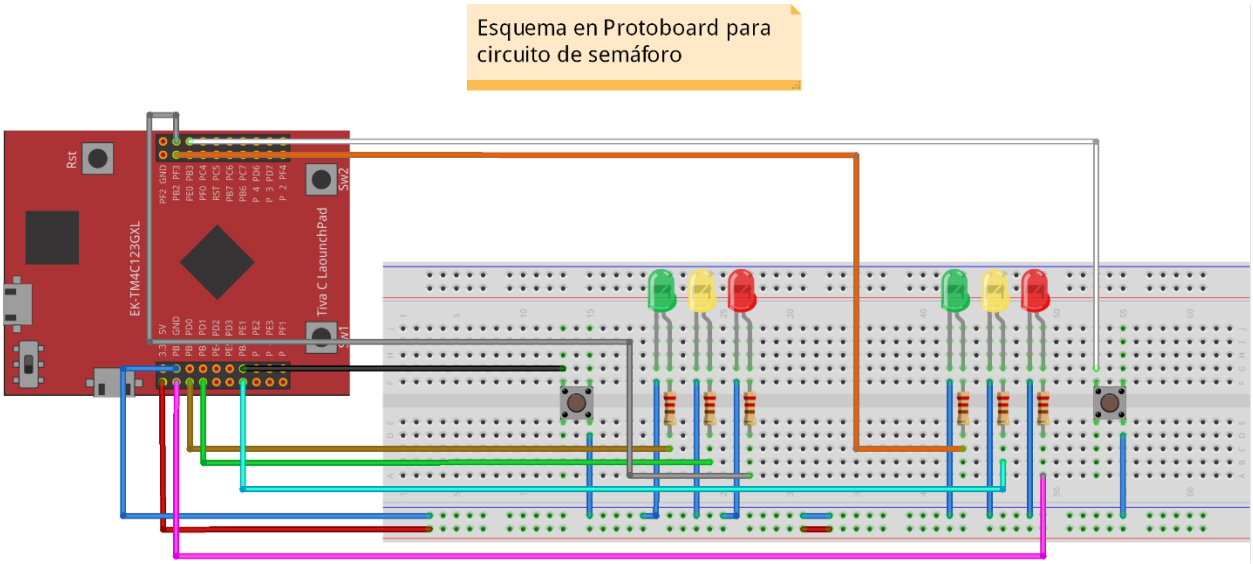

Figura 3: Esquema modelado del circuito

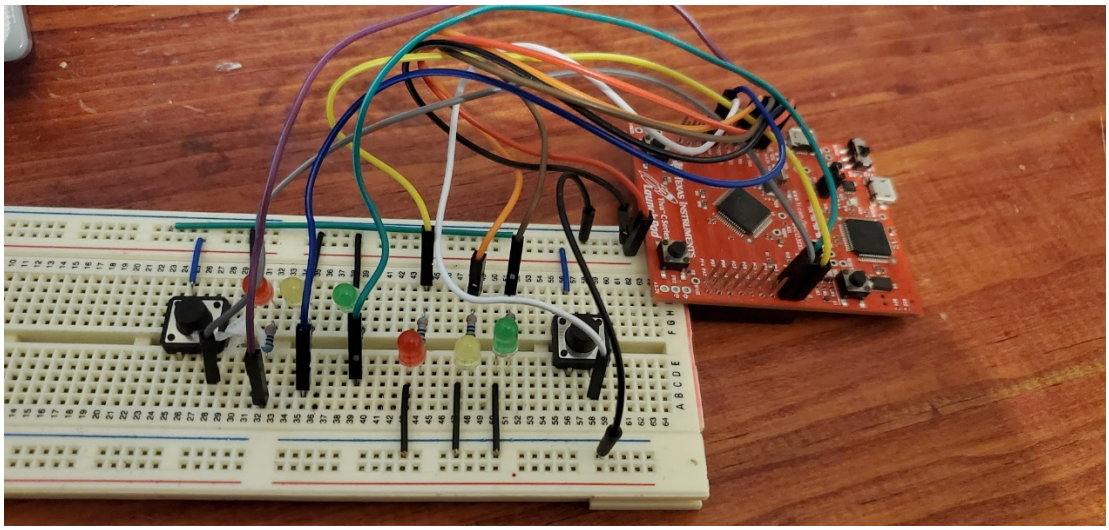

Figura 4. Circuito desarrollado

Análisis y conclusión

Una máquina de estados es una técnica de programación que permite ejecutar diferentes procesos sin generar conflicto alguno, permitiendo conocer el estado anterior y el estado próximo al que pasará la máquina. El desarrollo de esta practica nos ayudó a comprender de manera practica el funcionamiento de la maquina de estados finitos en los sistemas embebidos y como sucede la indepependencia de los procesos, ya que en el caso de la practica, si no habia una interrupcion externa (Pulsar algún botón), no había un cambio de estado hasta que el tiempo de espera transcurriera.

Bibliografía

(2022). Retrieved 12 November 2022, from http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C10_FiniteStateMachines.htm

Alvarez, R. (2022). Introducción a las Máquinas de Estado Finito. Retrieved 12 November 2022, from https://www.tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/13-introduccion-a-las-maquinas-de-estado-finito