PRÁCTICA 2: MÁQUINA DE ESTADOS FINITOS

MICROCONTROLADORES

RAMÍREZ ÁLVAREZ CARLO IVÁN - 280847
ONTIVEROS MARTÍNEZ BEATRIZ - 244784
TREJO DOMÍNGUEZ NELLY BIBIANA - 242494

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
INGENIERÍA BIOMÉDICA

MICROCONTROLADORES
ING. JOSÉ DE JESÚS SANTANA RAMÍREZ
ENERO, 2023

I. OBJETIVO

Diseñar 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.

II. MARCO TEÓRICO

a. Máquinas de estados Finitos

Las Máquinas de estados Finitos conocidas como Finite State Machines por su traducción al inglés, nos sirven para realizar procesos bien definidos en un tiempo discreto. Reciben una entrada, hacen un proceso y nos entregan una salida.

En otras palabras, imaginemos una máquina capaz de seguir una secuencia finita de pasos al introducir un conjunto de datos en ella, solo se puede leer un dato en cada paso que se realize, por tanto, el número de pasos a seguir está dado por el número de datos a introducir. Cada entrada diferente genera una salida diferente, pero siempre el mismo resultado con los mismos datos de entrada. Por lo tanto, una computación es capaz de resolver un problema, sí y solo sí tiene una solución algorítmica, es decir, puede ser descrito mediante una secuencia finita de pasos bien definidos. Mediante una computación podemos encontrar soluciones a problemas que teóricamente tienen una representación algorítmica, pero que pueden necesitar tal cantidad de recursos (factores como el tiempo y el espacio de almacenamiento) que desde el punto de vista práctico no se puede llegar a la solución.

Llamaremos una Máquina de Estados Finitos como Autómata Finito, el hecho es que un Autómata y una Máquina de Estados Finitos son lo mismo, podemos utilizar ambos términos de forma indistinta. Los Autómatas se caracterizan por tener un Estado inicial, reciben una cadena de símbolos, cambian de estado por cada elemento leído o pueden permanecer en el mismo estado. También tienen un conjunto de Estados Finales o Aceptables que nos indican si una cadena pertenece al lenguaje al final de una lectura.

Las FSM pueden utilizarse para reducir la complejidad del código separando la lógica en bloques discretos, mejorar la legibilidad del documento organizando el código en subrutinas separadas y facilitar la depuración identificando rutas de ejecución específicas a través del código.

Inicialmente podemos pensar que una máquina de estados no es necesaria, porque con un If, else if, o else…while…etc es suficiente. Sin embargo, con una máquina de estados podemos conocer dónde nos encontramos en cada momento, simplificar el código,

optimizar, ser eficiente y más profesional, etc.

La máquina de estados necesita de unas entradas y salidas para cada estado, y un conjunto de transiciones entre estados. De un estado, podemos ir a otro estado o a varios estados. Para llegar, por ejemplo, al estado 5, se puede llegar directamente o pasando por los estados 1, 2, 3 y 4. En cualquier caso debe cumplir con unas funciones de transición. Al cambio entre estados se le conocer por transición.

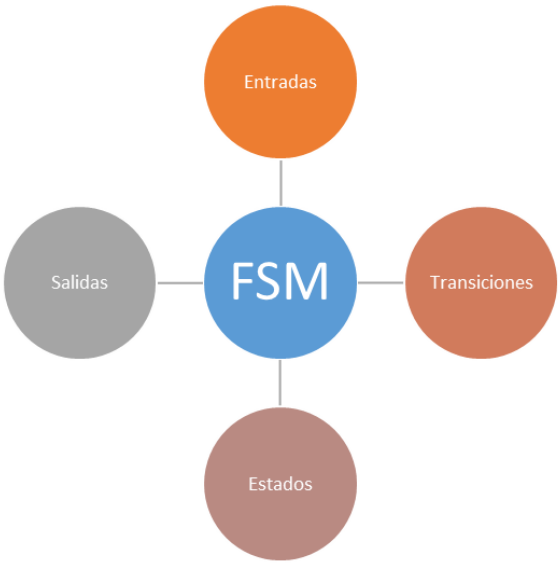

Figura 1. Composición básica de una máquina de estados finitos.

b. Tipos de máquinas de estados finitos.

- Las aceptadoras (reconocedoras o discriminadoras): Se llama aceptadora aquella que la salida es de forma binaria. Las máquinas aceptadoras se las conoce por autómatas finitos. La salida de cada estado de este tipo de máquinas es una aceptación o no aceptación/rechazo. Los estados con salida positiva se les conoce como estados finales.
- Transductoras: Son un tipo de máquinas que son una variación de las aceptadoras. Son máquinas más generales y trabajan tanto de forma binaria como con otro tipo de datos. A este tipo de máquinas se le conoce como autómata. Las máquinas transductoras transforman un conjunto de señales de entrada en una secuencia de salida, pudiendo ser esta binaria o más compleja, dependiendo de la entrada actual (no sólo del estado) y pudiendo ser no necesario un estado inicial. Este tipo de máquinas son usadas para el control de dispositivos y en el campo de la computación.
- Clasificadoras: Las máquinas clasificadoras son una generalización de las aceptadoras con un array de salida n dónde n > 2.

- Secuenciadoras (reconocedoras, generadoras o detectoras): Este tipo de máquinas realizan la detección de patrones o secuencias determinados en respuesta a las entradas recibidas. Transitan de un estado inicial a un estado final de “éxito”. Son útiles para la verificación de contraseñas, códigos, validación de paquetes de datos, etc.
- Deterministas (DFA): Cada estado tiene exactamente una transición para un input concreto. Las FSM deterministas siempre tienen una transición definida. Cuando la totalidad de las transiciones están determinadas en un Autómata, es decir para cada par de (estado, evento) existe uno y sólo un estado correspondiente, se tiene un Autómata Determinısta.
- No deterministas (NFA, GNFA): Una entrada para un estado concreto puede conducir a uno, más de uno o a ninguna transición. Si se tiene al menos una transición no definida o indeterminada entonces tenemos un Autómata No Determinısta.
- Combinacionales: Es aquella máquina de estados finita de un único estado que sólo transita dentro del mismo estado.

Figura 2. Tipos de máquinas de estados finitos.

Las máquinas de estados nos rodean en la sociedad moderna: Por ejemplo, las máquinas expendedoras, las luces de tráfico, los ascensores, las puertas de acceso automáticas, tornos de acceso, un horno doméstico, la lavadora, etc. Posiblemente los inodoros de nuestros hogares se podrían simplificar como máquinas de estados.

Además, el uso de las máquinas de estado finitas es muy importante en distintas áreas, incluyendo la ingeniería eléctrica, la lingüística, las ciencias de la computación, la biología, la matemática, los videojuegos y la lógica.

c. La máquina de Moore

Este tipo de máquina de estados la salida depende sólo del estado. Las salidas sólo cambian si existe un cambio de estado. Las entradas serían los sensores (de temperatura, presión, campo magnético, etc), las salidas los actuadores (motores, servos, etc). La máquina de Moore al igual que la de Mealy, tiene entradas, salidas y transiciones. La lógica de salidas depende del estado dónde se encuentra el sistema. La ventaja del modelo de Moore es una simplificación del comportamiento. La máquina de Moore viene caracterizada por la siguiente expresión:

- Estado siguiente = *f*(estado actual, inputs)
- Salida = *f*(estado actual)

La máquina de Moore se representa de la siguiente manera:

Figura 3. Representación simplificada Máquina de Moore.

En el siguiente ejemplo de máquina de Moore, cada estado está representado por una letra. Las entradas y las salidas se representan por un 0 o un 1.

ENTRADA	ESTADO ACTUAL	SIGUIENTE ESTADO	SALIDA
–	–	A	–
0	A	B	0
1	A	C	0
0	B	B	0
1	B	D	0
0	C	E	0
1	C	C	0
0	D	E	1
1	D	C	1
0	E	B	1
1	E	D	1

Figura 4. Tabla ejemplo Máquina de Moore.

En la primera fila, podemos observar que, si estamos en el estado A, con una entrada de 0, el siguiente estado será B y la salida 0.

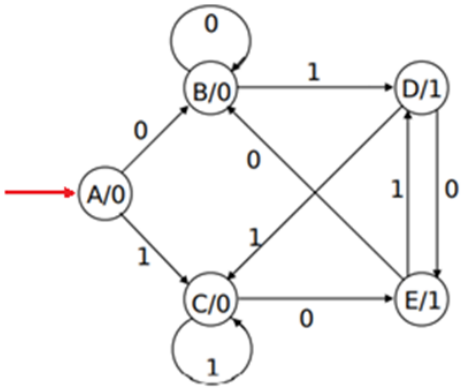

Figura 5. Ejemplo Máquina de Moore.

La representación de un proceso mediante un diagrama de estados (máquinas de Mealy o Moore), lenguaje UML o SDL (u otros) y su posterior codificación permite representar e implementar una secuencia lógica de estados, reduciendo el número de errores y aumentado el rendimiento.

III. MATERIAL

- 1 multímetro digital.
- 1 protoboard.
- 6 leds.
- 2 botones.
- 6 resistencias.
- Tiva LaunchPad TM4C123G.

IV. DESARROLLO DE LA PRÁCTICA

La intersección tiene dos caminos de sentido único con la misma cantidad de tráfico: Norte y Este, como se muestra en la Figura 6. 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).

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

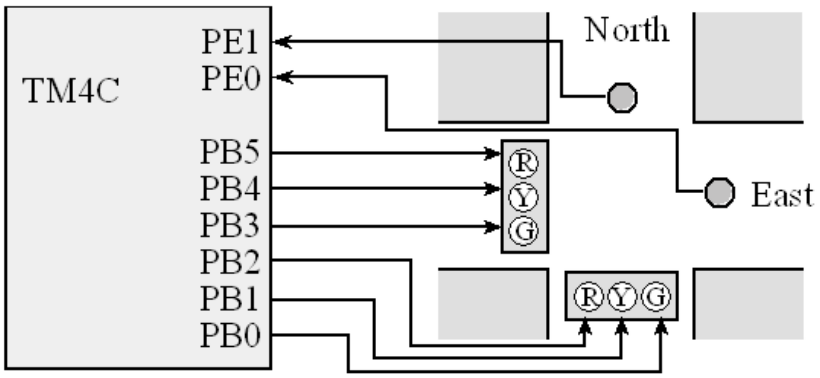

Figura 6. Interfaz de semáforo con dos sensores y seis luces.

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

Implementamos las heurísticas definiendo las transiciones de estado, como se ilustra en la figura 7.

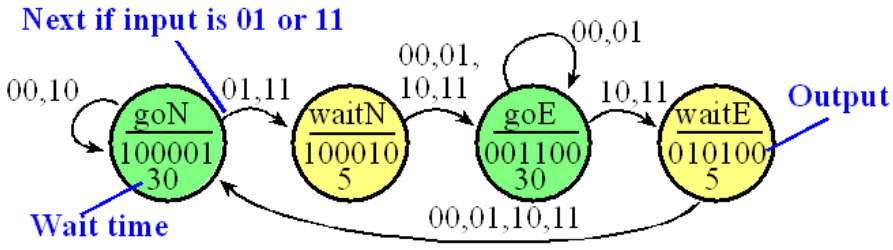

Figura 7. Forma gráfica de una FSM de Moore que implementa un semáforo.

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	Hora	en=0	en = 1	en = 2	en = 3
0	vamos	100001	30	vamos	esperaN	vamos	esperaN
1	esperaN	100010	5	irE	irE	irE	irE
2	irE	001100	30	irE	irE	esperaE	esperaE
3	esperaE	010100	5	vamos	vamos	vamos	vamos

Figura 8. 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 código 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. Las funciones del temporizador se presentaron anteriormente en el código de la figura 9. 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.

Para agregar más señales de salida (p. ej., luces para caminar y girar a la izquierda), simplemente aumentamos la precisión del campo Out . Para agregar dos líneas de

entrada más (por ejemplo, botón de espera, sensor de automóvil de giro a la izquierda), aumentamos el tamaño del siguiente campo a Siguiente[16] . Debido a que ahora hay cuatro líneas de entrada, hay 16 combinaciones posibles, donde cada posibilidad de entrada requiere un valor Siguiente que especifique a dónde ir si ocurre esta combinación. En este esquema simple, el tamaño del campo Siguiente[] será 2 elevado a la potencia del número de señales de entrada.

Código principal de la Interfaz de semáforo con los dos sensores y los 6 leds.

```
/**
 * main.c
 */
#include "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]={
 {0x21,500,{goN,waitN,goN,waitN}},
 {0x22, 250,{goE,goE,goE,goE}},
 {0x0C,500,{goE,goE,waitE,waitE}},
 {0x14, 250,{goN,goN,goN,goN}}};
uint32_t S;  // index to the current state
uint32_t Input;

void Delay(void){unsigned long volatile time;
  time = 1000000;
  while(time){
        time--;
  }
}

int main(void)
{
    SetSystemClock_80MHz(_80MHZ);
    Configurar_SysTick();
    Configurar_GPIO();
```

```
    S = goN;
     while(1){
       LIGHT = FSM[S].Out;  // set lights
      Delay();
       // SysTick_ms(FSM[S].Time/100000.0);
       Input = SENSOR;     // read sensors
       S = FSM[S].Next[Input];
     }
}
```

V. RESULTADOS

Inicialmente al correr nuestro código principal, nuestro semáforo inicia de un lado del cruce en verde y del otro lado en color rojo, de tal manera que al presionar uno de los botones va a cambiar nuestra luz de verde a amarillo y de amarillo a rojo. Del mismo modo, del otro lado del cruce el segundo semáforo funciona de igual manera, pero esta vez la luz cambia de rojo a amarillo y de amarillo a verde.

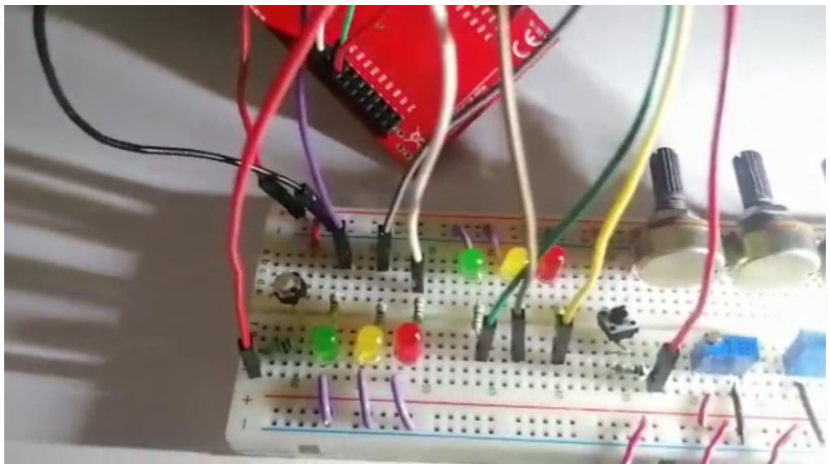

Figura 9. Circuito armado de nuestro semáforo con dos sensores (botones) y 6 leds.

De esta manera cuando se presiona el botón de nuestro primer semáforo, este cambia verde-amarillo-rojo, después el semáforo del otro lado del cruce cambia rojo-amarillo-verde.

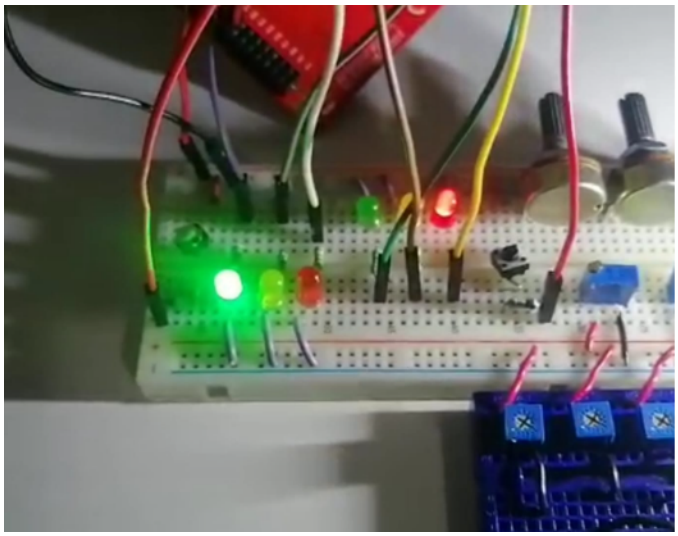

Figura 10. Semáforo con luz en verde y rojo.

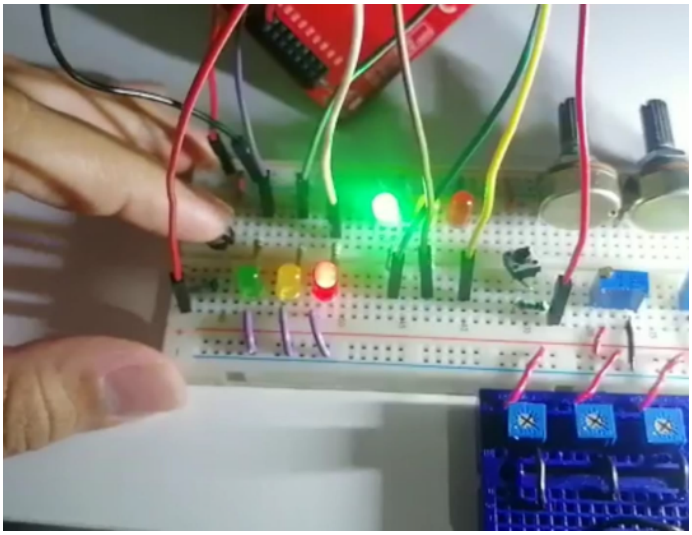

Figura 11. Cambio de luz de semáforo al presionar un botón.

De igual manera si presionamos al mismo tiempo ambos botones cambia el estado de nuestro semáforo, ya que al modificar las entradas también modificamos las salidas. En este caso la luz va cambiando en una secuencia en los 6 leds.

Las FSM pueden utilizarse para reducir la complejidad del código separando la lógica en bloques discretos, mejorar la legibilidad del documento organizando el código en subrutinas separadas y facilitar la depuración identificando rutas de ejecución específicas a través del código.

VI. BIBLIOGRAFÍA

Joober Technologies. (s.f.). *MÁQUINAS DE ESTADOS (FINITE STATE MACHINE – FSM)*. Obtenido de joober.eu: https://www.joober.eu/maquinas-de-estado-finitas/

Orozco, J. A. (2008). *Maquinas de Estados Finitos.* Obtenido de cinvestav: http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/maquinasef.pdf

Yerraballi, J. V. (s.f.). *Capítulo 10: Máquinas de estados finitos modificadas para ser compatibles con EE319K Lab 5*. Obtenido de utexas: http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C10_FiniteStateMachines.htm