UNIVERSIDAD TECNOLÓGICA DE QUERÉTARO

Diplomado en Software Embebido

Módulo 1: Bases de Ingeniería de Software

Tema: 3.2.6. MÁQUINAS DE ESTADO

José Pérez

Marcos Peña

Carlos Reyes

ÍNDICE GENERAL

3.2.6 MÁQUINAS DE ESTADO 1
3.2.6.1. Introducción 1
3.2.6.2. Máquinas Moore 3
3.2.6.3. Máquinas Mealy 4
3.2.6.4. Diagramas de estados 6
3.2.6.5. Deadlock 16
3.2.6.6. Implementación de máquinas de estado 18
3.2.6.7. Referencias previas 25
3.2.6.8. Autodiagnóstico 26
3.2.6.9 Referencias 28
3.2.6.10 Evaluación 28

TABLA DE FIGURAS

Figura 1. Máquina Moore 4
Figura 2. Máquina Mealy 5
Figura 3. Diagrama de estados genérico 7
Figura 4. Diagrama de estados Moore 8
Figura 5. Diagrama de estados Mealy 8
Figura 6. Ejemplo 1 Máquina Moore 9
Figura 7.Ejemplo 1 Máquina Mealy 10
Figura 8. Ejemplo 2. Máquina Display-Motor 12

Figura 9. Ejemplo 3. Máquina Jerarquía Alta 15

Figura 10. Ejemplo 4 Estado Deadlock 17

Figura 11. Diagrama de flujo base temporal 19

3.2.6 MÁQUINAS DE ESTADO

3.2.6.9 Introducción

Un estado es un conjunto particular de instrucciones las cuales serán ejecutadas en respuesta a la entrada de la máquina. Se puede pensar en el estado como algo análogo a la memoria principal de la computadora. El comportamiento del sistema es una función de: (a) la definición del autómata, (b) la entrada y (c) el estado actual. (Fundación Wikipedia, Inc, 2017).

Con base en la definición anterior se puede definir a una máquina de estados como un conjunto de estados o instrucciones que sirven para relacionar entradas y salidas, haciendo que el historial de entradas determine el nuevo estado de la máquina. Las máquinas de estado están compuestas con un elemento de memoria, el cual contiene el estado variable. Cuando la máquina de estado es excitada con señales de entrada válidas, el estado variable es actualizado con el valor de la próxima etapa, de tal manera que la salida dependa del estado actual y/o de las entradas actuales. Por consecuencia, una máquina de estado es una representación de un circuito secuencial particular.

Generalmente las máquinas de estado son aplicadas para especificar aspectos relacionados con tiempo real, protocolos o arquitecturas de software.

Se le llama máquina de estados finitos (FSM por finite state machine) a aquella que contiene un número finito de estados, tal es el caso de los sistemas embebidos, los

cuales contienen recursos limitados de memoria. Cualquier circuito con memoria puede ser considerado como una máquina de estados finita.

Las computadoras cuánticas o la Máquina Universal de Touring son ejemplos de máquinas de estado infinitas, las cuales trabajan con una cinta emulando una memoria infinita.

Las máquinas de estados se pueden dividir en máquinas aceptoras y máquinas transductoras.

Las **máquinas de estado aceptoras** son aquellas que regresan una salida binaria, la cual solamente depende del estado actual. En este tipo de máquinas la salida se activa solamente cuando la entrada es válida. Las máquinas aceptoras son de mayor interés en las teorías de la computación.

Las **máquinas de estado transductoras** consideran múltiples señales de entradas y éstas generan una secuencia como salida del sistema, la cual puede ser desde una secuencia binaria hasta una secuencia de mayor complejidad. A diferencia de las máquinas aceptoras, para este tipo de máquinas no es indispensable de un estado inicial, aunque es recomendable que todas las máquinas lo consideren.

Dada la complejidad en las señales de entrada y salida, las máquinas transductoras son utilizadas en sistemas digitales para el control de sistemas o la definición de arquitectura en sistemas embebidos.

Las máquinas de estado también se clasifican como **máquinas síncronas** cuando se requiere de transiciones a través de una señal de reloj, o **máquinas asíncronas** cuando no necesitan de esta señal.

Las máquinas de estado asíncronas requieren solamente de las entradas para cambiar de estado y/o la salida.

3.2.6.10 Máquinas Moore

Edward F. Moore, investigador de la universidad de Princeton y pionero de las máquinas de estado, expone en 1956 el concepto de máquinas de estado en el artículo "*Gedanken-experiments on Sequential Machines*".

Moore expone una arquitectura secuencial en la cual las salidas sólo dependen de los estados actuales de la máquina, quedando independientes de las entradas.

La arquitectura Moore considera:

- un conjunto finito de estados,
- un estado inicio (también llamado estado inicial),
- un conjunto finito llamado alfabeto entrada,
- un conjunto finito llamado el alfabeto salida,
- una función de transición, mapeando un estado y una entrada al siguiente estado,
- una función salida, mapeando cada estado al alfabeto salida.

Las máquinas Moore se explican a través de la ~~siguiente~~ figura 1.

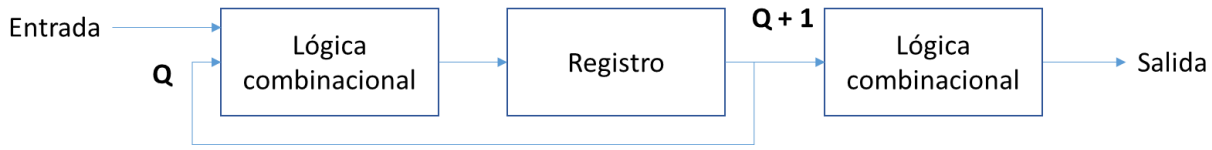

Figura 1. Máquina Moore

En donde se tiene que una señal de entrada y el estado actual **Q** se procesan dentro de un circuito de lógica combinacional para obtener una salida, la cual es almacenada en un registro (memoria) para generar el estado siguiente **Q+**1.

La salida del registro retroalimenta el circuito combinacional para generar el nuevo estado.

La salida solamente depende del estado de salida y la lógica combinacional que se pueda generar con este estado.

3.2.6.11 Máquinas Mealy

En 1955, George H. Mealy escribió Un *Método para sintetizar Circuitos Secuenciales*, en el cual se considera:

- un conjunto finito de estados,
- un estado inicio (también llamado estado inicial),
- un conjunto finito llamado alfabeto entrada,
- un conjunto finito llamado el alfabeto salida,

- una función de transición, mapeando un estado y una entrada al siguiente estado,
- una función salida.

A diferencia de las máquinas Moore, en las máquinas Mealy las salidas pueden ser determinadas por el estado presente solamente, o por el estado presente y las entradas presentes; es decir, las salidas se producen dependiendo de cómo la máquina realiza una transición de un estado a otro.

La figura 2 representa una máquina secuencial con arquitectura Mealy, en la cual la señal de entrada y el estado actual **Q** son procesados a través de un circuito de lógica combinacional, la salida del circuito genera el estado siguiente **Q+1** y la salida del sistema. El estado siguiente **Q+1** es almacenado en un registro y éste a su vez retroalimenta el circuito combinacional para generar el siguiente estado.

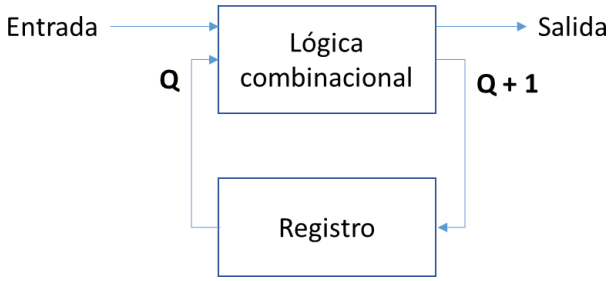

Figura 2. Máquina Mealy

3.2.6.12 Diagramas de estados

El diagrama de estados es la representación gráfica de la máquina de estados planteada. Se basa en círculos que representan cada estado y flechas que indican la transición hacia el estado siguiente.

La Figura 3 muestra un ejemplo de diagrama de estado, en donde se muestra el **Estado 1** como estado inicial. Ante un reinicio (reset) del sistema, la máquina de estados implementada deberá iniciar en este estado.

El estado *siguiente* al **Estado 1** es el **Estado 2**, en donde no se especifica ninguna condición de entrada para realizar la transición entre estos dos estados.

Adicionalmente, se identifica que el **Estado 2** tiene una transición al **Estado 3** o al **Estado 4** cuando se presenta una condición **Input1** o **Input2** respectivamente, en caso contrario, si la señal de entrada es una señal **Input 3** el **Estado 2** no se modifica.

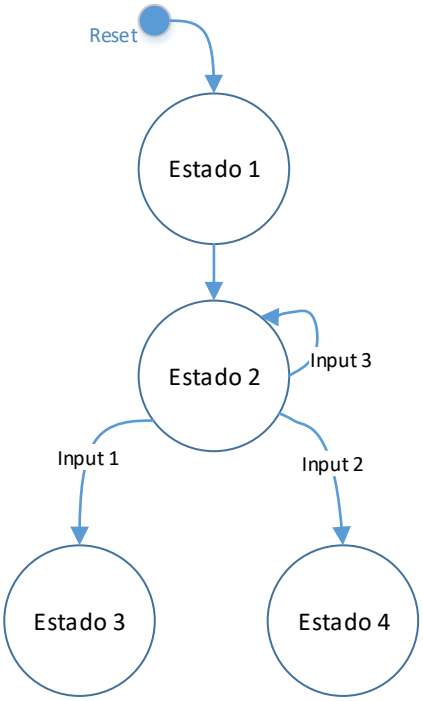

Figura *3. Diagrama de estados genérico*

En el apartado anterior se revisó la diferencia entre máquinas de estado Moore y Mealy. Aunque ambas máquinas pretenden llegar al mismo resultado, aunque con una arquitectura diferente, la notación de cada una de ellas también difiere en un diagrama de estados.

La salida en una Máquina de estado Moore se muestra dentro de la burbuja de estado debido a que la salida se mantiene igual mientras la máquina se mantenga en ese estado, tal como se observa en la figura 4. La salida puede ser arbitrariamente compleja, pero debe ser la misma cada vez que la máquina entra a ese estado.

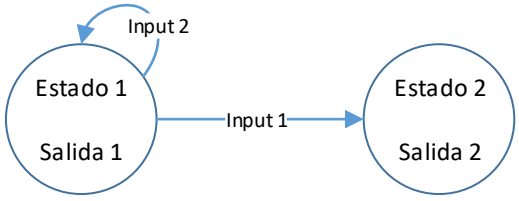

Figura 4. Diagrama de estados Moore

Dado que la máquina de Mealy genera las salidas basada en el estado presente y en las entradas a la máquina, la salida se representa en la misma línea de transición dividida por una diagonal, tal como se muestra en la figura 5.

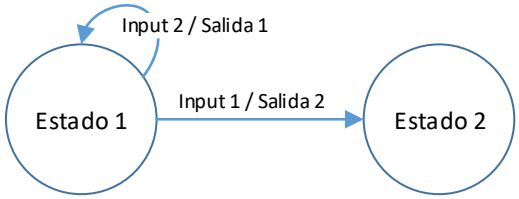

Figura 5. Diagrama de estados Mealy

A simple vista pareciera que se trata del mismo diagrama, incluso funcionalmente hablando ambas máquinas de estado generan el mismo resultado, sin embargo, las máquinas Mealy permiten optimizar el tiempo de respuesta y en algunos casos el número de estados. Analicemos el ejemplo 1:

Ejemplo 1.

Considere un sistema en el cual se requiera activar y desactivar un motor después de haber presionado y liberado un botón.

La salida es la señal de activación del motor, mientras que la entrada es el estado del botón.

Solución.

Con base en la descripción del problema se define la Salida Motor como 0 para representar apagado y 1 como encendido.

El botón se representa como 0 cuando está libre y 1 cuando se presiona.

Se desarrolla la máquina de estados basada en el modelo Moore en la figura 6.

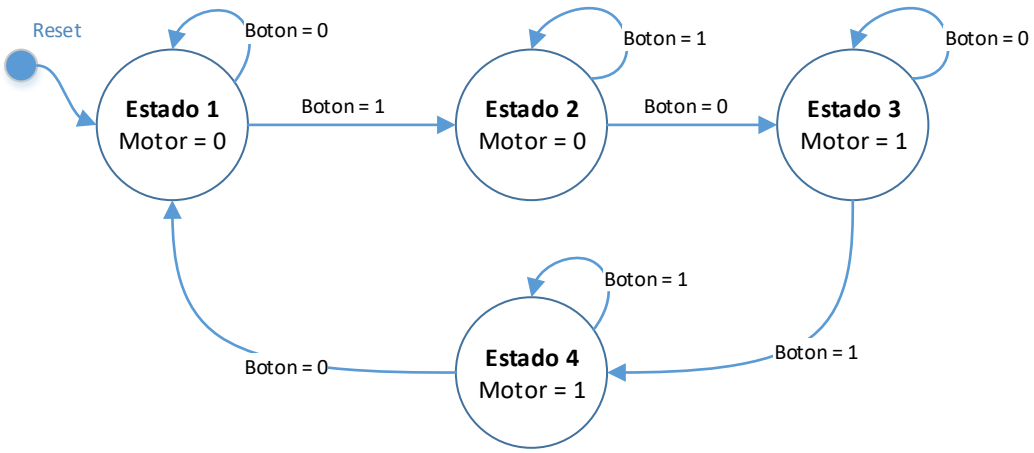

Figura 6. Ejemplo 1 Máquina Moore

La figura 7 muestra la máquina de estados basada en el Modelo Mealy.

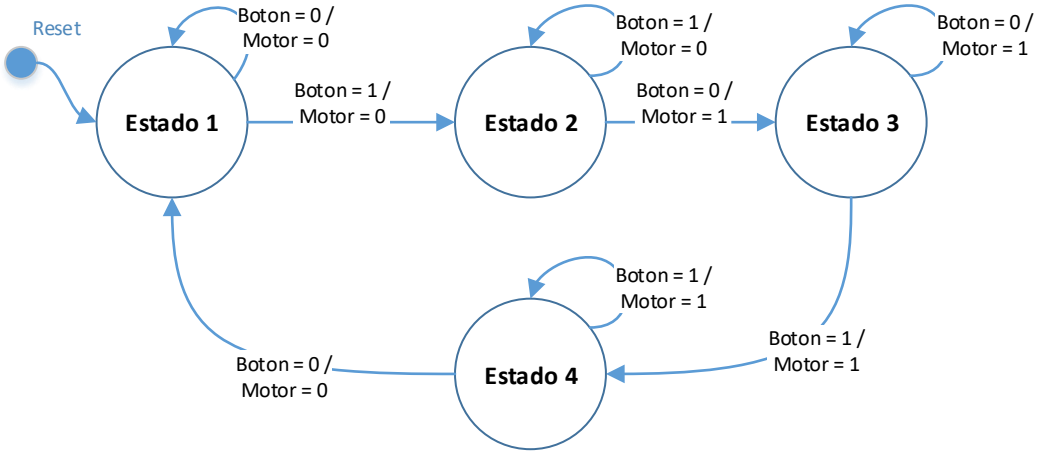

Figura 7.Ejemplo 1 Máquina Mealy

Al comparar ambos diagramas no se observa una diferencia significativa entre ellos, sin embargo, observe que en la máquina Moore el motor se enciende hasta que entra al **Estado 3**, mientras que en la máquina Mealy el motor se enciende al liberar el botón en el **Estado 2.**

Lo mismo sucede para apagar el motor, en el caso de Moore transcurren 2 estados para apagarlo, mientras que en Mealy se apaga en el **Estado 4** al liberar el botón.

Una de las ventajas de Moore es que la salida solamente depende del estado actual, por lo que podemos enfocar el control de los estados solamente con las entradas de cada uno de ellos.

Es importante analizar los requerimientos del proyecto antes de implementar una máquina de estados, ya que de éstos depende la definición de la arquitectura para el diseño de la máquina de estados.

Supongamos que se tiene un sistema, en el cual no se tienen consideradas todas las entradas o las condiciones que puedan modificar su comportamiento, en este caso los diagramas de estado permiten generar una única salida para una o múltiples condiciones esperadas.

Observe el ejemplo 2:

Ejemplo 2.

Considere un sistema en el cual, a través de un teclado matricial, se tiene el control del display y de un motor.

Diseñe una máquina de estados bajo las siguientes condiciones:

1. Al presionar el botón 1 el display cambia entre encendido y apagado.
2. Al presionar el botón 1, si el motor está activado, entonces se desactiva el motor y el display se apaga.
3. Al presionar el botón 2, solamente si el motor está detenido y el display está encendido, el motor se activa, y el display muestra la velocidad del motor.
4. Al presionar el botón 3, si el motor está en funcionamiento, el motor se detiene y el display muestra el mensaje “PAUSA”.
5. No se genera ninguna respuesta ante condiciones diferentes a las especificadas.

Solución.

Se definen 3 estados de operación con sus respectivas salidas:

- **Apagado**. Motor y display apagados,
- **Pausa.** Motor apagado y display muestra “PAUSA”,
- **Operación.** Motor encendido y display muestra RPM.

El tiempo de ejecución no es crítico para el sistema, por lo que se diseña una máquina de estados Moore, mostrada en la figura 8.

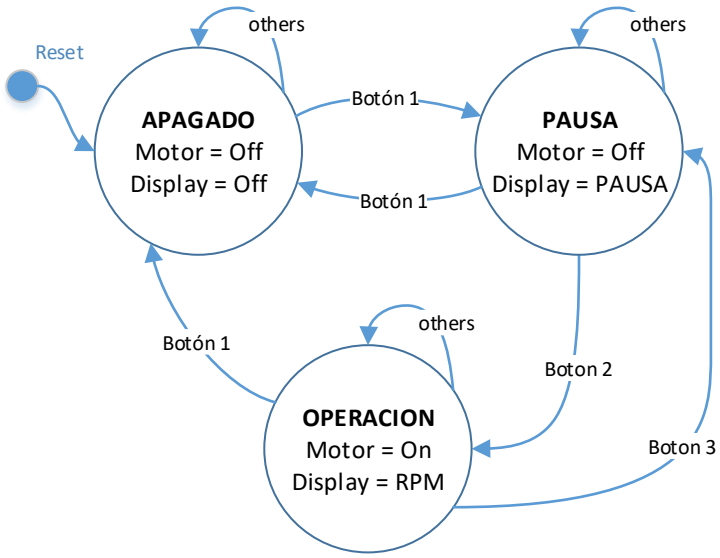

Figura 8. Ejemplo 2. Máquina Display-Motor

Observe que el control del sistema se mantiene definido y genera una respuesta definida para ciertas condiciones de operación, es decir, si el usuario presiona los botones 2 o 3 mientras el sistema se encuentra apagado, ni el motor ni el display

encenderán, o bien, si el motor está en operación y se presiona el botón 2, o se llega a presionar de manera simultánea 2 o más botones, el control continuará moviendo el motor con el patrón que se haya establecido.

Este tipo de arquitecturas permite que un se tenga certeza de la salida que genera los impulsos conocidos, por lo que entonces el enfoque se le da en cómo controlar las señales de entrada y cómo robustecer los algoritmos ante perturbaciones no deseadas, por ejemplo, ruido inducido a través de los botones.

Otra ventaja de utilizar máquinas de estados es que permite flexibilidad para el desarrollo de sistemas más complejos reduciendo el riesgo de fallas a estados que ya se hayan validado, de esta manera se estará desarrollando una arquitectura modular.

Ejemplo 3.

Considere el sistema desarrollado en el ejemplo 2, al cual se le deberá agregar una etapa que permita configurar el tipo de control (PID, FUZZY, RNA) durante los primeros 10 segundos después de haber energizado el sistema, transcurrido este tiempo, el control cambiará a modo usuario.

Para entrar al modo de configuración se requiere presionar de manera simultánea los botones 2 y 3.

Para salir del modo de configuración se deberá presionar el botón 1. El display se encenderá mostrando “PAUSA” y el motor permanecerá apagado.

Solución.

Si bien es cierto que se pueden ampliar los estados de la máquina desarrollada en el ejemplo 2, sin embargo, la manera de reducir la complejidad de la misma es diseñando una máquina de estados de un nivel superior, con la finalidad de que la máquina de estados de operación diseñada en el ejemplo 2 quede embebida en uno de los estados de mayor jerarquía.

Se proponen los siguientes estados.

Wait Config. Display apagado.

Config. Ejecuta configuración de sistema.

User Mode. Ejecuta operación del sistema.

Se diseña una máquina de estados de mayor jerarquía basado en máquinas Moore, la cual se muestra en la figura 9.

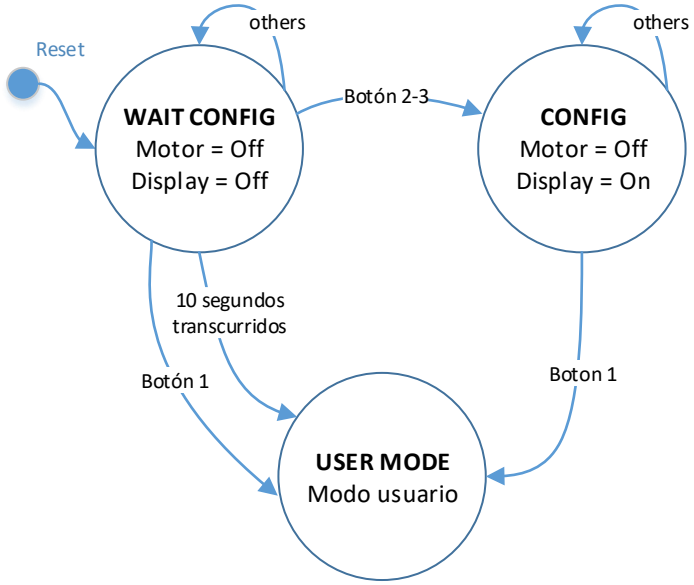

Figura 9. Ejemplo 3. Máquina Jerarquía Alta

La máquina desarrollada en este ejemplo muestra como con una máquina de estados de complejidad baja, y sin realizar cambios en el modo usuario, se puede establecer un sistema complejo que permita configurar el sistema sin modificar el modo usuario desarrollado en el ejemplo 2.

De manera similar se debería contemplar el diseño de una máquina de estados que permita mover un menú y configurar el sistema dentro del estado **CONFIG** definido en el ejemplo 3.

Se debe considerar que los diagramas de estado permiten crear un alto nivel de abstracción para identificar la arquitectura de un sistema, así como las entradas y salidas durante el tiempo.

Cada estado, a su vez, deberá detallarse a través de diagramas de flujo o se puede apoyar de diagramas de estado que se encuentren a un nivel inferior.

3.2.6.13 Deadlock

El término Deadlock se refiere al estado que no genera ninguna salida. Esto no significa que el diseño es incorrecto. Se recomienda que todos los estados tengan una salida, para que se tenga un control completo del sistema y evitar que el sistema se quede “colgado”. De acuerdo a los requerimientos del sistema, se podrán considerar estados *deadlock* cuando se considere necesario. Por ejemplo, cuando se requiera supervisión de algún técnico o un reset voluntario.

Ejemplo 4.

Considere el sistema del ejemplo 3, en el cual se requiere que, ante una falla detectada en el motor, el sistema permanezca en un modo de falla (**FAILURE**), donde el motor permanezca apagado y se despliegue un mensaje de falla en el display. El usuario no podrá manipular el sistema, sin embargo, se podrá restablecer el sistema al desenergizar el equipo. De esta manera se asegura que el usuario está consciente de que se generó una falla.

Solución.

Se establece un estado deadlock tal como se muestra en la figura 10.

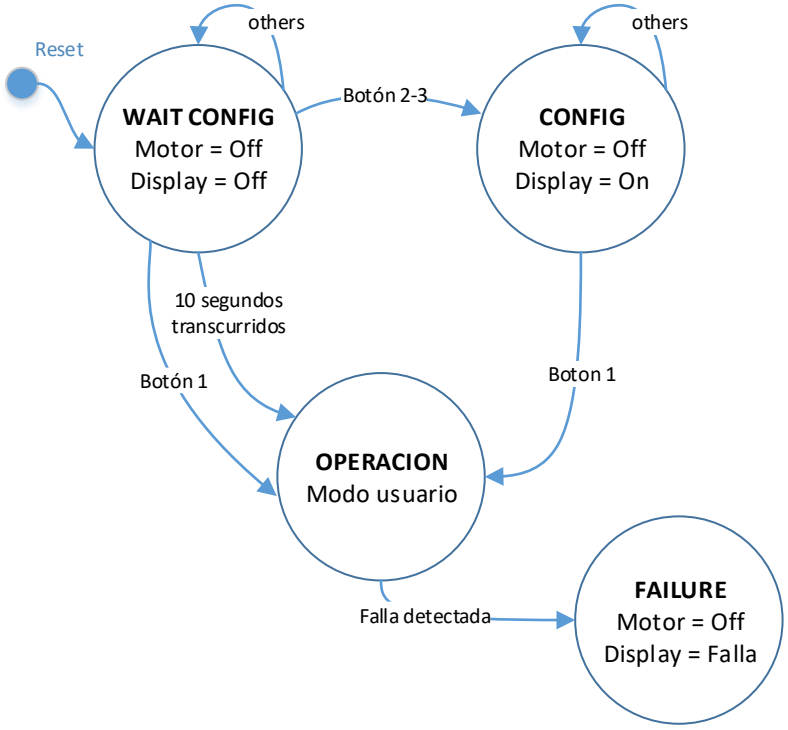

Figura 10. Ejemplo 4 Estado Deadlock

En el diagrama de estados se observa el estado **FAILURE**, el cual no tiene ninguna salida, sin embargo, al ~~desconectar~~ desenergizar el sistema, al ~~reconectar~~ reenergizar la máquina de estados se inicializará en el estado **WAIT CONFIG**, permitiendo que el usuario pueda manipular nuevamente el display.

3.2.6.14 Implementación de máquinas de estado

De acuerdo a lo analizado, las máquinas de estados son circuitos o arquitecturas secuenciales, es decir, que la salida de estos sistemas no depende solamente de las entradas actuales, sino que además dependen de las entradas anteriores, en este caso de los estados anteriores.

Los sistemas secuenciales y las máquinas de estados están basados en conceptos temporales, por lo que se requiere de sistemas de control basados en un reloj y en memorias o registros.

En esta sección se analizará el método para emular una máquina de estados utilizando lenguaje C estructurado.

Como ya se expuso, las máquinas de estado requieren de una base temporal por lo que se requiere definir un sistema que permita sincronizar estas máquinas.

Recomendación es desarrollar un *tick* de sistema, definido por el desarrollador de software basado en los requerimientos del proyecto, tal que, sea capaz de ejecutar la máquina de estados durante la ejecución del programa.

La figura 11 muestra la idea básica de desarrollar este sistema temporal.

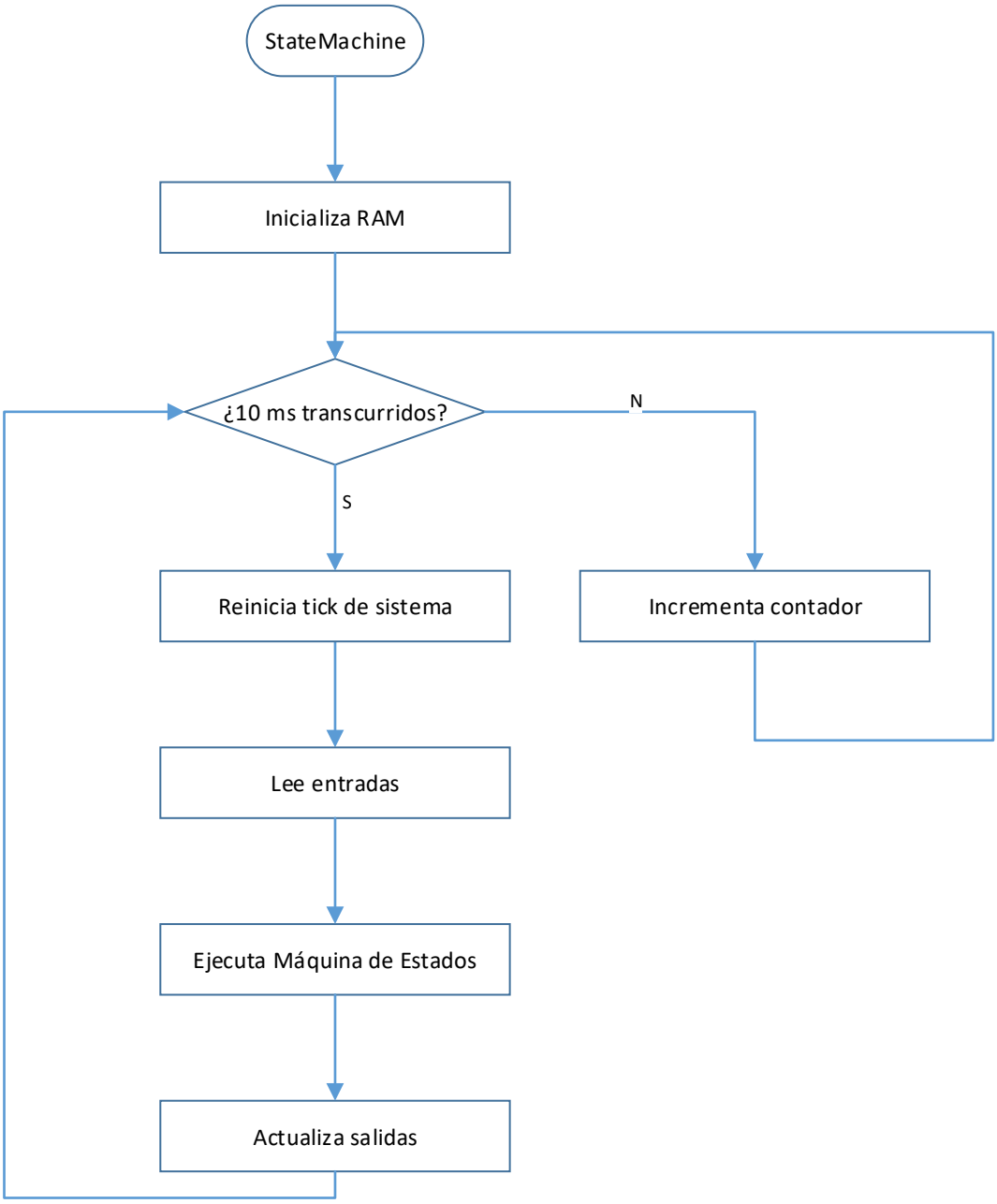

Figura 11. Diagrama de flujo base temporal

La figura 11 está conceptualizada para que el microcontrolador ejecute una serie de tareas de manera síncrona cada 10ms. De tal manera que cada vez que se complete este tiempo, el microcontrolador leerá las entradas, ejecutará la máquina de estados implementada y finalmente actualizará el estado del hardware de acuerdo a la salida establecida en el estado presente. Finalmente, termina el ciclo, y espera a que transcurra el tick de sistema definido.

Bajo este esquema, es posible desarrollar un sistema secuencial, basado en una estructura combinacional. El desarrollo de una máquina de estados se implementa a través de variables globales, que tomen la función de memoria y permitan identificar el estado actual y el estado siguiente.

El control de los estados por medio de la sentencia de control ***switch case*** de ANSI C.

La sentencia ***switch*** está formada por una serie de etiquetas ***case,*** y de un caso opcional ***default***. Esta sentencia evalúa por separado cada uno de los valores constantes enteros que puede asumir a través de las etiquetas ***case*** y toma diferentes acciones de acuerdo a la condición verdadera.

Haciendo uso de la sentencia ***switch case*** la máquina de estados queda implementada de la siguiente manera:

```
switch(u8State)
{
  case C_ESTADO_INICIAL:
        //Inicializa variables de estado
        // u8State = C_ESTADO_1
        break;

  case C_ESTADO_1:
        //Ejecuta estado 1
        //Evalúa condiciones de estado
        break;

  case C_ESTADO_2:
        //Ejecuta estado 2
        //Evalúa condiciones de estado
        break;

  case C_ESTADO_3:
        //Ejecuta estado 3
        //Evalúa condiciones de estado
        break;
        .
        .
        .
  case C_ESTADO_N:
        //Ejecuta estado 3
        //Evalúa condiciones de estado
        break;

  default:
        //Reinicia máquina de estados
        break;
}
```

Figura 12.

Observe que la variable de estado del ejemplo se está declarando como una variable de 8bits, por lo que la máquina de estados finita tendía un máximo de 255 estados posibles.

Se sugiere que las variables de estado se identifiquen con la terminación ***State*** o ***St*** para identificarlas del resto de las variables.

El estado C_ESTADO_INICIAL define la inicialización de las variables de la máquina de estados y establece el estado siguiente sin ninguna condición. La razón de esto es para tener una inicialización y asegurar el correcto funcionamiento de la máquina después de un Reset.

Seguido a la inicialización se establecen los ***n*** estados definidos con sus respectivas instrucciones y condiciones de transición.

El estado ***default*** contiene un reset de la máquina implementada, de esta manera, cuando la variable de estado toma un valor no definido; a esta variable se le asigna el estado inicial *C_ESTADO_INICIAL**,*** esto permite que el flujo de la máquina de estados no se vea interrumpido por un estado *Deadlock*.

Ejemplo 5.

Basado en el diagrama de estados desarrollado en el ejemplo 2, implemente el código en C para ejecutar la máquina de estados.

Solución

Se definen los estados de operación, se declara y se inicia la variable de estado:

```
#define C_INIT_ST       (0)
#define C_APAGADO_ST    (1)
#define C_PAUSA_ST      (2)
#define C_OPERACION_ST  (3)

U8 u8UserStateMachineSt = C_INIT_ST;
```

Figura 13.

Se implementa la máquina de estados mediante la sentencia ***switch case***, considerando un estado de inicialización y un estado default para reiniciar la máquina si fuese necesario:

```
void SR_PerformUserStateMachine(void)
{
  switch(u8UserStateMachineSt)
    {
      case C_INIT_ST:
          u8MotorControl = C_OFF;
          u8Display = C_CLEAR_DISPLAY;
          u8UserStateMachineSt = C_APAGADO_ST;
          break;

      case C_APAGADO_ST:
          u8MotorControl = C_OFF;
          u8Display = C_CLEAR_DISPLAY;
          if(u8Button1 == C_PRESSED)
            {
              u8UserStateMachineSt = C_PAUSA_ST;
            }
          break;

      case C_PAUSA_ST:
          u8MotorControl = C_OFF;
          u8Display = C_DISPLAY_PAUSA;
          if(u8Button1 == C_PRESSED)
            {
              u8UserStateMachineSt = C_APAGADO_ST;
            }
          else if(u8Button2 == C_PRESSED)
            {
              u8UserStateMachineSt = C_OPERACION_ST;
```

```
            }
          break;

      case C_OPERACION_ST:
          u8MotorControl = C_ON;
          u8Display = C_SHOW_RPM;
          if(u8Button1 == C_PRESSED)
            {
              u8UserStateMachineSt = C_APAGADO_ST;
            }
          else if(u8Button3 == C_PRESSED)
            {
              u8UserStateMachineSt = C_PAUSA_ST;
            }
          break;

      default:
          u8UserStateMachineSt = C_INIT_ST;
          break;
    }
}
```

Figura 14.

Observe que la salida (*u8MotorControl* y *u8Display*) depende solamente del estado de operación, implementando una máquina Moore.

Cada estado tiene asociado un valor para la salida, así como cada estado define sus propias condiciones de transición, indicando el estado siguiente. Los estados que contienen más de una línea de transición, se apoyan de la sentencia ***if-else*** para determinar el siguiente estado.

Finalmente, se implementa en el programa principal la máquina de estados *PerformUserStateMachine* dentro de una estructura temporal basado en un *tick de sistema* fijo,

```
void main(void)
{
  SR_InitRam();

  while(1)
  {
    if(u8TickCompleted == C_TRUE)
      {
        u8TickCompleted = C_FALSE;
        SR_ReadInputs();
        SR_PerformUserStateMachine();
        SR_RefreshOutputs();
      }
  }
}
```

Figura 15.

3.2.6.15 Referencias previas

https://www.youtube.com/watch?v=HhliQ9HVYXI

https://www.youtube.com/watch?v=UcFthbierrI

3.2.6.16 Autodiagnóstico

1. **¿Qué es un estado?**
    a) Conjunto de instrucciones que son ejecutadas como respuesta a una entrada
    b) Conjunto de bloques que describen una salida
    c) Conjunto de entradas y salidas que describen un algoritmo
    d) Conjunto de entradas que modifican un algoritmo

2. **¿Qué es una máquina de estados finita?**
    a) Es un conjunto ilimitado de estados que describen una arquitectura
    b) Es un conjunto ilimitado de estados que describen la relación de entradas y salidas
    c) Es un conjunto limitado de estados que describen una arquitectura
    d) Es un conjunto limitado de estados que describen la relación de entradas y salidas

3. **¿Qué es una máquina Moore?**
    a) Es la máquina de estados donde la salida sólo depende de las entradas
    b) Es la máquina de estados donde la salida sólo depende de las entradas y estado actual
    c) Es la máquina de estados donde la salida sólo depende del estado actual
    d) Es la máquina de estados donde la salida sólo depende del estado anterior

4. **¿Qué es una máquina Mealy?**
    a) Es la máquina de estados donde la salida sólo depende de las entradas

b) Es la máquina de estados donde la salida sólo depende de las entradas y estado actual
c) Es la máquina de estados donde la salida sólo depende del estado actual
d) Es la máquina de estados donde la salida sólo depende del estado anterior

5. ¿Qué representa un diagrama de estados?

a) El flujo de un circuito combinacional
b) Una secuencia lógica de software
c) Una relación entre entradas y salidas como circuito secuencial
d) Los principales bloques en la arquitectura de software

3.2.6.17 Referencias

Casasnovas, M. (Julio de 2014). *F.S.M. Maquinas de estado finitas.* Obtenido de Universidad Tecnolígica Nacional. Facultad Regional Córdoba: http://www.profesores.frc.utn.edu.ar/electronica/tecnicasdigitalesi/pub/file/AportesDelCudar/Maquinas%20de%20Estado%20MC%20V5.pdf

Cauca, U. d. (s.f.). *ftp://ftp.unicauca.edu.co.* Obtenido de Universidad del Cauca: ftp://ftp.unicauca.edu.co/Documentos_Publicos/Facultades/FIET/DEIC/Materias/SEDS/Material%20Auxiliar/FSM.pdf

Deitel, H. M. (1994). C How to program. Prentice Hall.

Fundación Wikipedia, Inc. (6 de Junio de 2017). *Wikipedia*. Obtenido de Estado (informática): https://es.wikipedia.org/wiki/Estado_(inform%C3%A1tica)

Mano, M. (2003). *Diseño Digital.* Pearson, Prentice Hall.