



Máquina de estados finitos





Modelos

- Un modelo es una **representación simplificada** de un sistema que **contempla** las **propiedades importantes** del mismo desde un **determinado punto de vista**
- El uso de modelos es una actividad arraigada en la ingeniería. Están expresados en **lenguajes cercanos al problema**. Frecuentemente son **gráficos o matemáticos**
- Cada vez es más frecuente el uso de modelos para ayudar al **desarrollo de software** de sistemas embebidos (en particular aquéllos basados en **microcontroladores**)



Modelos

- Los modelos no están pensados para visualizar **código** sino para representar un sistema con un **nivel de abstracción superior** al de los lenguajes de programación
- La migración de una metodología de programación basada en lenguaje C hacia el desarrollo de software basado en modelos, supone un **incremento en el nivel de abstracción y en productividad** similar al producido al cambiar desde assembler a C



Introducción

- El **comportamiento dinámico** de ciertas aplicaciones software puede **visualizarse** como la **ejecución de una serie de pasos** en los que, la mayor parte de las **actividades útiles** están asociadas a ciertos **estados** o a **estímulos externos o internos (eventos)** bien determinados. Ejemplo simple: cuando una aplicación enciende o apaga un LED, existen dos **estados**; uno cuando el **LED** está **encendido** y otro cuando está **apagado**. **Presionar un pulsador** puede ser el **evento** que haga **transicionar** al LED entre estos estados



Significado de estado

- El **estado** de un sistema **en un momento dado**, puede definirse como el **conjunto de propiedades relevantes** (de la **historia** del mismo) que el sistema exhibe en dicho instante.
- Implícita en el **estado** del sistema, se encuentra toda la **información** acerca de los **estados** y **entradas previas** y que, éste necesita **recordar** (o sea que tiene alguna clase de **memoria**) para evolucionar correctamente frente a nuevos **estímulos**
- Un sistema mientras está en un estado está **esperando un evento** o **realiza una actividad**

Significado de estado

State behavior

- Useful when an object
 - ▶ Exhibits discontinuous modes of behavior
 - ▶ Is reactive – waits for and responds to incoming events
- For example a Light can be :

▶ Off



▶ On



▶ Flashing



- A light can be characterized as having 3 states – On, Off & Flashing.

▶ The system can only reside in one of these states at a given time.

A state machine can be used to represent state behaviour.

Técnicas de descripción de estados



- En muchos casos, la **evolución y respuestas del sistema a eventos (mensajes recibidos, tiempo rebasado, errores, etc.)** pueden ser convenientemente expresados gráficamente mediante un **diagrama de estados**.
- El diagrama de estados no es más que una de las múltiples representaciones gráficas de una “**máquina de estados finitos**” que luego definiremos formalmente

Técnicas de descripción de estados



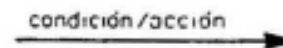
- Un diagrama de estados está compuesto por elementos de dos tipos: **nodos** y **arcos**.
- Los **nodos** son dibujados como **círculos** y se los usa para representar **estados**, los cuales son identificados mediante un **nombre** inscripto en el interior del círculo.

Técnicas de descripción de estados

- Los **arcos** se los dibuja como una flecha y representan **transiciones entre estados**. Asociada a cada flecha se encuentran:
 - una o más **condiciones o eventos**, los cuales, en caso de verificarse dan lugar a una **transición**
 - Y, una **lista de acciones** con las que el sistema responde cuando dicha transición ocurre.



Estado



Transición

Fig. 1 — Elementos de un diagrama de estado.



Ejemplo muy sencillo

- Máquina expendedora de golosinas: vende chicles a 0.15\$ y chocolates a 0.2\$ (ejemplo del 1986!!!). La máquina acepta monedas de 5 y 10 centavos. Cada vez que se introduce una moneda, la máquina emite un sonido (click), si dicha moneda es de 5 centavos o, dos (dos clicks) si es de 10 centavos. Se tienen 2 botones selectores para elegir la golosina deseada y uno de devolución que permite recuperar el dinero ingresado sino se efectuó la entrega de ninguna golosina.

Ejemplo muy sencillo

- Se llega al diagrama de estados con los siguientes pasos:
- Definición del estado inicial del sistema



Fig. 2 — Estado inicial del sistema.

- Estando en el estado "cero", el sistema es excitado mediante la inserción de una moneda. Si la moneda es de 5 centavos, el sistema pasa al estado "cinco" y emite un "click", por el contrario si la moneda es de 10 centavos se pasa al estado "diez" emitiendo un doble "click".

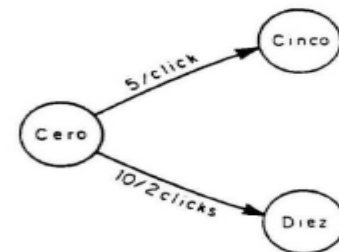


Fig. 3



Ejemplo muy sencillo

- Estando en el estado “cinco”, los estímulos que el sistema reconoce son:
 - Monedas de 5 y 10 centavos, las cuales en caso de introducirse, ocasionan una transición al estado “diez” o “quince”, respectivamente y, la emisión de 1 o 2 “clicks”, según corresponda.
 - Botón de devolución “dev”. En caso de oprimirse este botón, la máquina devuelve 5 centavos y vuelve al estado inicial “cero”

Ejemplo muy sencillo

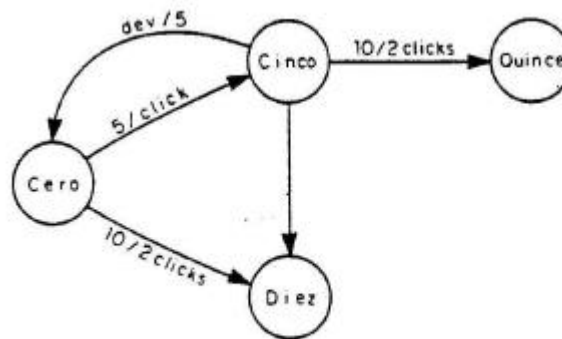


Fig. 4



Ejemplo muy sencillo

- En la figura 5 se aprecia el diagrama de estados completo de la máquina expendedora. En caso de presionar el botón de “chicle”, estando en el estado “veinte” la máquina responde entregando el chicle y además los 5 centavos de vuelto (“chicle” + “5”).

Ejemplo muy sencillo

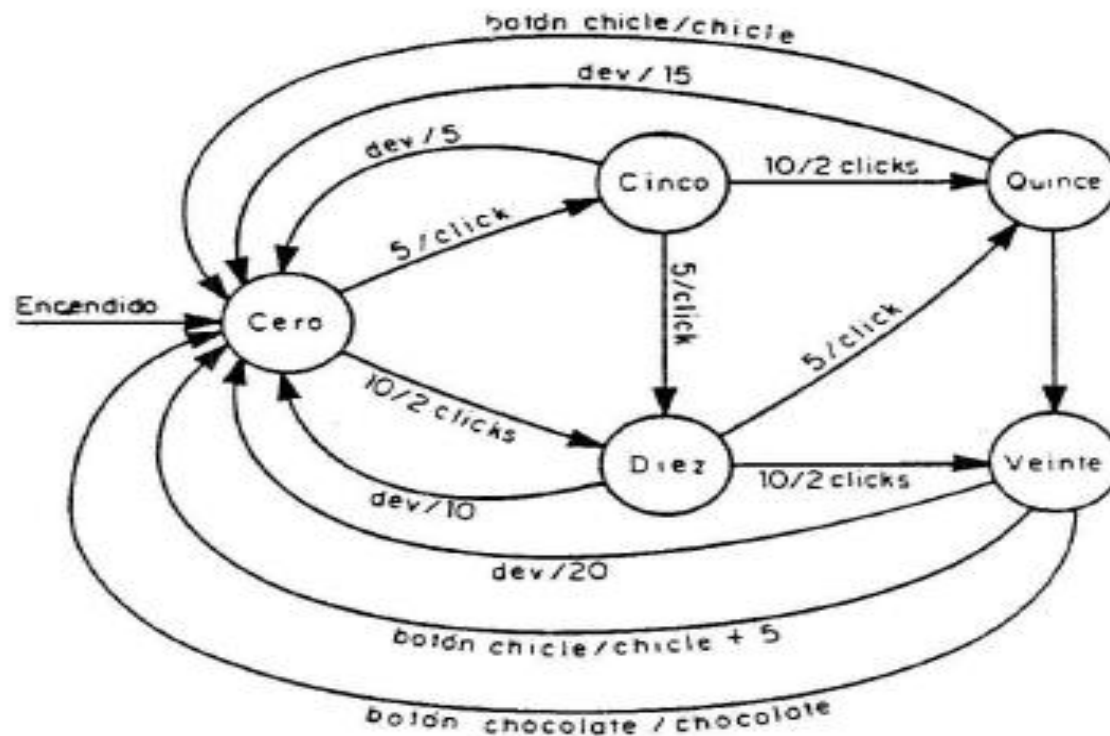
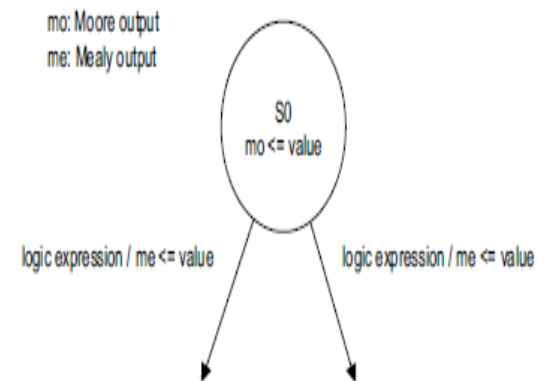


Fig. 5 — Diagrama de estados de una máquina expendedora de golosinas.

Diagramas de estado

- Este tipo de diagrama corresponde a las llamadas máquinas de Mealy (G. H. Mealy, 1955), en ellas las salidas son función del estado presente y las entradas presentes.
- En las máquinas de Moore, las salidas son sólo función del estado. Suelen usarse modelos mixtos.





Estados

- El sistema representado por el diagrama estará, en determinado instante, en un **estado estable** o **ejecutando las acciones especificadas en un transición**
- Un **estado** funciona como una **memoria** de la secuencias de eventos que ocurrieron
- El sistema **estará** exactamente en **único estado** en un diagrama de estado
- Cada estado tiene un nombre (sustantivo, **participio, gerundio...**)
- Estados distintos, implican **reacciones distintas** ante el mismo evento



Disparadores, eventos y transiciones

- Un evento representa la **ocurrencia de un suceso** que provoca un cambio de estado (dispara una transición)
 - La sucesión de transiciones marca el “camino” seguido por el sistema entre los estados
 - Estados y transiciones representan respectivamente **intervalos** e **instantes de tiempo**
 - Una transición puede tener varios eventos vinculados



Acciones en una transición

- Una acción (en hardware se llaman salidas) es un comportamiento que se ejecuta de modo **instantáneo** (con tiempo de ejecución despreciable) y **atómico** (no interrumpible)
 - Está asociada a un evento que dispara una transición
 - Es **completada** antes de que la transición alcance el nuevo estado. No se procesa o acepta ningún evento nuevo hasta que ello ocurra.

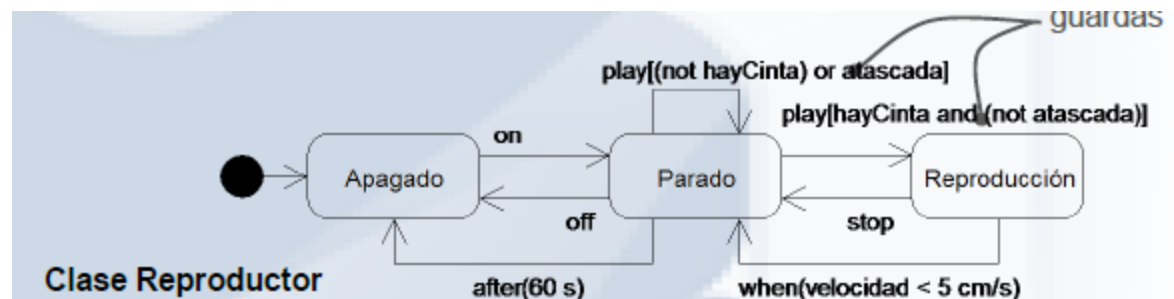


Algunas extensiones

- En 1987 David Harel propone una representación de MEF extendida llamada Statechart (o MEF jerárquicas) que extienden las MEF tradicionales MEF con características muy potentes
- En las transparencias siguientes se ven algunas de dichas extensiones que suelen agregarse a los diagramas de estados

Transiciones con guardas

- Una guarda es una condición lógica que es evaluada en el momento de recibir el evento y autoriza o no la transición de estado
 - Si la transición no es autorizada por la guarda, el evento no tiene efecto
 - Varias transiciones disparadas por un mismo evento pueden estar guardadas por condiciones mutuamente exclusivas





Estados con acciones y actividades

- Dentro de un estado pueden ejecutarse dos tipos de comportamientos:
 - **Acciones** asociadas a eventos puntuales que ocurren dentro de un estado: de entrada (entry)/de salida (exit)/dependientes de las entradas y el estado actual (input action). Las vistas hasta ahora son acciones realizadas durante el cambio de estado (transaction action)
 - **Actividades** asociadas al estado como tal
 - Una actividad es **duradera** e **interrumpible**. Ej: enviando un mensaje, esperando, calculando (función gerundio)
 - Puede modificar una condición que genere un evento de cambio de estado

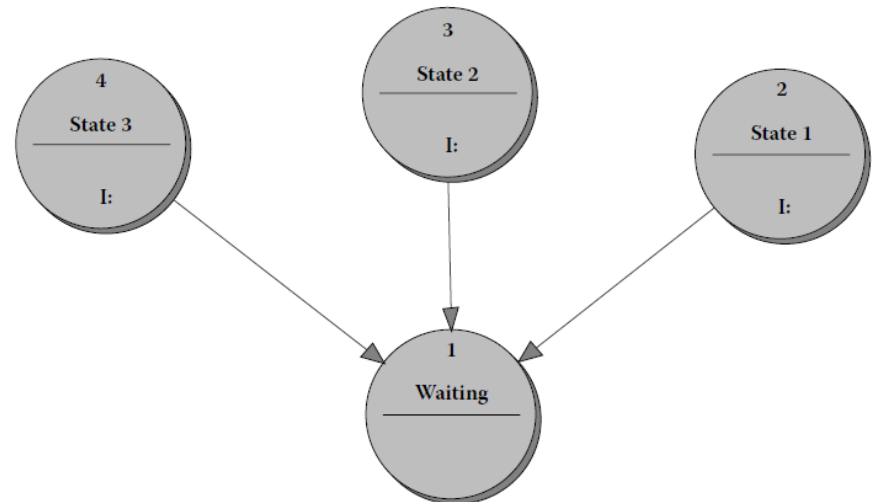


Estados con acciones y actividades

- El uso de acciones de entrada/salida asociadas un estado y, simples adiciones tomadas de las Statecharts, permite que al implementar una MEF en C, en nuestro caso, **no repetir código (y acciones en el diagrama)** cuando se tienen más de una transición de entrada (o salida) a (desde) un estado con la misma acción asociada
- Recordar que el estado no puede cambiar mientras esté haciendo este tipo de acciones (ininterrumpibles)

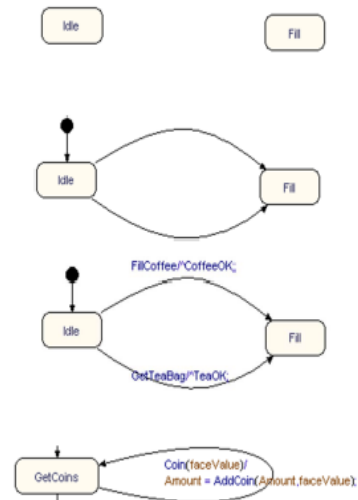
Estados con acciones y actividades

- En este trozo de diagrama, el sistema debe permanecer en el estado Waiting durante un determinado tiempo (*time-out*) controlado por un timer, que se arranca en la transición hacia Waiting desde cada uno de los 3 estados, a partir de los cuales puede alcanzarse Waiting; acción que debería poner en las 3 transiciones hacia Waiting. Una solución más elegante: colocar el disparo del timer como acción de entrada del estado Waiting



Para realizar un diagrama de estados

- Identifique estados del sistema (la decisión más importante)
- Identifique y defina las transiciones entre ellos
- Identifique eventos y acciones (no necesariamente se tienen en un diagrama todos los tipos de acciones y actividades mencionadas)
- Muy importante tener en cuenta el orden en que se ejecutan acciones y actividades al dispararse una transición (recordar al implementar en código)



¿Qué ocurre cuando se recibe un evento?



- ¿Es relevante para el estado actual?
 - Si no hay ninguna transición o evento interno disparados por ese evento, el evento se descarta
- ¿Existe una guarda asociada al evento?
 - Si la transición no es autorizada por la guarda, el evento no tiene efecto
- ¿Existe una actividad asociada al estado origen?
 - Se para
- ¿Existe una acción de salida del estado actual?
 - Se ejecuta la acción o secuencia de acciones
- ¿Existe una acción asociada al evento (transición)?
 - Se ejecuta la acción o secuencia de acciones
- ¿Existe una acción de entrada al próximo estado?
 - Se ejecuta la acción o secuencia de acciones
- ¿Existe una actividad asociada al estado destino?
 - Empieza a ejecutarse



Sistemas reactivos

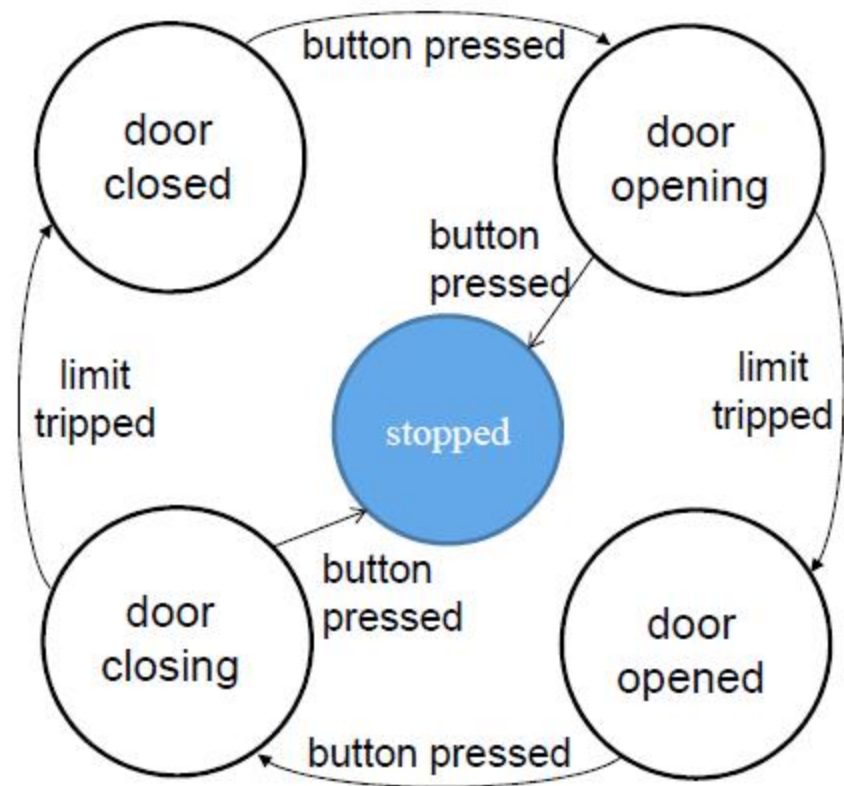
- Los diagramas de estado son particularmente útiles para modelar sistemas implementados con un MCU porque estos suelen ser **reactivos**
 - O sea, responden a **eventos externos o internos** que no necesariamente tienen orden o periodicidad, efectuando la **acción** apropiada en un contexto particular (**estado**).
 - **Eventos**: cambio de nivel de una señal en los pines de I/O, un mensaje que viene de otra parte del sistema a través de alguna interfaz o simplemente la expiración de un timer interno.
 - Por su estructura **simplifican la realización de tareas repetitivas**. Ej.: revisar las veces que se presiona un pulsador



Sistemas reactivos embebidos

- Ejecuta una porción de código en respuesta a un evento
- Permanece en reposo durante la ausencia de eventos
- En ciertos casos, se les requiere respuesta “inmediata”
- Manejan diversas actividades al mismo tiempo
- Se ejecutan en ambientes con recursos limitados
- Requieren lidiar con el acoplamiento y relación entre actividades del sistema
- Generalmente, el acceso a dispositivos y recursos se comparte entre actividades concurrentes...

Sistemas reactivos



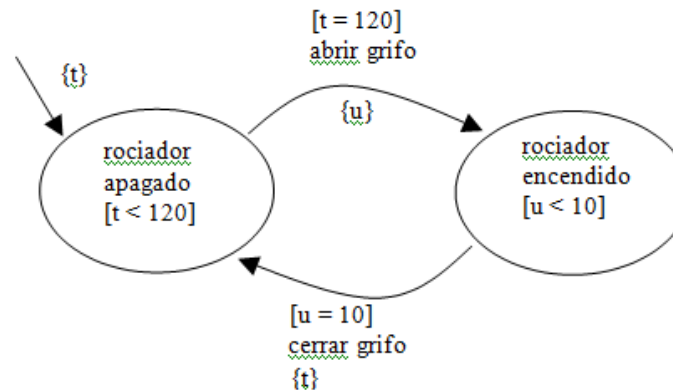
Sistemas con temporización

Ejemplo

t: timer. En minutos.

u: timer. En minutos.

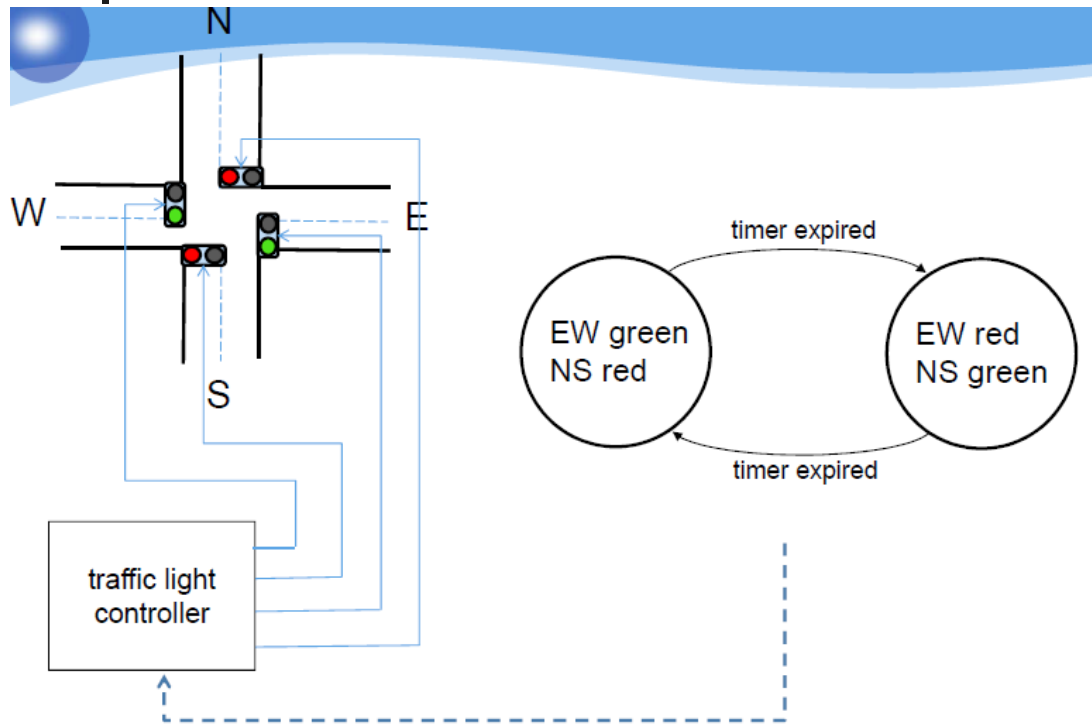
Rociador



Muchas veces los disparadores de los eventos que provocan los cambios de estado de las máquinas son iniciados por el paso del tiempo. Ej.: cambio de luz de un semáforo, o un sistema de dosificación de químicos en una cañería que cada determinada cantidad de tiempo inyecta distintas sustancias en un orden especial

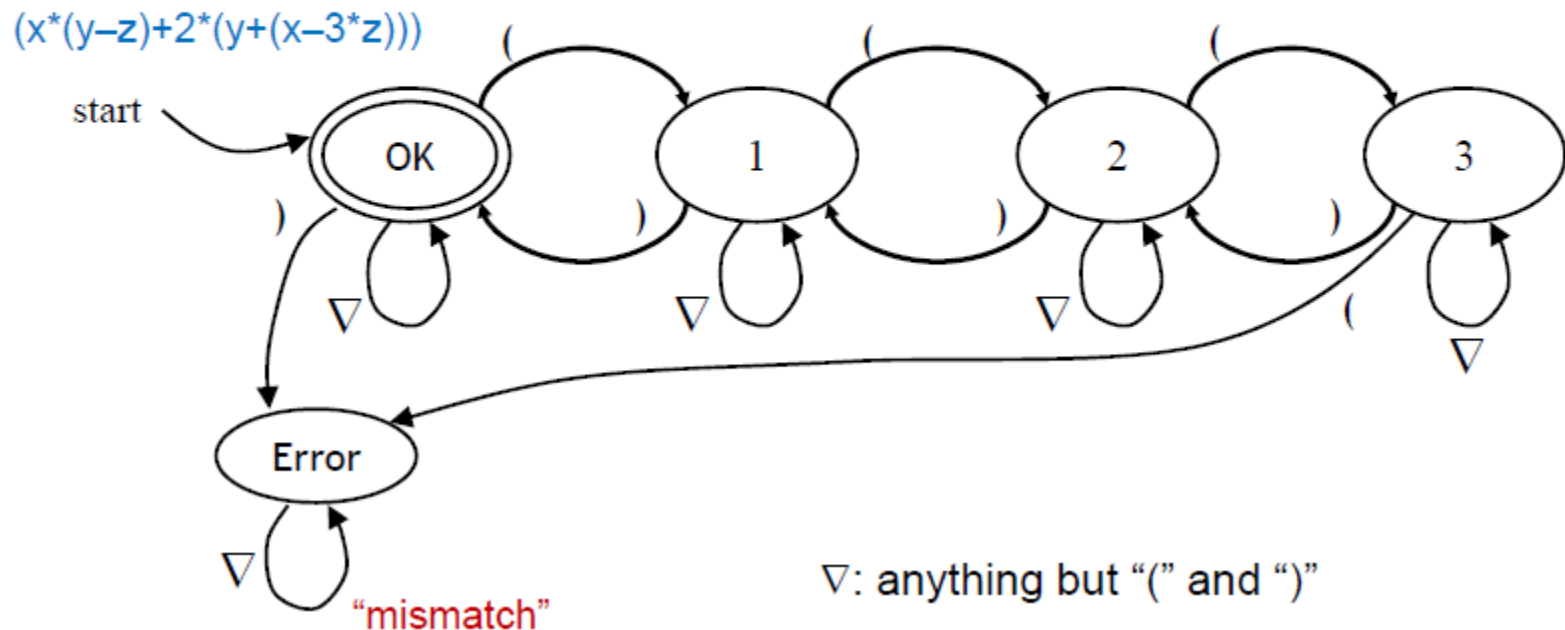
Descripción: inicialmente un rociador esta apagado y empieza la cuenta regresiva de t , pasan 120 minutos entonces el rociador empieza a funcionar durante 10 minutos, una vez que transcurren 10 minutos se cierra el grifo y vuelve a iniciar la cuenta regresiva.

Sistemas con temporización

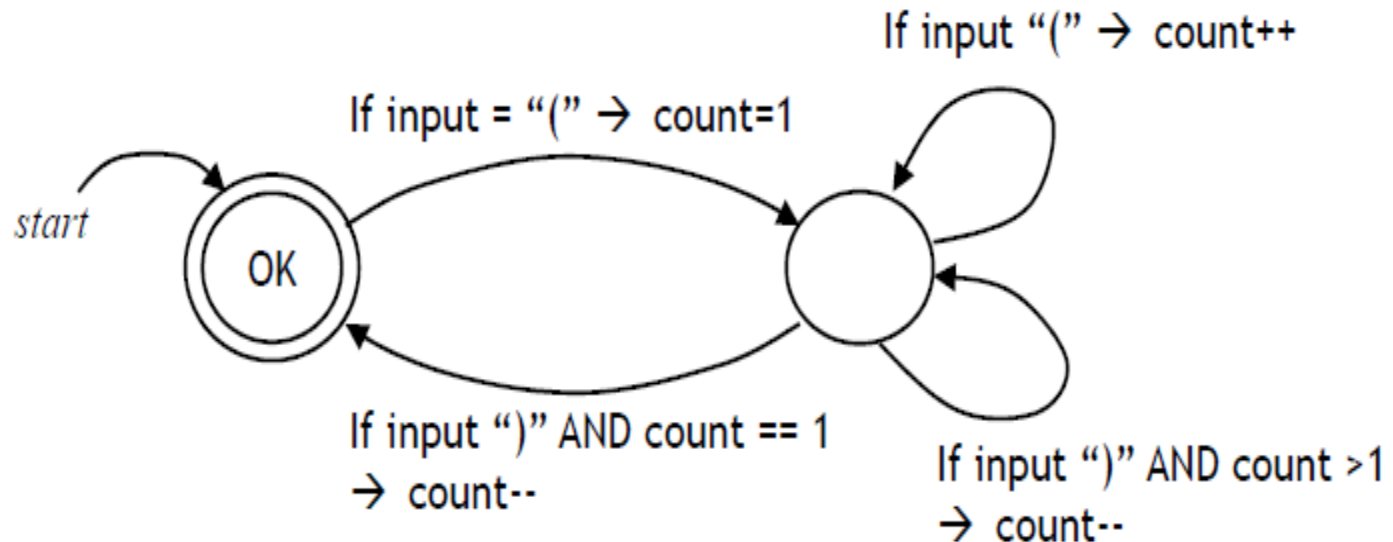


Detección de secuencias

- ❖ The following example tests whether parentheses are properly nested (up to 3 deep)

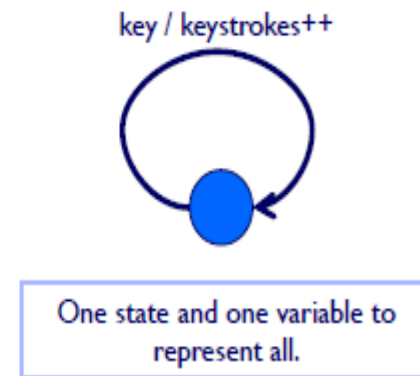
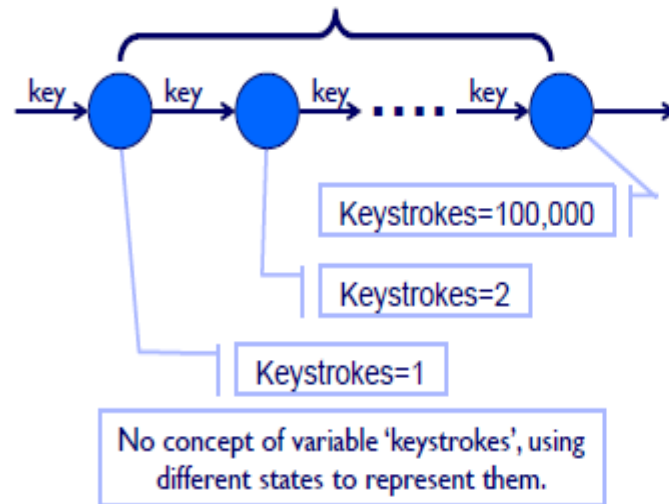


Para cualquier nivel de anidamiento de ()

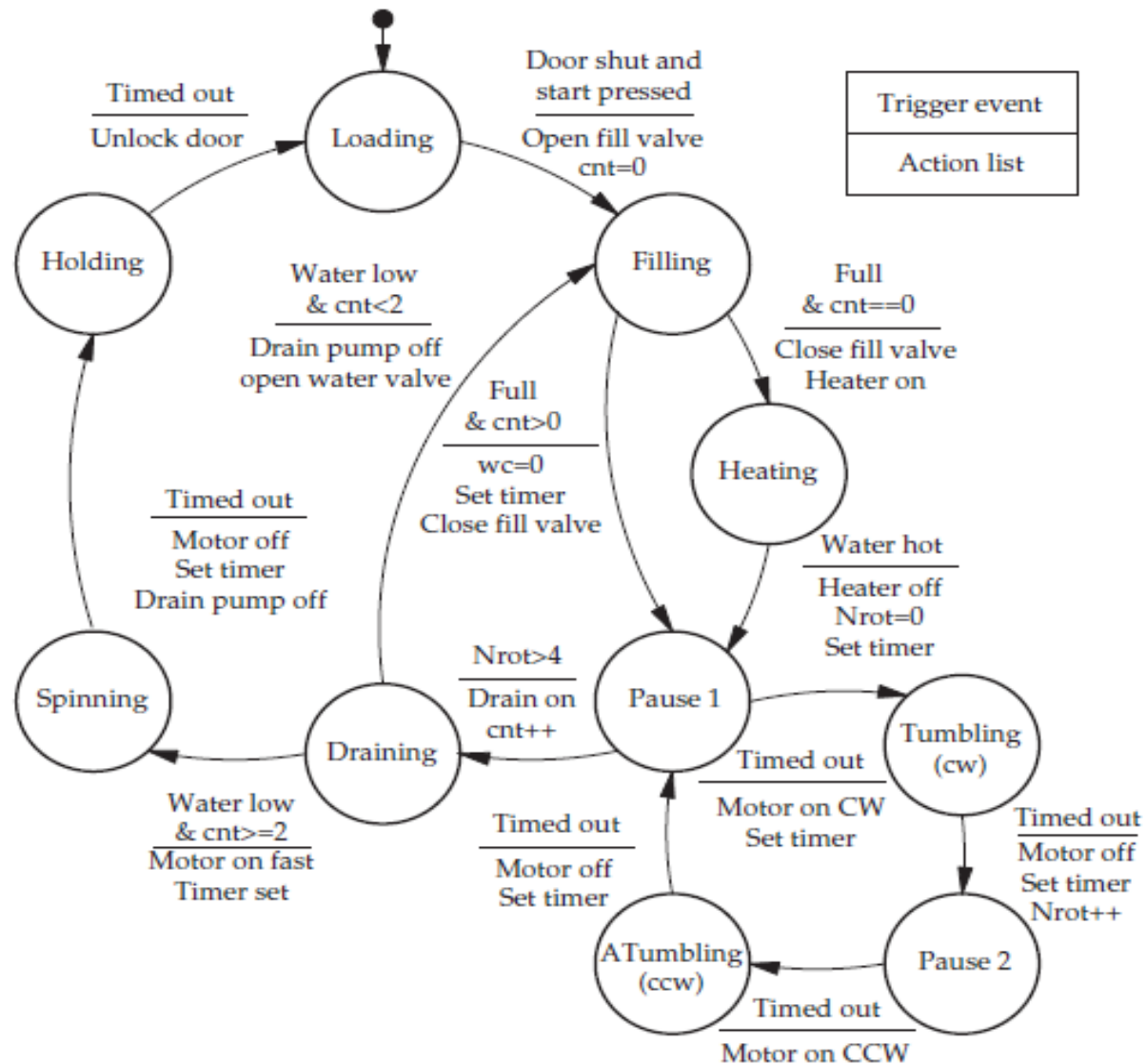


Variables auxiliares

- En el ejemplo anterior, aparte de las entradas se usa una **variable auxiliar** que funciona como un contador (de (0)), las mismas se introducen para reducir la complejidad del diagrama, sin introducir demasiada complejidad (igual no abusar de su uso). Otro ej: Si se necesitaría contar hasta 100.000 pulsados de una tecla!:

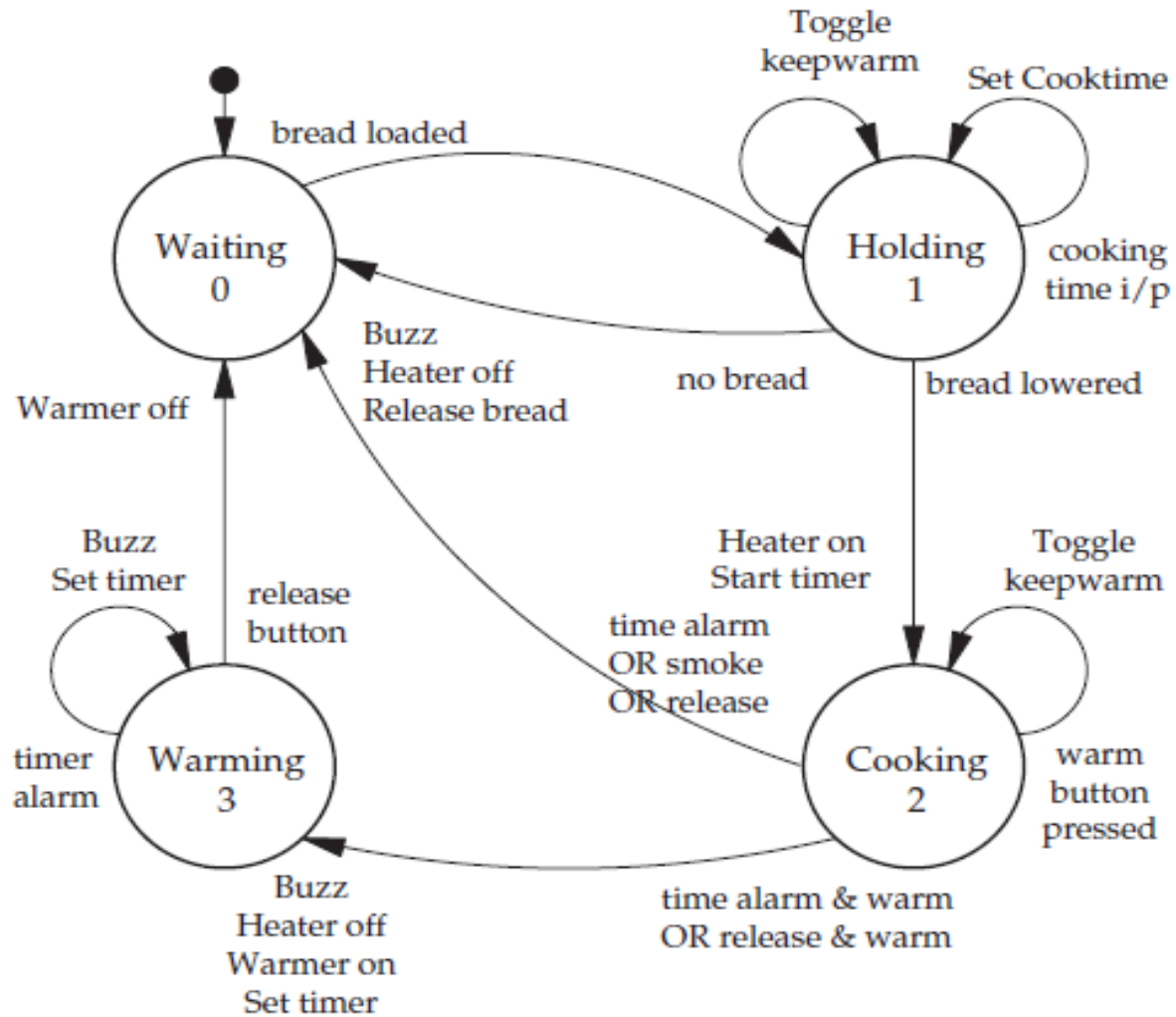


Otros ejemplos de sistemas reactivos: Lavarropa básico



FSD for a basic washing machine

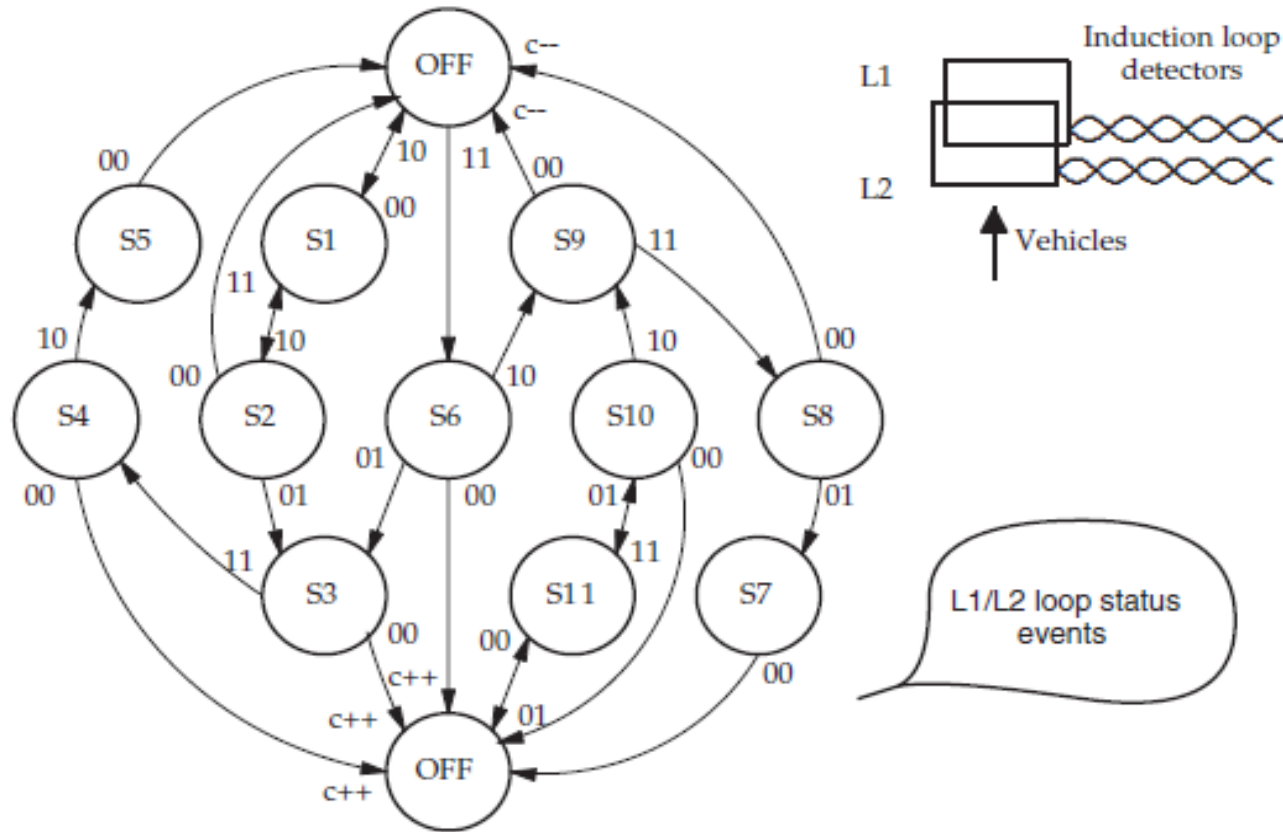
Otros ejemplos de sistemas reactivos: Tostadora



Example FSD: the toaster

Otros ejemplos de sistemas reactivos: Detección de vehículos con 2 sensores

Vehicle detection



FSD for a vehicle induction loop detector unit

Detección de estado de pulsador con anti-rebotes

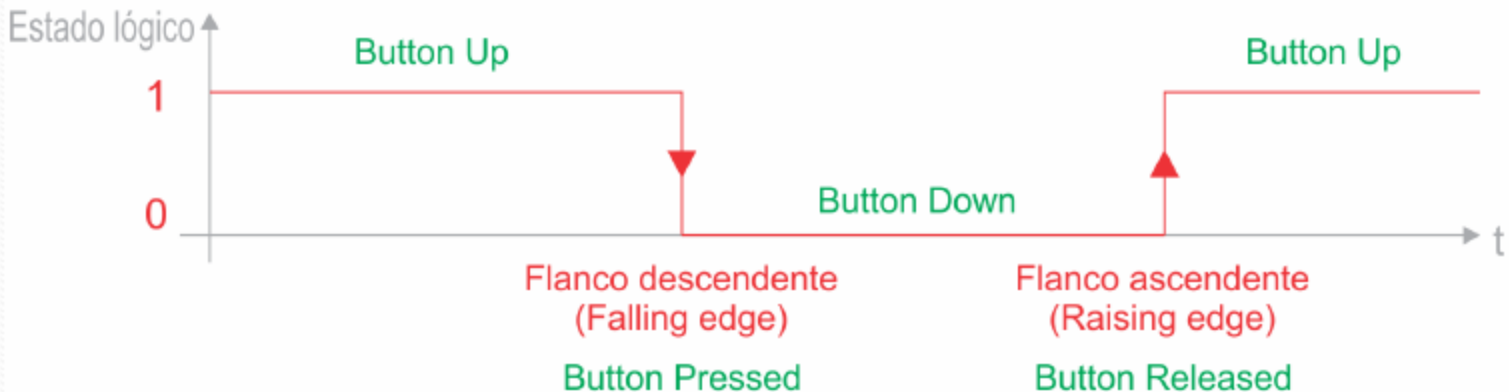
Fuente: Ing. Eric Pernia. Presentación, Clase3. Diagramas de Estado. Programación Orientada a Eventos

Voltaje en el pin del microcontrolador:



Detección de estado de pulsador con anti-rebotes

Queremos que el microcontrolador detecte a partir del voltaje Ingresado en el pin lo siguiente:





Detección de estado de pulsador con anti-rebotes

Estados:

- `BUTTON_UP`: Mientras el botón está liberado.
- `BUTTON_FALLING`: Mientras esta ocurriendo el flanco descendente, hace el anti-rebote.
- `BUTTON_RAISING`: Mientras esta ocurriendo el flanco ascendente, hace el anti-rebote.
- `BUTTON_DOWN`: Mientras el botón está presionado.



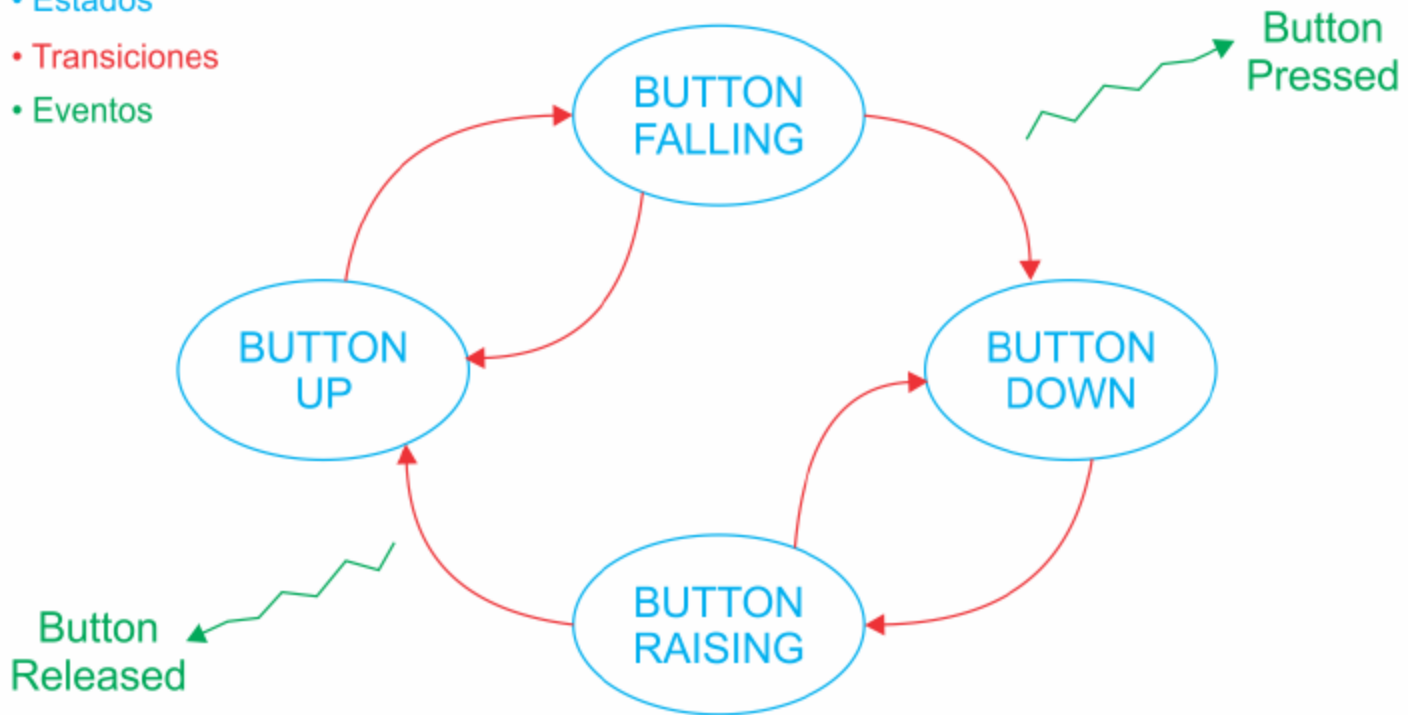
Detección de estado de pulsador con anti-rebotes

Condiciones de transición entre estados:

- Si está en estado UP y presionan el pulsador pasa al estado FALLING.
- Si está en estado FALLING, espera un tiempo de 40 ms y vuelve a leer el pulsador, si realmente estaba presionado pasa al estado DOWN y llama a la función (evento) `buttonPressed()`; si no estaba presionado vuelve al estado UP.
- Si está en estado DOWN y liberan el botón pasa al estado RAISING.
- Si está en estado RAISING, espera un tiempo de 40 ms y vuelve a leer el pulsador, si realmente estaba liberado pasa al estado UP y llama a la función (evento) `buttonReleased()`; si no estaba liberado vuelve al estado DOWN.

Detección de estado de pulsador con anti-rebotes

- Estados
- Transiciones
- Eventos





Máquina de estados finitos

- Formalmente una máquina de estados finitos es el arreglo séxtuple: (S, s_0, I, O, F, G) , donde,
 - S es el conjunto (finito) de **estados** posible
 - s_0 : es el **estado inicial**
 - I : es el conjunto de **entradas o estímulos** a los que el sistema responde
 - O : es el conjunto de **salidas o acciones** con las cuales el sistema responde
 - F : $(S \times I)$ función que determina, dados un cierto estado y un cierto estímulo, cual va a ser el próximo estado
 - G : $(S \times I)$ función que, dados un cierto estado y un cierto estímulo, determina la salida que el sistema produce en respuesta a dicho estímulo



¿Cómo se las conoce?

- Existen múltiples terminologías:
 - Statechart
 - FSM & EFSM (Finite State Machine & Extended FSM)
 - PFSM & CFSM (Partial & Complete FSM)
 - TFSM (Timed FSM)
 - DFSM & NFSM (Deterministic & Nondeterministic FSM)
 - LTS (Labelled Transition System)
 - Automata (teoría de lenguajes)
 - Timed Automata
 - IOA (Input Output Automata)
 - Etc., etc., etc.



¿Cómo se formalizan?

- Mediante diversos métodos de representación:
 - Funciones matemática
 - Diagramas / Tablas de Estados y Transiciones
 - Redes de Petri
- Existen métodos para minimizar estados
- Mediante diversos lenguajes de programación:
 - Assembly, C p/micros o PCs
 - C++, Java, UML
 - VHDL, etc., etc., etc.

El hecho de poder formalizarlas permite obtener modelos ejecutables. Aquí las representaremos gráficamente mediante diagramas de estado, e implementaremos en lenguaje C para el MCU; en otras materias se verán más formalizaciones



Máquina de estados finitos

- En el ejemplo de la máquina expendedora de golosinas:

$S = \{\text{cero, cinco, diez, quince, veinte}\}$

$s_0 = \text{cero}$

$I = \{5, 10, \text{botón, chicle, botón, chocolate, dev.}\}$

$O = \{\text{click, 2 clicks, chicle, chicle} + 5, \text{chocolate, 5, 10, 15, 20}\}$

- Las funciones F y G pueden expresarse en forma tabular

Máquina de estados finitos

F

| | 5 | 10 | bot. chicle | bot. choc. | dev. |
|--------|--------|--------|-------------|------------|------|
| cero | cinco | diez | | | |
| cinco | diez | quince | | | cero |
| diez | quince | veinte | | | cero |
| quince | veinte | | cero | | cero |
| veinte | | | cero | cero | cero |

Fig. 8 — Función F.

G

| | 5 | 10 | bot. chicle | bot. choc. | dev. |
|--------|-------|----------|-------------|------------|------|
| cero | click | 2 clicks | | | |
| cinco | click | 2 clicks | | | 5 |
| diez | click | 2 clicks | | | 10 |
| quince | click | | chicle | | 15 |
| veinte | | | chicle + 5 | chocolate | 20 |

Fig. 9 — Función G.

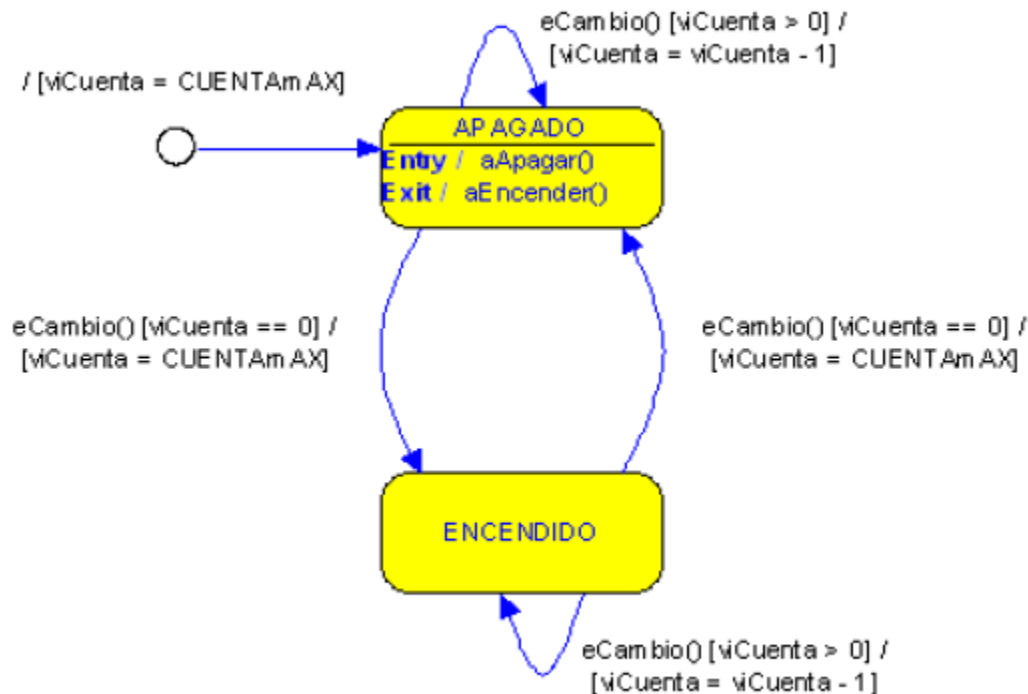
Limitaciones de las MEF tradicionales



- Los sistemas reactivos complejos se vuelven difíciles de representar mediante un modelo de MEF tal como el visto (plano y sin soporte de concurrencia), debido fundamentalmente al **crecimiento exponencial de estados y transiciones**, resultando en un diagrama caótico e inmanejable, perdiendo así las virtudes del modelo
- No captura eficientemente el comportamiento común, no permitiendo la **reutilización**

Statecharts

Control de LED con estado **Variable Perpetuo** manejado por **evento-cambio & condición** (control p/prog.) (dos estados simples)



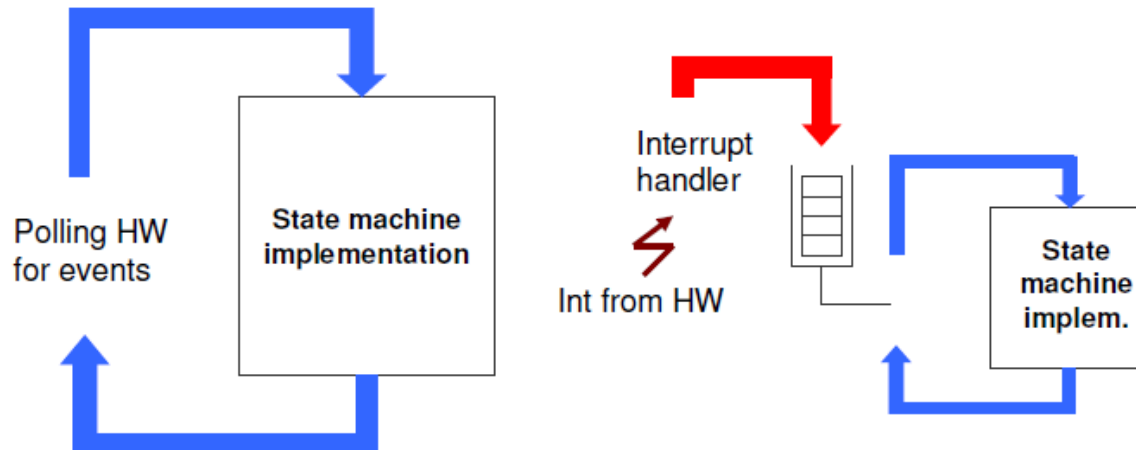


Implementaciones típicas en C

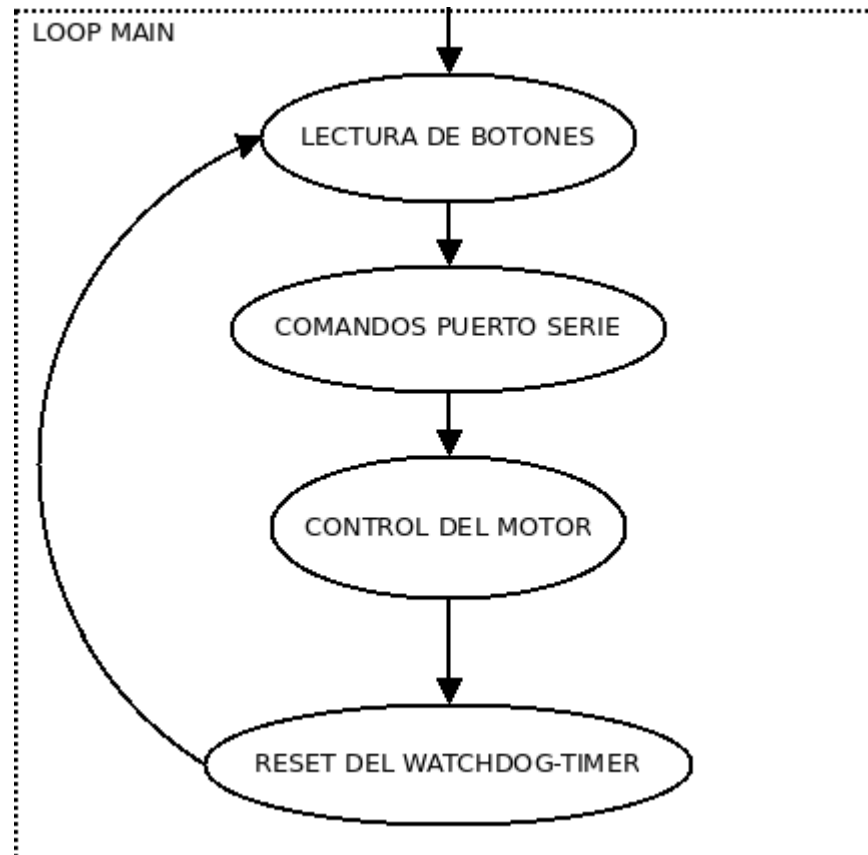
- Mediante **switch anidados** o **múltiples if**
- **Arrays de punteros a función.** Se representa la **tabla de transición de estados**, mediante una matriz (casi todos los elementos son vacíos); se usan **funciones (pequeñas y manejadas a través de punteros)** que se encargan de **responder a los eventos (generar acciones)** y de **actualizar el estado actual**
- Existen otras, aquí veremos estas 2

Implementaciones típicas en C

- La implementación de una FSM no es suficiente!
- Estas implementaciones típicamente son acopladas con un modelo de concurrencia y una política de despacho de los eventos



Implementaciones típicas en C





Implementación con switch anidados

- Se tiene un nivel de decisión superior (**switch externo**) referidos a los **estados** de la FSM (controlado por una variable de estado escalar)
- Se tiene un nivel de decisión inferior (**switch interno**) referido a las **señales, eventos o condiciones** que tienen significancia en ese estado. Si las señales son pocas pueden reemplazarse por ifs.
- A veces se invierte la jerarquía
- Suelen usarse enumeraciones para los estados y eventos (fácil añadir nuevos, aunque no es fácil añadir código nuevo para su procesamiento). Suelen declararse en los archivos de cabecera
- Implementado:
 - Directamente en el main (totalmente o como parte de él)
 - Haciendo uso de una función ("dispatcher") para ello

Implementación con switch anidados

► Modelo:

```
Proximo_Estado= Estado_Actual;
switch ( Estado_Actual)  {
    case EST_1:
        //aquí van acciones dentro del estado 1
        switch (evento ) { //solo eventos que cambian este estado
            case EVEN_1:
                Proximo_estado = EST_x;
                // aquí pueden ir acciones asociadas al evento para este cambio de estado
                break;
            case EVEN_4:
                Proximo_estado = EST_y;
                // aquí pueden ir acciones asociadas el evento para este cambio de estado
                break;
            . . .
        }
    break;
```

(continua en la siguiente diapositiva)

Implementación con switch anidados

```
case EST_n:
    //aquí van acciones dentro del estado n
    switch (evento ) { //solo eventos que cambian este estado
        case EVEN_3:
            Proximo_estado = EST_x;
            // aquí pueden ir acciones asociadas al evento para este cambio de estado
            break;

        ...
    }
    break;
}
if (Estado_actual!=Proximo_estado){ // solo en caso de utilidad particular
    //acciones para todo cambio de estado
    funcion_salida(Estado_actual); // solo en caso de utilidad particular
    funcion_entrada(Próximo_estado); // solo en caso de utilidad particular
}
```



Detalle plantilla

```
1 /*--- aqui van las variables de control de la(s) FSM y su inicializacion:
2 -----      o como globales (una o varias FSMs)
3 -----      o como estáticas locales a cada FSM implementada en el micro
4 -----*/
5
6 //--- este es el modelo para implementar la FSM
7 Proximo_Estado= Estado_Actual;
8 switch ( Estado_Actual) {
9     case EST_1:
10         //
11         // --- aqui van acciones que se desarrollan dentro del estado 1
12         // --- posiblemente atendiendo algunos eventos
13         //
14         switch (evento ) { //solo eventos que cambian este estado
15             case EVEN_1:
16                 Proximo_estado = EST_x;
17                 //
18                 // aquí pueden ir acciones asociadas al evento para este cambio PARTICULAR de estado
19                 //
20                 break;
21             case EVEN_4:
22                 Proximo_estado = EST_y;
23                 //
24                 // aquí pueden ir acciones asociadas el evento para este cambio PARTICULAR de estado
25                 //
26                 break;
27             . . .
28         }
```




Detalle plantilla

```
43 if (Estado_actual!=Proximo_estado){ // solo en caso de utilidad particular
44     //
45     //-- aqui van acciones para todo cambio de estado
46     //
47     funciones_salida(Estado_actual); // solo en caso de utilidad particular: funciones EXIT de la FSM
48     funciones_entrada(Próximo_estado); // solo en caso de utilidad particular: funciones ENTRY de la FSM
49     Estado_actual=Proximo_estado;
50 }
51 else
52     funciones_de_estado(Estado_actual)//solo en caso de utilidad particular: funciones INPUT de la FSM
53
```

Implementación con switch anidados

► Una variante (de muchas):

```
void dispatch(unsigned const evento) {  
    switch(estado) {  
        case EST_1:  
            ProcesarEst1(evento);  
            break;  
        case STATE_2:  
            ProcesarEst2(evento);  
            break;  
        ...  
    }
```

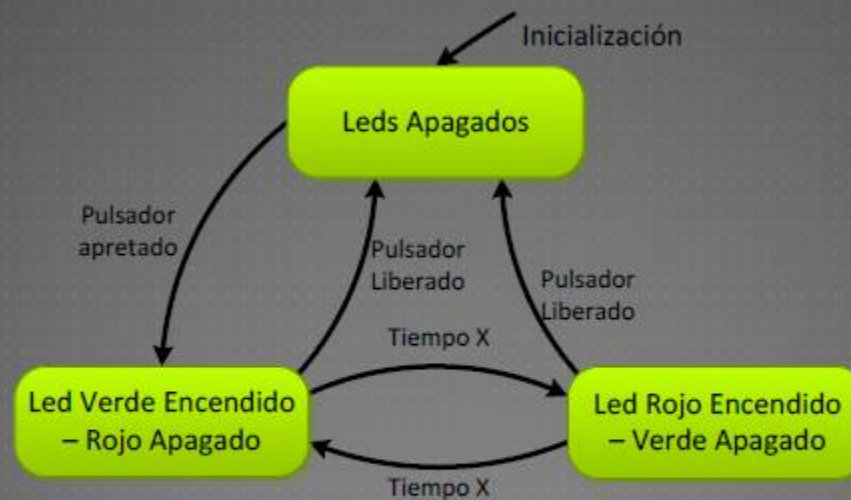


Características de este método

- Es simple
- Requiere enumerar estados y señales
- Poco uso de RAM (se requiere una única variable escalar para almacenar estado actual)
- No promueve la reutilización de código
- El tiempo de despacho de los eventos no es constante. Se incrementa con el número de casos.
- Implementación no jerárquica. Difícil de mantener cuando se efectúan cambios
- Compleja cuando los estados y eventos aumentan
- El código perteneciente a un estado está distribuido y repetido en muchos lugares (en cada transición que conduce a aquél estado)

Ejemplo

- Ejemplo: los leds titilan alternativamente sólo cuando está el pulsador accionado



(hacer código con switch anidados)

Ejemplo

main.c

```
/*=====ejercicio: mientras se mantiene apretado el pulsador sw1 los leds  
*=====parpadean alternativamente (dado como diagrama de estados de ejemplo)  
*===== */
```

```
#include "board.h"
```

```
#define RETARDO      50000
```

```
enum estados {NADA, VERDE, ROJO}; // estos son los estados segun el diagrama
```

```
int main(void)  
{  
    board_init();  
    enum estados estado_actual=NADA;  
    enum estados proximo_estado=NADA;  
    int8_t fin_tiempo;  
    volatile int32_t temporizando;  
    fin_tiempo=0;  
    temporizando=0;
```



¿Por qué uso volatile?

- El modificador **volatile** le indica al compilador que la variable puede sufrir modificaciones que no estén explícitas en el código
 - Registros de entrada
 - Variables globales modificadas por rutinas de interrupción
- De esta forma se evita que el compilador haga optimizaciones que podrían ser erróneas
- Facilitan el debug



¿Por qué uso volatile?

<http://www.indicart.com.ar/seminario-embbebidos/Elementos%20de%20C%20Embebido.pdf>

- Ej., Supongamos que la variable `clock` se incrementa regular y automáticamente, y la usamos en este código:

```
uint32_t volatile clock;  
  
foo = clock;  
while (clock < foo+10) { ... };
```

Si `clock` no fuera declarada como `volatile`, el compilador podría asumir que la condición del `while` ocurre siempre

- ...y entonces no generar código que la calcule, ejecutándose siempre el bloque del `while`

Código ejemplo

.c main.c ✕

```
fin_tiempo=0;
temporizando=0;
for (;;) {
    /* ===== tarea 1: implementa un mecanismo de retardo muy basico e impreciso, sin timers
    * ===== solo para su uso en la tarea 2 */
    if (temporizando>0){
        temporizando--;
        fin_tiempo=0;
    }
    else
        fin_tiempo=1;
    //nada mas
```




Código ejemplo

```
// ===== tarea 2: tarea implmentada mediante una MEF con switchs
switch (estado_actual){// switch de estados
    case NADA:
        LED_ROJO_OFF;           //todo apagado en este estado
        LED_VERDE_OFF;
        //hay pocas señales: uso if/else en vez de switch
        if (pulsadorSw1_get()) //pulsador apretado
            proximo_estado=VERDE;
        break;
```

Estando en estado inicial (NADA) y no pulsé pulsador sale del switch

```
        break;
    } //fin del switch de estados
    if (estado_actual != proximo_estado){
        if (proximo_estado != NADA) //cada vez que entro a VERDE o ROJO inicio el temporizador
            temporizando=RETARDO;
        //finalmente, cambio estado
        estado_actual=proximo_estado;
    }
} //end for
return 0;
} //end main
```



Código ejemplo

- Estando en NADA (estado_actual) cuando aprieto el pulsador, pongo en el primer case proximo_estado en VERDE, salgo del switch y, como hubo un cambio de estado y proximo_estado es !=de NADA, inicializo temporizando en 50000, pongo estado_actual=VERDE, en las siguientes iteraciones, si aún no solté el pulsador, temporizando>0, se decrementa en 1 pone fin_tiempo=0. En las siguientes iteraciones sigue decrementando temporizando hasta que llegue a 0 (proximo_estado=ROJO) o hasta que suelte el pulsador (proximo_estado=NADA)



Código ejemplo

```
case VERDE:
    LED_ROJO_OFF;           //el rojo permanece apagado en este estado
                             //////////////////////////////////////////////////
    LED_VERDE_ON;           //el verde permanece encendido en este estado
                             //////////////////////////////////////////////////
    if (!pulsadorSw1_get()) //veo estado pulsador (con prioridad frente al tiempo)
                             //////////////////////////////////////////////////
        proximo_estado=NADA;
    else if (fin_tiempo)     //veo si termino el tiempo
                             //////////////////////////////////////////////////
        proximo_estado=ROJO;
    break;
```



Código ejemplo

```
        break;
    case ROJO:
        LED_ROJO_ON;           //el rojo permanece encendido en este estado
        LED_VERDE_OFF;         //el verde permanece apagado en este estado
        if (!pulsadorSw1_get()) //veo estado pulsador (con prioridad frente al tiempo)
            proximo_estado=NADA;
        else if (fin_tiempo)    //veo si termino el tiempo
            proximo_estado=VERDE;
        break;
    } //fin del switch de estados
    if (estado_actual != proximo_estado){
        if (proximo_estado != NADA) //cada vez que entro a VERDE o ROJO inicio el temporizador
            temporizando=RETARDO;
        //finalmente, cambio estado
        estado_actual=proximo_estado;
    }
} //end for
return 0;
} //end main
```

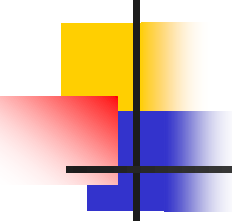


Código ejemplo

- El código del “titilado” de los leds es simétrico y al terminar la temporización en el estado VERDE y si aún no se soltó el pulsador, vuelve a temporizar en ROJO hasta que llegue a 0 y sigue “ciclando” entre estos dos estados, hasta que lo suelte donde el sistema vuelve al estado NADA y apaga los dos leds

Caso de más tareas con retardos

```
1 unsigned long int temporizadores [10]; //esto para contar diez "tiempos" distintos
2 int tiempo(int temporizador_id, int comando, unsigned long int parametro) //ejemplo de funcion que comanda la temporizacion
3
4 for(;;){
5     tarea0();
6     tarea1(); //cada tarea es una maquina de estado
7     tarea2();
8     ....
9     avanza_tiempo(); //es una tarea como cualquier otra
10 }
11
12
13 //ejemplo de implementacion
14 unsigned long int tiempo(int id_temporizador, int comando, int parametro) {
15     if (comando == RESET) temporizador[id] =0;
16     if (comando == DEV_TIEMPO) return temporizador[id];
17     if (comonda == INIT) temporizador[id]= parametro;
18     ....//otras posibilidades
19     return 0;
20 }
```



```
22 tarea1(){
23     proximo_estado=estado_actual;
24     switch (estado_actual){
25         case ESTADO1:
26             ledrojo_off;
27             ledverde_off;
28             if (Sw1) proximo_estado=ESTADO2;
29             break;
30         case ESTADO2:
31             ledrojo_on;
32             ledverde_off;
33             if (tiempo(1,DEV_TIEMPO,0)>100000) proximo_estado=ESTADO3;
34             if(!Sw1) proximo_estado=ESTADO1;
35             break;
36         case ESTADO3:
37             ledrojo_off;
38             ledverde_on;
39             if (tiempo(1,DEV_TIEMPO,0)>100000) proximo_estado=ESTADO2;
40             if(!Sw1) proximo_estado=ESTADO1;
41             break;
42     }
43     if (proximo_estado!=estado_actual{
44         estado_actual=proximo_estado;
45         tiempo(1,RESET,0);
46     }
47 }
48 void avanza_tiempo(){
49     int i;
50     for(i=0;i<=9;i++) temporizadores[i]++;
51 }
```



Implementación en C

- Siempre trate de separar el problema en tareas
- Ejemplo: Se desea controlar una lámpara (simulada por un led rojo en el MCU): Un pulsador SW1 al apretarlo la enciende, un pulsador SW2 al apretarlo la apaga, si a los 10 seg de encendida, la luz no se apagó, lo hace automáticamente, previamente a los 5 seg el led verde debe comenzar a titilar con una frecuencia de 500 mseg, avisando que la luz se apagará en breve automáticamente



Implementación en C

- Se podría dividir el problema en 2 tareas, la tarea1 controla los leds y las temporizaciones (tal vez esta podría ser otra tarea), la tarea2 se encargaría del control de los 2 pulsadores. Cuenta1 es la cuenta máxima que indica que se llegó a los 5 s. y Cuenta2 indica que se llegó a los 10 seg.
- Conectaría a las 2 tareas mediante 2 variables auxiliares, light_status (en 0 indicaría los 2 leds apagados, la pongo a 1 cuando pulso SW1, pongo a 0 cuando pulso SW2 (apagar los 2 leds). Uso otra variable auxiliar tick_count que pongo a 0 cuando pulso SW1, iniciando la temporización que controla el tiempo de encendido del led rojo.

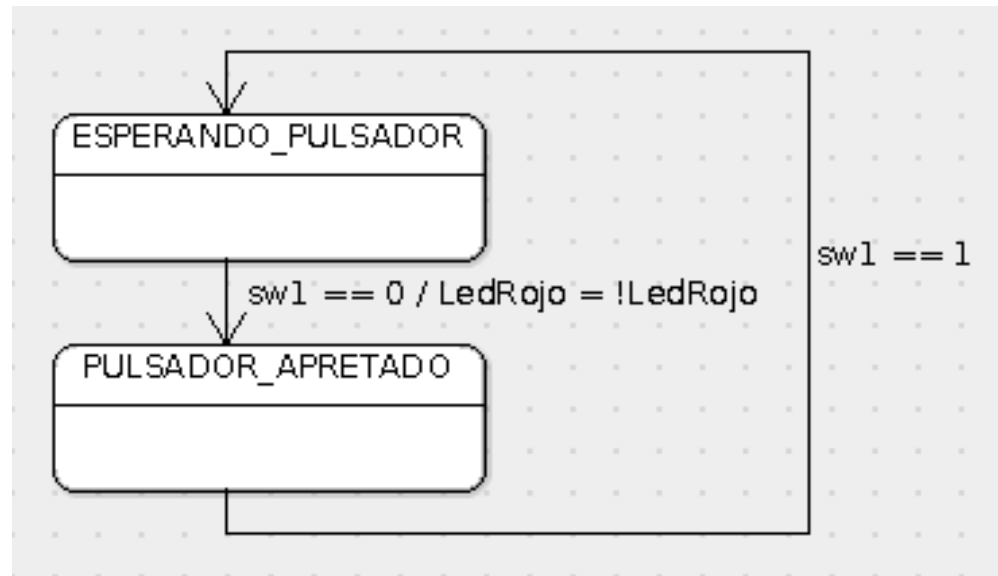


Implementación en C

- Cuando `light_status=1`, debe encenderse el led rojo, incrementar en 1 `tick_count` e ir controlando si llegó a `Cuenta1`, entonces debo hacer titilar el led verde y si llego a `Cuenta2` debo poner `light_status=0` y apagar los 2 leds.
- Con `light_status=0` apago los 2 leds y pongo `tick_count=0` dejando el sistema en estado inicial a la espera del pulsado de SW1

Implementación en C

- (Extraído de las clases prácticas de Sistemas Digitales II) Cada vez que se pulse SW1 el led rojo debe cambiar su estado (si estaba apagado debe encenderse y viceversa)





Implementación en C

- Introduzco la siguiente modificación: Una vez pulsado SW1, si éste se mantiene presionado por más de 2 segundos, a partir de ese momento el led rojo debe comenzar a titilar cada 300 milisegundos.
- El problema se dividió en 2 tareas: la lectura del pulsador y el control del led. En SDII las temporizaciones se resuelven usando timers e interrupciones
- Ambas tareas se conectaron mediante una variable tipo struct varsKey_struct (un campo contiene el próximo estado, según se pulse o se suelte el pulsador, otro que toma valores 0 o 1 (indica que el led debe cambiar de estado) y último que serían las variables para realizar las temporizaciones (de 2 seg y 300 msec.)



Implementación en C

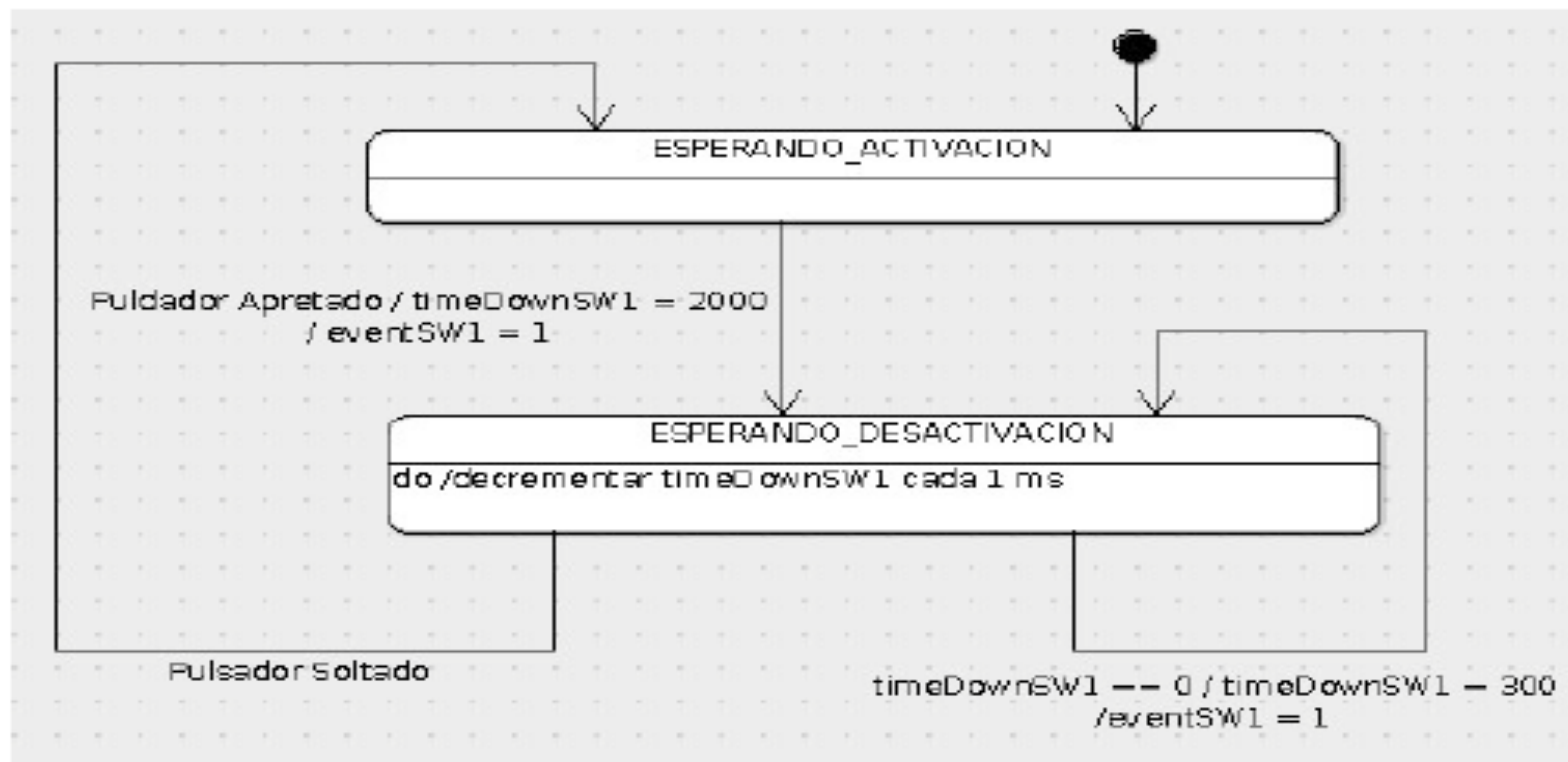
```
typedef enum
{
    EST_PUL_ESPERANDO_ACTIVACION = 0,
    EST_PUL_ESPERANDO_DESACTIVACION,
}estPul_enum;

typedef struct
{
    estPul_enum estSW;

    unsigned eventSW:1;
    uint16_t timeDownSW;
}varsKey_struct;
```

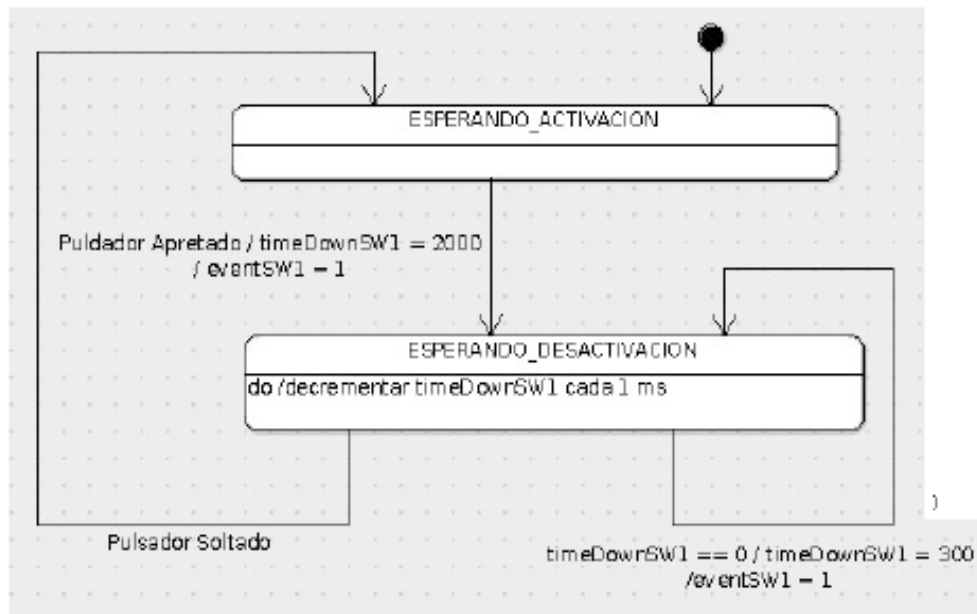
Implementación en C

LECTURA DE PULSADOR



Implementación en C

LECTURA DE PULSADORES



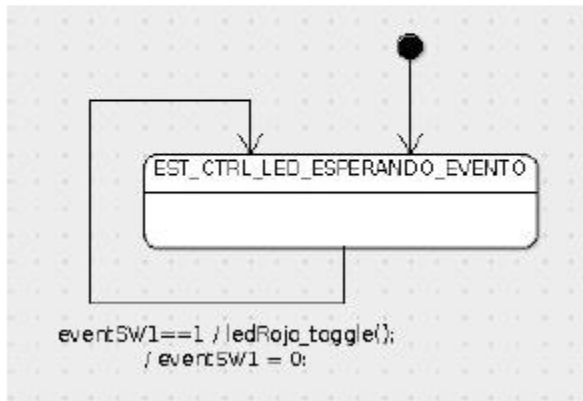
```
switch (varsKey.estSW1)
{
    case EST_PUL_ESPERANDO_ACTIVACION:
        if (pulsadorSw1_get())
        {
            varsKey.timeDownSW1 = 2000;
            varsKey.eventSW1 = 1;
            varsKey.estSW1 = EST_PUL_ESPERANDO_DESACTIVACION;
        }
        break;

    case EST_PUL_ESPERANDO_DESACTIVACION:
        varsKey.timeDownSW1--;
        if (!pulsadorSw1_get())
        {
            varsKey.estSW1 = EST_PUL_ESPERANDO_ACTIVACION;
        }
        if (varsKey.timeDownSW1 == 0)
        {
            varsKey.timeDownSW1 = 300;
            varsKey.eventSW1 = 1;
        }
        break;

    default :
        varsKey.estSW1 = EST_PUL_ESPERANDO_ACTIVACION;
        break;
}
```

Implementación en C

CONTROL DEL LED



```
if (varsKey.eventSW1)
{
    varsKey.eventSW1 = 0;
    ledRojo_toggle();
}
```




Un ejemplo interesante

- De: <https://sergioprado.org/maquina-de-estados-em-c/>
- Las fsm son muy utilizadas para rutinas de tratamientos de protocolos de comunicaciones. En este caso tiene el siguiente formato:

|STX|QTD_DADOS|DADOS|CHK|ETX|

STX (1 Byte) -> Inicio da transmissão (0x02)

QTD_DADOS (1 Byte) -> Quantidade de dados

DADOS (N Bytes) -> Dados

CHK (1 Byte) -> Checksum da transmissão

ETX (1 Byte) -> Fim da transmissão (0x03)



Ejemplo interesante

```
[-] #include <stdio.h>
    /* numero maximo de bytes do buffer de dados */
    #define MAX_BUFFER 512
    /* constantes usadas na comunicacao */
    #define STX      0x02
    #define ETX      0x03
    /* possiveis estados da maquina de estados de comunicacao */
    [-] typedef enum {
        ST_STX = 0, ST_QTD, ST_DATA, ST_CHK, ST_ETX
    [-] } States;
    /* trata dados recebidos */
    [-] void handlePackage(unsigned char *data, int qtd) {
        int i;
        printf("Imprimindo dados recebidos...\n");
        |   for (i = 0; i < qtd; i++)
            printf("Data[%d]=%d\n", i, data[i]);
        }
    }
```



Ejemplo interesante

```
/* Implementação da máquina de estados */  
void handleRx(unsigned char *data, int qtd) {  
    static States state = ST_STX;  
    static unsigned char buffer[MAX_BUFFER];  
    static int indBuffer = 0, qtdBuffer = 0;  
    static unsigned char chkBuffer = 0;  
    int i;  
    for (i = 0; i < qtd; i++) {  
        switch (state) {  
            case ST_STX:  
                if (data[i] == STX) {  
                    indBuffer = qtdBuffer = chkBuffer = 0;  
                    state = ST_QTD;  
                }  
                break;  
            case ST_QTD:  
                qtdBuffer = data[i];  
                state = ST_DATA;  
                break;  
        }  
    }  
}
```



Ejemplo interesante

```
case ST_DATA:
    buffer[indBuffer++] = data[i];
    chkBuffer ^= data[i];
    if (--qtdBuffer == 0) {
        state = ST_CHK;
    }
    break;
case ST_CHK:
    if (data[i] == chkBuffer) {
        state = ST_ETX;
    }
    else {
        state = ST_STX;
    }
    break;
```



Ejemplo interesante

```
        case ST_ETX:
            if (data[i] == ETX) {
                handlePackage(buffer, indBuffer);
            }
            state = ST_STX;
            break;
```

```
    }
```

```
}
```

```
}
```

```
/* main para simular o uso da maquina de estados */
```

```
int main() {
```

```
    unsigned char data1[] = { STX, 5, 11, 22, 33, 44 };
```

```
    unsigned char data2[] = { 55, 39, ETX };
```

```
    handleRx(data1, sizeof(data1));
```

```
    handleRx(data2, sizeof(data2));
```

```
    return 0;
```

```
}
```



Ejemplo interesante

Imprimindo dados recebidos...

Data[0]=11

Data[1]=22

Data[2]=33

Data[3]=44

Data[4]=55

Presione una tecla para continuar . . .



Implementación en C

- Siempre es deseable separar las funciones de **control** (qué tareas realizar) de las de **procesamiento** (las tareas propiamente dichas)
- Redunda en programas mejor estructurados (menores tiempos de programación y mantenimiento)
- En el siguiente método el uso de las tablas de transición de estados provee una forma de realización directa que permite implementar dicha separación



Usando punteros a función

ALTERNATIVA: MATRIZ DE TRANSICIONES

| | Señal 1 | Señal 2 | Señal 3 | Señal4 |
|----------|----------------------|----------------------|----------------------|----------------------|
| Estado X | | Accion X(), Estado_Y | | |
| Estado Y | | | | AccionY(), Estado_A |
| Estado A | Accion A(), Estado_Q | Accion B(), Estado_A | | AccionC (), Estado_A |
| Estado Q | | | Accion Q(), Estado_X | |

(Curiosidad: si las acciones son la misma en toda la línea=> Moore; si pueden variar=>Mealy)

Las tablas de estados contienen arrays de transiciones para cada estado.

El contenido de cada celda son transiciones representadas por el par {acción, próximo estado}

Usando punteros a función

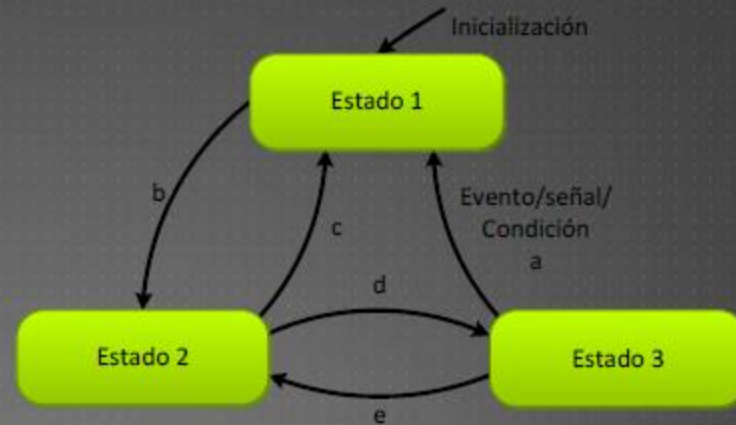
EJEMPLO:

```
int func_S1(void);  
int func_S2(void);  
int func_S3(void);
```

```
enum codigo_estados { S1, S2, S3};  
enum codigo_eventos {a, b, c, d, e, nada};
```

```
struct transicion {  
    int (*funcion_estado)(void);  
    enum codigo_estados proximo;  
};
```

```
struct transicion tabla_estado_evento [3][6]={  
    {{func_S1,S1},{func_S1,S2},{func_S1,S1},{func_S1,S1},{func_S1,S1},{func_S1,S1}},  
    {{func_S2,S2},{func_S2,S2},{func_S2,S1},{func_S2,S3},{func_S2,S2},{func_S2,S2}},  
    {{func_S3,S1},{func_S3,S3},{func_S3,S3},{func_S3,S3},{func_S3,S2},{func_S3,S3}}  
};
```





Usando punteros a función

EJEMPLO:

```
void dispatcher(enum codigo_estados, enum codigo_eventos);  
enum codigo_estados estado_actual=S1;  
enum codigo_eventos evento_actual=nada;
```

```
main() {  
    int (* accion)(void);  
    for (;;) {  
        dispatcher(estado_actual,evento_actual);  
    }  
    return 0;  
}
```

```
void dispatcher(enum codigo_estados, enum codigo_eventos){  
    int (*funcion)(void);  
    funcion=tabla_estado_evento[estado_actual][evento_actual].funcion_estado;  
    funcion();  
    estado_actual=tabla_estado_evento[estado_actual][evento_actual].proximo;  
}
```



Usando punteros a función

- Dispatcher es una función simple, realiza 3 pasos:
 - Identifica la transición que tendrá lugar realizando una búsqueda en la tabla de estados
 - Ejecuta la acción
 - Cambia el estado actual al próximo
- Esta implementación contiene una parte genérica y reutilizable y una parte específica de la aplicación.
- La parte específica de la aplicación requiere:
 - Usar enumeraciones para los estados y señales
 - Crear e inicializar una tabla de estados
 - Definir las funciones asociadas a las acciones



Características de la implementación

- Representación directa de la MEF
- Más modular, genera código más limpio (más fácil de leer)
- Provee una relativamente buena performance para el despacho de eventos
- Permite reutilizar la parte del procesador de eventos
- La tabla de estados puede ser muy grande y con desperdicio de celdas pero puede almacenarse en flash/ROM
- Puede requerir un gran número de pequeñas funciones asociadas a las acciones
- No es jerárquica
- Requiere una inicialización complicada

Ejemplo con leds revisitado...

```
+ /*=====ejercicio: mientras se mantiene apretado el pulsador sw1 los leds.

#include "board.h"

#define RETARDO      50000

enum estados {NADA, VERDE, ROJO};           // estos son los estados segun el diagrama
enum eventos {SUELTO, APRETADO, TIEMPO};    //estos son los eventos (por nivel)

- struct transicion {                       //contiene la funcion para el estado y el proximo estado
    void(*funcion_estado)(void);
    enum estados proximo_estado;
};
```



Ejemplo con leds revisitado...

```
void f_estado_VERDE(); // prototipos de las funciones de estados
void f_estado_ROJO();
void f_estado_NADA();

struct transicion tabla_estado_evento [3][3]={ //aqui se define la lógica
    {{f_estado_NADA,NADA},{f_estado_NADA,VERDE},{f_estado_NADA,NADA}},
    {{f_estado_VERDE,NADA},{f_estado_VERDE,VERDE},{f_estado_VERDE,ROJO}},
    {{f_estado_ROJO,NADA},{f_estado_ROJO,ROJO},{f_estado_ROJO,VERDE}}
};

enum estados estado_actual=NADA;
enum eventos evento_actual=SUELTO;
```

Ejemplo con leds revisitado...

```
int8_t fin_tiempo;
int32_t temporizando;

void dispatcher(enum estados, enum eventos);

int main() {
    board_init();
    fin_tiempo=0;
    temporizando=0;
    for (;;) {
        /* ===== tarea 1: implementa un mecanismo de retardo muy basico e impreciso, sin timers
         * ===== solo para su uso en la tarea 2 */
        if (temporizando>0){
            temporizando--;
            fin_tiempo=0;
        }
        else
            fin_tiempo=1;
        //nada mas

        // ===== tarea 2: tarea implmentada mediante una MEF con un tabla de transiciones y punteros
        dispatcher(estado_actual,evento_actual);
        //nada mas
    }
    return 0;
}
```

Ejemplo con leds revisitado...

```
void dispatcher(enum estados estado, enum eventos evento){
    void (*funcion) (void);
    funcion=tabla_estado_evento[estado][evento].funcion_estado;
    funcion();
    estado_actual=tabla_estado_evento[estado][evento].proximo_estado;

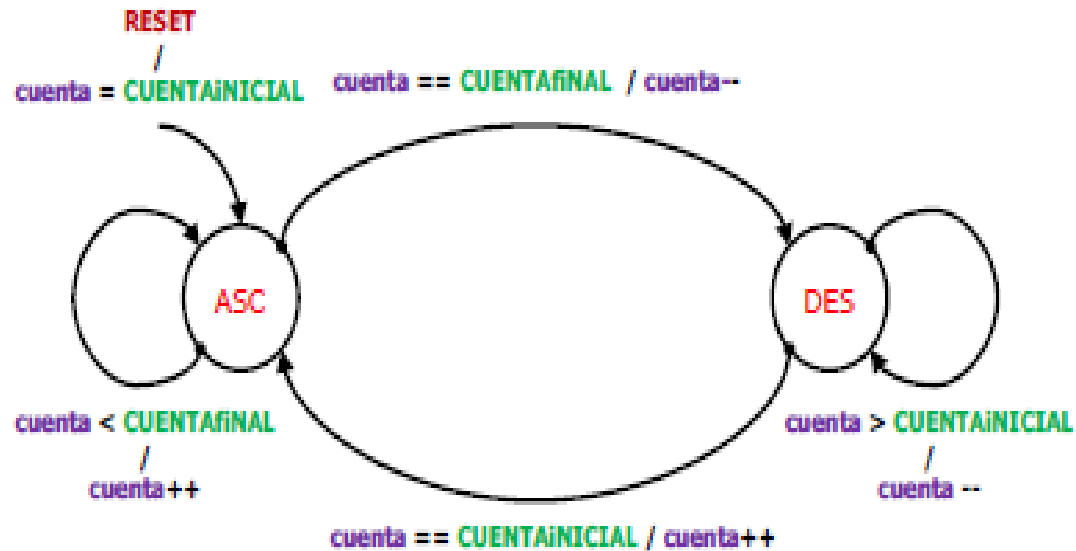
    /* No haria falta nada mas pues los eventos deberían analizarse en cada fucion de estado
     * segun los que interesan en cada estado.Sin embargo, como son poquitos, los ponemos acá.*/
    if (!pulsadorSw1_get()) //veo estado pulsador (con prioridad frente al tiempo)
        evento_actual=SUELTO;
    else if (fin_tiempo) { //veo si termino el tiempo
        evento_actual=TIEMPO;
        temporizando=RETARDO;
    }
    else
        evento_actual=APRETADO;
}
```


Ejemplo con leds revisitado...

```
void f_estado_VERDE() {  
    LED_ROJO_OFF;           //el rojo permanece apagado en este estado  
    LED_VERDE_ON;           //el verde permanece encendido en este estado  
}  
  
void f_estado_ROJO() {  
    LED_ROJO_ON;            //el rojo permanece encendido en este estado  
    LED_VERDE_OFF;          //el verde permanece apagado en este estado  
}  
  
void f_estado_NADA() {  
    LED_ROJO_OFF;           //todo apagado en este estado  
    LED_VERDE_OFF;  
}
```

Contador

- Ejemplo extraído de:
http://laboratorios.fi.uba.ar/lse/seminario/material-1erC2010/Tecnicas_Digitales_II-R4052-2010-Maquina_de_Estado.pdf



Contador

| Estado Actual | Excitación | Estado Futuro | Acción |
|---------------|--|---------------|------------------------|
| X | RESET estado >= ESTADOmAXIMO | ESTADOINICIAL | cuenta = CUENTAINICIAL |
| ASCENDENTE | estado < ESTADOmAXIMO && cuenta < CUENTAFINAL | ASCENDENTE | cuenta++ |
| | estado < ESTADOmAXIMO && cuenta == CUENTAFINAL | DESCENDENTE | cuenta-- |
| DESCENDENTE | estado < ESTADOmAXIMO && cuenta > CUENTAINICIAL | DESCENDENTE | cuenta-- |
| | estado < ESTADOmAXIMO && cuenta == CUENTAINICIAL | ASCENDENTE | cuenta++ |



Contador

```
/* Defines del correspondientes al Contador */
#define ASCENDENTE 0 // Estados
#define DESCENDENTE 1
#define ESTADOmAX DESCENDENTE + 1
#define ESTADOiNICIAL ASCENDENTE
#define CUENTAiNICIAL 0x00 // Límites de Cuenta
#define CUENTAfINAL 0xFF

/* Reserva de Variables */
unsigned char estado=ESTADOiNICIAL; // Estado de Contador de 8 bits Asc. / Desc.
unsigned char cuenta= CUENTAiNICIAL; // Contador de 8 bits Asc. / Desc.

void InicializarContador (void)
{ // Inicializa las Variables y Salidas de Control
    estado = ESTADOiNICIAL;
    cuenta = CUENTAiNICIAL;
    return;
}
```



Contador

```
= void Ascendiendo(void)
{
    if (cuenta < CUENTAfinal)
        cuenta++;
    else {
        estado = DESCENDENTE;
        cuenta--;
    }
    return;
}

= void Descendiendo (void)
{
    if (cuenta > CUENTAinICIAL)
        cuenta--;
    else {
        estado = ASCENDENTE;
        cuenta++;
    }
    return;}
.
```



Contador

```
typedef void (*Action)(void);
Action ArrayFuncionesEstadoContador [] = { Ascendiendo, Descendiendo};

void Contador (void)//se ejecuta para siempre
{ // Si el Estado esta fuera de rango reinicializa la M de E y retorna
  if (estado >= ESTADOMAX) {
    InicializarContador();
  }
  return;
  ArrayFuncionesEstadoContador [estado]();
}
```

El ejemplo interesante revisitado...

```
#include <stdio.h>
/* numero maximo de bytes do buffer de dados */
#define MAX_BUFFER 512
/* constantes usadas na comunicacao */
#define STX      0x02
#define ETX      0x03
/* possiveis estados da maquina de estados de comunicacao */
typedef enum {
    ST_STX = 0, ST_QTD, ST_DATA, ST_CHK, ST_ETX
} States;
typedef void (*Action)(unsigned char data);

struct StateMachine {
    States state;
    unsigned char buffer[MAX_BUFFER];
    unsigned char chkBuffer;
    int indBuffer;
    int qtdBuffer;
    Action action[5];
} sm;
```

El ejemplo interesante revisitado...

```
void handlePackage(unsigned char *data, int qtd) { /* trata dados recebidos */
    int i;
    printf("Imprimindo dados recebidos...\n");
    for (i = 0; i < qtd; i++)
        printf("Data[%d]=%d\n", i, data[i]);
}

void stSTX(unsigned char data){
    if (data == STX) {
        sm.indBuffer = sm.qtdBuffer = 0;
        sm.chkBuffer = 0;
        sm.state = ST_QTD;    }}

void stQtd(unsigned char data){
    sm.qtdBuffer = data;
    sm.state = ST_DATA;}

void stData(unsigned char data){
    sm.buffer[sm.indBuffer++] = data;
    sm.chkBuffer ^= data;
    if (--sm.qtdBuffer == 0) {
        sm.state = ST_CHK;
    }}
}
```


El ejemplo interesante revisitado...

```
void stChk(unsigned char data){
    if (data == sm.chkBuffer) {
        sm.state = ST_ETX;
    }
    else {
        sm.state = ST_STX;
    }
}

void stETX(unsigned char data){
    if (data == ETX) {
        handlePackage(sm.buffer, sm.indBuffer);
    }
    sm.state = ST_STX;
}

void handleRx(unsigned char *data, int qtd) {
    int i;

    for (i = 0; i < qtd; i++) {
        sm.action[sm.state](data[i]);
    }
}
```

El ejemplo interesante revisitado...

```
void initSM(){
    sm.state = ST_STX;
    sm.buffer[0] = 0;
    sm.chkBuffer = 0;
    sm.indBuffer = 0;
    sm.qtdBuffer = 0;
    sm.action[ST_STX] = stSTX;
    sm.action[ST_QTD] = stQtd;
    sm.action[ST_DATA] = stData;
    sm.action[ST_CHK] = stChk;
    sm.action[ST_ETX] = stETX;
}

int main() { /* main para simular o uso da maquina de estados */
    unsigned char data1[] = { STX, 5, 11, 22, 33, 44 };
    unsigned char data2[] = { 55, 39, ETX };
    initSM();
    handleRx(data1, sizeof(data1));
    handleRx(data2, sizeof(data2));
    return 0; }
```



Otros ejemplos que pueden consultar

- <http://amsekharkernel.blogspot.com.ar/2012/06/simple-finite-state-machine-for.html> (simple, prende y apaga una luz)
- <http://www.embarcados.com.br/maquina-de-estado/> (identifica que se ingresó por teclado la secuencia "abcd")
- <http://blog.ankurs.com/2010/04/simple-finite-state-machines-in-c/> (usa lista enlazadas)
- <http://www.gedan.net/2009/03/18/finite-state-machine-matrix-style-c-implementation-function-pointers-addon/> (genérico)
- No hay una única forma de implementar esta opción!