

Programación de SE en C

(Parte II: Máquinas de estados finitos)

Máquinas de estados finitos (FSM)

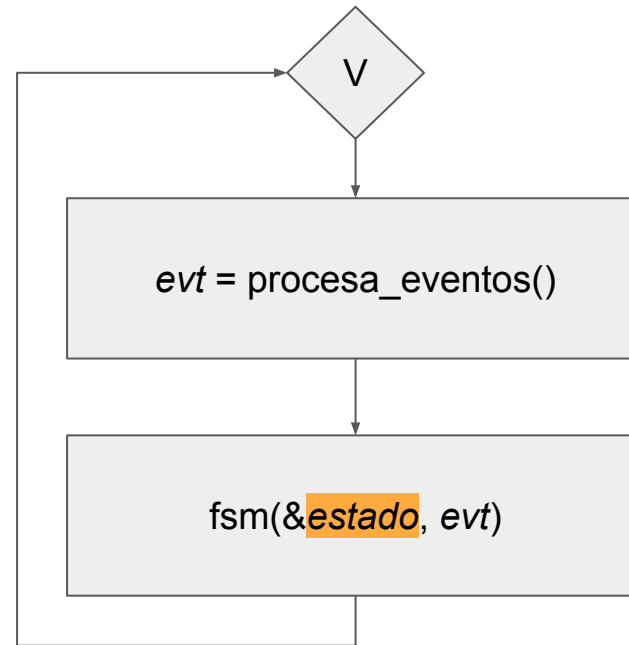
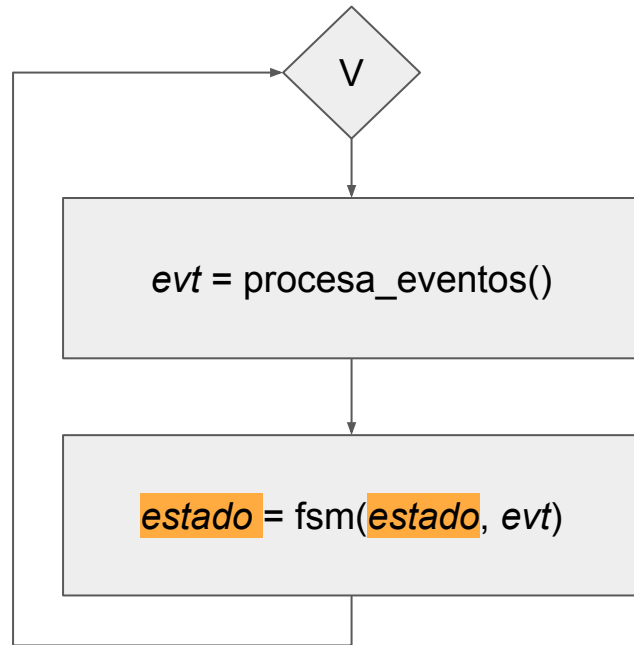
- Permiten modelar un problema particular, **con mayor nivel de abstracción**
- Muy útiles para representar sistemas **reactivos a eventos** o con **muchos modos de funcionamiento**
- Pueden **simularse** y evaluarse antes de implementarse en C
- Existen distintas maneras de implementar las FSM en código:
 - * switch/if anidados
 - * funciones de estados
 - * Tablas de transiciones

Máquinas de estados finitos (FSM)

- Son compatibles con las distintas arquitecturas de software:
 - * super-loop - foreground/background
 - * **event-driven**
 - * time triggered
 - * RTOS
- **Se debe procesar POR COMPLETO de a un evento a la vez.** Por lo que se debe tener especial cuidado con las tareas concurrentes e interrupciones.
- Suelen utilizarse colas/buffers para el procesamiento de eventos

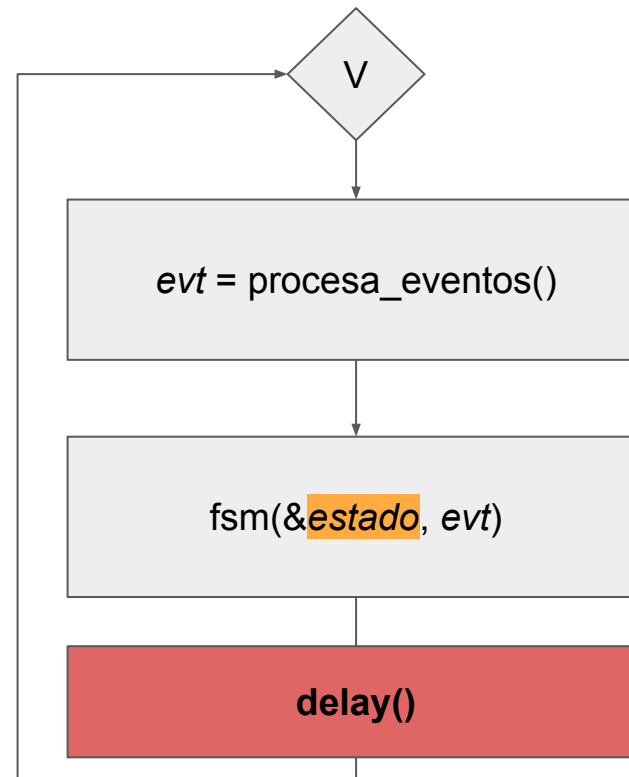
Máquinas de estados finitos (FSM)

- super-loop



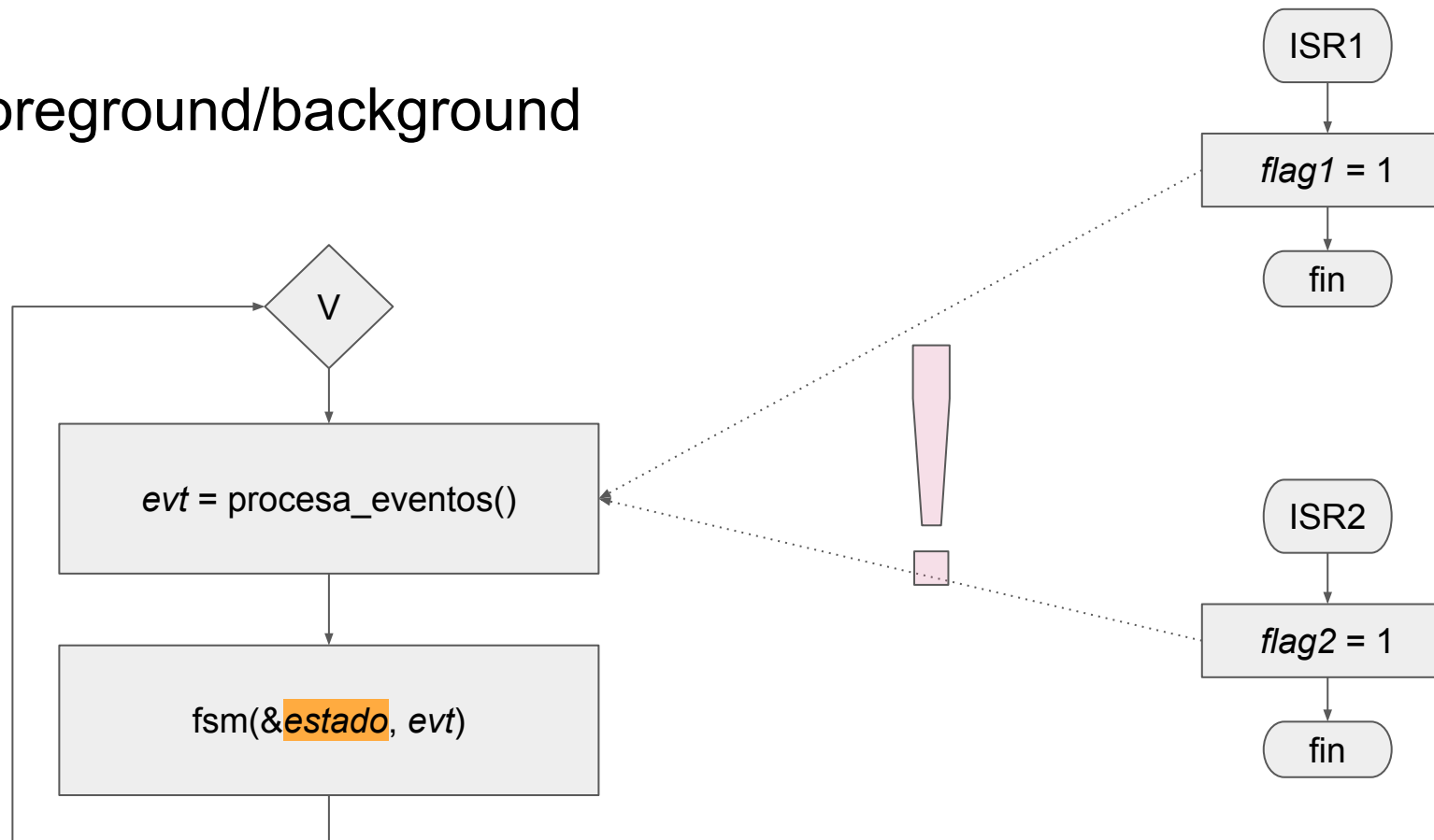
Máquinas de estados finitos (FSM)

- super-loop



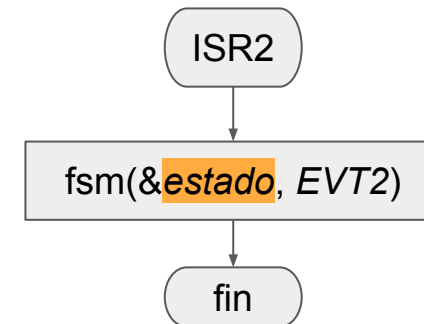
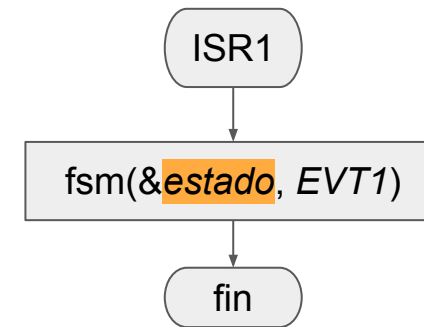
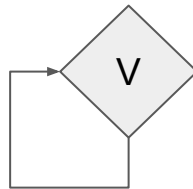
Máquinas de estados finitos (FSM)

- foreground/background



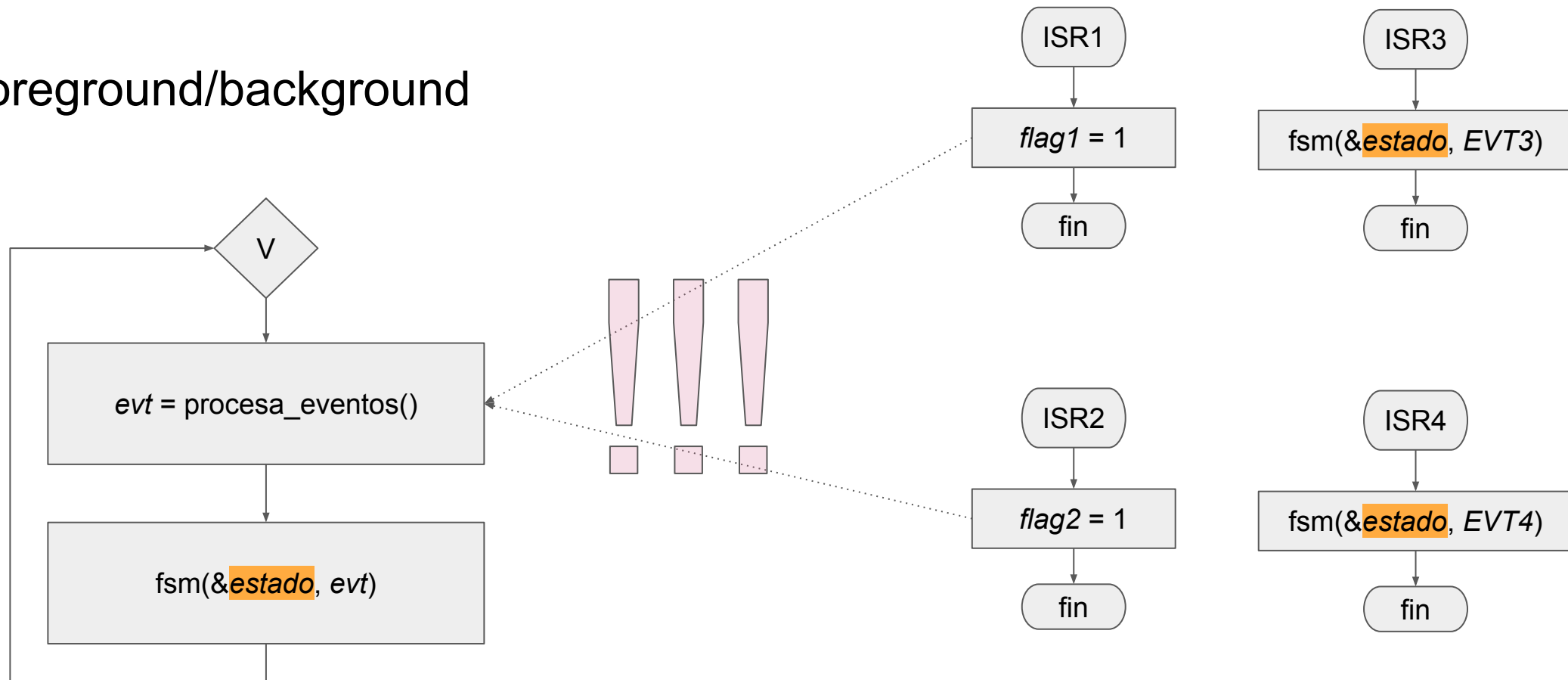
Máquinas de estados finitos (FSM)

- foreground/background



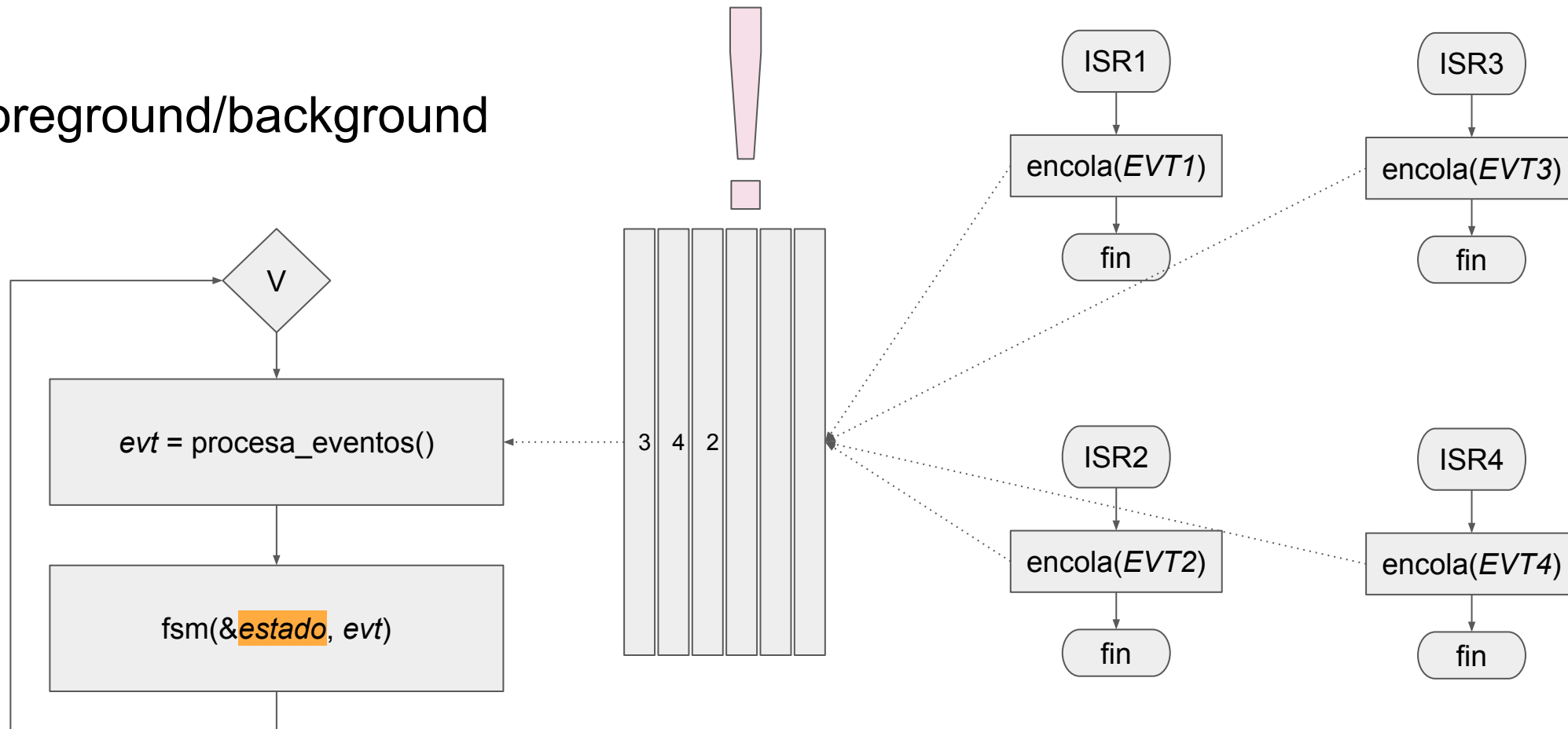
Máquinas de estados finitos (FSM)

- foreground/background



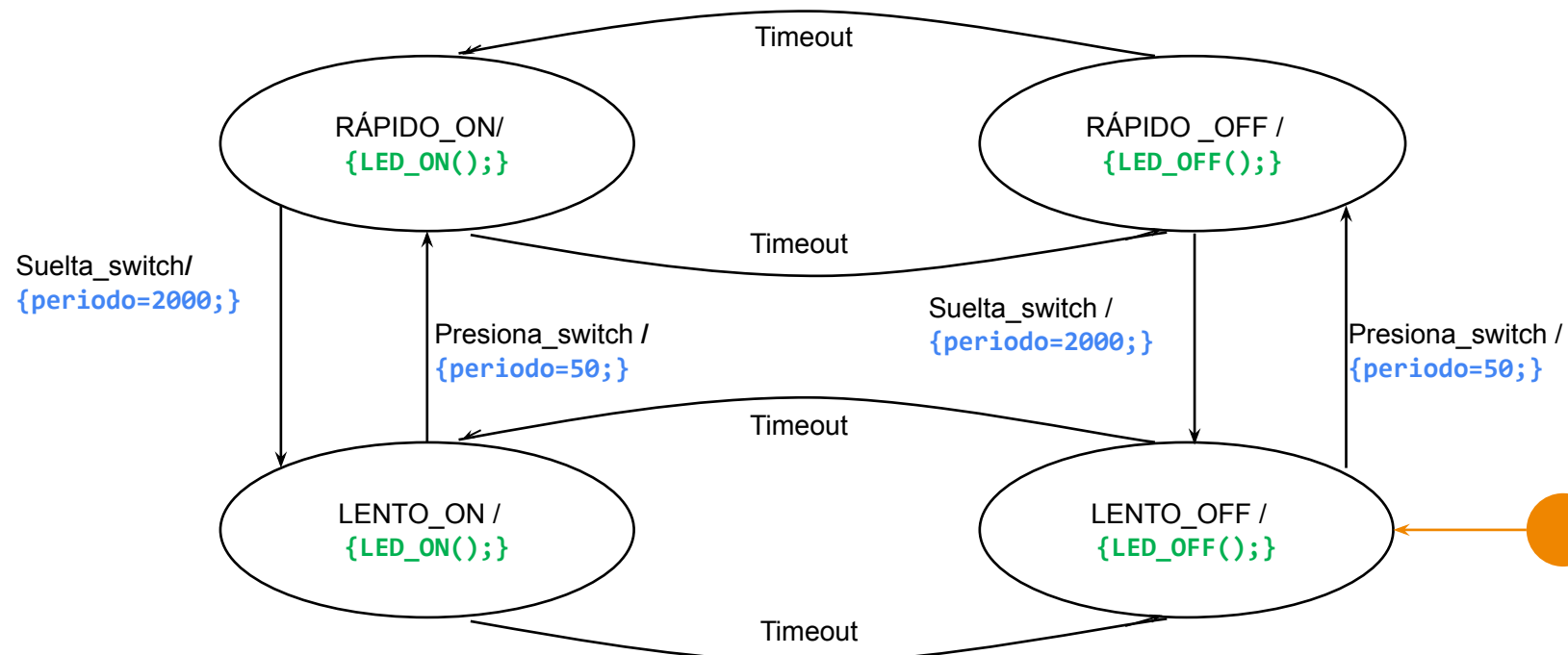
Máquinas de estados finitos (FSM)

- foreground/background



Máquinas de estados finitos (FSM)

- Las FSM siempre deben estar en un estado definido, **desde el inicio**
- Los estados suelen ser puntos de reposo donde no se modifican las salidas (no se ejecutan acciones)
- Las acciones (salidas) pueden estar asociadas a las **transiciones** (Mealy), al **ingreso al próximo estado** (Moore)



Máquinas de estados finitos (FSM)

Implementación con switch anidados

- Conviene crear un **tipo enum** para los estados y otro para los eventos*

```
typedef enum {RAPIDO_ON, RAPIDO_OFF, LENTO_ON, LENTO_OFF}estados_e;  
typedef enum {PRESIONA_SW, SUELTA_SW, TIMEOUT}eventos_e;*
```

- Y una **variable** que indicará el **estado actual** de la FSM (inicializada según el estado inicial)

```
estados_e estado_actual = LENTO_OFF;
```

- El código de la FSM puede modularizarse en una función que recibe el estado actual como parámetro por referencia y el evento que se desea procesar*

```
void fsm_switch(estados_e *estado, eventos_e evento);
```

(también puede usar el valor de retorno para devolver el nuevo estado)

```
estados_e fsm_switch(estados_e estado, eventos_e evento);
```

- Cada vez que se invoca, se evalúa **solo para el estado actual** el evento* pasado como parámetro y solo si corresponde realiza una transición hacia otro estado

Máquinas de estados finitos (FSM)

Implementación

- Conviene crear

*** hay máquinas de estados que hacen polling de entradas y no están manejadas por eventos. Otras sólo tienen un único evento.**

```
typedef enum {RAPIDO_ON, RAPIDO_OFF, LENTO_ON, LENTO_OFF}estados_e;  
typedef enum {PRESIONA_SW, SUELTA_SW, TIMEOUT}eventos_e;*
```

- Y una **variable** que indicará el **estado actual** de la FSM (inicializada según el estado inicial)

```
estados_e estado_actual = LENTO_OFF;
```

- El código de la FSM puede modularizarse en una función que recibe el estado actual como parámetro por referencia y el evento que se desea procesar*

```
void fsm_switch(estados_e *estado, eventos_e evento);
```

(también puede usar el valor de retorno para devolver el nuevo estado)

```
estados_e fsm_switch(estados_e estado, eventos_e evento);
```

- Cada vez que se invoca, se evalúa **solo para el estado actual** el evento* pasado como parámetro y solo si corresponde realiza una transición hacia otro estado

```

void fsm_switch(estados_e *actual, eventos_e evento){
    estados_e estado_anterior = *actual;
    //DETECTO TRANSICIONES Y EJECUTO ACCIONES EN LA TRANSICIÓN
    //TIPO MÁQUINA DE MEALY
    switch (*actual) {
        case LENTO_OFF:
            switch(evento){
                case PRESIONA_SW:
                    htim3.Instance->ARR = 50;
                    htim3.Instance->EGR |= 1;
                    *actual = RAPIDO_OFF;
                    break;
                case TIMEOUT:
                    *actual = LENTO_ON;
                    break;
            }
            break;
        case LENTO_ON:
            switch(evento){
                case PRESIONA_SW:
                    //...
                case TIMEOUT:
                    //...
            }
            break;
        case RAPIDO_OFF:
            //...
        case RAPIDO_ON:
            //...
    }
}

```

```

//SI HUBO UN CAMBIO DE ESTADO EJECUTO ACCIONES ASOCIADAS A LA ENTRADA AL ESTADO
//(TIPO MOORE)
if (*actual != estado_anterior) {
    switch (*actual) {
        case LENTO_OFF:
        case RAPIDO_OFF:
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
            break;
        case LENTO_ON:
        case RAPIDO_ON:
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
            break;
    }
}
}

```

```
void fsm_switch(estados_e *actual, eventos_e evento){
    estados_e estado_anterior = *actual;
    //DETECTO TRANSICIONES Y EJECUTO ACCIONES EN LA TRANSICIÓN
    //TIPO MÁQUINA DE MEALY
    switch (*actual) {
        case LENTO_OFF:
            switch(evento){
                case PRESIONA_SW:
                    htim3.Instance->ARR = 50;
                    htim3.Instance->EGR |= 1;
                    *actual = RAPIDO_OFF;
                    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
                    break;
                case TIMEOUT:
                    *actual = LENTO_ON;
                    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
                    break;
            }
            break;
        case LENTO_ON:
            switch(evento){
                case PRESIONA_SW:
                    //...
                case TIMEOUT:
                    //...
            }
            break;
        case RAPIDO_OFF:
            //...
        case RAPIDO_ON:
            //...
    }
}
```

Máquinas de estados finitos (FSM)

Implementación con switch anidados

- Puede invocarse desde el proceso de background

```

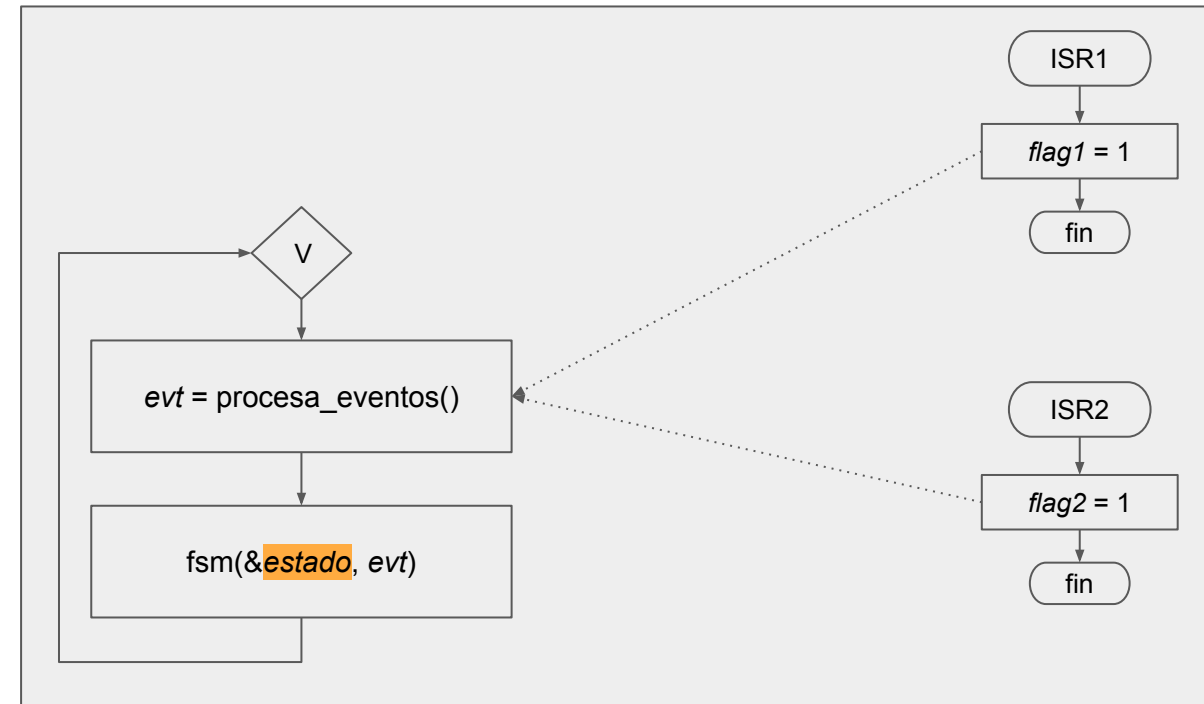
estados_e estado_actual = LENTO_OFF;
int main(void) {
    /* ... */
    while (1) {
        eventos_e evt;
        if(timeout>0){
            timeout=0;
            evt = TIMEOUT;
        }
        else
            evt = (HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin)==GPIO_PIN_SET)?PRESIONA_SW:SUELTA_SW;
        fsm_switch(&estado_actual, evt);
    }
}

```

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    timeout=1;
}

```



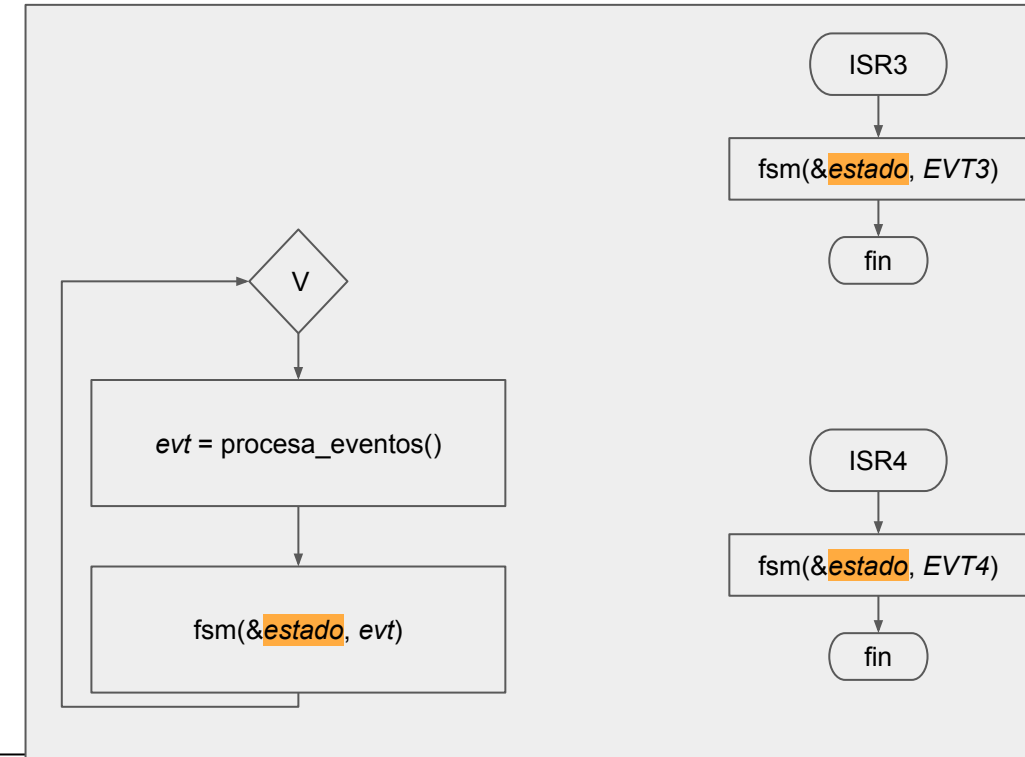
Máquinas de estados finitos (FSM)

Implementación con switch anidados

- O desde las ISR (**Con cuidado de no interrumpir transiciones u acciones en curso!!!**)

```
estados_e estado_actual = LENTO_OFF;
int main(void) {
    /* ... */
    while (1) {
        /* MECANISMO DE SINCRONIZACIÓN */
        if(HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin)==GPIO_PIN_SET)
            fsm_switch(&estado_actual, PRESIONA_SW);
        else
            fsm_switch(&estado_actual, SUELTA_SW);
    }
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    fsm_switch(&estado_actual, TIMEOUT);
}
```



Implementación de FSM con funciones de estados (usando *punteros a función*)

Máquinas de estados finitos (FSM) –

Implementación con funciones de estados (usando **punteros a función**)

- La FSM puede implementarse “rompiendo” el switch externo de la solución anterior en una función por estado del tipo:

```
void estado(estado_t *actual, eventos_e evento)
```

```
    void lento_off(estado_t *actual, eventos_e evento);  
    void lento_on(estado_t *actual, eventos_e evento);  
    void rapido_off(estado_t *actual, eventos_e evento);  
    void rapido_on(estado_t *actual, eventos_e evento);
```

```
o ... estado_t estado(estado_t *actual, eventos_e evento)
```

```
    estado_t lento_off(eventos_e evento);  
    estado_t lento_on(eventos_e evento);  
    estado_t rapido_off(eventos_e evento);  
    estado_t rapido_on(eventos_e evento);
```

Máquinas de estados finitos (FSM) – Implementación con funciones de estados (usando punteros a función)

- **estado_t** es el tipo de dato del puntero a las funciones estado, definido como:

```
typedef void(*estado_t)(void*, eventos_e);
```

O...

```
typedef estado_t(*estado_t)(eventos_e);
```

void* □ en la definición del tipo uso void*
como comodín de puntero

- Se define una variable que indicará el estado actual de la FSM (inicializada según el estado inicial)

```
estado_t fsm = lento_off;
```

Máquinas de estados finitos (FSM) –

Implementación con funciones de estados (usando **punteros a función**)

- La “variable” fsm a la vez que almacena el estado actual, es un puntero a la función que debe ejecutarse de la máquina de estados cuando sucede un evento:

`fsm(&fsm, evt);`

`fsm = fsm(evt);`

- La ventaja sobre la implementación con switch es una *mayor velocidad de ejecución y menor complejidad de implementación* para **máquinas de estados grandes** (con muchos estados)

Máquinas de estados finitos (FSM) – Implementación con funciones de estados (usando punteros a función)

```
void lento_off(estado_t *actual, eventos_e evento)
{
    switch(evento){
        case PRESIONA_SW:
            htim3.Instance->ARR = 50;
            htim3.Instance->EGR |= 1;
            *actual = rapido_off;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
            break;
        case TIMEOUT:
            *actual = lento_on;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
            break;
    }
}
```

```
estado_t lento_off(eventos_e evento)
{
    estado_t actual = lento_off;
    switch(evento){
        case PRESIONA_SW:
            htim3.Instance->ARR = 50;
            htim3.Instance->EGR |= 1;
            actual = rapido_off;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
            break;
        case TIMEOUT:
            actual = lento_on;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
            break;
    }
    return actual;
}
```

Implementación de FSM con tabla de transiciones (usando *punteros a función*)

Máquinas de estados finitos (FSM) – Implementación con **tabla de transiciones** (punteros a función)

- Vimos que para un gran número de estados es más eficiente utilizar una función por estado y punteros a función.
- Pero si el número de eventos/entradas a evaluar también es grande se pierde eficiencia en los switch o if/else que verifican la existencia de eventos
- Se logra una implementación más eficiente con una función por cada combinación de estado-evento posible y construir una tabla de doble entrada (una matriz 2D) que a partir del estado actual y del evento ocurrido nos de acceso a un manejador de evento que puede o no terminar en una transición.
- Es decir, hay que construir una matriz 2D de punteros a función, donde la fila indica el estado y la columna el evento

Máquinas de estados finitos (FSM) – Implementación con **tabla de transiciones** (punteros a función)

```
typedef void* (*estado_t)();
```

```
estado_t* rapido_on_ssw();  
estado_t* rapido_on_to();  
estado_t* rapido_off_ssw();  
estado_t* rapido_off_to();  
estado_t* lento_on_psw();  
estado_t* lento_on_to();  
estado_t* lento_off_psw();  
estado_t* lento_off_to();  
estado_t* null(){};
```

```
typedef enum {RAPIDO_ON, RAPIDO_OFF, LENTO_ON, LENTO_OFF}estados_e;
```

```
typedef enum{PRESIONA_SW, SUELTA_SW, TIMEOUT}eventos_e;
```

```
#define NUM_EVENTOS 3
```

```
#define NUM_ESTADOS 4
```

```
estado_t const tabla[NUM_ESTADOS][NUM_EVENTOS] = {  
    {null,          rapido_on_ssw,  rapido_on_to},    /* RAPIDO_ON */  
    {null,          rapido_off_ssw, rapido_off_to},    /* RAPIDO_OFF */  
    {lento_on_psw,  null,           lento_on_to},    /* LENTO_ON */  
    {lento_off_psw, null,           lento_off_to}};    /* LENTO_OFF */  
/* PRESIONA_SW    SUELTA_SW        TIMEOUT */
```

```
estado_t* rapido_on_ssw(){  
    htim3.Instance->ARR = 2000;  
    htim3.Instance->EGR |= 1;  
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);  
    return (estado_t*)tabla[LENTO_ON];  
}
```

```
estado_t* rapido_on_to(){  
    HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);  
    return (estado_t*)tabla[RAPIDO_OFF];  
}
```

```
estado_t *fsm = (estado_t)tabla[LENTO_OFF];
```


Máquinas de estados finitos (FSM) – Implementación con **tabla de transiciones** (punteros a función)

```
while (1) {  
    eventos_e evt;  
    if(timeout>0){  
        timeout=0;  
        evt = TIMEOUT;  
    }  
    else{  
        evt=(HAL_GPIO_ReadPin(SWITCH_GPIO_Port,SWITCH_Pin)==GPIO_PIN_SET)?PRESIONA_SW:SUELTA_SW;  
    }  
    fsm = fsm[evt]();  
}
```

- El estado actual queda determinado por la fila a la que apunta fsm.
- Al ocurrir un evento se ejecuta la función apuntada por fsm[evt] y se retorna el nuevo estado como un puntero a otra fila

tabla		evt	
fsm →			
		fsm[evt]	

Statecharts: FSM con extensiones

Extensiones de las FSM

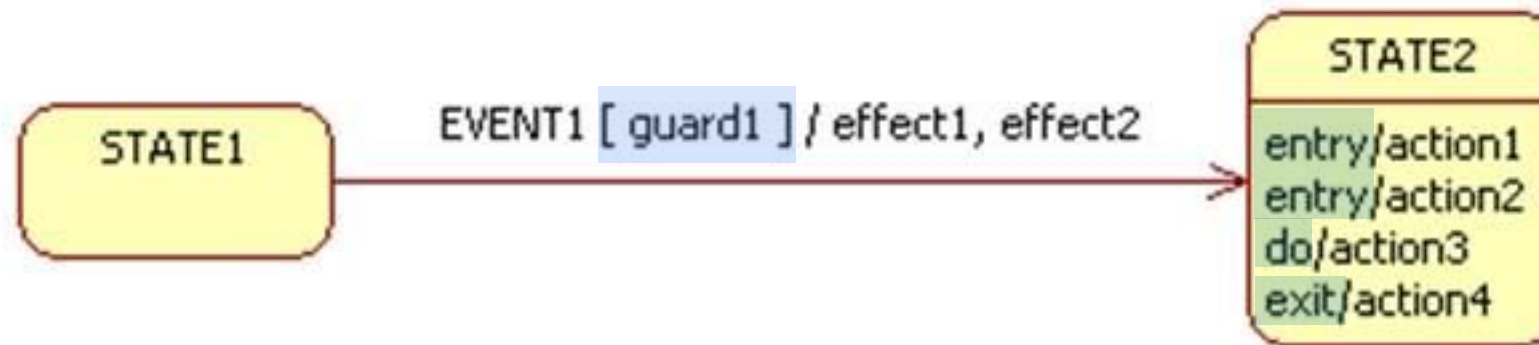
- Guardas (condiciones lógicas)
- Variables
- Estados jerárquicos (anidados)
- Estados concurrentes
- Pseudoestados
- Eventos internos

UML Statecharts

D. Harel, "Statecharts: A visual formalism for complex systems", Science of Computer Programming, vol. 8 num. 3, pp. 231–274, junio de 1987

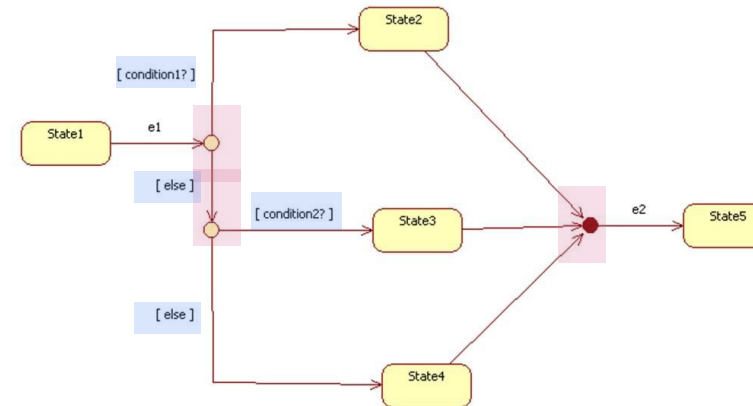
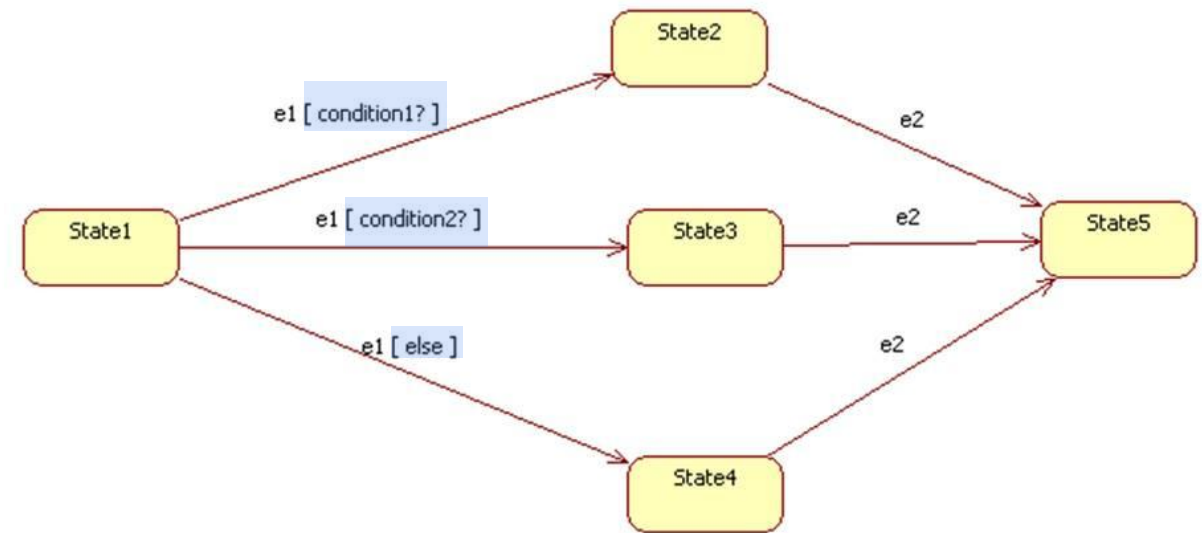
Extensiones de las FSM

- Guardas (condiciones lógicas)
- Variables
- Estados jerárquicos (anidados)
- Estados concurrentes
- Pseudoestados
- Eventos internos



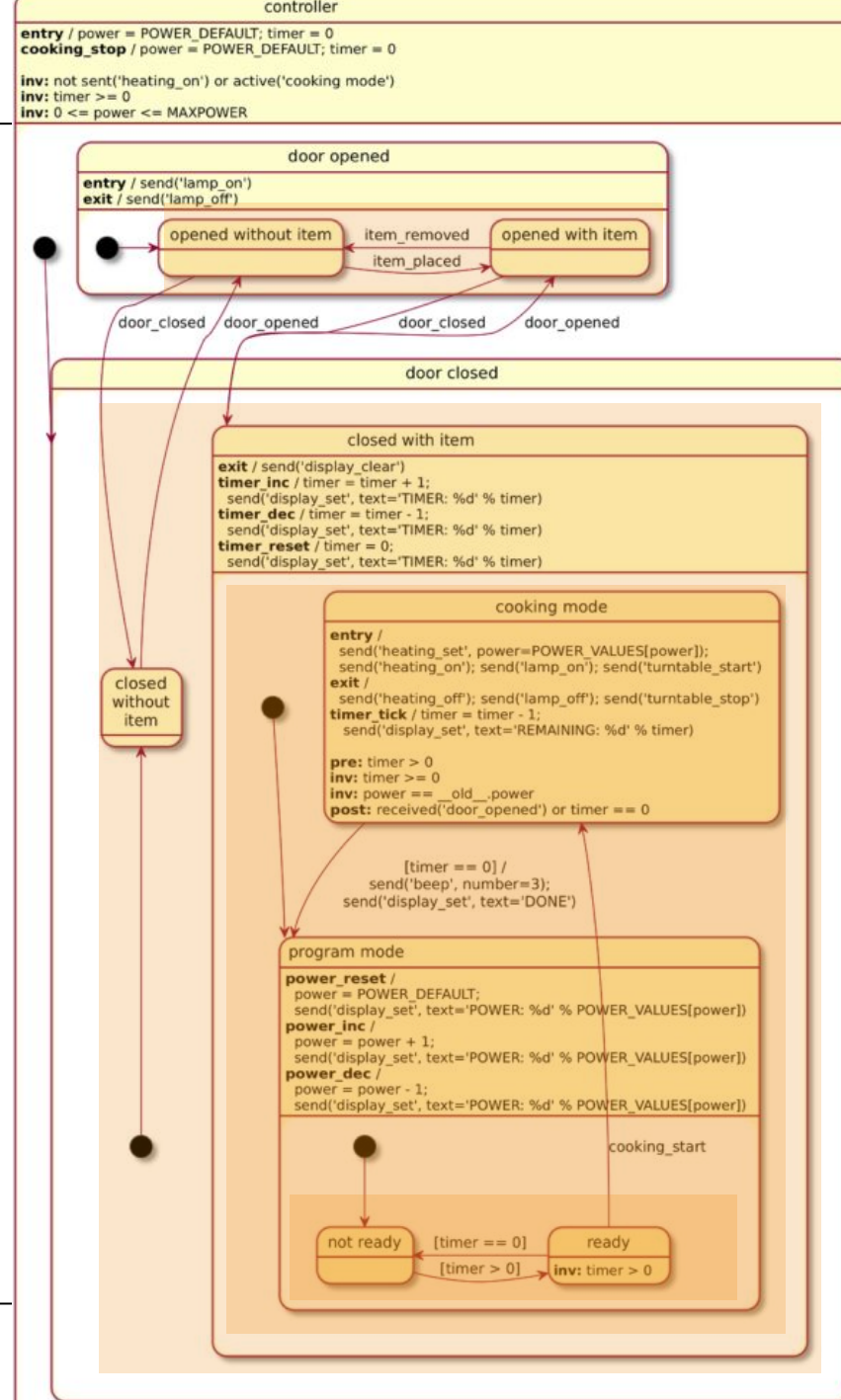
Extensiones de las FSM

- Guardas (condiciones lógicas)
- Variables
- Estados jerárquicos (anidados)
- Estados concurrentes
- Pseudoestados
- Eventos internos

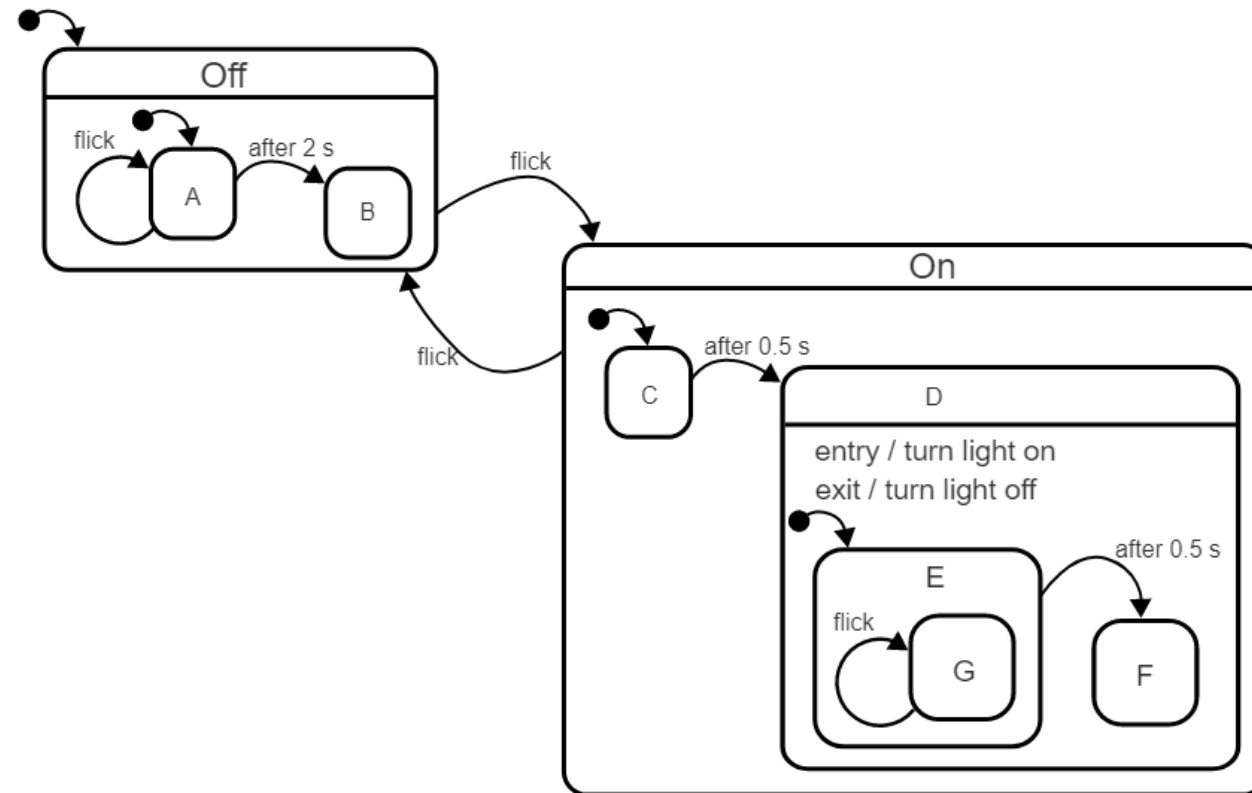


Extensiones de las FSM

- Guardas (condiciones lógicas)
- Variables
- Estados jerárquicos (anidados)
- Estados concurrentes
- Pseudoestados
- Eventos internos

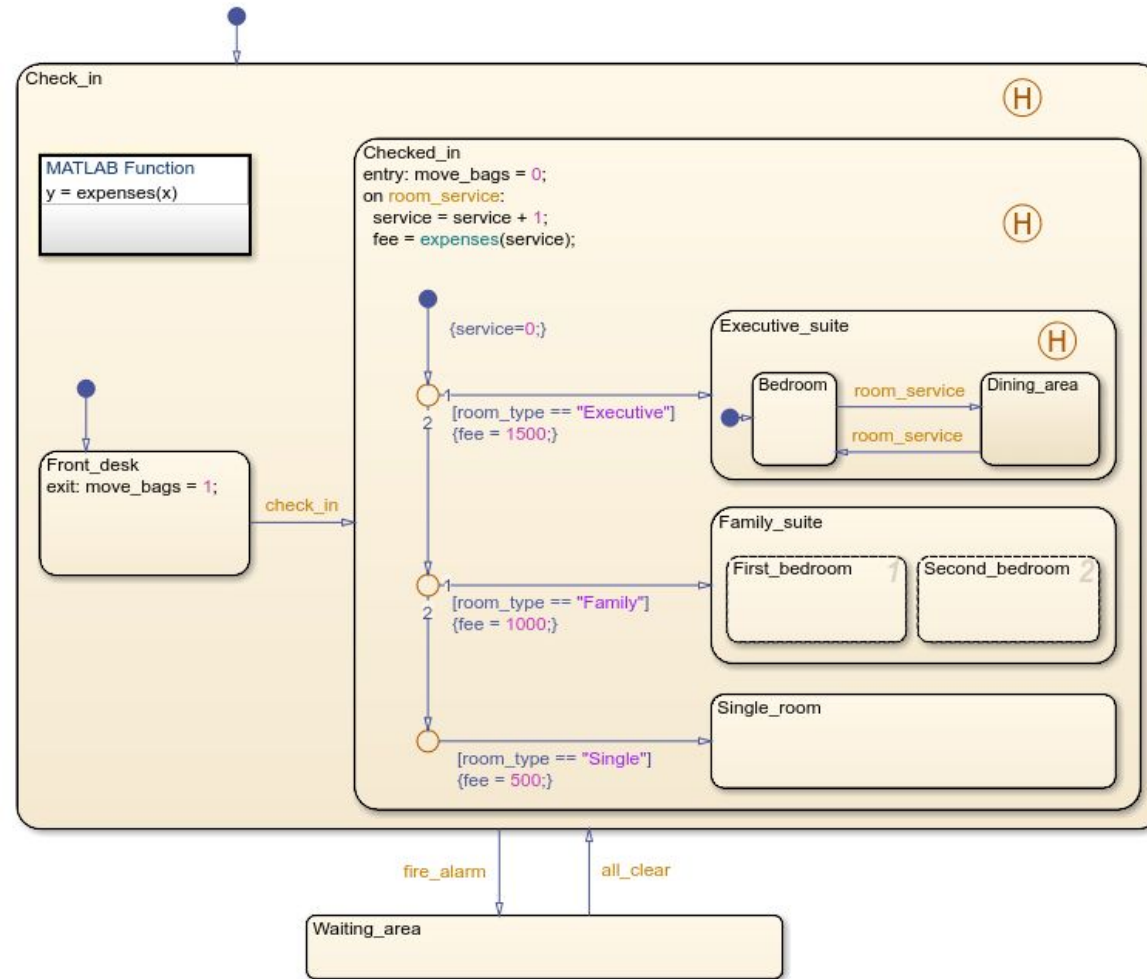


Extensiones de las FSM



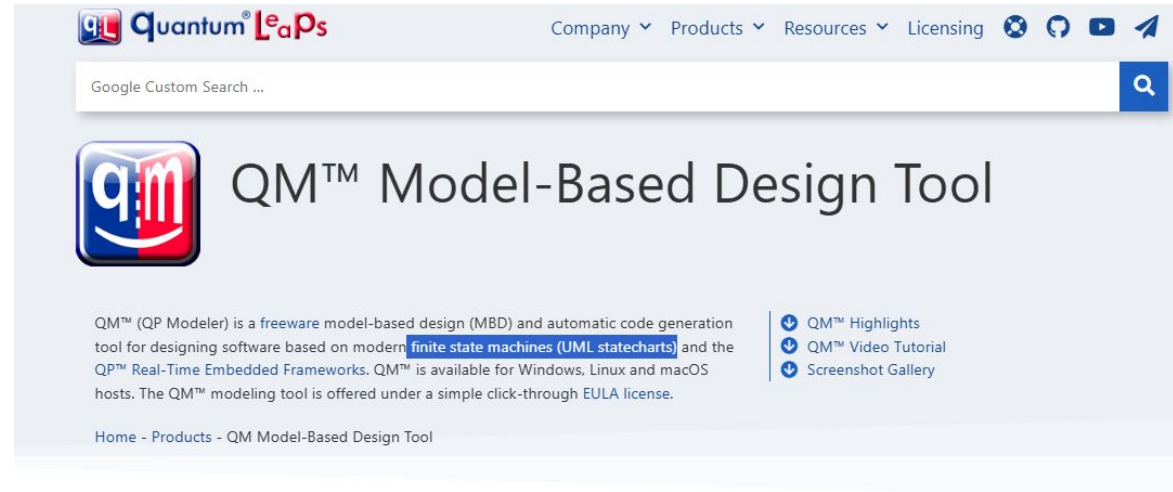
Extensiones de las FSM

Stateflow de Simulink



Extensiones de las FSM

QM Modeler de Quantum Leaps



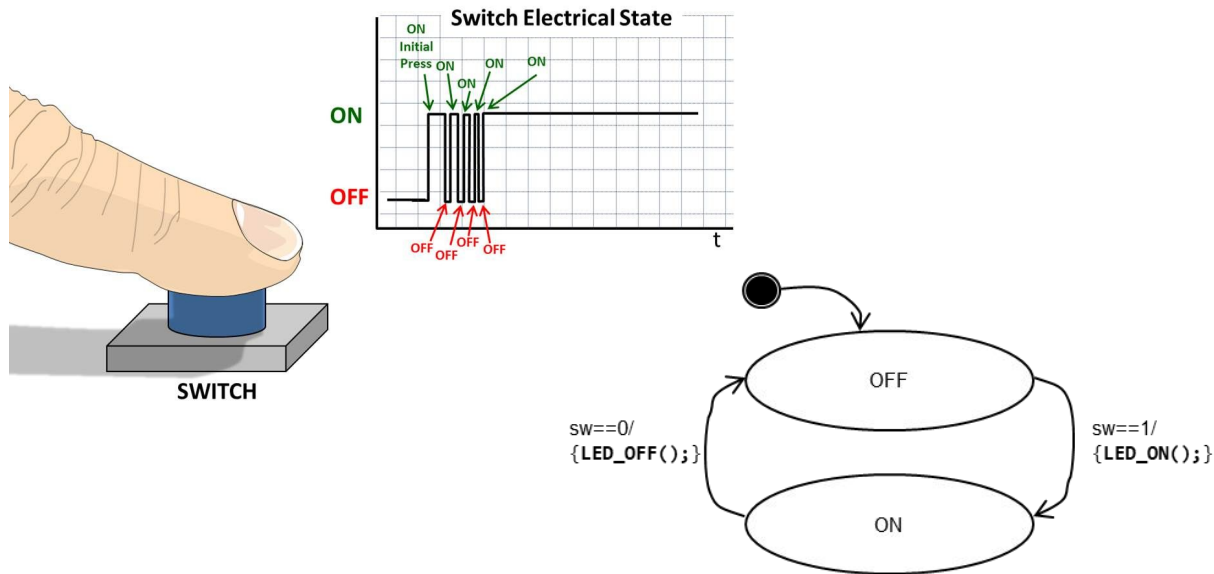
The screenshot shows the Quantum Leaps website. At the top is the Quantum Leaps logo and navigation links: Company, Products, Resources, Licensing. Below the navigation is a Google Custom Search bar. The main heading is "QM™ Model-Based Design Tool" with a logo. Below the heading is a paragraph describing the tool: "QM™ (QP Modeler) is a freeware model-based design (MBD) and automatic code generation tool for designing software based on modern **finite state machines (UML statecharts)** and the QP™ Real-Time Embedded Frameworks. QM™ is available for Windows, Linux and macOS hosts. The QM™ modeling tool is offered under a simple click-through EULA license." To the right of the paragraph are three links: "QM™ Highlights", "QM™ Video Tutorial", and "Screenshot Gallery". At the bottom of the page is a breadcrumb trail: "Home - Products - QM Model-Based Design Tool".



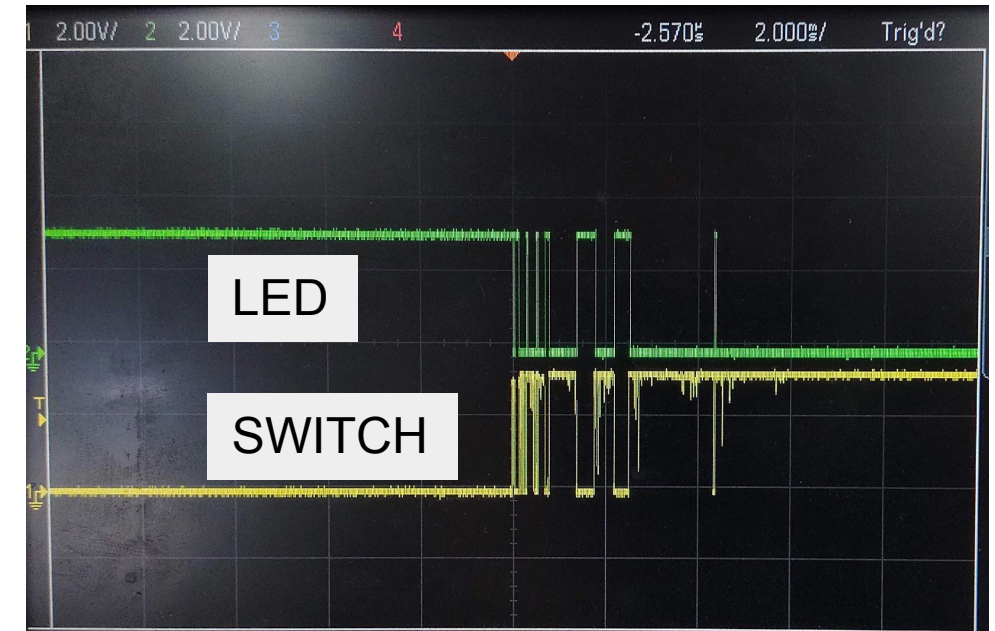
The image shows three computer monitors displaying the QM Model-Based Design Tool interface. In the foreground, there is a large, 3D-style QM logo. Below the logo are three circular icons representing the operating systems: Windows, Linux (Tux), and macOS (Apple logo). At the bottom of the image, there is a blue thumbs-up icon and a text box that reads: "The QM™ Model-Based Design tool runs on Windows, Linux and macOS. However, we recommend **Windows**, because we use it as the main platform for development and testing of our host-based tools."

Máquinas de estados finitos (FSM) – Ejemplo anti-rebote

- Cada vez que se presiona o se libera un pulsador ocurren rebotes mecánicos, que normalmente se dan en los primeros milisegundos

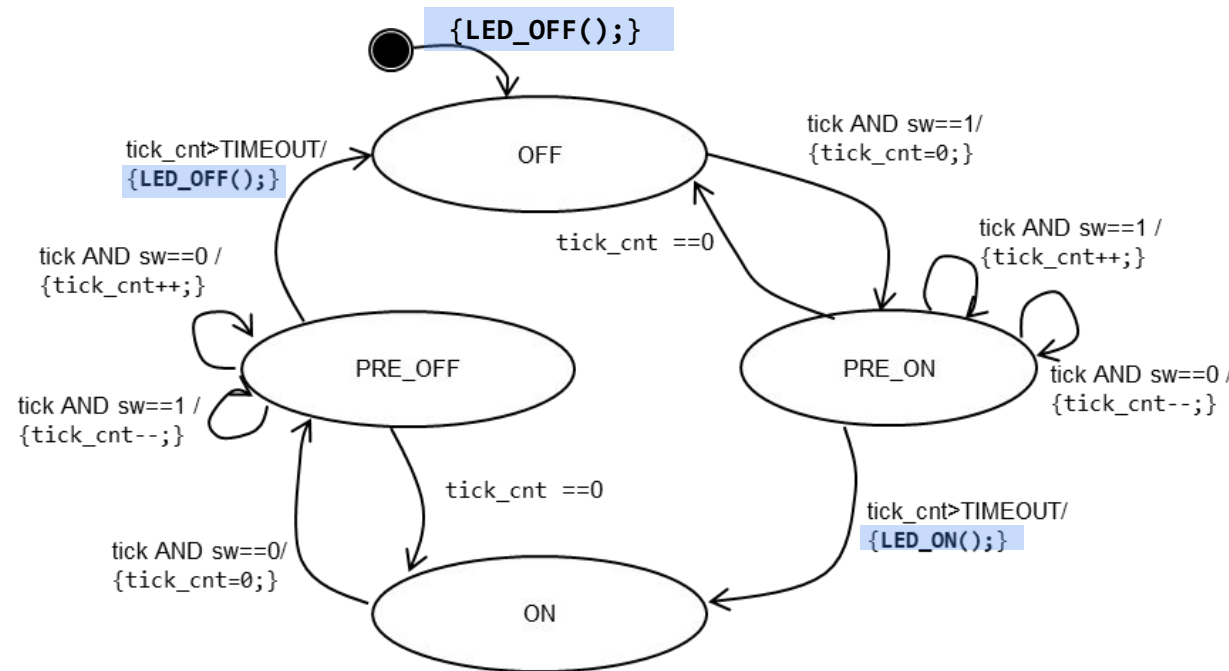


```
while (1) {
    if(HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin)==GPIO_PIN_RESET)
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
    else
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
}
```



Máquinas de estados finitos (FSM) – Ejemplo anti-rebote

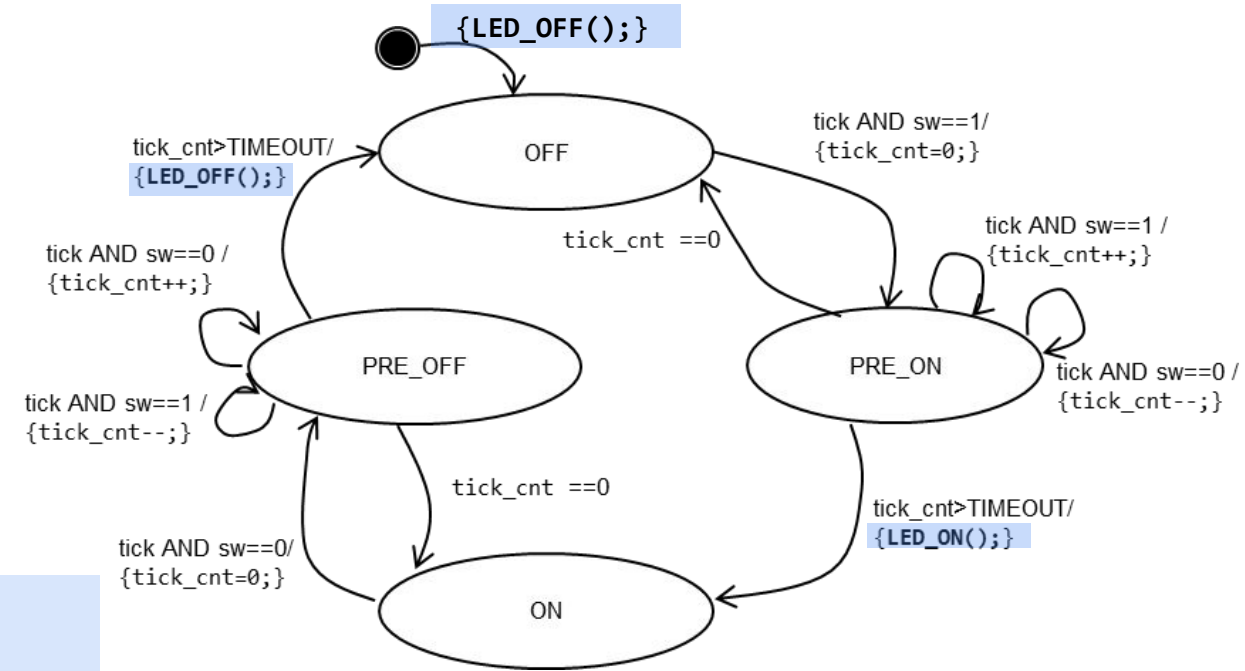
- Una posible solución es implementar un procesamiento de la entrada “ruidosa” basado en una FSM, que modela el switch con 4 estados:
 - * dos estados estables: ON y OFF
 - * dos estados transitorios: PRE_ON y PRE_OFF, en los cuales estará el switch mientras haya rebotes
- El criterio para realizar la transición a los estados estables es que la entrada se mantenga estable más de 5 ms en promedio.
- En este caso se optó por controlar la temporización con un *tick* periódico de 1 ms (disparado por una interrupción)



Máquinas de estados finitos (FSM) – Ejemplo anti-rebote

```
volatile int8_t tick = 0, tick_cnt = 0;
typedef enum {OFF, PRE_OFF, PRE_ON, ON} estado_t;
estado_t actual = OFF;
HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
void fsm(estado_t*);
#define TIMEOUT 5
```

```
int main(void) {
    /* ...inicialización... */
    while (1) {
        fsm(&actual);
        HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
    }
}
```

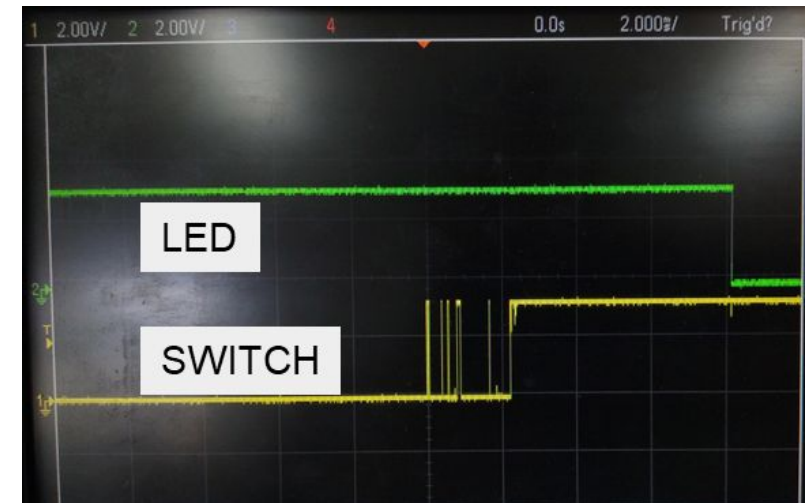
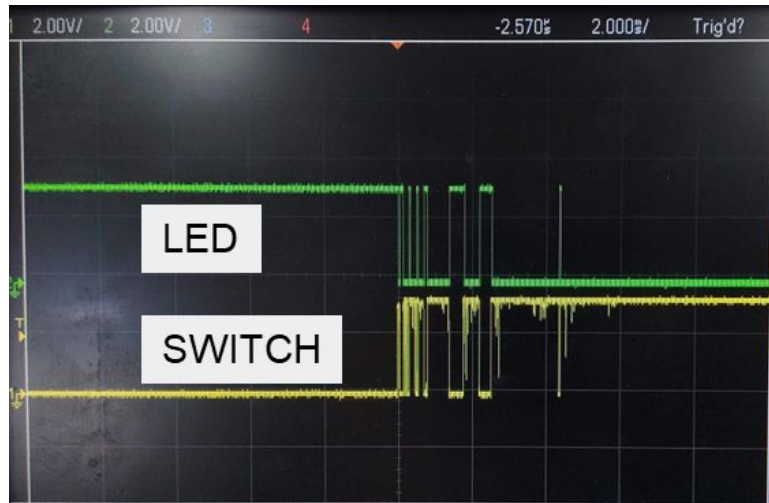
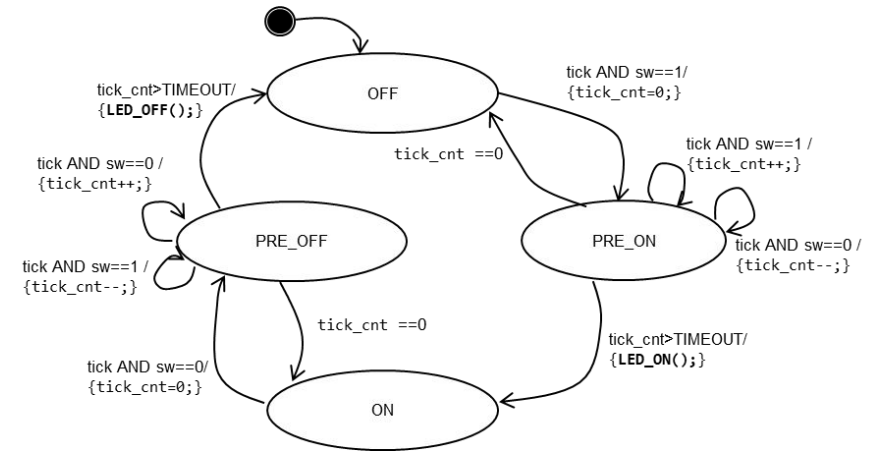
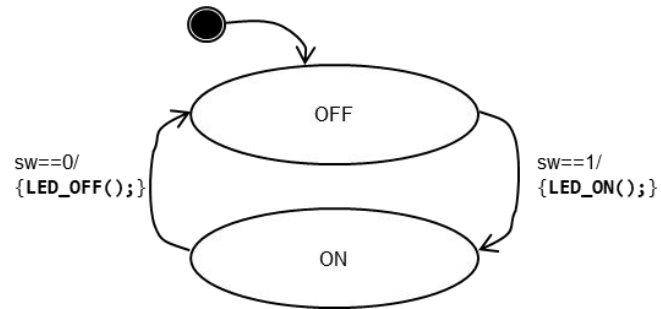


```
void fsm(estado_t *actual) {
    estado_t anterior = *actual;
    switch (*actual) {
    case OFF:
        if (tick) {
            tick = 0;
            if (HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_SET) {
                tick_cnt = 0;
                *actual = PRE_ON;
            }
            break; }
        break;
    case ON:
        if (tick) {
            tick = 0;
            if (HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_RESET) {
                tick_cnt = 0;
                *actual = PRE_OFF;
            }
            break;}
        break;
    case PRE_OFF:
        if (tick_cnt >= TIMEOUT) {
            *actual = OFF;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
            break;}
        if (tick_cnt < 0) {
            *actual = ON;
            break;}
        if (tick) {
            tick = 0;
            if (HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_RESET) {
                tick_cnt++;
            } else {
                tick_cnt--;
            }
            break;}
        break;
    }
```

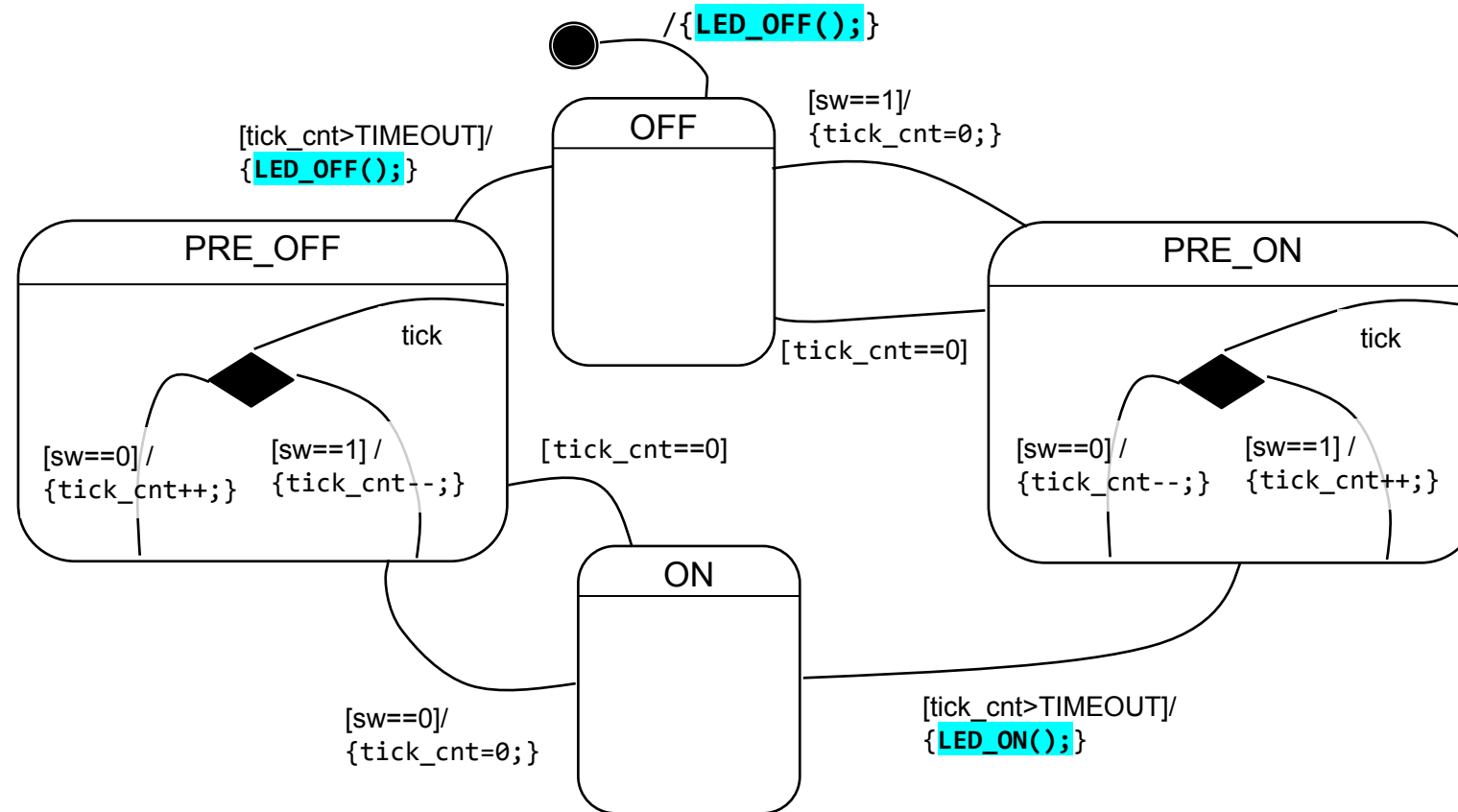
```
case PRE_ON:
    if (tick_cnt >= TIMEOUT) {
        *actual = ON;
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
        break;}
    if (tick_cnt < 0) {
        *actual = OFF;
        break;}
    if (tick) {
        tick = 0;
        if (HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_RESET) {
            tick_cnt--;
        } else {
            tick_cnt++;
        }
        break;}
    break;
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    tick = 1;
}
```

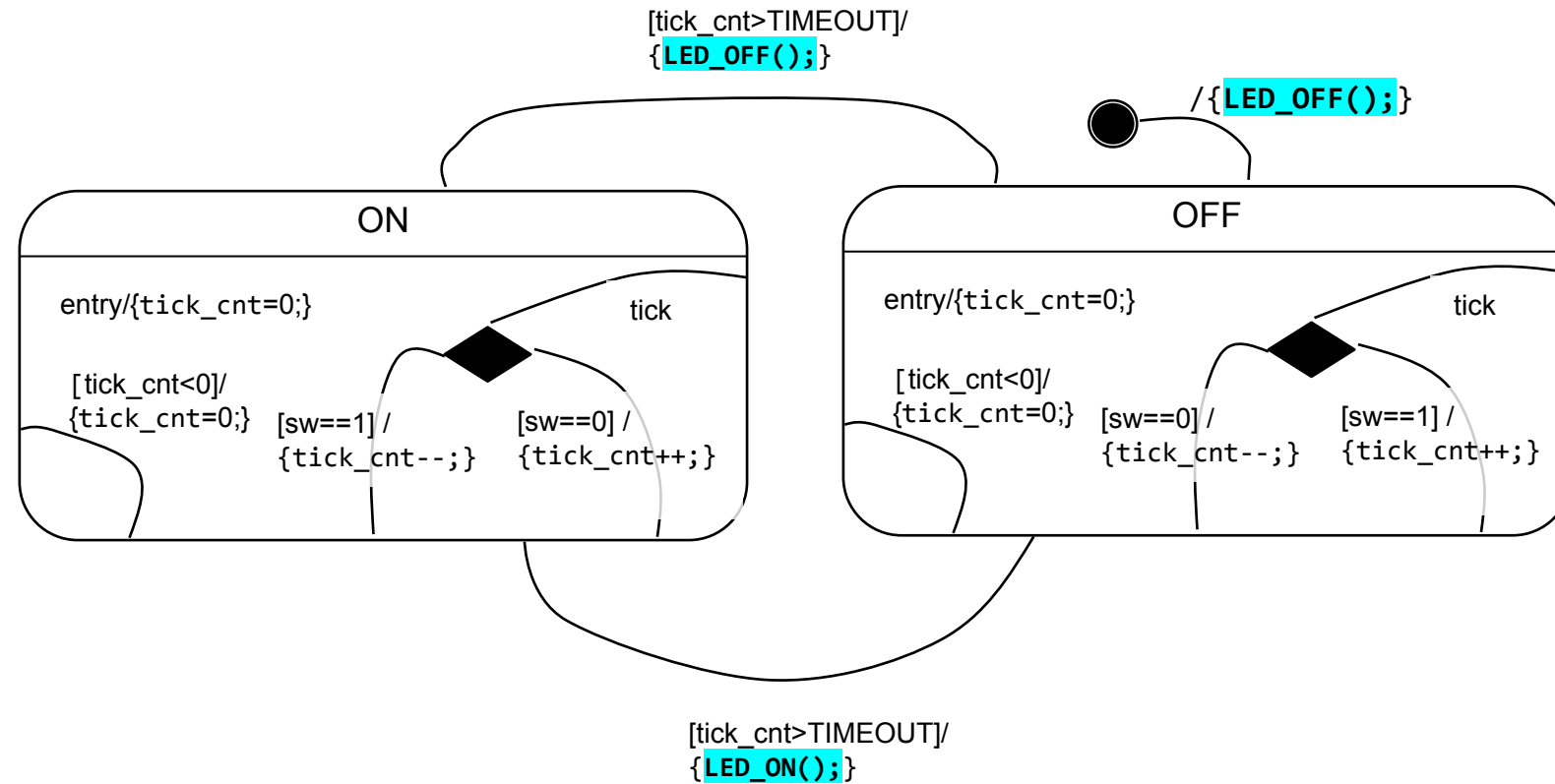
Máquinas de estados finitos (FSM) – Ejemplo anti-rebote



Máquinas de estados finitos (FSM) – Ejemplo anti-rebote



Máquinas de estados finitos (FSM) – Ejemplo anti-rebote



Máquinas de estados finitos (FSM) – Ejemplo anti-rebote

