Programación de SE en C

(Parte I: arquitecturas de firmware)

Arquitecturas de firmware

- Organización elemental
  - Superloop
  - Superloop con Interrupciones: foreground/ background
- Organización según patrones
  - Sistema controlado por eventos
  - Sistemas disparados por tiempo y por eventos
- Máquina de estados (controladas por entradas y/o eventos)
- RTOS
- Frameworks

Superloop básico

```
void main (void)
{
   Inicializar_y_configurar();

   for(;;){
       tarea1();
       tarea2();
       :
       .
       tareaN();

   }
}
```

En los Sistemas Embebidos *Bare-metal* no tenemos SO al cual retornar desde main().

Por lo tanto, Es natural implementar un bucle eterno.

Cada tarea que deba realizar el procesador se puede gestionar mediante una función

Superloop básico

Ventajas:

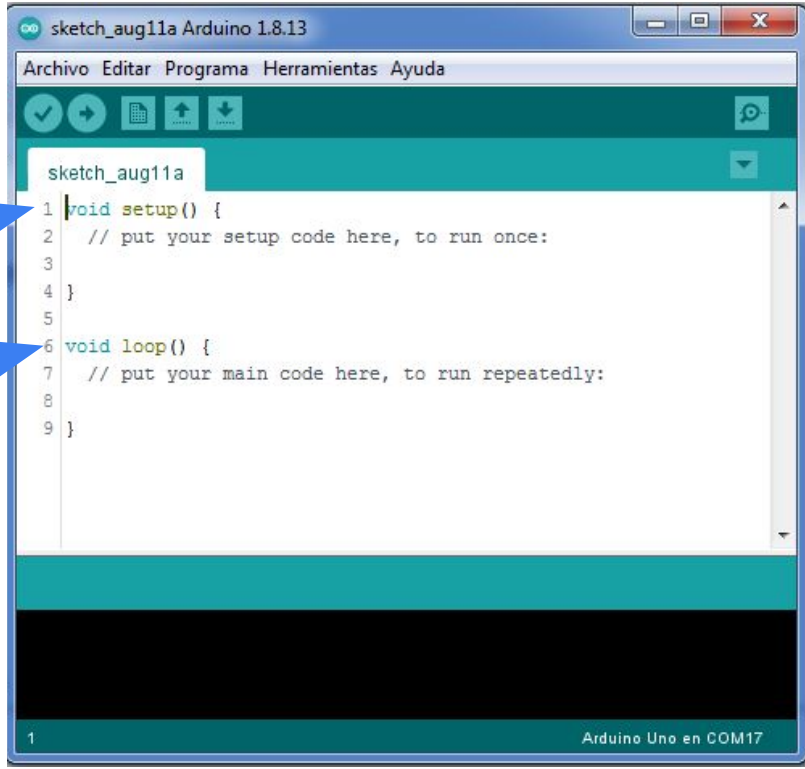
- *Fácil de escribir y depurar*
- *Fácil de interpretar*
- *Flujo de ejecución único*
- *Más fácilmente portable entre arquitecturas de HW como esquema de alto nivel (No usa IRQs ni presupone el uso de recursos de HW)*

Desventajas:

- *Difícil asegurar el cumplimiento de restricciones temporales*
- *Latencia entre entrada/salida variable*
- *El agregado de una nueva tarea o la modificación de las existentes pueden llevar a resultados imprevistos -> difícil de mantener y escalar*
- *Mayor consumo de energía que otras estructuras de FW*

Superloop básico

```
/*   main.cpp - Main loop for Arduino sketches    ... */
#include <Arduino.h>

int main(void){
      init();
      initVariant();
      ...
      setup();

      for (;;) {
            loop();
            if (serialEventRun) serialEventRun();
      }
      return 0;
}
```



Superloop básico

- *Ejemplo*
  - *Salida: LED on-board en GPIOC-13*
  - *El led debe parpadear a ¼ Hz (2 seg. encendido/2 seg. apagado).*

***Nota:** Para los retardos se utiliza HAL_Delay() que se apoya en un contador global que se incrementa con las interrupciones del SysTick, cada 1ms. Si bien se usa una interrupción para tener una buena referencia temporal, la solución se comporta como un superloop básico, dado que HAL_Delay() es bloqueante.*

```
void HAL_Delay(uint32_t Delay){
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

  while ((HAL_GetTick() - tickstart) < wait) {}
}
```

Superloop básico

```
uint16_t periodo= 4000;
while (1)
{
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
   HAL_Delay(periodo/2);
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
   HAL_Delay(periodo/2);

}
```

Superloop básico

- *Ejemplo*
  - *Entrada: SWITCH en GPIOA-9 (conectado a 3V3 y pin configurado con pull-down interno)*
  - *Salida: LED on-board en GPIOC-13*
  - *Si el switch está presionado, el led debe parpadear a 10 Hz, caso contrario a ¼ Hz.*

Superloop básico

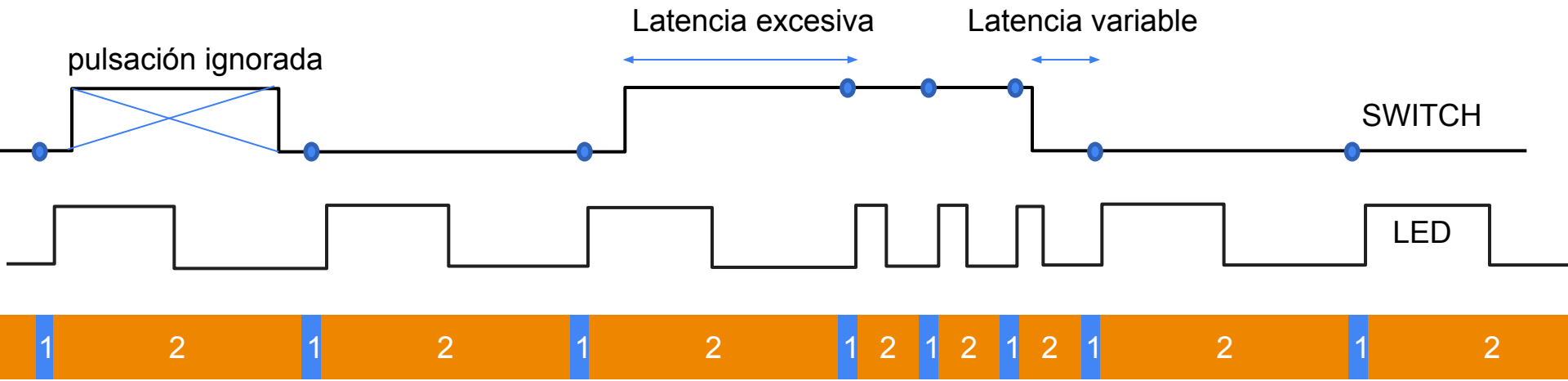
```
uint16_t periodo;
while (1)
{
   /* Tarea 1: Leer switch y calcular período*/
   if(HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_SET)
      periodo = 100;
   else
      periodo = 4000;

   /* Tarea 2: Parpadear un período */
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
   HAL_Delay(periodo/2);
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
   HAL_Delay(periodo/2);
}
```

Superloop básico

```
uint16_t periodo;
while (1)
{
   /* Tarea 1: Leer switch y calcular período*/
   if(HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_SET)
      periodo = 100;
   else
      periodo = 4000;

   /* Tarea 2: Parpadear un período */
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
   HAL_Delay(periodo/2);
   HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
   HAL_Delay(periodo/2);
}
```

La tarea 2 deja al sistema **bloqueado**… no puede reaccionar al cambio de estado del switch

Superloop básico



Superloop básico

```
uint16_t periodo;
while (1)
{
   /* Tarea 1: Leer switch y calcular período*/
   if(HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin) == GPIO_PIN_SET)
      periodo = 100;
   else
      periodo = 4000;

   /* Tarea 2: Invertir LED cada semiperíodo */
   HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
   HAL_Delay(periodo/2);
}
```

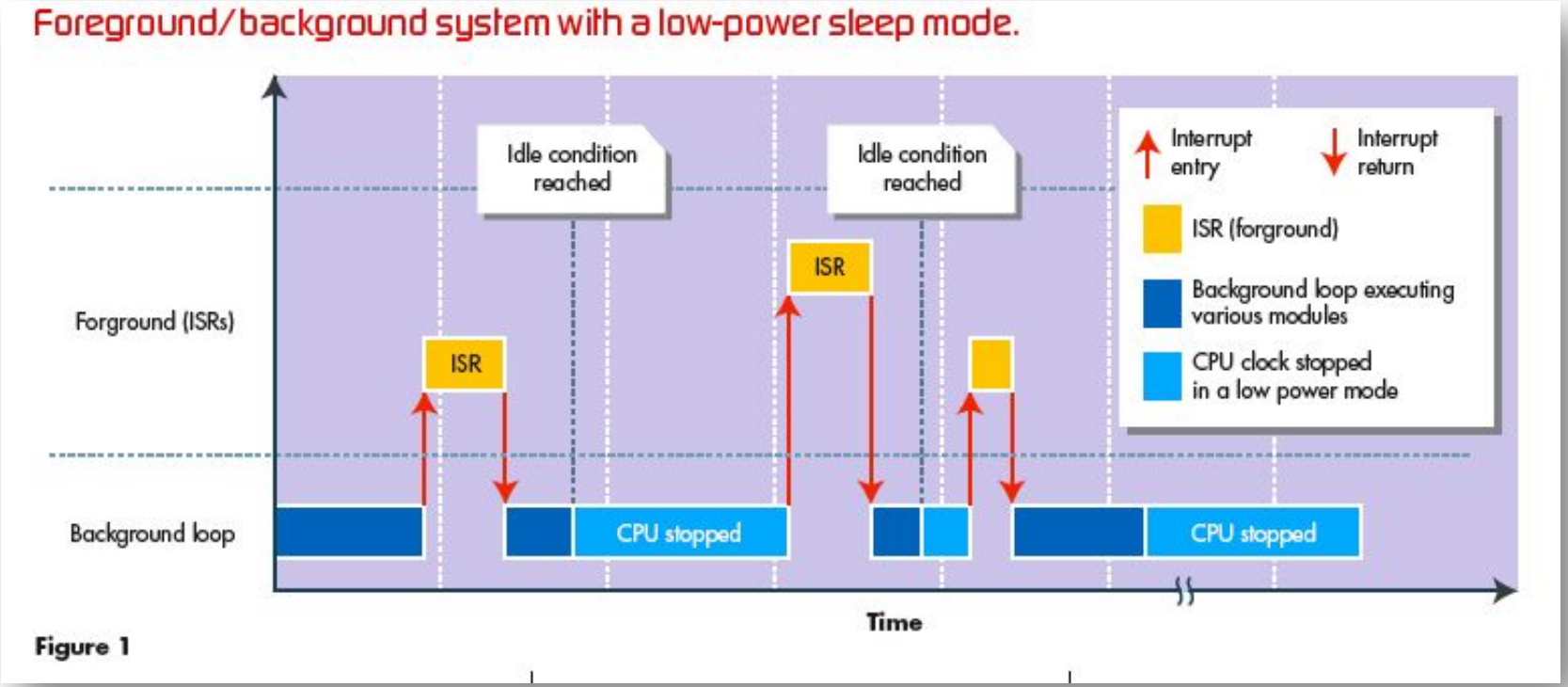
Solución mejorada…pero sigue siendo bloqueante!

Superloop con interrupciones – (Foregroung/Background)

```
void main (void)
{
   Inicializar_y_configurar();

   for(;;){
       tareaBG1();
       tareaBG2();
       :
       .
       tareaBGN();
       /* Dormir(); */

   }
}
```

```
void ISR_FG1 (void)
{
   ...
}

void ISR_FG2 (void)
{
   ...
}
```

Superloop con interrupciones – (Foregroung/Background)



https://www.embedded.com/use-an-mcus-low-power-modes-in-foreground-background-systems/

Superloop con interrupciones – (Foreground/Background)

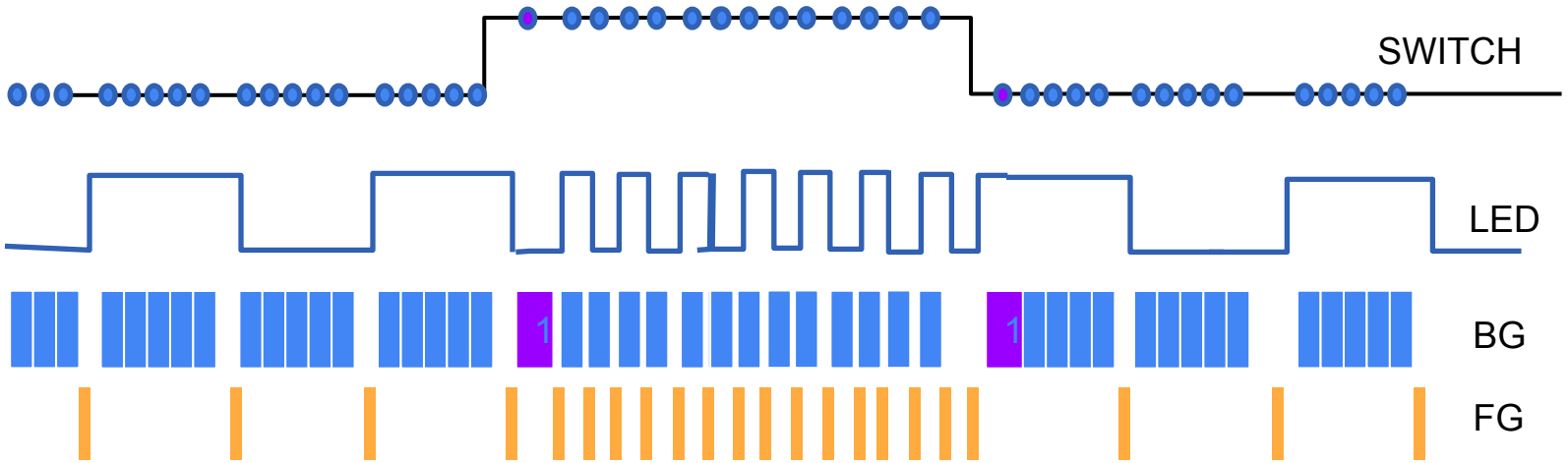
Background

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  HAL_TIM_Base_Start_IT(&htim3);

  GPIO_PinState sw_anterior=GPIO_PIN_RESET, sw_actual;
  while (1)
  {
    /* Tarea 1: Leer switch y calcular período*/
    sw_actual = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);
    /* Transición a estado alto */
    if(sw_anterior == GPIO_PIN_RESET && sw_actual == GPIO_PIN_SET) {
      htim3.Instance->ARR = 50;
      htim3.Instance->EGR |= 1; /* actualizo Timer (ARR buffereado)*/}
    /* Transición a estado bajo */
    else if(sw_anterior == GPIO_PIN_SET && sw_actual == GPIO_PIN_RESET){
      htim3.Instance->ARR = 2000;
      htim3.Instance->EGR |= 1; /* actualizo Timer (ARR buffereado)*/}
    sw_anterior = sw_actual;
  }
}
```

Foreground

```
/* Tarea 2: Invertir LED cada semiperíodo */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}
```

Superloop con interrupciones – (Foreground/Background)



Superloop con interrupciones – (Foreground/Background)

También se puede hacer todo en foreground

Background

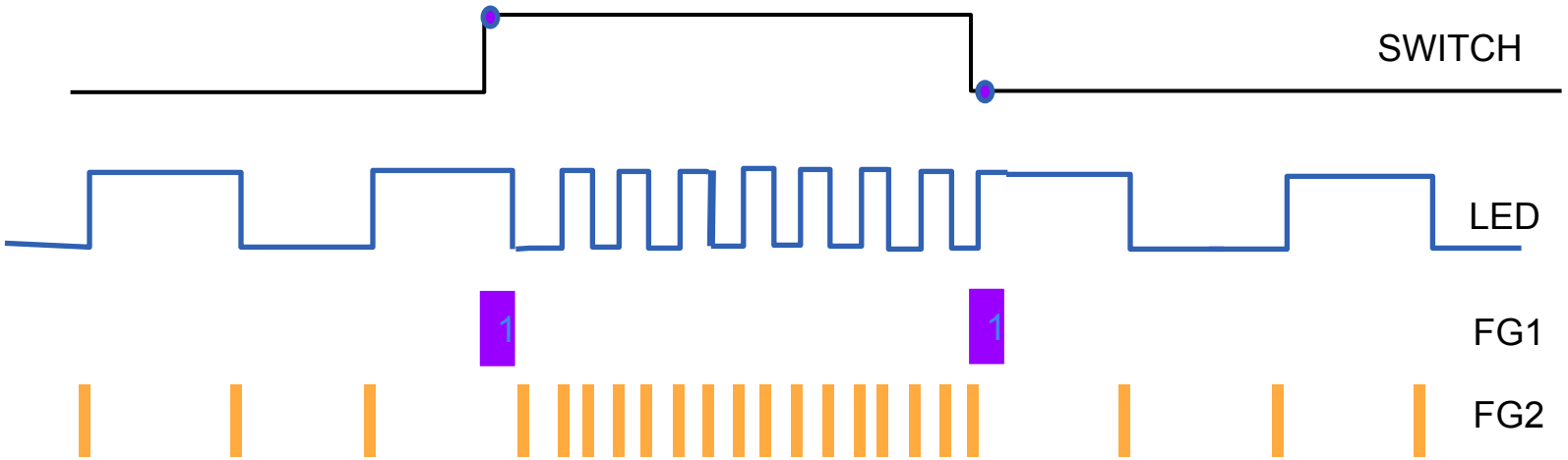
```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  HAL_TIM_Base_Start_IT(&htim3);

  while (1)
  {
  }
}
```

Foreground

```
/* Tarea 1: Leer switch y calcular período*/
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

  GPIO_PinState sw = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);

  /* Transición a estado alto */
  if(sw == GPIO_PIN_SET) htim3.Instance->ARR = 50;
  /* Transición a estado bajo */
  else htim3.Instance->ARR = 2000;

  /* actualizo Timer (ARR buffereado)*/
  htim3.Instance->EGR |= 1;
}
```

```
/* Tarea 2: Invertir LED cada semiperíodo */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}
```

Superloop con interrupciones – (Foreground/Background)



Superloop con interrupciones – (Foreground/Background)

También se puede hacer todo en foreground…**y con consumo de energía reducido**

Background

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  HAL_TIM_Base_Start_IT(&htim3);

  while (1)
  {
    HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON,PWR_SLEEPENTRY_WFI);
  }
}
```

Foreground

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){

  GPIO_PinState sw = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);

  /* Transición a estado alto */
  if(sw == GPIO_PIN_SET) htim3.Instance->ARR = 50;
  /* Transición a estado bajo */
  else htim3.Instance->ARR = 2000;

  /* actualizo Timer (ARR buffereado)*/
  htim3.Instance->EGR |= 1;
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
}
```

if(Interrupciones){problemas_de_concurrencia = **true**;}

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
      ...
      flag = 1; //escritura (0)
 }
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  ...
      flag = 1; //escritura (2)
}
```

```
volatile uint8_t flag=0;
int main(void){
 while (1){
    if(flag){ //lectura (1)
      procesa_evt();
      flag=0; //escritura (3)
    }
  }
}
```

Condición de carrera: Si los eventos se dan en una secuencia determinada, el sistema se perdería una interrupción.

if(Interrupciones){problemas_de_concurrencia = **true**;}

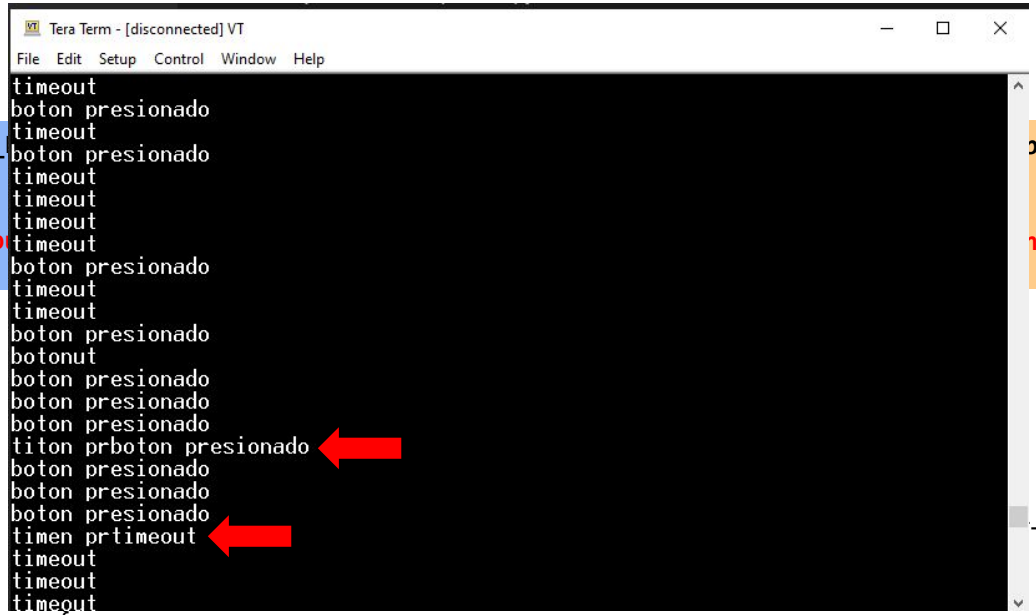
```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
  ...
  flag = 1;
  strncpy(str_buffer,“botón presionado\n”,50);
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  ...
  flag = 1;
  strncpy(str_buffer,“timeout\n”,50);
}
```

```
volatile char str_buffer[50];
volatile uint8_t flag=0;
int main(void){
 while (1){
    if(flag){
      HAL_UART_Transmit(&huart1, str_buffer, strlen(str_buffer), HAL_MAX_DELAY);
      flag=0;
    }
  }
}
```

if(Interrupciones){problemas_de_concurrencia = **true**;}

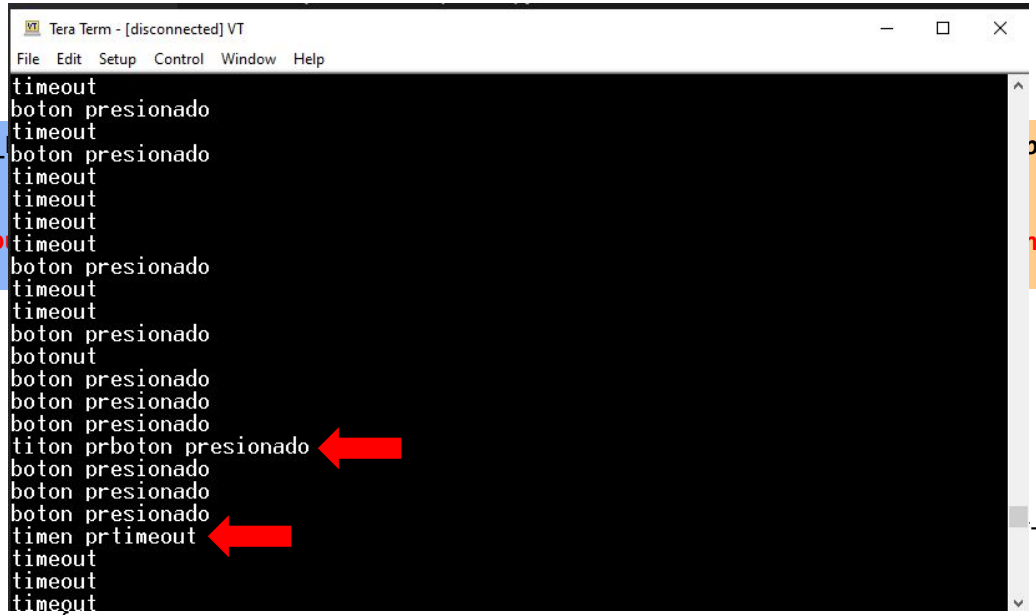
```
void HAL_GPIO_
  ...
  flag = 1;
  strncpy(str_b
}
```




```
back(TIM_HandleTypeDef *htim){

n",50);

_MAX_DELAY);
    }
  }
```

if(Interrupciones){problemas_de_concurrencia = **true**;}

```
void HAL_GPIO_
  ...
  flag = 1;
  strncpy(str_b
}
```




```
back(TIM_HandleTypeDef *htim){

n",50);

_MAX_DELAY);
    }
  }
```

if(Interrupciones){problemas_de_concurrencia = **true**;}

- **Usar `volatile`** para evitar optimizaciones del compilador.
- **Deshabilitar interrupciones temporalmente** al modificar/leer variables compartidas.
- **Usar copias** de variables
- **Estructuras de datos seguras** como colas de eventos o semáforos.

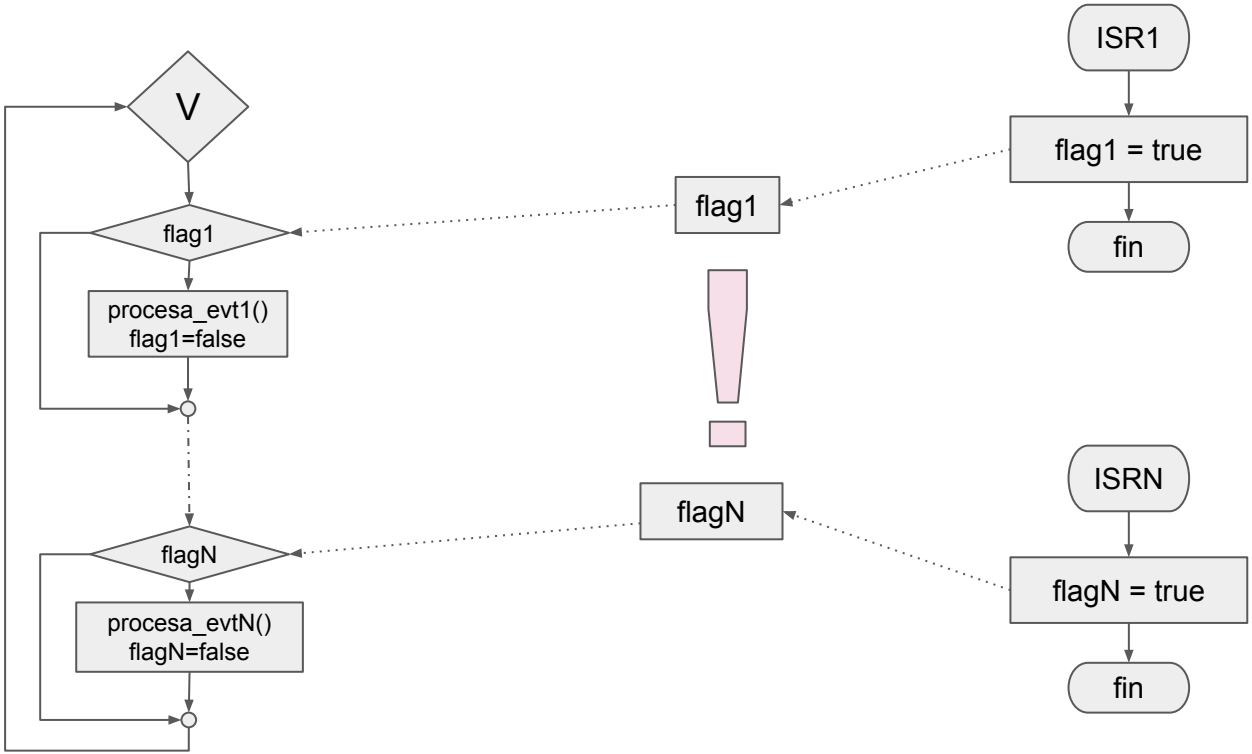
if(Interrupciones){problemas_de_concurrencia = **true**;}

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
 …
__disable_irq();
 flag = 1;
 strncpy(str_buffer,“botón presionado\n”,50);
__enable_irq();
}
```

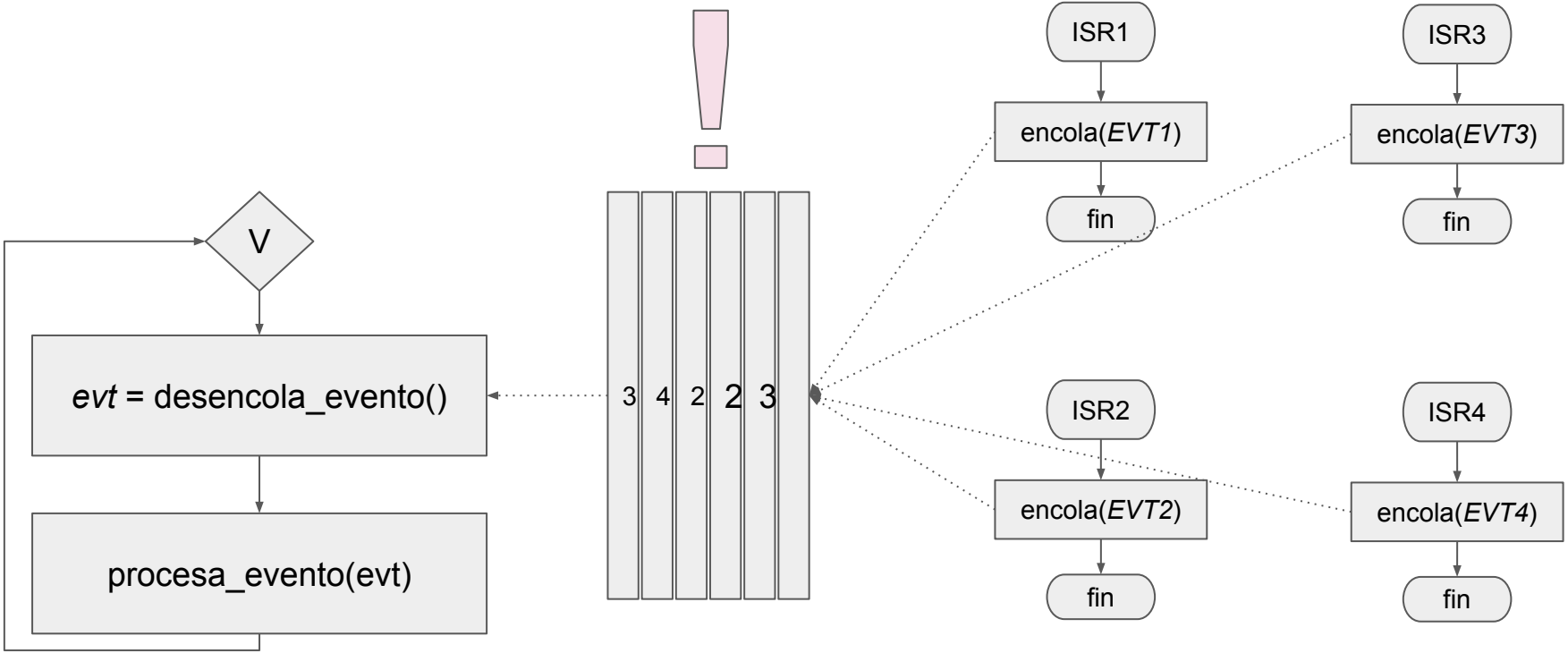
```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  …
__disable_irq();
  flag = 1;
  strncpy(str_buffer,“timeout\n”,50);
__enable_irq();
}
```

```
volatile char str_buffer[50];
volatile uint8_t flag=0;
int main(void){
 while (1){
    char local_buffer[50];
    uint8_t local_flag;
    __disable_irq();
    local_flag = flag;
    if(local_flag) { strncpy(local_buffer,str_buffer,50); flag=0;}
    __enable_irq();
    if(local_flag){ HAL_UART_Transmit(&huart1, local_buffer, strlen(local_buffer), HAL_MAX_DELAY);
    }
  }
}
```

Sistemas controlados por eventos (Event-driven architecture)



Sistemas controlados por eventos (Event-driven architecture)

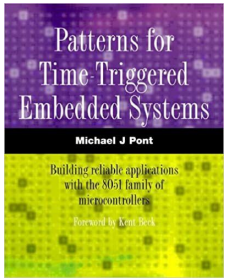


Sistemas controlados por tiempo (Time-Triggered Architecture)

La ***Time-Triggered Architecture*** (gatillada por tiempo) es una arquitectura derivada de la anterior, donde solo se activa una interrupción temporal periódica y sólo en estos tiempos conocidos, el sistema puede evaluar las entradas (*polling*), cambiar su estado y ejecutar las tareas correspondientes.

En general este último esquema es menos eficiente energéticamente y menos reactivo, pero ofrece previsibilidad y determinismo en el comportamiento y por eso se lo utiliza en aplicaciones críticas.

Se recomienda la lectura de *Patterns for Time Triggered Embedded Systems* de Michael J Point.



Time triggered

```
volatile uint32_t tick = 0;
volatile uint32_t old_tick = 0;
volatile uint32_t old_tick2 = 0;
volatile uint32_t periodo = 2000;
int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_TIM3_Init();
 HAL_TIM_Base_Start_IT(&htim3);

 while (1)
 {
  if(tick - old_tick >= periodo){ //cada periodo
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    old_tick=tick;}
  if(tick - old_tick2 >= 100){ //cada 100 milisegundos
    GPIO_PinState input = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);
    if(input == GPIO_PIN_SET)periodo = 50;
    else periodo = 2000;
    old_tick2=tick;}
 }
}
```

interrupción cada 1 ms

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

Time triggered

```
volatile uint32_t tick = 0;
volatile uint32_t old_tick = 0;
volatile uint32_t old_tick2 = 0;
volatile uint32_t periodo = 2000;
int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 MX_TIM3_Init();
 HAL_TIM_Base_Start_IT(&htim3);

 while (1)
 {
  if(tick - old_tick >= periodo){ //cada periodo
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    old_tick=tick;}
  if(tick - old_tick2 >= 100){ //cada 100 milisegundos
    GPIO_PinState input = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);
    if(input == GPIO_PIN_SET)periodo = 50;
    else periodo = 2000;
    old_tick2=tick;}
 }
}
```

interrupción cada 10 ms

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  tick+=10;
}
```

Time triggered

```
volatile uint32_t tick = 0;
volatile uint32_t old_tick = 0;
volatile uint32_t old_tick2 = 0;
volatile uint32_t periodo = 2000;
int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();


 while (1)
 {
  tick = HAL_GetTick();
  if(tick - old_tick >= periodo){ //cada periodo
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
    old_tick=tick;}
  if(tick - old_tick2 >= 100){ //cada 100 milisegundos
    GPIO_PinState input = HAL_GPIO_ReadPin(SWITCH_GPIO_Port, SWITCH_Pin);
    if(input == GPIO_PIN_SET)periodo = 50;
    else periodo = 2000;
    old_tick2=tick;}
 }
}
```

```
/**
* @brief Provides a tick value
in millisecond.
* @note This function is
declared as __weak to be
overwritten in case of other
* implementations in user file.
* @retval tick value
*/
__weak uint32_t
HAL_GetTick(void)
{
return uwTick;
}
```

```
/**
* @brief This function is called to
increment a global variable "uwTick"
* used as application time base.
* @note In the default implementation, this
variable is incremented each 1ms
* in SysTick ISR.
* @note This function is declared as __weak
to be overwritten in case of other
* implementations in user file.
* @retval None
*/
__weak void HAL_IncTick(void)
{
uwTick += uwTickFreq;
}
```

```
/**
* @brief This function handles System
tick timer.
*/
void SysTick_Handler(void)
{
/* USER CODE BEGIN SysTick_IRQn 0 */
/* USER CODE END SysTick_IRQn 0 */
HAL_IncTick();
/* USER CODE BEGIN SysTick_IRQn 1 */
/* USER CODE END SysTick_IRQn 1 */
}
```