Electrónica Digital II

Santiago Rúa Pérez, PhD.

18 de septiembre de 2022

CONCURRENCIA BÁSICA

Objetivos

- Entender como realizar un programa modular.
- Medira la sensibilidad de un sistema o software.
- Comprender la sobrecarga de la CPU.
- Trabajar con interrupciones y eventos.

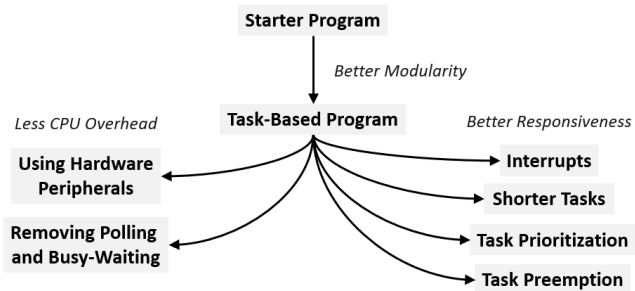
Reto de planificación

Cómo compartir los recursos de la CPU entre diferentes actividades o tareas que tiene el software?

- Típicamente un sistema embebido tiene múltiples tareas que debe administrar.
  - Algunas son realizadas por los periféricos, otras por software en la CPU.
- Cada actividad tiene requerimientos de tiempo, y algunos son más críticos que otros.
  - Ejemplos: lecturas de pulsadores, escanear LEDs.
- Se desea que solo una CPU parezca que puede hacer muchas cosas al tiempo de forma concurrente.
  - Inclusive con procesadores multinúcleo, se tiene mas tareas que procesadores.
- Como compartir la CPU entre diferentes tareas? Como simulamos concurrencia entre las diferentes actividades? **Como hacemos planificación de tareas**?

Roadmap

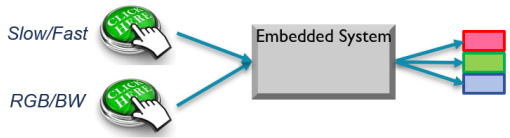
- Modularidad: que tanto el programa está estructurado en pequeñas porciones funcionales.
- Respuesta: medir que tan rápido el sistema responde ante un evento.
- Sobrecarga de la CPU: cuanto tiempo se demora la CPU ejecutando instrucciones fuera de las principales.

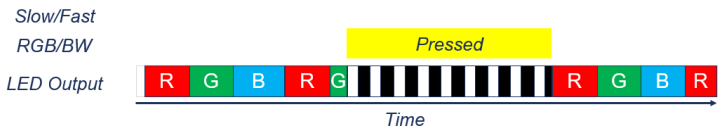


Problema a resolver

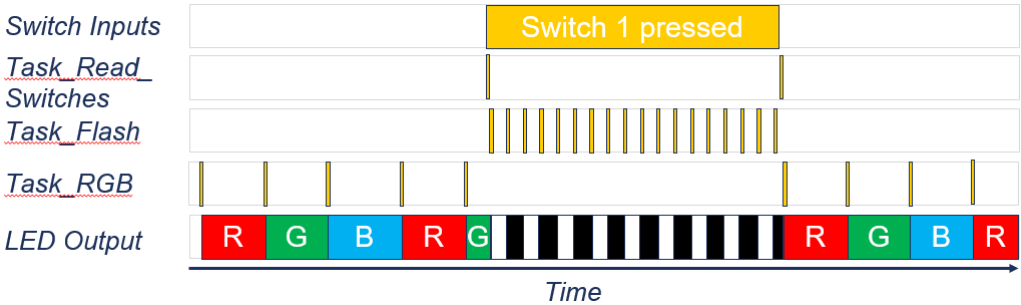
Se tiene un sistema con dos pulsadores y un LED RGB.

- Cuando el suiche 1 no se presiona, el sistema muestra una secuencia de forma repetitiva (R-G-B).
- Cuando se presiona 1, comienza a parpadear en blanco (todos prendidos), hasta que se libera
- Mientras se mantenga presionado 2, es más rápido el parpadeo y la secuencia RGB.





Planificación ideal - Manejado por eventos



- Note que la ejecución de cada tarea consume poco tiempo dentro del diagrama.
- Inmediatamente se presiona el suiche el estado del Led cambia y solo tarda lo que demora la tarea de lectura y de parpadeo.

Solución 1

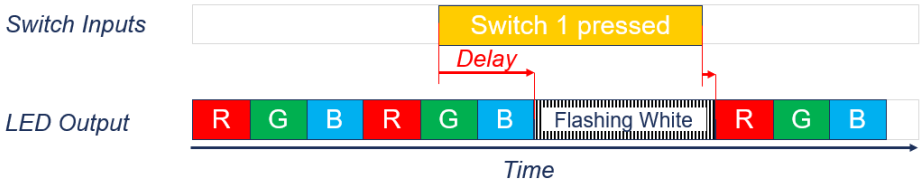
```
#define W_DELAY_SLOW 400
#define W_DELAY_FAST 200
#define RGB_DELAY_SLOW 4000
#define RGB_DELAY_FAST 1000

void Flasher(void) {
  uint32_t w_delay = W_DELAY_SLOW;
  uint32_t RGB_delay = RGB_DELAY_SLOW;

  Init_GPIO_RGB();
  Init_GPIO_Switches();
  while (1) {
    if (SWITCH_PRESSED(SW1_POS)) {  // flash white
      Control_RGB_LEDs(1, 1, 1);
      Delay(w_delay);
      Control_RGB_LEDs(0, 0, 0);
      Delay(w_delay);
    } else {                        // sequence R, G, B
      Control_RGB_LEDs(1, 0, 0);
      Delay(RGB_delay);
      Control_RGB_LEDs(0, 1, 0);
      Delay(RGB_delay);
      Control_RGB_LEDs(0, 0, 1);
      Delay(RGB_delay);
    }
    if (SWITCH_PRESSED(SW2_POS)) {
      w_delay = W_DELAY_FAST;
      RGB_delay = RGB_DELAY_FAST;
    } else {
      w_delay = W_DELAY_SLOW;
      RGB_delay = RGB_DELAY_SLOW;
    }
  }
}
```

Qué pasa cuando se presiona el suiche 1 cuando esta en verde el led?

Análisis - Solución 1



- Considere el programa realizado.
- Problemas:
  - Tiene mantener presionado el interruptor hasta que el programa lo sondea, puede haber una gran demora.
  - Algunas veces que se presiona el interruptor pueden ignorarse.
- El programa solo lee los interruptores por cada ciclo del LED.
- El ciclo de escaneo del LED puede tomar mayor tiempo.

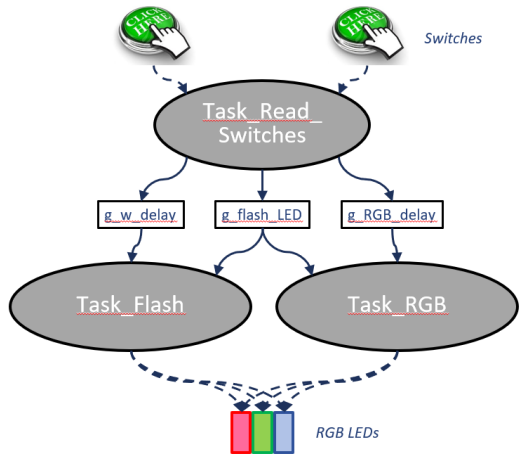
Creando y usando tareas

Una estructura buena de codificación simplifica el desarrollo

- Tareas: código que desarrolla una actividad, o conjunto de actividades.
  - Parpadear leds.
  - Leer interruptores.
- Simplifica el desarrollo.
  - Agrupar por funcionalidades similares.
  - Separar funcionalidades diferentes.
  - Mejora el debug, mantenimiento y desarrollo.
- Cada tarea es llamada desde otra tarea o función principal.
  - La tarea principal puede llamar a otras funciones que necesite.

Solución 2

- Se utilizan variables globales para compartir información.
- Las flechas indican quien modifica y lee las variables globales.
- Este tipo de esquema se llama multitarea cooperativa, si una tarea se demora mucho entonces las demás les toca esperar.



Solución 2 - Código

Tarea de lectura de los suiches

```
1  volatile uint8_t g_flash_LED = 0;
2  volatile uint32_t g_w_delay = W_DELAY_SLOW;
3  volatile uint32_t g_RGB_delay = RGB_DELAY_SLOW;
4
5  void Task_Read_Switches(void) {
6    if (SWITCH_PRESSED(SW1_POS)) {
7      g_flash_LED = 1; // flash white
8    } else {
9      g_flash_LED = 0; // RGB sequence
10   }
11   if (SWITCH_PRESSED(SW2_POS)) {
12     g_w_delay = W_DELAY_FAST;
13     g_RGB_delay = RGB_DELAY_FAST;
14   } else {
15     g_w_delay = W_DELAY_SLOW;
16     g_RGB_delay = RGB_DELAY_SLOW;
17   }
18 }
19
```

Solución 2 - Código

Tarea de parpadear el led y secuencia RGB

```
1  void Task_Flash(void) {
2    if (g_flash_LED == 1) { // Only run task when in flash mode
3      Control_RGB_LEDs(1, 1, 1);
4      Delay(g_w_delay);
5      Control_RGB_LEDs(0, 0, 0);
6      Delay(g_w_delay);
7    }
8  }
9  void Task_RGB(void) {
10   if (g_flash_LED == 0) { //only run task when NOT in flash mode
11     Control_RGB_LEDs(1, 0, 0);
12     Delay(g_RGB_delay);
13     Control_RGB_LEDs(0, 1, 0);
14     Delay(g_RGB_delay);
15     Control_RGB_LEDs(0, 0, 1);
16     Delay(g_RGB_delay);
17   }
18 }
19
```

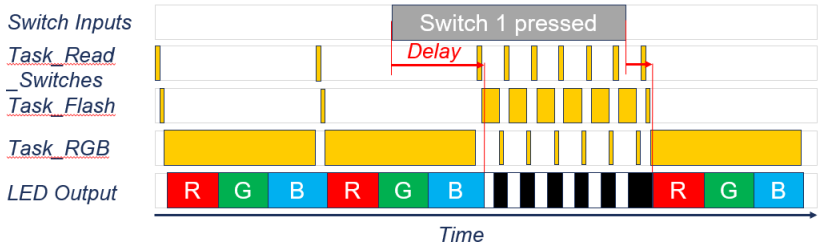
Solución 2 - Código

Programa principal o tarea principal

```
1  void Flasher(void) {
2    Init_GPIO_RGB();
3    Init_GPIO_Switches();
4    while (1) {
5      Task_Read_Switches();
6      Task_Flash();
7      Task_RGB();
8    }
9  }
10
```

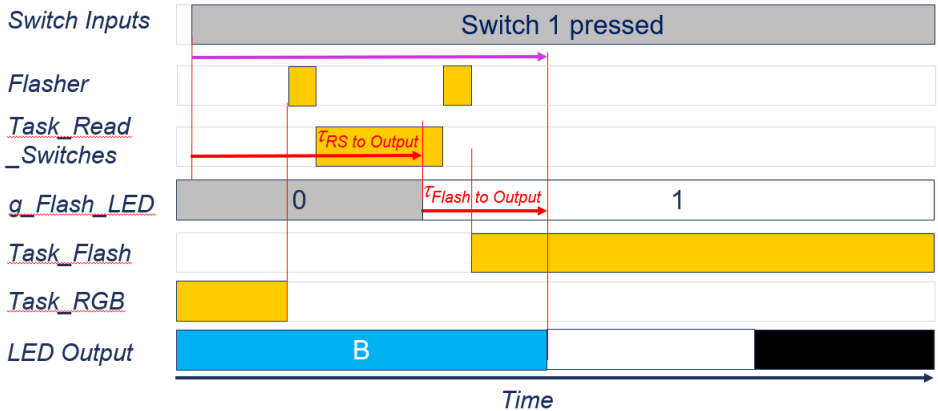
Cada tarea es llamada de forma secuencial, por lo que la ejecución dependerá de la más lenta.

Análisis - Solución 2



- Flasher es un planificador de tareas (**task scheduler**), el cual hace un llamado a cada tarea. Cada tarea se ejecuta y devuelve el control al planificador.
- Máximo retardo ocurre cuando se presiona el interruptor justo cuando inicia la secuencia RGB.
- Cada tarea se ejecuta después

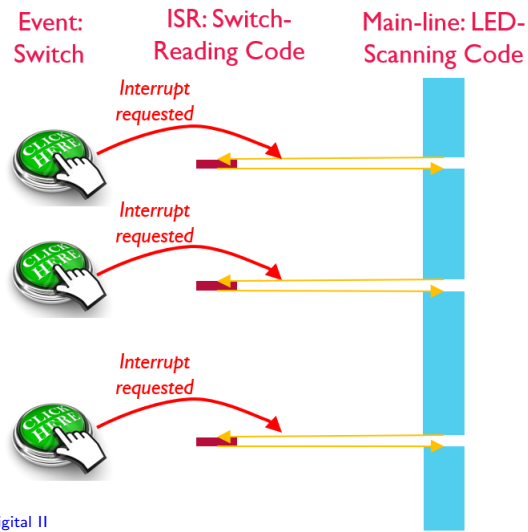
Considerar retrasos entre tareas



- La tarea de lectura de interruptores cambia la salida despues de τ_{RS}.
- La tarea de parpadear cambia el led despues del retraso τ_{Flash}
- Se puede mejorar utilizando interrupciones en vez de polling.

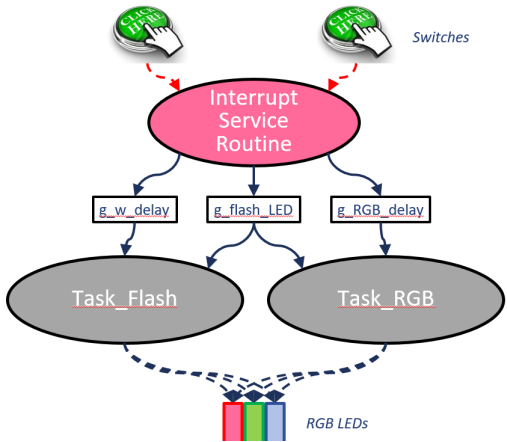
Conceptos de interrupciones

- Interrupción
  - Señal que indica que un evento particular pasó.
- Comportamiento del sistema.
  - CPU está ejecutando programas.
  - La interrupción fuerza al sistema a parar y correr una función especial (Interrupt Service Routine ISR), el cual atiende el evento.
  - Después de que finaliza, el control vuelve a la CPU.
- Nunca se llamada a una función ISR desde el código.



Solución 3

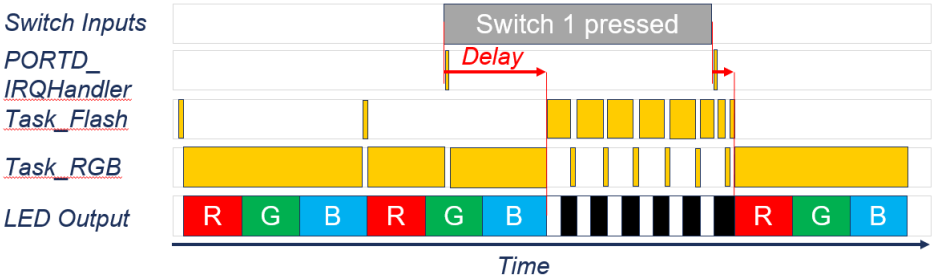
- Utilizar rutina de servicio de interrupción para los suiches.
- Comunicarse a través de variables globales.



Solución 3 - Código

```
1  volatile uint8_t g_flash_LED = 0;
2  volatile uint32_t g_w_delay = W_DELAY_SLOW;
3  volatile uint32_t g_RGB_delay = RGB_DELAY_SLOW;
4
5  void PORTD_IRQHandler(void){
6    if ((PORTD->ISRF & MASK(SW1_POS))) {
7      if (SWITCH_PRESSED(SW1_POS)) {
8        g_flash_LED = 1;          //
9      } else {
10       g_flash_LED = 0;          // RGB
11     }
12   }
13   if ((PORTD->ISRF & MASK(SW2_POS))) {
14     if (SWITCH_PRESSED(SW2_POS)) {  // Short delays
15       g_w_delay = W_DELAY_FAST;
16       g_RGB_delay = RGB_DELAY_FAST;
17     } else {
18       g_w_delay = W_DELAY_SLOW;
19       g_RGB_delay = RGB_DELAY_SLOW;
20     }
21   }
22   PORTD->ISFR = 0XFFFFFFFF;
23 }
24
```

Análisis - Solución 3



- Mejoramiento limitado del ISR, ya que la tarea de RGB sigue su ejecución.
  - La tarea de RGB pone limitantes.
  - Ejemplo de un sistema no preventivo (non-preemptive): las tareas no puede apropiarse de los recursos mientras se ejecuta otra.

Como podemos mejorar tiempo de respuesta?

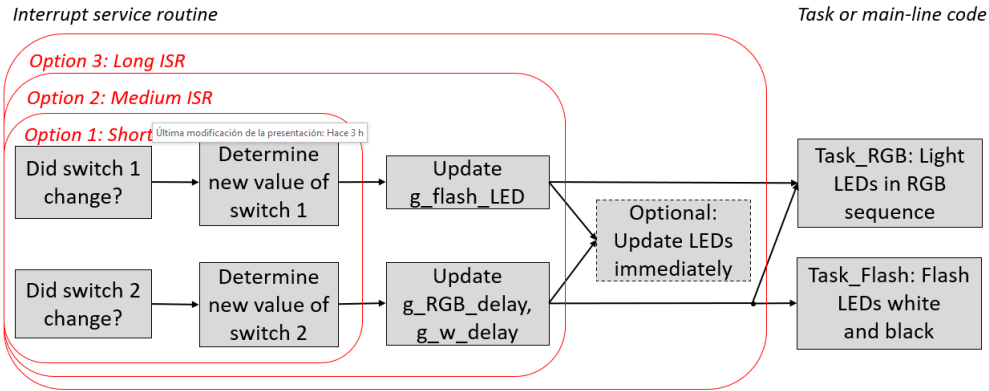
- Adicionar mas test dentro de las tareas?
  - Posibilita que la tarea termine antes.
- Mala idea
  - Mezcla diferente tareas para el planificador.
  - Código extra de ejecución.
  - Dificil de mantener en el tiempo.
- Modificar la función de retraso para terminar antes? Código spaghetti

```
void Task_Flash(void) {
    if (g_flash_LED == 1) {      // Only run task when in flash mode
        Control_RGB_LEDs(1, 1, 1);
    }
    if (g_flash_LED == 1) {      // Only run task when in flash mode
        Delay(g_w_delay);
    }
    if (g_flash_LED == 1) {      // Only run task when in flash mode
        Control_RGB_LEDs(0, 0, 0);
    }
    if (g_flash_LED == 1) {      // Only run task when in flash mode
        Delay(g_w_delay);
    }
}

void Task_RGB(void) {
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Control_RGB_LEDs(1, 0, 0);
    }
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Delay(g_RGB_delay);
    }
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Control_RGB_LEDs(0, 1, 0);
    }
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Delay(g_RGB_delay);
    }
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Control_RGB_LEDs(0, 0, 1);
    }
    if (g_flash_LED == 0) {      // only run task when NOT in flash mo
        Delay(g_RGB_delay);
```

Como podemos mejorar tiempo de respuesta?

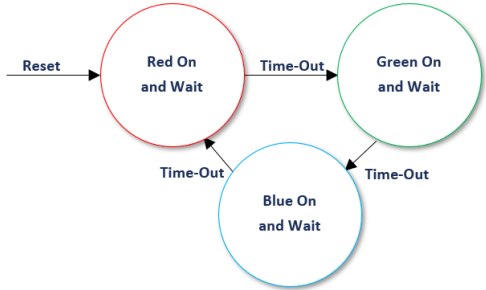
Mover más código al ISR?



Solución: Utilizar máquinas de estado finito.

Reducir tiempo de respuesta con FSM

- Reescribir cada tarea para returno el control al planificador cuando antes.
- Cada llamado ejecuta un estado al tiempo.
- **Objetivo**: reducir tiempos largos de consumo de una tarea.
- Utilizar un case por cada estado.



Solución 4 - FSM

```
void Task_RGB(void) {
    if (g_flash_LED == 0) {          // c
        // Red state
        Control_RGB_LEDs(1, 0, 0);
        Delay(g_RGB_delay);

        // Green state
        Control_RGB_LEDs(0, 1, 0);
        Delay(g_RGB_delay);

        // Blue state
        Control_RGB_LEDs(0, 0, 1);
        Delay(g_RGB_delay);
    }
}
```

```
void Task_RGB_FSM(void) {
    static enum {ST_RED, ST_GREEN, ST_BLUE} next_state=ST_RED;

    if (g_flash_LED == 0) {      // only run task when NOT in flash mode
        switch (next_state) {
            case ST_RED:
                Control_RGB_LEDs(1, 0, 0);
                Delay(g_RGB_delay);
                next_state = ST_GREEN;
                break;
            case ST_GREEN:
                Control_RGB_LEDs(0, 1, 0);
                Delay(g_RGB_delay);
                next_state = ST_BLUE;
                break;
            case ST_BLUE:
                Control_RGB_LEDs(0, 0, 1);
                Delay(g_RGB_delay);
                next_state = ST_RED;
                break;
            default:
                next_state = ST_RED;
                break;
        }
    }
}
```

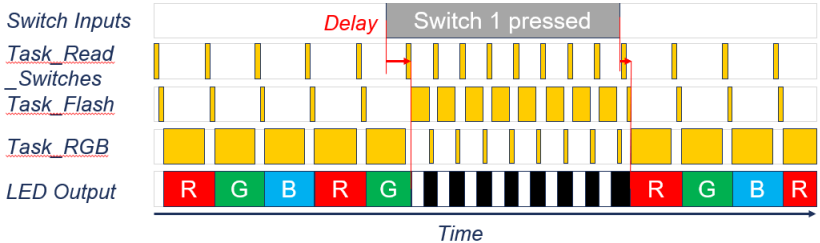
Solución 4 - FSM

```
void Task_Flash(void) {
        if (g_flash_LED == 1) {          // Or
            Control_RGB_LEDs(1, 1, 1);
            Delay(g_w_delay);
            Control_RGB_LEDs(0, 0, 0);
            Delay(g_w_delay);
        }
}
```

```
void Task_Flash_FSM(void) {
    static enum {ST_WHITE, ST_BLACK} next_state = ST_WHITE;

    if (g_flash_LED == 1) {      // Only run task when in flash mode
        switch (next_state) {
            case ST_WHITE:
                Control_RGB_LEDs(1, 1, 1);
                Delay(g_w_delay);
                next_state = ST_BLACK;
                break;
            case ST_BLACK:
                Control_RGB_LEDs(0, 0, 0);
                Delay(g_w_delay);
                next_state = ST_WHITE;
                break;
            default:
                next_state = ST_WHITE;
                break;
        }
    }
}
```

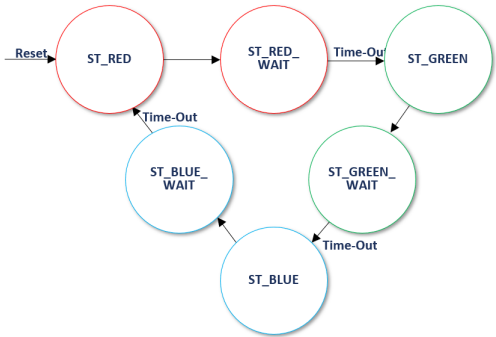
Análisis - Solución 4



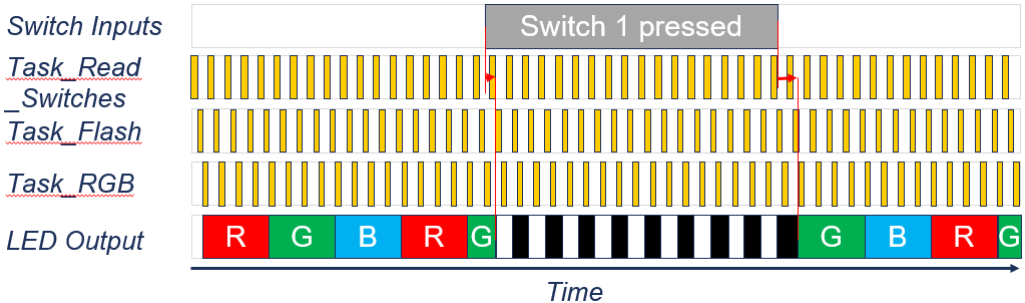
- Máximo retardo después de presionar el LED, aprox 1/3 mas corto.
- **Nota**: se esta realizando polling de nuevo para permitir comparación directa.
- Un poco de sobrecarga con FSM, pero aceptable.
- Software organizado en tareas.

Usar hardware para conservar tiempo de CPU - Solución 5

- La función de retardo usa un loop que retarda el programa.
- Mejoría:
  - Usar timer por hardware para llevar el tiempo.
  - Software puede consultar el timer para determinar si ya paso el tiempo.
  - Periódicamente preguntar por el timer.

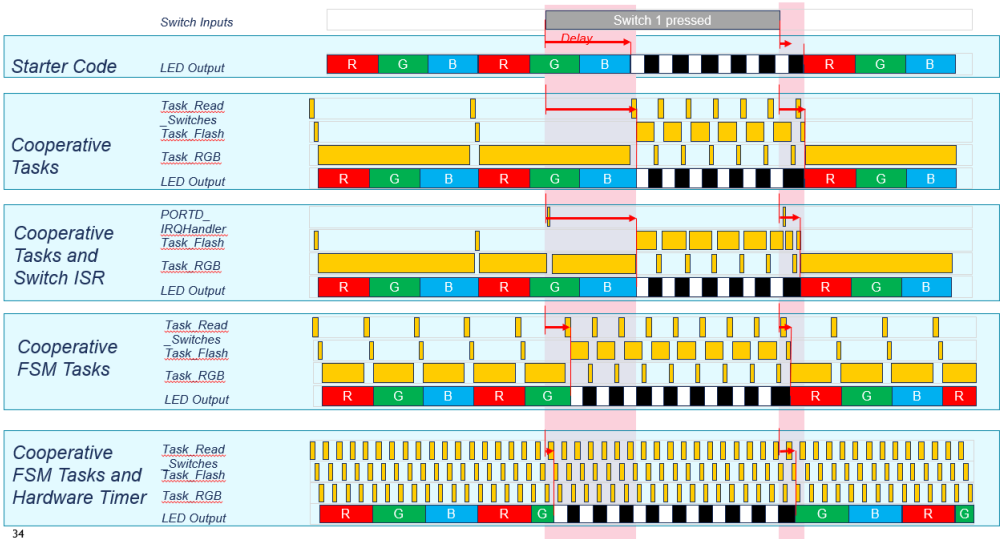


Análisis - Solución 5



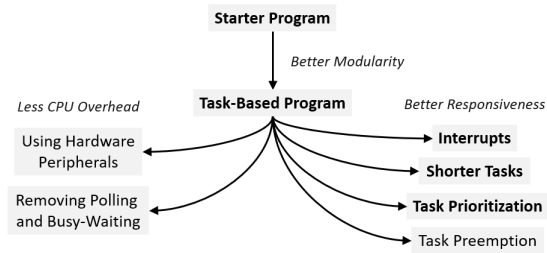
- Todos los estados se ejecutan y retornan rápidamente el control al planificador.
- El planificador da una vuelta más rápido.

Comparación Final



Roadmap Final

- Agregue priorización a las tareas.
- Tareas con multipropósitos mas fácil de implementar.
- Posibilita el control de tareas más fácil.



Laboratorio 2 (15 %)

Implementar el programa de la tarea RGB y pulsadores utilizando Máquinas de estado finito.

- Crear FSM para la secuenciación del led RGB.
- Crear FSM para el parpadeo blanco y negro del led.

CONCURRENCIA BÁSICA

GRACIAS