Punto 6

6. Agregar al ejemplo anterior (5.a) sensado de Push Buttons c/sAPI

a. Mediante compilación condicional, mantener en el archivo TP1.c los fuentes del TP1-1, TP1-2, TP1-3, TP1-4, TP1-5 y TP1-6

b. Identificar funciones de librería sAPI útiles para el sensado de un pulsador

c. Modificar ejemplo de aplicación para soportar todos los LEDs (apagar/encender uno a la vez en secuencia al oprimir un pulsador)

d. En caso que no funcione correctamente el ejemplo de aplicación documentar la forma de la señal digital "pulsador" a sensar: no oprimido // transición a oprimido // mantener oprimido // transición a no oprimido) // ...

El ejemplo provisto responde como debe pero resulta poco práctico ya que tiene una velocidad de muestreo muy rápida. Si bien se tiene un contador para contrarestar esto, por defecto también viene con un valor bastante pequeño por lo que al apretar el pulsador se tiene poco control humano sobre el LED que quedará encendido.

Las funciones que serán útiles para el manejo de los leds y de los botones son los de la biblioteca *sapi_gpio* (figura 1). La inicialización de los pines como salidas (para los LED) y como entradas (para los botones) se realiza en *boardConfig()* como ya se vio en los ejercicios anteriores.

```
85
86 /*==================[external data declaration]==============================*/
87
88 /*==================[external functions declaration]=========================*/
89
90 bool_t gpioConfig( gpioMap_t pin, gpioConfig_t config );
91 bool_t gpioRead( gpioMap_t pin );
92 bool_t gpioWrite( gpioMap_t pin, bool_t value );
93 bool_t gpioToggle( gpioMap_t pin );
94
95 /*==================[cplusplus]==============================================*/
```

Figura 1

La configuración por defecto es TICKRATE_MS 1 y BUTTON_STATUS_MS 10 como se ve en la figura 2. A modo de ejemplo lo que sucedería es que si se aprieta durante medio segundo se estaría registrando unas 500 veces el botón presionado y como se tiene un contador que cada 10ms permite manejar los LEDS se estarían realizando unas 50 acciones en vez de una única acción esperada.

```
85    Code that uses the DEBUG* functions will have their I/O routed to
86    the sAPI DEBUG UART. */
87 DEBUG_PRINT_ENABLE;
88
89 #define TICKRATE_1MS    (1)             /* 1000 ticks per second */
90 #define TICKRATE_10MS   (10)            /* 100 ticks per second */
91 #define TICKRATE_100MS  (100)           /* 10 ticks per second */
92 #define TICKRATE_MS     (TICKRATE_1MS)  /* ¿? ticks per second */
93
94 #define BUTTON_STATUS_10MS  (10)
95 #define BUTTON_STATUS_100MS (50)
96 #define BUTTON_STATUS_500MS (100)
97 #define BUTTON_STATUS_MS    (BUTTON_STATUS_10MS / TICKRATE_MS)
98
```

Figura 2

Para solventar esto se puede aumentar BUTTON_STATUS_MS para que el tiempo antes de procesar los LEDS sea mayor. De todas formas, como no es necesario muestrear cada 1ms se puede aumentar el valor de TICKRATE_MS tal de tener menor cantidad de interrupciones y gastar menor cantidad de recursos. BUTTON_STATUS_MS queda en función de TICKRATE_MS y hará que el contador sea menor si la frecuencia de muestreo es menor.

Algo a notar es que en "sapi_tick.h"se establece que el máximo tickrate es de 50ms (figura 3) por lo que se modificó el ejemplo tal de no usar un *tickrate* 100 ms. Y como el contador para el botón de 100 ms resultaba a veces incómoda se agregó una opción BUTTON_STATUS_200MS ya que usar 500 ms es bastante lento (figura 4).

```
/*==================[external functions definition]==========================*/

// Tick Initialization and rate configuration from 1 to 50 ms
bool_t tickInit( tick_t tickRateMSvalue ) {

   bool_t ret_val = 1;

   if( tickRateMSvalue == 0 ){
      tickPowerSet( OFF );
      ret_val = 0;
   } else{
      if( (tickRateMSvalue >= 1) && (tickRateMSvalue <= 50) ){
```

Figura 3

```
#include "sapi.h"
/* The DEBUG* functions are sAPI debug print functions.
   Code that uses the DEBUG* functions will have their I/O routed to
   the sAPI DEBUG UART. */
DEBUG_PRINT_ENABLE;

#define TICKRATE_1MS    (1)             /* 1000 ticks per second */
#define TICKRATE_10MS   (10)            /* 100 ticks per second */
// #define TICKRATE_100MS   (100)           /* 10 ticks per second */
#define TICKRATE_50MS (50)
#define TICKRATE_MS     (TICKRATE_50MS) /* ¿? ticks per second */

#define BUTTON_STATUS_10MS  (10)
#define BUTTON_STATUS_100MS (100)
#define BUTTON_STATUS_200MS (200)
#define BUTTON_STATUS_500MS (500)
#define BUTTON_STATUS_MS    (BUTTON_STATUS_200MS / TICKRATE_MS)

volatile bool LED_Time_Flag = false;

volatile bool BUTTON_Status_Flag = false;
volatile bool BUTTON_Time_Flag = false;
```

Figura 4

Para que se prenda y se apague el led al oprimir el botón se modificó ligeramente el código tal de que el cambio de índice del LED se haga sólo cuando el led esté apagado. En la próxima pulsación del botón se llamará a gpioToggle con el nuevo índice (donde el led estaría previamente apagado). El código se puede ver en la figura 5.

```
/* ------------- REPETIR POR SIEMPRE ------------- */
while(1) {
    __WFI();

    if (BUTTON_Time_Flag == true) {
        BUTTON_Time_Flag = false;

        if (BUTTON_Status_Flag == true) {
            BUTTON_Status_Flag = false;

            if (BUTTON_Status_Counter == 0) {
                BUTTON_Status_Counter = BUTTON_STATUS_MS;

                gpioToggle(idx);

                if(!gpioRead(idx))
                    ((idx > LEDR) ? idx-- : (idx = LED3));

                debugPrintString( "LED Toggle\n" );
            }
            else
                BUTTON_Status_Counter--;
        }

    }
}
```

Figura 5

Estados del pulsador

- No oprimido: Mientras el botón no esté oprimido sucede siempre que BUTTON_Status_Flag es falso por lo que el programa siempre hará dicha consulta y no avanzará de ese punto.
- Mantener oprimido: Cuando el botón se encuentre oprimido sucederá que BUTTON_Time_Flag es verdadero y BUTTON_Status_Flag también lo es. Para evitar una cantidad incontrolable de acciones se implementa un contador de veces que se verifica el estado del botón en el *main*, el cual está directamente relacionado a la frecuencia de muestreo del pulsador (en este ejemplo sería el *tickrate*).
- Transiciones a oprimido (y no oprimido): Se debe tener en cuenta que los interruptores son dispositivos mecánicos con dinámicas de tiempo mayor a las de un circuito electrónico por lo que en caso de haber ligeros rebotes mecánicos del interruptor (imperceptibles para un humano), estos serían captados perfectamente por el microcontrolador generando pulsaciones indeseadas. Este fenómeno tiene el nombre de *bouncing* y se puede solucionar a través de hardware y o de software. La placa EDU-CIAA posee un circuito RC en cada pulsador para reducir dicho efecto y podría agregarse un algoritmo de *debouncing* si se desea.