Proyecto 2
Laboratorio de Microcontroladores
Temporizador y GPIOs

Marco Antonio Montero Chavarría Carné: A94000
Francisco Molina Carné: B14194

15 de septiembre de 2015

1. Problema

Desarrollar un "semáforo inteligente"simplificado utlizando los leds, la botonera, y los temporizadores del microcontrolador STM32F4,e los cuatro leds disponibles en el microcontrolador los leds LD3 y LD5 representarán el semáforo vehicular, mientras que los LD4 y LD6 serían el semáforo peatonal. El botón B1 (user_button) sería el puerto con el que el usuario solicitará la activación de las luces peatonales. La asignación de funcionalidades de los leds es la siguiente:

- LD3:paso de vehículos.
- LD5:vehículos detenidos.
- LD4:paso de peatones.
- LD6:peatones deternidos

El botón se puede presionar inclusive antes que se acaben los 10s del funcionamiento de LD3, pero debería permanecer encendido hasta que terminen los 10s. Si no se presiona el botón LD3 debería permancer encendido indefinidamente. La temporización de los leds se debe realizar utilizando uno de los temporizadores del microcontrolador. Se recomienda utilizar el Advanced Timer 1 con interrupciones de tiempo habilitados. Como innovación se podrían agregar leds externos para representar adecuadamente con colores verde y rojo ambos semáforos, o agregar una luz amarilla externa con su respectivo control adicional. En cualquiera de los se deberían utilizar resistencias de pull-up para los leds alimentados por el mismo micro-controlador para garantizar la seguridad de la tarjeta.

2. Solución Propuesta

Generar un código para el semáforo utilizando el código visto en clase que contiene métodos de definición para el botón, para el temporizador y para encender los leds.

3. Procedimiento

Se utilizó el archivo glove.c en la carpeta scr_glove de opencoroco. Se removió las partes del código inecesarias para el semáforo y se añadió lo necesario para darle funcionalidad al temporizador, al botón y a los leds.

Para el temporizador se utilizó el siguiente código:

```
void DTC_SVM_tim_init(void)
{
/* Enable TIM1 clock. and Port E clock (for outputs) */
rcc_peripheral_enable_clock(&RCC_APB2ENR, RCC_APB2ENR_TIM1EN);
rcc_peripheral_enable_clock(&RCC_AHB1ENR, RCC_AHB1ENR_IOPEEN);

//Set TIM1 channel (and complementary)
//output to alternate function push-pull'.
//f4 TIM1=> GIO9: CH1, GPIO11: CH2, GPIO13: CH3
//f4 TIM1=> GIO8: CH1N, GPIO10: CH2N, GPIO12: CH3N
gpio_mode_setup(GPIOE, GPIO_MODE_AF,GPIO_PUPD_NONE,GPIO9 | GPIO11 | GPIO13);
gpio_set_af(GPIOE, GPIO_AF1, GPIO9 | GPIO11 | GPIO13);
gpio_mode_setup(GPIOE, GPIO_MODE_AF,GPIO_PUPD_NONE,GPIO8 | GPIO10 | GPIO12);
gpio_set_af(GPIOE, GPIO_AF1, GPIO8 | GPIO10 | GPIO12);

        /* Enable TIM1 commutation interrupt. */
        //nvic_enable_irq(NVIC_TIM1_TRG_COM_TIM11_IRQ); //f4

        /* Reset TIM1 peripheral. */
        timer_reset(TIM1);

        /* Timer global mode:
         * - No divider
         * - Alignment edge
         * - Direction up
         */
        timer_set_mode(TIM1, TIM_CR1_CKD_CK_INT,
//For dead time and filter sampling, not important for now.
TIM_CR1_CMS_CENTER_1,
//TIM_CR1_CMS_EDGE

//TIM_CR1_CMS_CENTER_1
```

```
//TIM_CR1_CMS_CENTER_2

//TIM_CR1_CMS_CENTER_3 la frequencia del pwm se divide a la mitad.
//(frecuencia senoidal)
        TIM_CR1_DIR_UP);

        timer_set_prescaler(TIM1, PRESCALE);
//1 = disabled (max speed)
        timer_set_repetition_counter(TIM1, 0);
//disabled
        timer_enable_preload(TIM1);
        timer_continuous_mode(TIM1);

        /* Period (32kHz). */
        timer_set_period(TIM1, PWM_PERIOD_ARR);
//ARR (value compared against main counter to reload counter
//aka: period of counter)

        /* Configure break and deadtime. */
//timer_set_deadtime(TIM1, deadtime_percentage*pwm_period_ARR);
//timer_set_deadtime(TIM1, 1100.0f*PWM_PERIOD_ARR);
//timer_set_deadtime(TIM1, DEAD_TIME_PERCENTAGE*PWM_PERIOD_ARR);
        timer_set_enabled_off_state_in_idle_mode(TIM1);
        timer_set_enabled_off_state_in_run_mode(TIM1);
        timer_disable_break(TIM1);
        timer_set_break_polarity_high(TIM1);
        timer_disable_break_automatic_output(TIM1);
        timer_set_break_lock(TIM1, TIM_BDTR_LOCK_OFF);

        /* Disable outputs. */
        timer_disable_oc_output(TIM1, TIM_OC1);
        timer_disable_oc_output(TIM1, TIM_OC1N);
        timer_disable_oc_output(TIM1, TIM_OC2);
        timer_disable_oc_output(TIM1, TIM_OC2N);
        timer_disable_oc_output(TIM1, TIM_OC3);
        timer_disable_oc_output(TIM1, TIM_OC3N);

        /* -- OC1 and OC1N configuration -- */
        /* Configure global mode of line 1. */
        timer_enable_oc_preload(TIM1, TIM_OC1);
        timer_set_oc_mode(TIM1, TIM_OC1, TIM_OCM_PWM1);
        /* Configure OC1. */
        timer_set_oc_polarity_high(TIM1, TIM_OC1);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC1);
//When idle (braked) put 0 on output
```

```
        /* Configure OC1N. */
        timer_set_oc_polarity_high(TIM1, TIM_OC1N);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC1N);
        /* Set the capture compare value for OC1. */
timer_set_oc_value(TIM1, TIM_OC1, INIT_DUTY*PWM_PERIOD_ARR);
//initial_duty_cycle*pwm_period_ARR);

        /* -- OC2 and OC2N configuration -- */
        /* Configure global mode of line 2. */
        timer_enable_oc_preload(TIM1, TIM_OC2);
        timer_set_oc_mode(TIM1, TIM_OC2, TIM_OCM_PWM1);
        /* Configure OC2. */
        timer_set_oc_polarity_high(TIM1, TIM_OC2);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC2);
        /* Configure OC2N. */
        timer_set_oc_polarity_high(TIM1, TIM_OC2N);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC2N);
        /* Set the capture compare value for OC2. */
timer_set_oc_value(TIM1, TIM_OC2, INIT_DUTY*PWM_PERIOD_ARR);
//initial_duty_cycle*pwm_period_ARR);

        /* -- OC3 and OC3N configuration -- */
        /* Configure global mode of line 3. */
        timer_enable_oc_preload(TIM1, TIM_OC3);
        timer_set_oc_mode(TIM1, TIM_OC3, TIM_OCM_PWM1);
        /* Configure OC3. */
        timer_set_oc_polarity_high(TIM1, TIM_OC3);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC3);
        /* Configure OC3N. */
        timer_set_oc_polarity_high(TIM1, TIM_OC3N);
        timer_set_oc_idle_state_unset(TIM1, TIM_OC3N);
        /* Set the capture compare value for OC3. */
timer_set_oc_value(TIM1, TIM_OC3, INIT_DUTY*PWM_PERIOD_ARR);
//initial_duty_cycle*pwm_period_ARR);//100);

        /* Reenable outputs. */
        timer_enable_oc_output(TIM1, TIM_OC1);
        timer_enable_oc_output(TIM1, TIM_OC1N);
        timer_enable_oc_output(TIM1, TIM_OC2);
        timer_enable_oc_output(TIM1, TIM_OC2N);
        timer_enable_oc_output(TIM1, TIM_OC3);
        timer_enable_oc_output(TIM1, TIM_OC3N);

        /* ---- */

        /* ARR reload enable. */
```

```
        timer_enable_preload(TIM1);

/*
 * Enable preload of complementary channel configurations and
 * update on COM event.
 */
        //timer_enable_preload_complementry_enable_bits(TIM1);
        timer_disable_preload_complementry_enable_bits(TIM1);

        /* Enable outputs in the break subsystem. */
        timer_enable_break_main_output(TIM1);

/* Generate update event to reload all registers before starting*/
        timer_generate_event(TIM1, TIM_EGR_UG);

        /* Counter enable. */
        timer_enable_counter(TIM1);

        /* Enable commutation interrupt. */
        //timer_enable_irq(TIM1, TIM_DIER_COMIE);

        /*********/
        /*Capture compare interrupt*/

        //enable capture compare interrupt
        timer_enable_update_event(TIM1);

        /* Enable commutation interrupt. */
        //timer_enable_irq(TIM1, TIM_DIER_CC1IE);
//Capture/compare 1 interrupt enable
        /* Enable commutation interrupt. */
        //timer_enable_irq(TIM1, TIM_DIER_CC1IE);
        timer_enable_irq(TIM1, TIM_DIER_UIE);
        nvic_enable_irq(NVIC_TIM1_UP_TIM10_IRQ);
}

void tim1_up_tim10_isr(void)
{
  //Clear the update interrupt flag
  timer_clear_flag(TIM1,TIM_SR_UIF);



static int counter = 0;
counter +=1 ;
if(counter >=1000)
```

```
{
gpio_toggle(GPIOD,GPIO12);
counter=0;
}
}
```

Para el botón se utilizó el siguiente código:

```
void button_init(void)
{
 rcc_peripheral_enable_clock(&RCC_AHB1ENR, RCC_AHB1ENR_IOPAEN);
 gpio_mode_setup(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO0);
 gpio_set_af(GPIOA, GPIO_AF0,GPIO0);
}
```

```
Y dentro del método void tim1_up_tim10_isr(void) \\

int button = 0;

        button = gpio_get(GPIOA,GPIO0);

        if(button > 0)
        {
        gpio_set(GPIOD,GPIO12);
        gpio_clear(GPIOD,GPIO13);
        gpio_set(GPIOD,GPIO14);
        gpio_clear(GPIOD,GPIO15);
        }
        else
        {
        gpio_clear(GPIOD,GPIO12);
        gpio_clear(GPIOD,GPIO13);
        gpio_clear(GPIOD,GPIO14);
        gpio_clear(GPIOD,GPIO15);
```

Para los leds se utilizó el siguiente código:

```
void leds_init(void)
{
  rcc_peripheral_enable_clock(&RCC_AHB1ENR, RCC_AHB1ENR_IOPDEN);
  gpio_mode_setup(GPIOD, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO12);
  gpio_mode_setup(GPIOD, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO13);
  gpio_mode_setup(GPIOD, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO14);
  gpio_mode_setup(GPIOD, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO15);
```

```
}
```

Además para inicializar los métodos, estos deben ser llamados en el system init de la siguiente manera:

```
void system_init(void)
{
  rcc_clock_setup_hse_3v3(&hse_8mhz_3v3[CLOCK_3V3_168MHZ]);
  leds_init();
  button_init();
  DTC_SVM_tim_init();
  tim1_up_tim10_isr();

}
```

4. Conclusiones