**SISTEMAS ELECTRÓNICOS DIGITALES. 2º CONTROL, CURSO 20-21**

**SOLUCIÓN**

**Ejercicio 2** [Ver Anexo]

Debido a una nueva ola de frío, la temperatura de las aulas de examen debe ser controlada de forma continua. Por ello, el director de la ETSIDI le ha encargado el diseño de un sistema de monitorización de la temperatura. Para ello, se le pide aprovechar todas las placas STM32F4Discovery que hay en la ETSIDI, junto con un sensor de temperatura analógico y un módulo WiFi para transmitir la información a un servidor central. El sensor de temperatura analógico está conectado al puerto PA1 y el módulo WiFi al puerto UART1. El funcionamiento debe ser el siguiente:

- La placa estará leyendo continuamente el valor del sensor de temperatura.
- Si el valor de la temperatura cae por debajo de 18 grados, deberá mandar un mensaje por puerto serie al módulo WiFi, consistente en la palabra "ALARMA". Suponga que existe una función "int val_a_grados(int val)" que devuelve la temperatura asociada al valor del ADC.
- Esperará la contestación. Un 'OK' indica que todo está correcto, otro valor indicará error en la conexión y se deberá mandar de nuevo el mensaje de alarma hasta que se reciba un 'OK'.

Notas:

- No tiene que preocuparse por el módulo WiFi. Suponga que está configurado y que lo que manda por puerto serie se manda al servidor correspondiente, y que la respuesta del servidor le llega directamente por puerto serie a continuación.

Se pide:

1. Argumente cual es la mejor forma de implementar la comunicación entre el sensor de temperatura y el micro, y entre el módulo WiFi y el micro: por polling o por interrupciones.

   Entre el sensor de temperatura y el micro: interrupciones
   Entre el módulo WiFi y el micro: polling

2. Indique el número de bits que debe tener el ADC para que la resolución sea del orden de 3mV. Suponer Vref=3,3V.

   11bits.

   ¿Cuál es el mínimo tiempo de muestreo que se debe seleccionar para tener medidas a 17,86kHz? Puede elegir entre 3, 15, 28, 56, 84, 112, 144 ó 480 ciclos. Suponga un reloj de 8Mhz y un prescaler de 4.

   Fm >=17,86kHz => tm<1/17,86kHz
   Tm=(nº bits + x) x 1 ciclo = (11+x) x (1/2MHz) < 1/17,86kHz
   Tm<=(2MHZ/17,96kHz) – 11 = 101

   Por tanto 84.

3. Si el módulo WiFi trabaja a 115200bps, ¿cual es la máxima velocidad de transmisión que se puede seleccionar en el micro?

   115200

4. Escriba el código necesario para realizar las tareas pedidas (no hace falta escribir el código que genera automáticamente el STM32CubeIDE). Use interrupciones para el CAD y polling para la UART (independientemente de lo respondido anteriormente);

   Suponemos:

```
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart1;

--
int16_t adcval;
char enviado[]="ALARMA";
char recibido[2];

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
   if (hadc->Instance == ADC1){
      adcval = HAL_ADC_GetValue(&hadc1);
   }
}

int main(){
   HAL_ADC_Start_IT(&hadc1);

   while (1) {
      if (val_a_grados(adcval)<18)
         while (strcmp(recibido,"OK")!=0)) {
            HAL_UART_Transmit(&huart1,(uint8_t *) enviado, 6, HAL_MAX_DELAY);
            HAL_UART_Receive(&huart1, recibido, 2, HAL_MAX_DELAY);
         }

   }
}
```

5. Si en vez de “ALARMA” se quisiera mandar el valor de temperatura (entero positivo de 2 cifras) por puerto serie, ¿cómo lo haría sin utilizar librerías?

   Forma 1: dígito a dígito

```
int digit[2];
i = 0;
j = 0;
char ch[2];
```

```
while( temp > 0) {
   digit [i++] = temp % 10;
   temp = temp / 10;
}
while(j < i){
   ch[1-j] = digit[j] + '0';
   j++;
}
```

Forma 2: usando UNION

```
union {
  int temp;
  char temp_array[sizeof(int)];
} converter;
```

## Anexo: Instrucciones e identificadores

```
GPIOA, GPIOB, GPIOC, GPIOD, GPIOE
GPIO_PIN_0 .. GPIO_PIN_10
GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_IT_RISING, GPIO_MODE_IT_RISING, GPIO_MODE_IT_FALLING
GPIO_NOPULL, GPIO_PULLDOWN, GPIO_PULLUP

TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3, TIM_CHANNEL_4
TIM1, TIM2, TIM3, TIM4
TIM_COUNTERMODE_UP
TIM_COUNTERMODE_DOWN
TIM_COUNTERMODE_CENTERALIGNED1

EXTI0_IRQn

void HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)

void HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
void HAL_NVIC_SetPriority (IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
void HAL_NVIC_EnableIRQ (IRQn_Type IRQn)

void HAL_DELAY(uint32_t Delay)
void HAL_GetTick()

HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_TIM_Base_Stop(TIM_HandleTypeDef *htim)
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);

__HAL_TIM_SET_COUNTER(TIM_HandleTypeDef * htim, uint32_t value)
__HAL_TIM_SET_PRESCALER(TIM_HandleTypeDef * htim, uint32_t value)
__HAL_TIM_SET_COMPARE(TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t value);

ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart1;

HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size, uint32_t Timeout)
```

```
HAL_StatusTypeDef HAL_UART_Transmit_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
HAL_StatusTypeDef HAL_UART_Receive_IT (UART_HandleTypeDef * huart, uint8_t * pData, uint16_t Size)
void HAL_UART_TxCpltCallback (UART_HandleTypeDef * huart)
void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)

HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef * hadc)
HAL_StatusTypeDef HAL_ADC_Stop (ADC_HandleTypeDef * hadc)
HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef* hadc, uint32_t Timeout);
uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc);

HAL_StatusTypeDef HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)
HAL_StatusTypeDef HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)

int strcmp ( const char * str1, const char * str2 ); \\ Devuelve 0 si son iguales
```

## **Ejercicio 3** (60%)

Se quiere diseñar un módulo PWM denominado PWMGEN en el que sean configurables el periodo y el ciclo de trabajo de la señal de salida. Para ello el módulo debe contar con tres registros: period, duty y count y dos comparadores. El número de bits de estos registros WIDTH debe ser configurable. La señal PWM se generará de la siguiente manera: count debe funcionar como un contador ascendente en el intervalo [0, period], cuando la cuenta llegue al máximo, debe volver a empezar desde 0; la salida PWM debe permanecer a '1' mientras count < duty y a '0' en cualquier otro caso. Existe un caso especial: si period = 0, el contador no debe contar.

Los valores de los registros ***period*** y ***duty*** se pueden modificar de forma síncrona a través de las entradas ***REGSEL*** y ***VALUE***. Cuando se modifica el valor de cualquiera de estos registros, ***count*** se reinicia a 0.

Las entradas y salidas del sistema serán las siguientes:

| | | |
|---|---|---|
| RESET | E | Reset síncrono. Pone a 0 los registros ***count***, ***period*** y ***duty***. |
| CLK_N | E | Reloj activo en el flanco de bajada. |
| REGSEL | E | "01": carga ***VALUE*** en el registro ***period*** y pone ***count*** a 0 en el siguiente flanco activo de ***CLK_N***.<br>"10": carga ***VALUE*** en el registro ***duty*** y pone ***count*** a 0 en el siguiente flanco activo de ***CLK_N***.<br>"00" y "11": ***count*** avanza en cada flanco activo de ***CLK_N*** como se ha descrito más arriba. |
| VALUE[WIDTH-1..0] | E | Entrada paralela del valor a cargar en los registros del módulo. |
| PWM | S | Salida PWM: '1' si ***count*** < ***duty***, '0' en cualquier otro caso. |

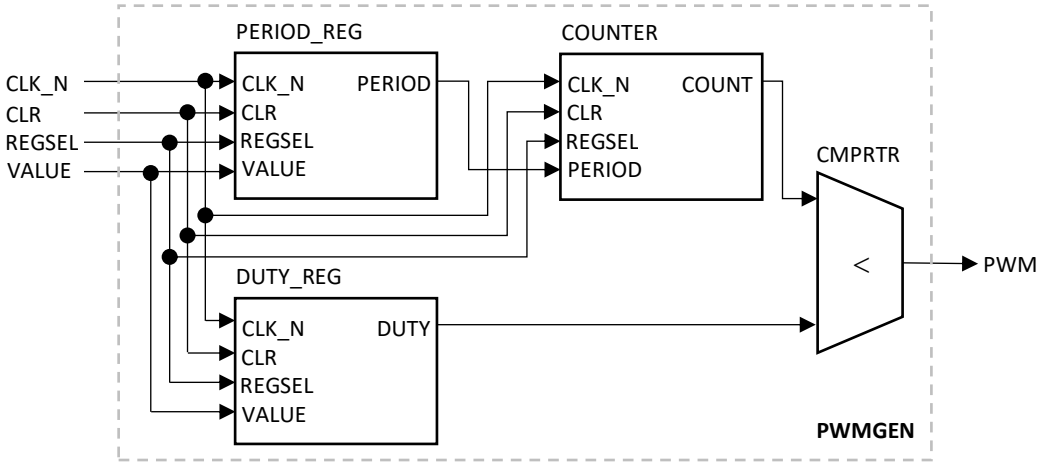
Se pide:

1. Identificar las distintas funcionalidades independientes necesarias para el trabajo del módulo.
2. Dibujar el diagrama de bloques del módulo, explicitando las señales de entrada, salida e internas y la función de cada bloque.
3. Elaborar una descripción en VHDL sintetizable del sistema con la funcionalidad descrita en el enunciado.

SOLUCIÓN

1. Funcionalidades independientes del módulo:
   - Memorización del periodo (*period*).
   - Memorización del ciclo de trabajo (*duty*)
   - Cuenta ascendente (*count*) con periodo ajustable.
   - Comparación cuenta / ciclo de trabajo.

2. Diagrama de bloques:




3. Código VHDL:

***pwmgem.vhd:***

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PWMGEN is
  generic (
    WIDTH : positive := 16
  );
  port (
    CLK_N  : in  std_logic;
    CLR    : in  std_logic;
    REGSEL : in  std_logic_vector(1 downto 0);
    VALUE  : in  std_logic_vector(WIDTH - 1 downto 0);
    PWM    : out std_logic
  );
end PWMGEN;

architecture BEHAVIORAL of PWMGEN is
  signal period : unsigned(VALUE'range);
  signal duty   : unsigned(VALUE'range);
  signal count  : unsigned(VALUE'range);
begin
```

```
  period_reg: process (CLK_N)
  begin
    if falling_edge(CLK_N) then
      if CLR = '1' then
        period <= (others => '0');
      elsif REGSEL = "01" then
        period <= unsigned(VALUE);
      end if;
    end if;
  end process;

  duty_reg: process (CLK_N)
  begin
    if falling_edge(CLK_N) then
      if CLR = '1' then
        duty <= (others => '0');
      elsif REGSEL = "10" then
        duty <= unsigned(VALUE);
      end if;
    end if;
  end process;

  counter: process (CLK_N)
  begin
    if falling_edge(CLK_N) then
      if CLR = '1' then
        count <= (others => '0');
      else
        case REGSEL is
          when "01" | "10" =>
            count <= (others => '0');
          when others =>
            if period /= 0 then
              if count < period then
                count <= count + 1;
              else
                count <= (others => '0');
              end if;
            end if;
        end case;
      end if;
    end if;
  end process;

  cmprtr: PWM <= '1' when count < duty else
                 '0';
end BEHAVIORAL;
```

***pwmgem_tb.vhd:***

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PWMGEN_TB is
end PWMGEN_TB;

architecture TEST of PWMGEN_TB is

  component PWMGEN is
    generic (
      WIDTH : positive
    );
    port (
      CLK_N  : in  std_logic;
      CLR    : in  std_logic;
      REGSEL : in  std_logic_vector(1 downto 0);
      VALUE  : in  std_logic_vector(WIDTH - 1 downto 0);
```

```
      PWM    : out std_logic
    );
  end component;

  constant WIDTH : positive := 4;
  constant CLK_PERIOD : time := 10 ns;

  -- Inputs
  signal clk_n  : std_logic;
  signal clr    : std_logic;
  signal regsel : std_logic_vector(1 downto 0);
  signal value  : std_logic_vector(WIDTH - 1 downto 0);

  -- Outputs
  signal pwm    : std_logic;

begin
  dut: PWMGEN
    generic map (
      WIDTH => WIDTH
    )
    port map (
      CLK_N  => clk_n,
      CLR    => clr,
      REGSEL => regsel,
      VALUE  => value,
      PWM    => pwm
  );

  clk_gnrtr: process
  begin
    clk_n <= '0';
    wait for 0.5 * CLK_PERIOD;
    clk_n <= '1';
    wait for 0.5 * CLK_PERIOD;
  end process;

  stim_gnrtr: process
  begin
    -- Reset
    clr <= '0' after 0.25 * CLK_PERIOD;
    wait until clk_n = '1';
    clr <= '1';
    regsel <= "00";
    wait until clk_n = '1';
    clr <= '0';

    -- Don't count if period = 0
    for i in 1 to 2 loop
      wait until clk_n = '1';
    end loop;

    -- Load period
    regsel <= "01";
    value  <= std_logic_vector(to_unsigned(9, VALUE'length));
    wait until clk_n = '1';
    regsel <= "00";

    -- No pulse if duty = 0
    for i in 1 to 12 loop
      wait until clk_n = '0';
    end loop;

   -- Load duty
    wait until clk_n = '1';
    regsel <= "10";
    value  <= std_logic_vector(to_unsigned(3, VALUE'length));
```

```
    wait until clk_n = '1';
    regsel <= "00";

    -- 3 cycle pulse
    for i in 1 to 12 loop
      wait until clk_n = '0';
    end loop;

    wait for 2 * CLK_PERIOD;
    assert false
      report "[SUCCESS]: simulation finished."
      severity failure;
  end process;
end TEST;
```