Nombre:	**Nº Matrícula:**

SISTEMAS ELECTRÓNICOS DIGITALES. 2º CONTROL, CURSO 19-20, HOJA 1 de 2

NOTAS IMPORTANTES:

- El único resultado válido será la que se indique en la casilla de solución.
- La resolución de las cuestiones sin una mínima explicación escrita de los pasos seguidos para la obtención de los resultados producirá penalización en la calificación, considerándose no valida la respuesta.
- La fecha de publicación de las calificaciones y de la revisión estará dentro de los márgenes establecidos según normativa.
- La puntuación del ejercicio se indica en cada parte.
- **CADA EJERCICIO SE CONTESTA EN UNA HOJA SEPARADA**

Ejercicio 1.

En determinada aplicación se usa un micrófono que produce un flujo de bits codificado en PDM (*Pulse Density Modulation*). En esta modulación, el número de pulsos (1's) por unidad de tiempo es proporcional a la amplitud de la señal digitalizada. Se quiere diseñar un sistema que convierta este flujo de bits en otro de bytes muestreando el flujo de bits y contando cuántos 1's hay en 256 muestras. La suma debe saturar a 255 (el valor máximo de la suma debe ser 255). Una vez finalizado el recuento, el resultado debe pasarse a un registro de salida de forma que el valor de la salida permanezca constante durante el recuento del siguiente valor. A modo de ejemplo, a continuación, se muestra cómo sería una conversión de PDM a 2 bits (valores de salida entre 0 y 3):

0	1	0	1	0	1	1	1	0	0	1	0	1	1	1	1	…	⇒	2	3	1	3	…

Las entradas y salidas del sistema serán las siguientes:

RESET_N	E	Reset asíncrono negado. Reinicia el muestreo y carga el registro de salida a 127.
CLK	E	Reloj de 2,4 MHz activo en el flanco de subida.
CE_N	E	Habilitación del reloj. El circuito de conversión sólo trabaja cuando esta entrada está a '0'.
SIN	E	Entrada serie por la que se recibe el flujo de datos codificado en PDM. Muestreada en los flancos activos del reloj.
POUT[7..0]	S	Salida paralela con el resultado de la conversión. Actualizada cada 255 muestreas.
READY	S	Se pone a '1' durante un ciclo de reloj cada vez que se actualiza el registro de salida. El resto del tiempo permanece a '0'.

Se pide:

1. Elaborar una descripción en VHDL sintetizable del sistema con la funcionalidad descrita en el enunciado.

SOLUCIÓN

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

entity pdm2par is
  port (
    rst_n : in  std_logic;
    clk   : in  std_logic;
    ce_n  : in  std_logic;
    sin   : in  std_logic;
    pout  : out std_logic_vector(7 downto 0);
    rdy   : out std_logic
  );
end pdm2par;

architecture behavioral of pdm2par is
begin
  process (rst_n, clk)
    subtype count_t is integer range 0 to 255;
    variable rem_smpls : count_t;
    variable ones      : count_t;
  begin
    if rst_n = '0' then
      pout      <= std_logic_vector(to_unsigned(127, 8));
      rdy       <= '0';
      ones      := 0;
      rem_smpls := 255;
    elsif rising_edge(clk) then
      if ce_n = '0' then
        rdy <= '0';
        if sin = '1' and ones < 255 then
          ones := ones + 1;
        end if;
        if rem_smpls = 0 then
          pout      <= std_logic_vector(to_unsigned(ones, 8));
          rdy       <= '1';
          ones      := 0;
          rem_smpls := 255;
        else
          rem_smpls := rem_smpls - 1;
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

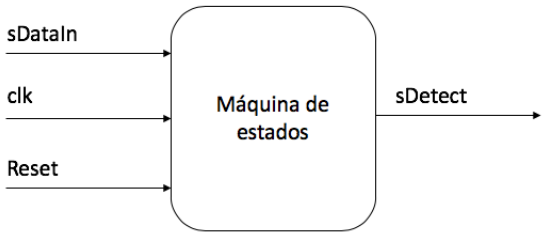
Ejercicio 2.

Se desea diseñar un circuito que acepte una entrada de datos serie y presente una salida que se activará (tomará el valor lógico '1') cuando en los instantes de muestreo aparezca la secuencia "1011".

Para ello se pide realizar una entidad que será una máquina de estados con la siguiente interfaz:

- Entradas: Una entrada de datos (sDataIn) más una señal de reloj (clk) y otra de inicialización (Reset a nivel alto asíncrono), asociadas a la parte secuencial.
- Salidas: Una sDetect, encargada de activarse cuando se ha detectado la secuencia.

La detección se realiza deforma continua: la secuencia '1011' activará la señal sDetect dos veces (después del segundo y tercer unos).



Complete el código siguiente en otra hoja.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SecMoore IS
    PORT ( sDataIn: ...
           CLK: ...
           sResetH: ...
           sDetect: ...
         );
END SecMoore;
ARCHITECTURE SecMooreArch OF SecMoore IS
  TYPE TipoEstados IS ...
  SIGNAL tEstadoActual, tEstadoSiguiente: TipoEstados;

  BEGIN
  ...

END SecMooreArch;
```

SOLUCIÓN

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SecMoore IS
    PORT ( sDataIn: IN std_logic; --Señal de entrada.
           CLK: IN std_logic; --Señal de reloj.
           sResetH: IN std_logic; --Señal de inicialización
           sDetect: OUT std_logic --Salida
         );
```

```
END SecMoore;

ARCHITECTURE SecMooreArch OF SecMoore IS

  TYPE TipoEstados IS (E0, E1, E10, E101, E1011); --Declaración del tipo asociado a los estados.
  SIGNAL tEstadoActual, tEstadoSiguiente: TipoEstados; --Señales auxiliares para la codificación del
estado actual y siguiente.

  BEGIN

  -- Proceso dedicado a la lógica de estado:
  LOGICA_ESTADO: PROCESS(tEstadoActual, sDataIn)
  BEGIN
  tEstadoSiguiente <= tEstadoActual;
    CASE (tEstadoActual) IS
      WHEN E0 =>
          IF (sDataIn = '0') THEN tEstadoSiguiente <= E0;
          ELSE tEstadoSiguiente <= E1;
          END IF;
      WHEN E1 =>
          IF (sDataIn = '0') THEN tEstadoSiguiente <= E10;
          ELSE tEstadoSiguiente <= E1;
          END IF;
      WHEN E10 =>
          IF (sDataIn = '0') THEN tEstadoSiguiente <= E0;
          ELSE tEstadoSiguiente <= E101;
          END IF;
      WHEN E101 =>
          IF (sDataIn = '0') THEN tEstadoSiguiente <= E10;
          ELSE tEstadoSiguiente <= E1011;
          END IF;
      WHEN E1011 =>
          IF (sDataIn = '0') THEN tEstadoSiguiente <= E10;
          ELSE tEstadoSiguiente <= E1;
          END IF;

    END CASE;
  END PROCESS LOGICA_ESTADO;

  -- Proceso dedicado a la Memoria de Estado
  MEM_ESTADO: PROCESS(CLK, sResetH)
  BEGIN
    --Inicialización con RESET_H
    IF (sResetH ='1') THEN tEstadoActual <= E0;
    ELSIF (rising_edge(CLK)) THEN tEstadoActual <= tEstadoSiguiente;
    END IF;
  END PROCESS MEM_ESTADO;

  --Zona concurrente dedicada a modelar la lógica de salida.
  sDetect <= '1' WHEN (tEstadoActual = E1011)
  ELSE '0';

END SecMooreArch;
```

Ejercicio 5.

Disponemos de un STM32F4 para procesar la señal de un sensor de iluminación. El sensor es fuertemente no lineal, de forma que podemos establecer de forma aproximada que en el margen 1 – 100 lux la sensibilidad del dispositivo polarizado es de 2 mV/lux y en el margen 100 – 1000 lux es de 3,2 mV/lux. Si se desea que la precisión en la medida sea mejor que 0,5 lux para cualquier entrada, hallar el número de bits necesarios en cada margen en que se ha dividido la entrada y evaluar si el convertidor A/D del micro puede realizar la conversión cumpliendo las especificaciones. En caso afirmativo evalúe la mejora producida (en la resolución de la iluminación) respecto de lo especificado.

Datos: Vref = 3,6 V.

SOLUCIÓN

En el margen 1-100 lux, un incremento de 0,5 lux implica un incremento de la tensión de salida del transductor de:

$$\Delta V = 0{,}002\ \frac{V}{lux} \cdot 0{,}5\ lux = 1mV$$

El convertidor A/D ha de ser sensible a este incremento de tensión de entrada, por lo que el valor del LSB deberá ser equivalente a:

$$LSB = \frac{V_{FS}}{2^N} = \Delta V = 1mV$$

Si el rango de medida de iluminación es hasta 1000 lux, la máxima tensión que entrega el transductor, que será igual o menos que la tensión de referencia del convertidor, resulta:

$$V_{max} = 0{,}002\ \frac{V}{lux} \cdot 1000\ lux = 2V < V_{ref}$$

Con lo que es posible utilizar el convertidor con dicha tensión de referencia.
Por tanto, el número de bits necesario para mantener esta precisión en la medida resulta ser:

$$\Delta V \geq \frac{V_{ref}}{2^N} \Rightarrow 2^N \geq \frac{V_{ref}}{\Delta V} \Rightarrow$$

$$N \geq \frac{\log \frac{V_{ref}}{\Delta V}}{\log 2} = 11{,}8$$

Por lo tanto, N será 12 bits. Es posible utilizar el convertidor ya que éste es de 12 bits.

Si repetimos el proceso para el margen 100-1000 lux, obtenemos:

$$\Delta V = 0{,}0032\ \frac{V}{lux} \cdot 0{,}5\ lux = 1'6mV = LSB$$

$$V_{max} = 0{,}0032\ \frac{V}{lux} \cdot 1000\ lux = 3'2V < V_{ref}$$

Con lo que es posible utilizar el convertidor con dicha tensión de referencia.
El número de bits necesario para mantener esta precisión en la medida resulta ser en este caso:

$$\Delta V \geq \frac{V_{ref}}{2^N} \Rightarrow 2^N \geq \frac{V_{ref}}{\Delta V} \Rightarrow$$

$$N \geq \frac{\log \frac{V_{ref}}{\Delta V}}{\log 2} = 11{,}13$$

Por lo tanto, N será también 12 bits. Por tanto es posible utilizar el convertidor.

La mejora se produce en la resolución en la medida:

En el margen 1-100 lux: $\Delta V = \frac{V_{ref}}{2^N} = 0'88mV \Rightarrow \Delta G = \frac{\Delta V}{S} = \frac{0'88mV}{0{,}002\,\frac{V}{lux}} = 0'44\ lux$

En el margen 100-1000 lux: $\Delta V = \frac{V_{ref}}{2^N} = 0'88mV \Rightarrow \Delta G = \frac{\Delta V}{S} = \frac{0'88mV}{0{,}0032\,\frac{V}{lux}} = 0'27\ lux$

La resolución en el margen 100-1000 lux es ostensiblemente mejor.

Ejercicio 6.

Se pretende desarrollar un sistema de control de las luces de un árbol de Navidad con el STM32F411. Además del micro STM32F411, el sistema se compone de un pulsador conectado a la entrada PA0 (emite una señal digital, '0' si no se pulsa, '1' si se pulsa) y una tira de LEDs conectados a la salida PD15. Suponga que toda la tira se comporta como un solo LED. La idea es que la tira de LEDs tenga varios modos de funcionamiento, y que vayan cambiando según presionemos el pulsador. Los modos son: encendido, parpadeando modo A (1s encendidos y 4s apagados), parpadeando modo B (2s encendidos y 3s apagados), y apagados.

Suponer que el Timer TIM4, Channel 4, está configurado para controlar el pin PD15. Suponga que la señal de reloj que llega a los temporizadores (APB1/APB2) es de 16MHz.

A. Complete el siguiente código para realizar la tarea pedida, leyendo el pulsador por nivel. Utilice el temporizador TIM4 para realizar la tarea.
B. Modifique el programa anterior para utilizar interrupciones con el pulsador.
C. Desde el punto de vista teórico, comente el tipo de temporizador utilizado y razónelo.

En ambos casos, suponga que no hay rebotes.

Complete el código del main() y las instrucciones en negrita (marcadas con ->).

```
TIM_HandleTypeDef htim4;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM4_Init(void);


int main(void)
{
  HAL_Init();
  SystemClock_Config();

  ...

  while (1)
  {

  ...

  }
}
```

```
static void MX_TIM4_Init(void)
{
  htim4.Instance = TIM4;
  ->htim4.Init.Prescaler =       ;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  ->htim4.Init.Period =          ;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  ->GPIO_InitStruct.Mode =              ;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

Anexo: Instrucciones e identificadores

```
void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
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_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
void HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
void HAL_TIM_OC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
void HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)
void HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
void HAL_TIM_PWM_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)
__HAL_TIM_SET_COMPARE(TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t value);
void HAL_TIM_IC_Start (TIM_HandleTypeDef *htim, uint32_t Channel)
void HAL_TIM_IC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)
void HAL_TIM_IC_Start_DMA (TIM_HandleTypeDef * htim, uint32_t Channel, uint32_t * pData, uint16_t Length)

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)

GPIOA, GPIOB, GPIOC, GPIOD, GPIOE
GPIO_PIN_0 .. GPIO_PIN_10
TIM_CHANNEL_1, TIM_CHANNEL_2, TIM_CHANNEL_3. TIM_CHANNEL_4
TIM1, TIM2, TIM3, TIM4
EXTI0_IRQn
GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_IT_RISING, GPIO_MODE_IT_RISING, GPIO_MODE_IT_FALLING
```

SOLUCIÓN

A)

```
TIM_HandleTypeDef htim4;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM4_Init(void);

int count=0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM4_Init();

  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);

  while (1)
  {
    if (HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)==1){
     count++;
     if (count==4) count=0;
     switch (count)
     {
       case 0: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 50000); break;
       case 1: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 10000); break;
       case 2: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 20000); break;
       case 3: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 0); break;
     }
    }
  }
}
```

```
static void MX_TIM4_Init(void)
{


  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 1600;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 50000;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;


}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
```

B)

Añadir:

```
volatile int8_t pressed=0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
  if (GPIO_Pin == GPIO_PIN_0){
    pressed=1;
  }
}
```

Y en el main():

```
  while (1)
  {
    if (pressed ==1){
     count++;
     if (count==4) count=0;
     switch (count)
     {
       case 0: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 50000); break;
       case 1: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 10000); break;
       case 2: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 20000); break;
       case 3: __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 0); break;
     }
     pressed=0;
    }
  }
```

```
static void MX_GPIO_Init(void)
{

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

}
```

C) Es un temporizador en modo PWM. Las características se explican en la teoría.