Ecodiseño de dispositivos electrónicos

UNIDAD 9: Sistemas de microcontroladores, parte 2

Autor: Andrej Sarjaš

Resumen del capítulo:
- Sistemas en tiempo real
- Componentes de sistemas en tiempo real
- Diseñando programas en sistemas en tiempo real

Funded by the Erasmus+ Programme of the European Union

Ecoinnovation Skills for European Designers, Project number: 562573-EPP-1-2015-1-SI-EPPKA2-SSA. El presente proyecto ha sido financiado con el apoyo de la Unión Europea. Esta publicación (comunicación) es responsabilidad exclusiva de su autor. La Comisión no es responsable del uso que pueda hacerse de la información aquí difundida

9.1. Uso de la tarjeta microcontroladora SMT32F4discovery

Placa de desarrollo STM32F4 Discovery es una placa de microcontrolador económica y de bajo precio, adecuada para el desarrollo rápido de aplicaciones integradas y pruebas del potente microcontrolador ARM STM 32F407VG con unidad de FPU incorporada (FPU-unidad de coma flotante, una unidad para computación con números de coma flotante). La placa de desarrollo contiene el microcontrolador ARM STM 32F407VG, programa FTAG ST-link V2. Acelerómetro SB, códec de audio, micrófono, 4 diodos LED de control, interruptor y conector micro-U. La imagen 1 presenta la placa de desarrollo STM32F4 Discovery.

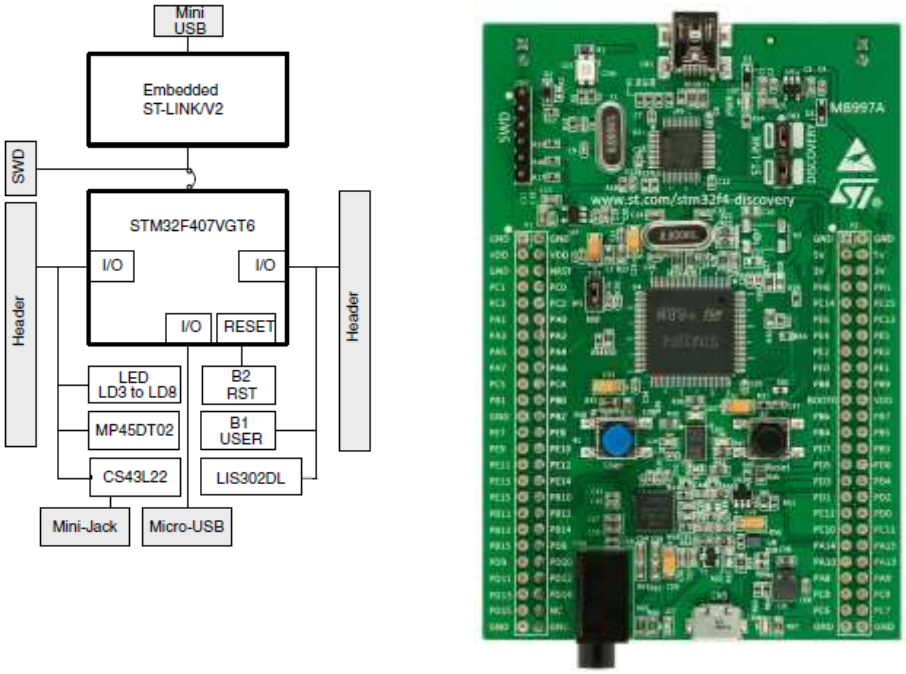

IMAGEN 1: JUNTA DE DESARROLLO STM32F4DISOCVERY

El microcontrolador STM32F4 está basado en el núcleo Corex-M4 y puedefuncionar a una frecuencia máxima de 168MHz. Core Cortex-M4 ofrece una unidad de precoesador de 32 bits y la FPU mecánica esá diseñada para el procesamiento digital de señal y tiene funcionalidades de procesador DSP. En la frecuencia máxima, alcanza 210 DMIPS (instrucción de millón de Dyrestone por segundo). El microcontrolador también contiene una amplia gama de dispositivos periféricos:

- 2 x USB OTG (en camino).
- Bucle cerrado de audio (bucle de bloque de fase PLL).
- Contiene 15 interfases de comunicación (4 x UART, 2 x USART, 3 x SPI, 3 x I2C, 2x CAN, SDIO).

- Contiene 2x 12-bit DAC en 3 x 12-bit ADC velocidad 2.4Msps.
- Tiene 17 temporizadores. Todos los temporizadores son de 16 bits, excepto dos que son interfases de 32 bits par sistemas de memoria adicionales, SRAM, NAND, etc.

El controlador tiene una memoria SRAM de 192kByte integrada y memoria FLASH de 1Mbyte. Ara programar el controlador, usaremos la herramienta profesional KEIL uVisión que se usa en muchos centros de desarrollo. Para la programación, utilizaremos el lenguaje de programa C.

9.2. Setting discrete inputs and outputs -GPIO

GPIO es una abreviatura para pines de entradas / salidas de propósito general. Los pines bajo este acrónimo presentan entradas o salidas al microcontrolador. En los sistemas de microcontroladores, conocemos múltiples tipos de entradas / salidas que generalmente se dividen en discretas y analógicas. Las entradas / salidas discretas presentan niveles lógicos (0-1) donde entradas / salidas analógicas presentan una resolución de señal multinivel. Las señales analógicas están limitadas por la tensión de alimentación delos microcontroladores, precisamente por la fuente de alimentación de la unidad AD dentro del chip. Las entradas analógicas genrealmente se etiquetan como señales AD (analógicas a digitales) y las salidas analógicas como señales DA (de digital a analógico). Los pines del microcontrolador están divididos en grupos que son puertos con nombre y están etiquetados con PA, PB, PC, PD, PE, donde cada grupo captura pinens físicos de 0-15 (por ejemplo, PAO-PA15, etc). La configuración del módulo GPIO y todas las funcionalidades conrrespondientes deben incluirse en la biblioteca **stm32fxxx_gpio.c.**

- **Configuración de salida GPIO discreta.**

Determinación discreta de entrada:

```
GPIO_InitTypeDef GPIO_InitStructure; //Structure

//GPIO out
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13|
                              GPIO_Pin_14 | GPIO_Pin_15;// Configuration of pins 12-15
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;           // Output definition
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       // Module speed (2/10/50 MHz)
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;   //  push / pull (opposed to open drain)
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pulldown resistor is not
active
GPIO_Init(GPIOD, &GPIO_InitStructure);                  // Port D setting
```

Significado de la etiqueta:

```
GPIO_Pin_xx     -  Sequential pin number 1-15 on port -(A,B,C,D,E)
GPIOx           - Port name A,B,C,D…
```

- **Configuración de entrada GPIO discreta:**

Determinación de entrada discreta:

```
  GPIO_InitTypeDef GPIO_InitStructure; //Structure
  //GPIO IN
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;         //  Configuration of pin 0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;      // GPIO as input
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Speed of GPIO module(2/10/50 MHz)
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;    // push/pull (opposed to open drain)
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pullup resistor is not
active
  GPIO_Init(GPIOA, &GPIO_InitStructure);            // Port A setting
```

- **Ejemplo de entrada / salida discreta**.

En la placa del controlador, tenemos cuatro diodos que están en el puerto D y los pines 12-15. También tenemos un botón de usuario que está conectado al puerto A y al pin 0. La imagen 2 presenta el esquema de cableado de los diodos LED y un botón.

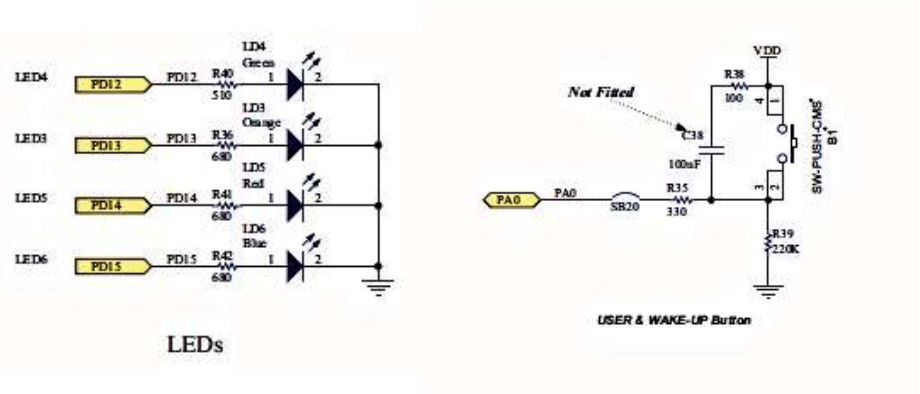

IMAGEN 2: ESQUEMA DE CABLEADO DE DIODOS Y BOTONES.

Ejemplo de programa:

```
//MACRO
#define LED1           GPIO_Pin_12
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)

void RCC_Configuration(void);
void GPIO_Configuration(void);
int  button=0;


int main(void)
{

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();

    //Programm
  GPIO_SetBits(GPIOD, GPIO_Pin_12);
  GPIO_SetBits(GPIOD, GPIO_Pin_13);
  GPIO_ResetBits(GPIOD, GPIO_Pin_14);
```

```
  GPIO_ResetBits(GPIOD, GPIO_Pin_15);

    //Infinite loop
    while(1)
    {

           button = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0); //Read input bit(Button
value)

                    if(button == 1)
                    {
                            GPIO_SetBits(GPIOD, GPIO_Pin_14);
                            GPIO_SetBits(GPIOD, GPIO_Pin_15);
                            LEDon(LED1); //MACRO function
                     }else
                     {
                            GPIO ResetBits(GPIOD, GPIO Pin 14);
                            GPIO_ResetBits(GPIOD, GPIO_Pin_15);
                            LEDoff(LED1); //MACRO function
                     }

                }//Infinite loop


}//End MAIN
```

```
//Function RCC_Configuration
void RCC_Configuration(void)
{
  /* GPIO clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
}

//Function GPIO_Configuration
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;


    //LED OUT
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
                                  GPIO_Pin_14 | GPIO_Pin_15; // configurate all I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;            // GPIO as output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;        // Speed of GPIO module
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // push / pull(opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pullup resistor is not
active
    GPIO_Init(GPIOD, &GPIO_InitStructure);               // Port D setting

     //BUTTON
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;        // PINA 0 configuration
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;     // GPIO as input
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //Speed of GPIO module (2/10/50 MH
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;   //push/pull (opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  // pullup / pullup resistor is not
active
    GPIO_Init(GPIOA, &GPIO_InitStructure);           // Port A setting

}
```

9.3. Comunicación UART

La comunicación UART es más corta para el recetor-transmisor asíncromo universal. Se utiliza con mayor frecuencia como comunicación RS232. Para RS232 necesitamos un circuito adaptativo periférico que convierta los niveles de señal para la

lógica de nivel TTL con un chip (MAX232) o use un módulo USB-Com-Port que emule la conexión estándar RS232 a través de la interfaz USB ((FTDi, CP2102, etc.). Esto se puede ver en la imagen 3. La placa de desarrollo contiene cuatro módulos internos de USART que se encuentran en los siguientes puertos:

UART 1 - Rx: PA10 Tx: PA9
UART 2 - Rx: PA3 Tx: PA2
UART 3 - Rx: PB11 Tx: PB10
UART 4 - Rx: PA1 Tx: PA0

- **Ejemplo de uso de USART2:**

En el proyecto, incluimos la biblioteca **stm32fxxx_usart.c.** Conectamos PIN PA2 con pin RX en el módulo CP2012 Y PIN PA2 con módulo de pin TX CP2012.

IMAGEN 3: MODULO CP2012

```
  //MACRO

#define LED1 GPIO_Pin_15
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)


/******Functions*********************************/
void Delay(int ms);
void RCC_Configuration(void);
void GPIO_Configuration(void);
void USART2_Configuration(void);
void NVIC_Configuration(void); //Interrupt controller
int  button=0;
char buffer[200];
int j,i=0;



/*******************************************************************************
* Function Name  : Main
* Description    : Main function
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
int main(void)
{

  // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  NVIC_Configuration();
```

```
    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON

    USART_Send_Str(USART2, "Hello from StmF4Discovery!\n\r" );


    //Infinite loop
    while(1)
    {
                if(Button==1)
                 {
                     LEDon(LED1);
                 }else
                 {
                      LEDoff(LED1);
                 }

     }//Infinite loop


}//End MAIN

/*******************************************************************************
* Function Name  : RCC_configuration
* Description    : Clock enable
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RCC_Configuration(void)
{
  /* --------------------------- System Clocks Configuration -----------------*/


   /* USART2 clock enable */
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
   /* GPIO clock enable */
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
}


/*******************************************************************************
* Function Name  : GPIO_Configuration
* Description    : Configures the different GPIO ports.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;


    //LED OUT
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
                                  GPIO_Pin_14 | GPIO_Pin_15; // Configurate all I/O pins for LED
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;          // GPIO as output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;    // HitrSpeed of GPIO module (2/10/50 MHz)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;       // push / pull (opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;      // pullup / pullup resistor is not active
    GPIO_Init(GPIOD, &GPIO_InitStructure);                 // Port D setting

  //BUTTON
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;          //  Configuration of PIN 0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;           //  GPIO as input
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;     //  Speed of GPIO module(2/10/50 MHz)
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;     //  push / pull (opposed to open drain)
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;     // pullup / pullup resistor is not active
  GPIO_Init(GPIOA, &GPIO_InitStructure);                 // Port A setting

     // UART2 GPIO
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2| GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Connect USART pins to AF */
```

```
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2); // USART2_TX
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2); // USART2_RX


}


/*******************************************************************************
* Function Name  : USART_Configuration
* Description    : Configures the USAR module.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USART2_Configuration(void)
{
    USART_InitTypeDef USART_InitStructure;


  //USART2 configuration
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART2, &USART_InitStructure);
  USART_Cmd(USART2, ENABLE);
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE); // Receive Interrupt enable

}
```

9.4. Conversión AD

La placa de desarrollo permite 16 entradas analógicas externas y dos internas (sensor de temperatura y voltaje de la batería). El microcontrolador contiene 3 convectores AD separados, donde cada AD1-3 tiene una resolución ajustable por separado de 6, 8, 10 a 12 bits (el valor predeterminado es 12 bits). Tres convectores AD se pueden configurar en diferentes unidades GPlo y pines con la biblioteca adicional **stm32fxxx_adc.c**.

Nombres de puestos y pines que están en convertidores AD:

Channel APB	ADC1 2	ADC2 2	ADC3 2
ADC Channel 0	PA0	PA0	PA0
ADC Channel 1	PA1	PA1	PA1
ADC Channel 2	PA2	PA2	PA2
ADC Channel 3	PA3	PA3	PA3
ADC Channel 4	PA4	PA4	PF6
ADC Channel 5	PA5	PA5	PF7
ADC Channel 6	PA6	PA6	PF8
ADC Channel 7	PA7	PA7	PF9
ADC Channel 8	PB0	PB0	PF10
ADC Channel 9	PB1	PB1	PF3

ADC Channel 10	PC0	PC0	PC0
ADC Channel 11	PC1	PC1	PC1
ADC Channel 12	PC2	PC2	PC2
ADC Channel 13	PC3	PC3	PC3
ADC Channel 14	PC4	PC4	PF4
ADC Channel 15	PC5	PC5	PF5

- **Ejemplo de configuración del convertidor AD**.

Lectura de valores analógicos en los canales 1, 4 y 5 con la ayuda de dos convertidores AS separados (AD1 y AD2).

```
int main(void)
{
   uint16_t read_AD[3];

  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  ADC_init();

  USART_Send_Str(USART2,"ADC test!\n\r");


    while(1)
     {

          read_AD[0] = Read_ADC(ADC1,1); //Read ADC value with ADC1 on channel 1 - PA1
          read_AD[1] = Read_ADC(ADC2,4); //Read ADC value with ADC2 on channel 4 - PA4
          read_AD[2] = Read_ADC(ADC2,5); //Read ADC value with ADC2 on channel 5 - PA5

      //Convert number to string
      sprintf(buffer,"PA1: %d   PA4: %d  PA5: %d
\n\r",read_AD[0],read_AD[1],read_AD[2]);

          USART_Send_Str(USART2,buffer);

          Delay(300);
          GPIO_ToggleBits(GPIOD, GPIO_Pin_12);

     }//Infinite loop


}//End MAIN

// ADC configuration
void ADC_init()
{
   GPIO_InitTypeDef        GPIO_InitStructure;
   ADC_InitTypeDef         ADC_InitStructure;
   ADC_CommonInitTypeDef   ADC_CommonInitStructure;

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1  | RCC_APB2Periph_ADC2 ,  ENABLE);

    //Setup GIPO pins
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);


    //Common settings of ADC
  ADC_CommonInitStructure.ADC_DMAAccessMode    = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_Mode             = ADC_Mode_Independent ;
```

```
  ADC_CommonInitStructure.ADC_Prescaler        = ADC_Prescaler_Div4;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_10Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);


    //Common settings of ADC
  ADC_InitStructure.ADC_Resolution             = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode           = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode     = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge   = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign              = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion        = 1;

    //Connect GPIO to the ADC
  ADC_Init(ADC1, &ADC_InitStructure);
  ADC_Init(ADC2, &ADC_InitStructure);



   //Enable ADC1 and ADC2
  ADC_Cmd(ADC1, ENABLE);
  ADC_Cmd(ADC2, ENABLE);

}

//READ ADC function
uint16_t Read_ADC(ADC_TypeDef* ADCx, uint8_t channel)
{
        uint16_t timeout = 0xFFF;

            ADC_RegularChannelConfig(ADCx, channel, 1, ADC_SampleTime_28Cycles);

            /* Start software conversion */
            ADCx->CR2 |= (uint32_t)ADC_CR2_SWSTART;

            /* Wait till done */
            while (!(ADCx->SR & ADC_SR_EOC)) {
                if (timeout-- == 0x00) {
                    return 0;
                }
            }

            /* Return result */
            return (uint16_t)ADCx->DR;
}
```

9.6. Modulación de ancho de pulso PWM

Modulación de ancho de pulso PWM es un tipo de modulación con el cual cambiamos el ciclo de trabajo a la frecuencia de señal constante. La modulación del ancho de pulso está relacionada con el temporizador del microcontrolador. El microcontrolador STM32F407VG contiene 14 temporizadores (TIM1-TIM14) con resolución de tiempo de 16 bits y dos con resolución de 32 bits (TIM2 y TIM5). Con un microcontrolador, podemos generar 14 diferentes señales PWM independientes con diferentes frecuencias y 32 señales PWM donde ciertos grupos tienen la misma frecuencia y ciclo de trabajo independiente.

Cada temporizador se puede unir físicamente a GPIO, donde determinamos el pin de salida en el puerto A, B, C, D, E al configurar el PWM con el temporizador y la elección del canal.

PWM Module Pinouts for ST				
	Channel 1	**Channel 2**	**Channel 3**	**Channel 4**
Timer1	_GPIO_MODULE_TIM1_CH1_PA8 _GPIO_MODULE_TIM1_CH1_PE9	_GPIO_MODULE_TIM1_CH2_PA9 _GPIO_MODULE_TIM1_CH2_PE11	_GPIO_MODULE_TIM1_CH3_PA10 _GPIO_MODULE_TIM1_CH3_PE13	_GPIO_MODULE_TIM1_CH4_PA11 _GPIO_MODULE_TIM1_CH4_PE14
Timer2	_GPIO_MODULE_TIM2_CH1_PA0 _GPIO_MODULE_TIM2_CH1_PA5	_GPIO_MODULE_TIM2_CH2_PA1 _GPIO_MODULE_TIM2_CH2_PB3	_GPIO_MODULE_TIM2_CH3_PA2 _GPIO_MODULE_TIM2_CH3_PB10	_GPIO_MODULE_TIM2_CH4_PA3 _GPIO_MODULE_TIM2_CH4_PB11
Timer3	_GPIO_MODULE_TIM3_CH1_PA6 _GPIO_MODULE_TIM3_CH1_PB4 _GPIO_MODULE_TIM3_CH1_PC6	_GPIO_MODULE_TIM3_CH2_PA7 _GPIO_MODULE_TIM3_CH2_PB5 _GPIO_MODULE_TIM3_CH2_PC7	_GPIO_MODULE_TIM3_CH3_PB0 _GPIO_MODULE_TIM3_CH3_PC8	_GPIO_MODULE_TIM3_CH4_PB1 _GPIO_MODULE_TIM3_CH4_PC9
Timer4	_GPIO_MODULE_TIM4_CH1_PB6 _GPIO_MODULE_TIM4_CH1_PD12	_GPIO_MODULE_TIM4_CH2_PB7 _GPIO_MODULE_TIM4_CH2_PD13	_GPIO_MODULE_TIM4_CH3_PB8 _GPIO_MODULE_TIM4_CH3_PD14	_GPIO_MODULE_TIM4_CH4_PB9 _GPIO_MODULE_TIM4_CH4_PD15
Timer5	_GPIO_MODULE_TIM5_CH1_PH10	_GPIO_MODULE_TIM5_CH2_PA1 _GPIO_MODULE_TIM5_CH2_PH11	_GPIO_MODULE_TIM5_CH3_PA2 _GPIO_MODULE_TIM5_CH3_PH12	_GPIO_MODULE_TIM5_CH4_PA3 _GPIO_MODULE_TIM5_CH4_PI0
Timer8	_GPIO_MODULE_TIM8_CH1_PC6 _GPIO_MODULE_TIM8_CH1_PI5	_GPIO_MODULE_TIM8_CH2_PC7 _GPIO_MODULE_TIM8_CH2_PI6	_GPIO_MODULE_TIM8_CH3_PC8 _GPIO_MODULE_TIM8_CH3_PI7	_GPIO_MODULE_TIM8_CH4_PC9 _GPIO_MODULE_TIM8_CH4_PI2
Timer9	_GPIO_MODULE_TIM9_CH1_PA2 _GPIO_MODULE_TIM9_CH1_PE5	_GPIO_MODULE_TIM9_CH2_PA3 _GPIO_MODULE_TIM9_CH2_PE6		
Timer10	_GPIO_MODULE_TIM10_CH1_PB8 _GPIO_MODULE_TIM10_CH1_PF6			
Timer11	_GPIO_MODULE_TIM11_CH1_PB9 _GPIO_MODULE_TIM11_CH1_PF7			
Timer12	_GPIO_MODULE_TIM12_CH1_PB14 _GPIO_MODULE_TIM12_CH1_PH6	_GPIO_MODULE_TIM12_CH2_PB15 _GPIO_MODULE_TIM12_CH2_PH9		
Timer13	_GPIO_MODULE_TIM13_CH1_PA6 _GPIO_MODULE_TIM13_CH1_PF8			
Timer14	_GPIO_MODULE_TIM14_CH1_PA7 _GPIO_MODULE_TIM14_CH1_PF9			

TABLA 1: TABLA DE PINES DE SALIDA PARA DETERMINAR LA MODULACIÓN DE PWM DEPENDIENDO DEL TEMPORIZADOR Y EL CANAL.

Determinación del período de modulación PWM (frecuencia) y ajuste del valor ARR (registro de recarga automática).

$$ARR(peridoa) = (TimerX\_frequency / PWM\_frequency) - 1$$

Según a configuración del proyecto, la frecuencia del temporizador es igual a la mitad de la frecuencia del reloj principal (168MHz), que es 84MHz. El valor ARR calculado es el valor de la conversión máxima.

- **Ejemplo de ajuste de modulación PWM**

Ajuse de 10kHz de la señal PWM en el pin PD14 y PD15.

```
int main(void)
{
   int duty1=4200,duty2=0;

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  PWM_Configuration();
```

```
    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON

    USART_Send_Str(USART2,"PWM test!\n\r" );


    while(1)
     {
          Delay(10);
          duty1+=10;
          duty2+=10;

           if(duty1>=8399)
           {duty1=0;}
           TIM4->CCR3=duty1; //Write duty cycle to register, pin PD14

           if(duty2>=8399)
           {duty2=0;}
           TIM4->CCR4=duty2; //Write duty cycle to register, pin PD15

      }//Infinite loop


}//End MAIN


//PWM configuration
void PWM_Configuration(void)
{

    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;


    //PWM on pin PD14 PD15
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_15; // configurate I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;       // GPIO as output (Alternate
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  // Speed of GPIO module (2/10/50)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      // push / pull
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;     // pullup / pullup
    GPIO_Init(GPIOD, &GPIO_InitStructure);

      /* Connect pins PD14 and PD15 on timer TIM4 */
   GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4); // PD14 on Channel 3
   GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4); // PD14 on Channel 4



     /* TIM4 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    /* Time base configuration 10kHz */
  TIM_TimeBaseStructure.TIM_Period = 8399; // ARR+1=(TIM3
clock/PWM_frequncy)=(84MHz/10kHz)=8400
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

  TIM TimeBaseInit(TIM4, &TIM TimeBaseStructure);


  /* PWM1 Mode configuration: Channel3 */
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 4200; //Duty 50%
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OC3Init(TIM4, &TIM_OCInitStructure);
  TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);
```

```
    /* PWM1 Mode configuration: Channel4 */
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0; //Duty 0%
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OC4Init(TIM4, &TIM_OCInitStructure);
  TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);


    TIM_ARRPreloadConfig(TIM4, ENABLE);

   /* TIM4 enable counter */
   TIM_Cmd(TIM4, ENABLE);
}
```

9.7. Rutina de interrupción del temporizador

Las rutinas de interrupción son una parte importante de los sistemas en tiempo real. En estos sistemas y dependiendo del sistema de microcontroladores, conocemos diferentes tipos de rutinas de interrupción que se desencadenan por diferentes eventos. Algunos de estos son: interrupción del temporizador, interrupciones externas (en el cambio de valores en el pin de entrada o salida), interrupción del convertidor AD, interrupción de eventos de comunicación (recepción de datos, transmisión de datos SPI, 12C, UART, error en el bus de comunicación, etc). En nuestro caso, presentaremos interrupción de tiempo, interrupción externa e interrupción en la recepción de datos.

- **Configuración de interrupción en el temporizador 3 (TIM3).**

Periodos de tiempo del temporizador igual a periodo

$$Period = (TimerX\_frequency / TimerX\_prescaler + 1)^{-1} \cdot TimerX\_period$$

En nuestro caso se usa el factor de escala 2, lo que significa que la frecuencia básica del temporizador es igual a la mitad del reloj MCU principal, la frecuencia básica del reloj del temporizador es 84MHz.

Ejemplo:

Set timer period to 10ms. Choose value TimerX_prescaler=209, TimerX_period=4000;

$$Period = (TimerX\_frequency / TimerX\_prescaler + 1)^{-1} \cdot TimerX\_period$$
$$= (84MHz / 209 + 1)^{-1} \cdot 4000 = 0.01s$$

El tiempo de interrupción del disparo se establece con la frecuencia del temporizador.

1) Active la MCU del reloj del sistema en el temporizador determinado (***RCC_APB1ENR.TIM2EN = 1***)

```
/* TIMx clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
```

2) Determine el preescalador de división de reloj. La frecuencia del reloj se divide en MCU ***TIMx_PSC***

```
/* TIM3 clock prescaler */

TIM_TimeBaseStructure.TIM_Prescaler = 210;
```

3) Determine el número de valores contados TIMX ARR (auto reload zegister), cuya frecuencia depende del valor en el registro TIMx PSC y frecuencia de la MCU.ARR

```
/* TIM3 period*/
TIM_TimeBaseStructure.TIM_Period = 4000;
```

4) Habilite los modos contando hacia arriba o hacia atrás y escriba la estructura en los registros temporizadores TM3.

```
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //Counter mode up

        TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);     //Structure entry
```

5) Habilite el temporizador y edite la rutina de interrupción.

```
TIM_ClearFlag(TIM3, TIM_FLAG_Update);   //Clear flag
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //Interrupt on auto reload register
TIM_Cmd(TIM3, ENABLE); //Enable timer
```

Para configurar los disparadores de interrupción, también debemos establecer la prioridad en NVIC (controlador de interrupción vectorial anidado).

```
NVIC_InitTypeDef NVIC_InitStruct;

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);        //TIM3 Priority
NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

Las prioridades se pueden seleccionar en los niveles 0-4, donde un número más bajo significa que la función de interrupción es más importunate. El subprogram de interrupción se ejecuta en **TIM3 IRQHandler preparado previamente** (vacío). Los nombres de las funciones de interrupción están preestablecidos y escrito en el ducomento **startup_stm32f40_41xxx.s.**

```
//MACRO
#define  LED1 GPIO_Pin_15
#define  LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define  LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define  Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)


int main(void)
{

    // MCU and GPIO setting
    RCC_Configuration();
    GPIO_Configuration();
    USART2_Configuration();
    TIM_Configuration();
    NVIC_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON
```

```
    USART_Send_Str(USART2,"TIMER test!\n\r");


    while(1)
                {


                }//Infinite loop


}//End MAIN

/******************************************************************************
Timer3 configuration
******************************************************************************/
void TIM Configuration(void)
{

    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;



     /* TIM3 clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    /* Time base configuration */
    TIM_TimeBaseStructure.TIM_Period = 4000;     //Timer3 period =
((TIM3_clock_freq)/(TIM_prescaler+1))^-1 * TIM_Preiod= ((84MHz)/(209+1))^-1 * 4000=10ms
    TIM_TimeBaseStructure.TIM_Prescaler =209;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;   //Counter mode up
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);


    TIM_ClearFlag(TIM3, TIM_FLAG_Update);   //Clear flag
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //Interrupt on auto reload register
    TIM_Cmd(TIM3, ENABLE); //Enable timer
}


/******************************************************************************
 NVIC-vector
******************************************************************************/
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStruct;

        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);        //

      //TIM3 Priority
        NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);

}

/******************************************************************************
 Interrupt function
******************************************************************************/
void TIM3_IRQHandler(void)
{

    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
    {
           TIM_ClearITPendingBit(TIM3, TIM_IT_Update);


              GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
```

```
            GPIO_ToggleBits(GPIOD, GPIO_Pin_13);

    }
}
```

- **Configuración de interrupción en el botón A0.**

Las inerrupciones externas que se desencadenan mediante dispositivos externos, los protocolos de comunicación de cinmutadores, etc. Deben habilitarse con un conjunto de entradas en certos registros del Sistema. Estas interrupciones se dividen entre las unidades EXTIO EXTI15 y se determinan mediante programación en el registro de control SYSCFG EXTICRx. Cada EXTI0-15 corresponde a un número de Puerto secuencal O-15, como se vee n la imagen 4.

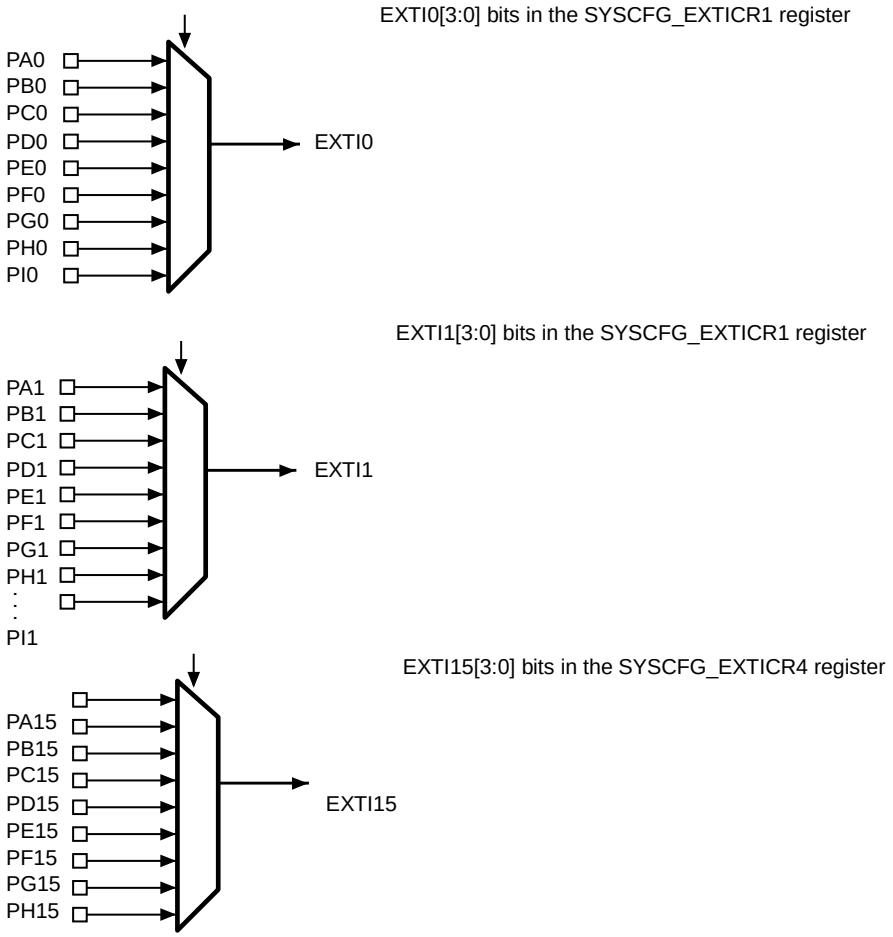

IMAGEN 4: GRUPOS PARA INTERRUPCIONES EXTERNAS

Ejemplo de interrupción externa a través del botón PAO.

Determine la másccara que ingresamos para registrar SYSCFG EXTICRX. Usaremos el botón en el puerto A y el número secuencial 0 (PAO). Como se ve en la imagen de arriba, necesitaremos usar la concexion de pin PAO a la interrupción eyternal:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Configuración de interrupción externa:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

Configuración de interrupción externa:

```
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
```

Dependiendo del tipo de disparador (ya sea que esté configurado en el registro como positivo o negativo) determinaremos el tipo de interrupción:

```
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;

EXTI_Init(&EXTI_InitStructure);

EXTI_GenerateSWInterrupt(EXTI_Line0);
```

Ejemplo de configuración completa del programa:

```
//MACRO
#define LED1 GPIO_Pin_15
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)

int main(void)
{


  // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  EXTILine0_Configuartion();
  NVIC_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON

    USART_Send_Str(USART2,"EXTI interrupt test!\n\r");


    while(1)
        {


        }//Infinite loop
```

```
}//End MAIN

/*******************************************************************************
 EXTI0 configuration
*******************************************************************************/
void EXTILine0_Configuartion(void)
{
       EXTI_InitTypeDef   EXTI_InitStructure;
       GPIO_InitTypeDef   GPIO_InitStructure;

        /* Enable SYSCFG clock */
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

      //BUTTON
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;          //  configuration PIN_A 0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;           //  GPIO as input
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;    //  Speed of GPIO module
(2/10/50 MHz)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;    //  push / pull (opposed to open
drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;    // pullup / pullup resistor is
not active
    GPIO_Init(GPIOA, &GPIO_InitStructure);                // Setting of port A

     /* Connect EXTI Line0 to PA0 pin */
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

        /* Configure EXTI Line0 */
    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

     /* Generate software interrupt: simulate a rising edge applied on EXTI0 line */
     EXTI_GenerateSWInterrupt(EXTI_Line0);

}

/*******************************************************************************
 NVIC configuration for EXTI0
*******************************************************************************/
void NVIC_Configuration(void)
{

    NVIC_InitTypeDef NVIC_InitStruct;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);        //

   /* Enable and set EXTI Line0 Interrupt  */
  NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);

}


/*******************************************************************************
EXTI0 Interrupt function
*******************************************************************************/
void EXTI0_IRQHandler(void)
{

     if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {


            GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
            GPIO_ToggleBits(GPIOD, GPIO_Pin_13);
```

```
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
```

9.8. Codificador incremental

El codificador incremental es un sensor de dispositivo que detecta cambios de sistema o rotaciones. De esta manera podemos diferenciar codificadores incrementales lineales y rotatorios. Muy a menudo codificadores de ángulos, velocímetros o botones de control en dispositivos electrónicos. Los codificadores giratorios a menudo se montan directamente en el eje del motor o en el elemento de contacto. La medición de los cambios y la rotación con codificadores incrementales es una práctica industrial general. La mayoría de los sistemas incorporados ya contienen módulos para la concesión del codificador con el sistema de microcontroladores. El principio de medición se basa en el principio de recuento incremental causado por el cambio de valor medido. Los sistemas más preciosos tienen de dos a cuatro canales y el índice del cociente B, A, B, índice, los más simples solo uno (A). El funcionamiento del codificador incremental se presenta en la imagen 5:

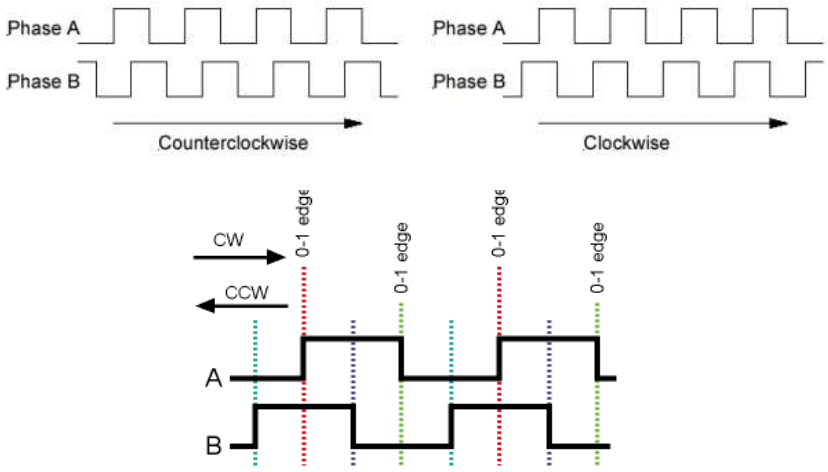

IMAGEN 5: PRINCIPIO DE FUNCIONAMIENTO DEL ENCODER INCREMENTO

En la inicialización del codificador incrementeal, es importante conectar el temporizador de foram similar a los pines PWM y GPIO. En este caso, solo se pueden usar los pines de los canales 1 y 2 (vistos en la tabla 1). Los pines de los canales 3 y 4 no se pueden usar en el codificador incremental de modo. En este caso, el conteo del temporizador no está relacionado con la resolución y el reloj predefinidos del temporizador, sino solo con el codificador incrementeal. Para esto, el temporizador necesita configurarse en modo codificador incremental con la función 'TIM EncoderInterfaceConfig(xxx)'. El temporizador se puede usar en un circuito con una interrupción o sin él, lo que significa que la interrupción se active dependiendo de un cierto número de incrementos preestablecidos. El siguiente ejemplo de Código muestra

los ajustes del codifiados en el pin PC6(A) y PC7(B) con un temporizador TIM3 y con una interrupción con 3200 incrementos contados.

Código ejemplo:

```
int main(void)
{
  int16_t encoder;
  char Buffer[5];

   RCC_Configuration();
   ENCODER_Configuration();
   NVIC_Configuration();

    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    USART_Send_Str(USART2,"ENCODER TEST\n\r");

    //Infinite loop
    while(1)
      {

             encoder = TIM3->CNT; //Read encoder value
             sprintf(Buffer,"%d ",encoder); //Convert to string
             USART_Send_Str(USART2,Buffer);USART_Send_Str(USART2,"\n\r"); //Send data
USART2
             Delay(200);

      }//END - Infinite loop
}//END - MAIN


/*******************************************************************************
ENCODER_Configuration
*******************************************************************************/
void ENCODER_Configuration()
{

        TIM_TimeBaseInitTypeDef TIM3_TimeBaseStructure;
        GPIO_InitTypeDef GPIO_InitStructure;

        /*ENCODER PIN Configuration ( PC6 & PC7 )*/
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7; //PIN PC6-T1(A) PC7-T2(B)
        GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;  //Trigger only on + voltage
        GPIO_InitStructure.GPIO_OType= GPIO_AF_TIM3;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        /*Connect PIN PC6 and PC7 on TIM 3 (Channel 1&2)*/
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM3); //Enable GPIO PC6 To alternate
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_TIM3); //Enable GPIO PC7 To alternate


        /*TIM3 setting*/
        TIM_DeInit(TIM3);
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
        TIM3_TimeBaseStructure.TIM_Period = 3200;   //Number of encoder counts
        TIM3_TimeBaseStructure.TIM_Prescaler = 0;
        TIM3_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM3_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
        //ENCODER MODE
       TIM_EncoderInterfaceConfig(TIM3,
                                  TIM_EncoderMode_TI12, //Count on both channel A in B
                                   TIM_ICPolarity_Rising,
                                   TIM_ICPolarity_Rising);
        TIM_TimeBaseInit(TIM3, &TIM3_TimeBaseStructure);
```

```
          TIM3->CNT=0; //Inital value of the encoder timer

        //Enable upadate flag
         TIM_ClearFlag(TIM3, TIM_FLAG_Update);
        //Timer interrupt enable, for one revolution
        TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
        //Start timer TIM3
        TIM_Cmd(TIM3, ENABLE);

}


/*******************************************************************************
*  Interrupt  priority configuration
*******************************************************************************/
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStruct;


        //Timer 3 Priority, with encoder
        NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);


}



/*******************************************************************************
Encoder interrupt function, occur after 3200 increments
*******************************************************************************/
    void TIM3_IRQHandler(void)
    {

                    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
                    {
                            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);




                    }
    }
```

9.8. Comunicación SPI con el acelerómetro LIS302DL

En la placa de desarrollo se encuentra el acelerómetro de eje triple LiS302DL con rango de medición entre t2 y t8g y resolución de 8bits. El sensor en la MCU está conectado al bus SPI a través de las patillas PA7(MOSI), PA6(MISO), PA5(SCK), PE3(CS or SS). SPI (interfaz periférica serial) puede operar en modo de 3 o 4 hilos. SC / SS (selección de chip, selección de esclavo). Para la iniciación del sector esté calibrado correctamente. Establecer rango de sensibilidad y rango de frecuencia. Los datos de la biblioteca ya son valores calculados mm/s2-mg.

Ejemplo del programa:

```
int main(void)
```

```
{
    TM_LIS302DL_LIS3DSH_t Axes_Data;
    char Buffer[200];

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  NVIC_Configuration();


  GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON

  USART_Send_Str(USART2,"Hello from StmF4Discovery!\n\r");


     /*Initialization of LIS302DL*/
     TM_LIS302DL_LIS3DSH_Init(TM_LIS3DSH_Sensitivity_2G, TM_LIS3DSH_Filter_800Hz);

    //Infinite loop
    while(1)
    {
         TM_LIS302DL_LIS3DSH_ReadAxes(&Axes_Data); //READ data from SPI
         /*Write to string*/
         sprintf(Buffer,"x: %d  y: %d   z: %d ",Axes_Data.X, Axes_Data.Y, Axes_Data.Z);
         /*Send string*/
         USART_Send_Str(USART2,Buffer); USART_Send_Str(USART2,"\n\r");

        Delay(200);
    }//Infinite loop


}//Konec MAIN
```

9.9. Puerto serie virtual VCP

En la placa de prueba dada no tenemos un módulo especial para comunicación USART (FTDI, CP2012…), por lo tanto, usaremos USB 2.0 como un módulo en el microconrolador, que emularemos como un puerto COM virtual. Si queremos usar el módulo USB como VCP, entonces debemos incluir una biblioteca adicional por parte del fabricante ST-Microelectronics. En el siguiente ejemplo, se muestra un código de programa que muestra la codificación de la interfaz USBVSP en la placa de desarrollo SRM32DiscoveryF407. El conector micro USB se encuentra en los pines PA11 y PA12 funciona como líneas de datos (D- D+). Para facilitar la compresión y el uso de la interfaz USB, hemos preparado una bliblioteca rediseñada por Tilen Majerle que es de acceso libre en http://stm32f4-discovery.com/tag/tilen-majerle/. El programa describe la recepcion de datos co nel signo final % y una pantalla del contenido recibido con un clic en el botón. Con la función USBD Int () y el argumento USE USB OTG FS incializamos la interfaz USB-VCP en los pines PA10, PA11,PA12,PA13.

Ejemplo de programa.

```
USB_OTG_CORE_HANDLE USB_OTG_dev; //USB VCP structure

int main(void)
{
   uint8_t b;
   uint8_t i=0;
   char USB_read[100];
```

```
   //MCU and GPIO setting
   RCC_Configuration();
   GPIO_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //  LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  // LED ON


      /* Initialize USB VCP */
       USBD_Init(  &USB_OTG_dev,
                   USE_USB_OTG_FS
                   USB_OTG_FS_CORE_ID,
                   &USR_desc,
                   &USBD_CDC_cb,
                   &USR cb);

    //Infinite loop
    while(1)
    {


           if(Button==1)//Send received data on the button press
           {
                  if (TM_USB_VCP_Getc(&b) == TM_USB_VCP_DATA_OK) {


                        USB_read[i]=(char)b;
                        i++;

                           if(b=='%')//Terminal character
                            {
                                USB_read[i-1]=' ';
                                TM_USB_VCP_Puts(USB_read);TM_USB_VCP_Puts("\n\r");
//Send data

                                      for(int j=0;j<=i;j++) //Clear out buffer
                                      USB_read[j]=' ';

                                      i=0;  //Clear array index
                            }
                   }

           }


      }//Infinite loop-END


}//MAIN - END
```