2013-2014

Juan Domingo Jiménez Jerez

2013-2014

1.-OBJETIVOS DEL PROYECTO

1.1.- Objetivos del proyecto

El objetivo de este trabajo es la realización de un control PWM mediante la placa de pruebas Discovery del microcontrolador STM 32F4 de ARM. El control de la señal PWM se realiza mediante una fotorresistencia LDR o interrupción externa mediante el botón USER integrado en la placa de pruebas.

1.2.- Especificaciones del sistema

El control del ciclo de trabajo de la señal PWM generada puede realizarse de dos formas:

-Automaticamente: mediante la fotoresistencia LDR.

-Manualmente: mediante el Boton USER integrado en la placa de pruebas. Podemos volver al modo anterior mediante el otro botón integrado en la placa de pruebas (Botón RESET).

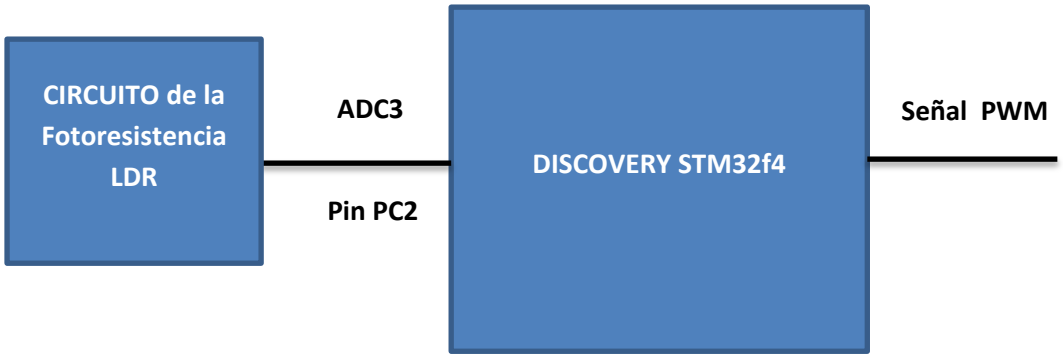
1.2.1.-Modo de control Automático mediante Fotoresistencia LDR

El ciclo de trabajo de la señal PWM dependerá del valor de una Fotoresistencia LDR.
A mayor luminosidad captada por el LDR la variación de la resistencia de este componente dará lugar a un menor ciclo de trabajo de la señal PWM y viceversa. Esto será gestionado en tiempo real por el Microcontrolador STM 32F4.

1.2.2.- Modo de control Manual mediante el botón User

Se dispone de otra posibilidad de control, mediante el botón USER integrado en la placa de pruebas Discovery. Mediante dicho botón podemos ir aumentando el Ciclo de trabajo secuencialmente al doble del ciclo de trabajo anterior. Los valores del ciclo de trabajo serán 12.5%, 25%, 50% y 100%. Partiendo inicialmente de 12,5% al sobrepasar el valor máximo (100%) volverá cíclicamente a 12,5%.

2.-HARDWARE

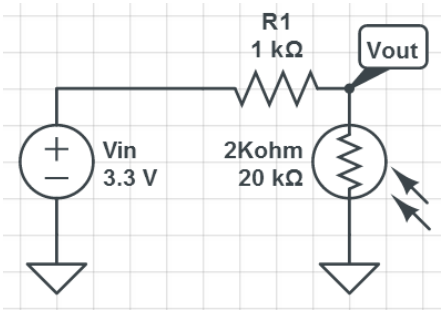


2.1.-Diseño del circuito de la Fotoresistencia LDR:

Especificaciones de la Fotoresistencias LDR:

-Resistencia dependiente de la luz (mín. - máx.): 2-20 kohm
-Resistencia de oscuridad (después de 10 seg.): >2 Mohm
-Valor gamma a 10-100 Lux: 0.7
-Máx. disipación de potencia: 100mW
-Máx. tensión de ruptura: 150Vdc
-Respuesta espectral pico: 540nm
-Tiempo de respuesta (subir): 20ms
-Tiempo de respuesta (bajar): 30ms
-Temperatura ambiente: de -35°C a +70°C
-Dimensiones:
-D: 4.0 ± 0.2mm
-d: 3.5 ± 0.2mm
-H: 35.5 ± 2mm
-T: 1.5mm
-t: 0.40 ± 0.01mm
-W: 2.5 ± 0.2mm

La Fotoresistencia varía su resistencia en función de la luminosidad entre 2-20 kohm. Para analizar la luminosidad con el Microcontrolador creamos un divisor de tensión mediante otra resistencias y alimentamos con 3.3V que es el valor máximo que detecta el conversor ADC que emplearemos en el Microcontrolador.



$$V_{OMAX} = \frac{3.3 \times 20k}{1k + 20k} = 3.143V \qquad V_{OMIN} = \frac{3.3 \times 2k}{1k + 2k} = 2.2V$$

Por lo que a la entrada del Microcontrolador tendremos cualquier valor de tensión entre 3.143V y 2.2V.

2.2.-Señal PWM:

La señal PWM generada por el Microcontrolador la podemos encontrar en el Pin PB0 del mismo, aquí colocamos un diodo led para comprobar como varía su iluminación en función del ciclo de trabajo recibido o también podríamos colocar un algún circuito del tipo mosfet driver para poder controlar aplicaciones de potencia mediante la señal PWM generada que es donde mejor se podría apreciar el potencial del trabajo y que no se ha realizado porque escapa del objetivo de la asignatura.

2.-Software

Para la implementación software se ha utilizado el ya antes mencionado Microcontrolador STM32F4 de ARM, a partir de este dispositivo gestionamos el ciclo de trabajo que tendrá nuestra señal de salida a partir del valor recibido por el circuito de la Fotoresistencia LDR o lo incrementamos simplemente mediante el botón USER de la placa de pruebas mediante interrupción externa.

El entorno de programación elegido ha sido IAR Embedded Workbench for ARM y el trabajo se ha realizado a partir de dos de los ejemplos del firmware package que incluye 22 ejemplos explicados en un PDF que proporciona el fabricante. TIM PWM output y ADC DMA.

2.1-Recursos del Microcontrolador empleados:

TIM periférico en PWM:

Para obtener la señal PWM se usa el Timer 3. La frecuencia TIM3CLK se fija mediante SystemCoreClock / 2 (Hz). El reloj contador de TIM3 se fija a 28 MHz, para ello fijamos el prescaler de la siguiente forma:

● Prescaler = (TIM3CLK / TIM3 reloj contador) - 1

● SystemCoreClock se fija a 168 MHz mediante STM32F4xx Devices Revision A.

● TIM3 funciona a 42 kHz: frecuencia de TIM3 = TIM3 reloj contador/(ARR + 1) = 28 MHz / 666 = 42 kHz

● El valor del registro TIM3 CCR4 es el encargado de modificar el ciclo de trabajo de la señal PWM y será el valor que variara en función de los parámetros de entrada.

● Generaremos a partir de TIM3 Channel 3 una señal PWM de frecuencia 30 kHz y con un ciclo de trabajo inicial de 12.5% y que irá variando en función del valor del registro CCR4: TIM3 Channel3 Ciclo de trabajo = (TIM3_CCR4/ TIM3_ARR + 1)* 100.

DMA y ADC.
Usamos ADC3 y DMA para tranferir continuamente datos convertidos continuamente mendiante ADC3 directamente a memoria usando DMA.

● ADC3 está configurado para convertir continuamente valores de analógico a digital

● Cada vez que finaliza la conversión, DMA transfiere los datos convertidos del registro ADC3 DR a la variable ADC3ConvertedValue en modo circular que posteriormente a partir de ADC3ConvertedValue podemos obtener el valor analógico que se tiene a la entrada mediante una formula.

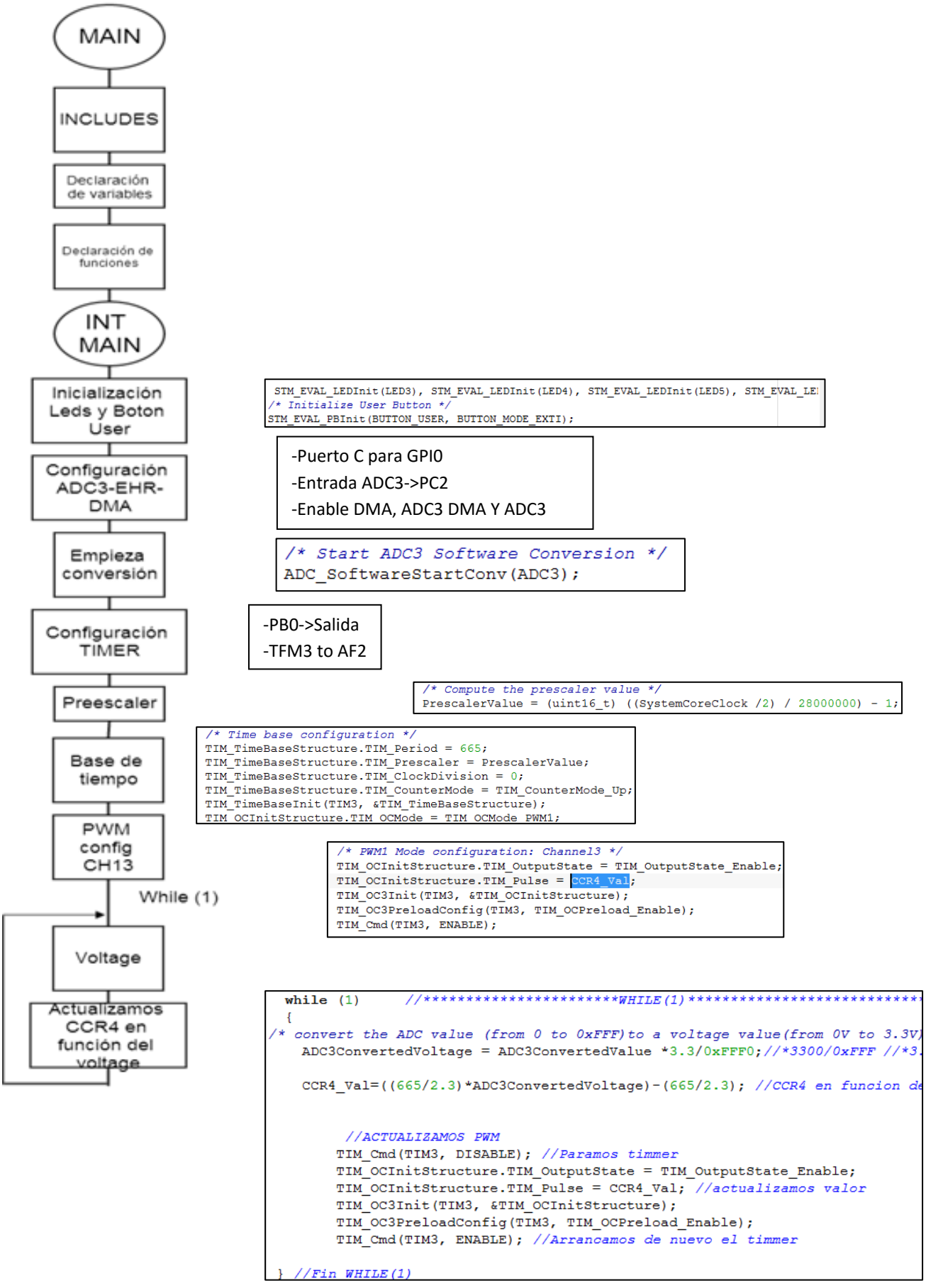
● El reloj del sistema tiene una frecuencia de 144 MHz, APB2 se fija en 72 MHz y el reloj para la conversión ADC toma el valor APB2/2.

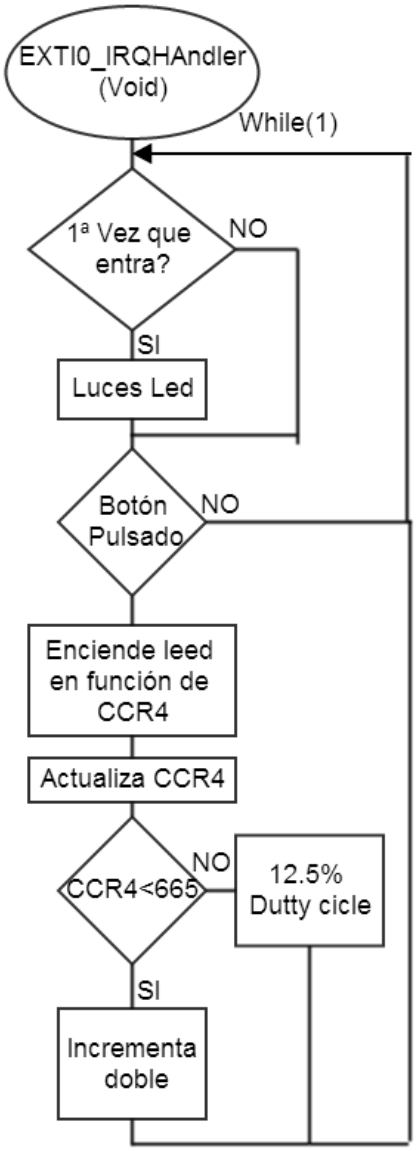
● Dado que el reloj ADC3 es de 36 MHz y el tiempo de muestreo se establece en 3 ciclos, el tiempo de conversión de datos de 12 bits es de 12 ciclos, por lo que el tiempo de conversión total es de (12 +3) / 36 = 0.41 us (2.4 Msps).

GPIO
Configuramos PC0 como entrada ADC y PBO como salida PWM

EXTI
El Modo de control Manual mediante el botón USER produce una interrupción externa y en la rutina de atención a esta es donde se aumentará el ciclo de trabajo cíclicamente cada vez que se pulse el botón USER y la única forma de salir de la rutina de atención es pulsando el botón RESET que activaría el modo automático.

2.1-Implementación Software: Flujogramas






```
void EXTI0_IRQHandler(void)  //Interrupcion externa boton
{
  UserButtonPressed = 0x01;
   while (1) {  //************ WHILE(1) **********************
   while(semaforo==0&&STM_EVAL_PBGetState(BUTTON_USER) != Bit_SET){ //lo hace solo la primera vez
      STM_EVAL_LEDOff(LED4);
      STM_EVAL_LEDOff(LED3);
      STM_EVAL_LEDOff(LED5);
      STM_EVAL_LEDOff(LED6);
            Delay(1000000);
          STM_EVAL_LEDToggle(LED4);
            Delay(1000000);
          STM_EVAL_LEDToggle(LED3);
            Delay(1000000);
          STM_EVAL_LEDToggle(LED5);
            Delay(1000000);
          STM_EVAL_LEDToggle(LED6);
            Delay(1000000);
        }
    semaforo=1;

    if (STM_EVAL_PBGetState(BUTTON_USER) == Bit_SET)
      {
    STM_EVAL_LEDOff(LED4), STM_EVAL_LEDOff(LED3), STM_EVAL_LEDOff(LED5), STM_EVAL_LEDOff(LED6);

        while (STM_EVAL_PBGetState(BUTTON_USER) == Bit_SET)  //Espera a soltar boton

         if (CCR4_Val>3){
           STM_EVAL_LEDOn(LED4);
         }if (CCR4_Val>83){
           STM_EVAL_LEDOn(LED3);
         }if (CCR4_Val>166){
           STM_EVAL_LEDOn(LED5);
         }if (CCR4_Val>332){
           STM_EVAL_LEDOn(LED6);
         }
         //ACTUALIZAMOS PWM
        TIM_Cmd(TIM3, DISABLE); //Paramos timmer
        TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = CCR4_Val; //actualizamos valor
        TIM_OC3Init(TIM3, &TIM_OCInitStructure);
        TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
        TIM_Cmd(TIM3, ENABLE); //Arrancamos de nuevo el timmer

        if (CCR4_Val<664)
          {
              CCR4_Val= CCR4_Val+CCR4_Val;  // CCR4=CCR4*2
          }
        else
          {
              CCR4_Val=83;
          }
          Delay(1000000); //MUY IMPORTANTE!!, sin este delay tiene algunos fallos
     } //fin IF
  }
  /* Clear the EXTI line pending bit */
  EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE); //No salimos de la interrupcion hasta pulsar e
}
```

ANEXO1: Código MAIN.C

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"
#include "stm32f4_discovery.h"
#include <stdio.h>
#include "stm32f4xx_it.h"

#define ADC3_DR_ADDRESS     ((uint32_t)0x4001224C)

 TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
TIM_OCInitTypeDef  TIM_OCInitStructure;

 uint16_t CCR4_Val = 83;
uint16_t PrescalerValue = 0;

 __IO uint16_t ADC3ConvertedValue = 0;
__IO double ADC3ConvertedVoltage = 0.0;
__IO double voltageLCD = 0.0;

/* Private function prototypes -----------------------------------------------*/
void TIM_Config(void);
/* Private functions ---------------------------------------------------------*/
void ADC3_CH12_DMA_Config(void); //ADC
uint8_t semaforo = 0;
RCC_ClocksTypeDef RCC_Clocks;
__IO uint8_t RepeatState = 0;
__IO uint16_t CCR_Val = 16826;
extern __IO uint8_t LED_Toggle;
__IO uint8_t UserButtonPressed = 0x00;
__IO uint32_t TimingDelay;
void Delay(__IO uint32_t nTime);   //Declaración funcion delay

int main(void)          //INT MAIN(VOID)

{ STM_EVAL_LEDInit(LED3), STM_EVAL_LEDInit(LED4), STM_EVAL_LEDInit(LED5),
STM_EVAL_LEDInit(LED6);

  /* Initialize User Button */
STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);

    /* ADC3 configuration ******************************************************
/*  - Enable peripheral clocks                                 */
/*  - DMA2_Stream0 channel2 configuration                      */
/*  - Configure ADC Channel12 pin as analog input              */
/*  - Configure ADC3 Channel12                                 */
ADC3_CH12_DMA_Config();
/* Start ADC3 Software Conversion */
ADC_SoftwareStartConv(ADC3);
// SysTick end of count event each 10ms
RCC_GetClocksFreq(&RCC_Clocks);
SysTick_Config(RCC_Clocks.HCLK_Frequency / 100);
```

```
  /*!< At this stage the microcontroller clock setting is already configured,  this is done through
SystemInit() function which is called from startup file (startup_stm32f4xx.s) before to branch to
application main.
To reconfigure the default setting of SystemInit() function, refer to system_stm32f4xx.c file  */

  /* TIM Configuration */
TIM_Config();
/* ---------------------------------------------------------------------
TIM3 Configuration: generate 4 PWM signals with 4 different duty cycles.  In this example TIM3 input
clock (TIM3CLK) is set to 2 * APB1 clock (PCLK1), since APB1 prescaler is different from 1.

 TIM3CLK = 2 * PCLK1
PCLK1 = HCLK / 4  => TIM3CLK = HCLK / 2 = SystemCoreClock /2

 To get TIM3 counter clock at 28 MHz, the prescaler is computed as follows:
          Prescaler = (TIM3CLK / TIM3 counter clock) – 1
          [Prescaler = ((SystemCoreClock /2) /28 MHz) - 1] // utilizamos esta
To get TIM3 output clock at 30 KHz, the period (ARR)) is computed as follows:
          ARR = (TIM3 counter clock / TIM3 output clock) – 1 = 665

 TIM3 Channel3 duty cycle = (TIM3_CCR4*8/ TIM3_ARR)* 100 = 100%
TIM3 Channel3 duty cycle = (TIM3_CCR4*4/ TIM3_ARR)* 100 = 50%
TIM3 Channel3 duty cycle = (TIM3_CCR4*2/ TIM3_ARR)* 100 = 25%
TIM3 Channel3 duty cycle = (TIM3_CCR4/ TIM3_ARR)* 100 = 12.5%

Note:     SystemCoreClock variable holds HCLK frequency and is defined in system_stm32f4xx.c file
Each time the core clock (HCLK) changes, user had to call SystemCoreClockUpdate()   function to
update SystemCoreClock variable value. Otherwise, any configuratin based on this variable will be
incorrect.

  ---------------------------------------------------------------------- */

  /* Compute the prescaler value */

  PrescalerValue = (uint16_t) ((SystemCoreClock /2) / 28000000) - 1;

  /* Time base configuration */

  TIM_TimeBaseStructure.TIM_Period = 665;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;

  /* PWM1 Mode configuration: Channel3 */

  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR4_Val;
TIM_OC3Init(TIM3, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
TIM_Cmd(TIM3, ENABLE);
```

```
 while (1)     //***********************WHILE(1)************************************
{
/* convert the ADC value (from 0 to 0xFFF)to a voltage value(from 0V to 3.3V)*/
ADC3ConvertedVoltage = ADC3ConvertedValue *3.3/0xFFF0;//*3300/0xFFF //*3.3/65535
CCR4_Val=((665/2.3)*ADC3ConvertedVoltage)-(665/2.3); //CCR4 en funcion del voltage

       //ACTUALIZAMOS PWM

      TIM_Cmd(TIM3, DISABLE); //Paramos timmer
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR4_Val; //actualizamos valor
TIM_OC3Init(TIM3, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
TIM_Cmd(TIM3, ENABLE); //Arrancamos de nuevo el timmer
} //Fin WHILE(1)
} //fin Main

void Delay(__IO uint32_t nTime) //Funcion delay
{
while(nTime--)
{}
}
void ADC3_CH12_DMA_Config(void){

 ADC_InitTypeDef       ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
DMA_InitTypeDef       DMA_InitStructure;
GPIO_InitTypeDef      GPIO_InitStructure;

/* Enable ADC3, DMA2 and GPIO clocks ****************************************/

 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2 | RCC_AHB1Periph_GPIOC, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
/* DMA2 Stream0 channel0 configuration **************************************/

 DMA_InitStructure.DMA_Channel = DMA_Channel_2;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC3_DR_ADDRESS;
DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADC3ConvertedValue
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
DMA_InitStructure.DMA_BufferSize = 1;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA2_Stream0, &DMA_InitStructure);
DMA_Cmd(DMA2_Stream0, ENABLE);
```

```
  /* Configure ADC3 Channel12 pin as analog input ******************************/

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOC, &GPIO_InitStructure);
/* ADC Common Init **********************************************************/
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
ADC_CommonInit(&ADC_CommonInitStructure);

/* ADC3 Init ****************************************************************/
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC3, &ADC_InitStructure);

/* ADC3 regular channel12 configuration *************************************/
ADC_RegularChannelConfig(ADC3, ADC_Channel_12, 1, ADC_SampleTime_3Cycles);
/* Enable DMA request after last transfer (Single-ADC mode) */
ADC_DMARequestAfterLastTransferCmd(ADC3, ENABLE);
/* Enable ADC3 DMA */
ADC_DMACmd(ADC3, ENABLE);
/* Enable ADC3 */
ADC_Cmd(ADC3, ENABLE);
}
void TIM_Config(void)   //Configuración Timer
{
GPIO_InitTypeDef GPIO_InitStructure;
/* TIM3 clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
/* GPIOC and GPIOB clock enable */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC | RCC_AHB1Periph_GPIOB, ENABLE);
/* GPIOB Configuration:  TIM3 CH3 (PB0) */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
GPIO_Init(GPIOB, &GPIO_InitStructure);
/* Connect TIM3 pin to AF2 */
GPIO_PinAFConfig(GPIOB, GPIO_PinSource0, GPIO_AF_TIM3);
}
#ifdef  USE_FULL_ASSERT
/**
* @brief  Reports the name of the source file and the source line number
*         where the assert_param error has occurred.
* @param  file: pointer to the source file name
```

```
* @param  line: assert_param error line source numbe
* @retval None*/
void assert_failed(uint8_t* file, uint32_t line){
/* User can add his own implementation to report the file name and line number, ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
while (1){}
}
#endif
```

ANEXO1: Código stm32f4xx_it.c

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_it.h"
#include "stm32f4_discovery.h"
#include "stm32f4xx.h"
#include <stdio.h>

extern  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
extern  TIM_OCInitTypeDef  TIM_OCInitStructure;
extern  uint16_t CCR4_Val ;
extern  uint16_t PrescalerValue ;
extern  uint32_t TimingDelay;
extern void Delay(__IO uint32_t nTime);
extern uint8_t semaforo ;

/** @addtogroup STM32F4_Discovery_Peripheral_Examples
* @{
*/

/** @addtogroup TIM_PWM_Output
* @{  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

extern __IO uint8_t UserButtonPressed; //Interrupcion externa boton

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M4 Processor Exceptions Handlers                         */
/******************************************************************************/
/**
* @brief  This function handles NMI exception.
* @param  None
* @retval None
*/
void NMI_Handler(void){}
/**
* @brief  This function handles Hard Fault exception.
* @param  None
```

```
* @retval None   */
void HardFault_Handler(void){                  /* Go to infinite loop when Hard Fault exception occurs */

while (1) {}
}
/**
* @brief  This function handles Memory Manage exception.
* @param  None
* @retval None   */

void MemManage_Handler(void){  /* Go to infinite loop when Memory Manage exception occurs */

  while (1) {}
}
/**
* @brief  This function handles Bus Fault exception.
* @param  None
* @retval None   */

void BusFault_Handler(void){     /* Go to infinite loop when Bus Fault exception occurs */

  while (1){}
                        }

/**
* @brief  This function handles Usage Fault exception.
* @param  None
* @retval None    */

void UsageFault_Handler(void){    /* Go to infinite loop when Usage Fault exception occurs */

  while (1) {}
}



/**
* @brief  This function handles Debug Monitor exception.
* @param  None
* @retval None   */

void DebugMon_Handler(void){}

/**
* @brief  This function handles SVCall exception.
* @param  None
* @retval None   */

void SVC_Handler(void){}

/**
* @brief  This function handles PendSV_Handler exception.
* @param  None
* @retval None    */

void PendSV_Handler(void){}
```

```
/**
* @brief  This function handles SysTick Handler.
* @param  None
* @retval None    */

void SysTick_Handler(void){}

void EXTI0_IRQHandler(void)  //Interrupcion externa boton

{
UserButtonPressed = 0x01;

  while (1) {  //************ WHILE(1) **********************
while(semaforo==0&&STM_EVAL_PBGetState(BUTTON_USER) != Bit_SET){ //lo hace solo la 1 vez

STM_EVAL_LEDOff(LED4)
STM_EVAL_LEDOff(LED3);
STM_EVAL_LEDOff(LED5);
STM_EVAL_LEDOff(LED6);
          Delay(1000000);
STM_EVAL_LEDToggle(LED4);
          Delay(1000000);
STM_EVAL_LEDToggle(LED3);
          Delay(1000000);
STM_EVAL_LEDToggle(LED5);
          Delay(1000000);
STM_EVAL_LEDToggle(LED6);
          Delay(1000000);
}
semaforo=1;
if (STM_EVAL_PBGetState(BUTTON_USER) == Bit_SET)

     {     STM_EVAL_LEDOff(LED4), STM_EVAL_LEDOff(LED3), STM_EVAL_LEDOff(LED5),
STM_EVAL_LEDOff(LED6);

 while (STM_EVAL_PBGetState(BUTTON_USER) == Bit_SET)  //Espera a soltar boton

            if (CCR4_Val>3){
                  STM_EVAL_LEDOn(LED4);
           }if (CCR4_Val>83){
                    STM_EVAL_LEDOn(LED3);
           }if (CCR4_Val>166){
                  STM_EVAL_LEDOn(LED5);
           }if (CCR4_Val>332){
           STM_EVAL_LEDOn(LED6);

       }

      //ACTUALIZAMOS PWM

      TIM_Cmd(TIM3, DISABLE); //Paramos timmer
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR4_Val; //actualizamos valor
TIM_OC3Init(TIM3, &TIM_OCInitStructure);
```

```
TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
TIM_Cmd(TIM3, ENABLE); //Arrancamos de nuevo el timmer

      if (CCR4_Val<664){
           CCR4_Val= CCR4_Val+CCR4_Val;  // CCR4=CCR4*2
}
else { CCR4_Val=83;
}

          Delay(1000000); //MUY IMPORTANTE!!, sin este delay tiene algunos fallos

     } //fin IF

  }

  /* Clear the EXTI line pending bit */

  EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE); //No salimos de la interrupcion hasta pulsar el otro boton (Reset)

}
```