Control de iluminación PWM mediante fotoresistencia LDR o interrupción Externa (Botton User)

Memoria Trabajo Microcontrolador SEA

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

**DISCOVERY STM32f4**

**Señal PWM** dfpqpppsdfsdfPWMP

**CIRCUITO de la Fotoresistencia LDR**

**ADC3**

**Pin PC2**

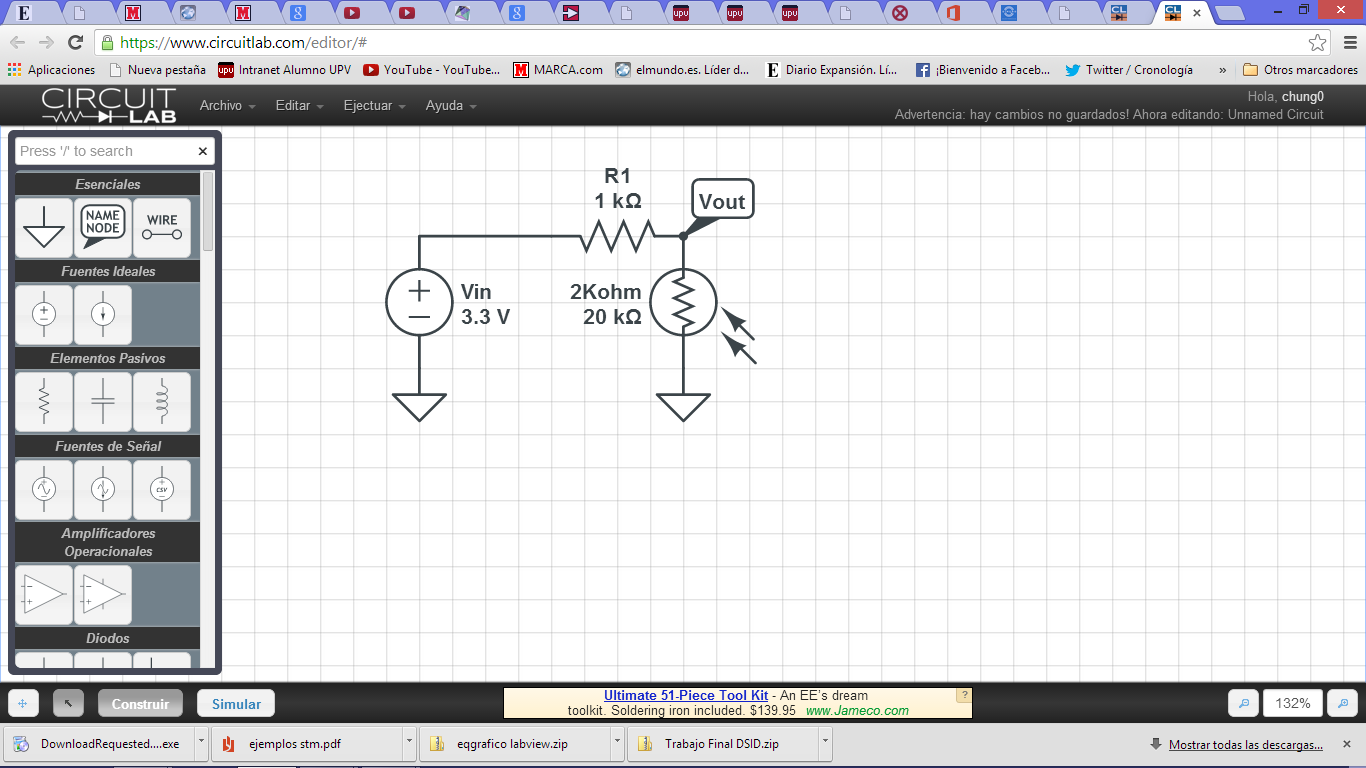
# 2.1.-Diseño del circuito de la Fotoresistencia LDR:

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.





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.

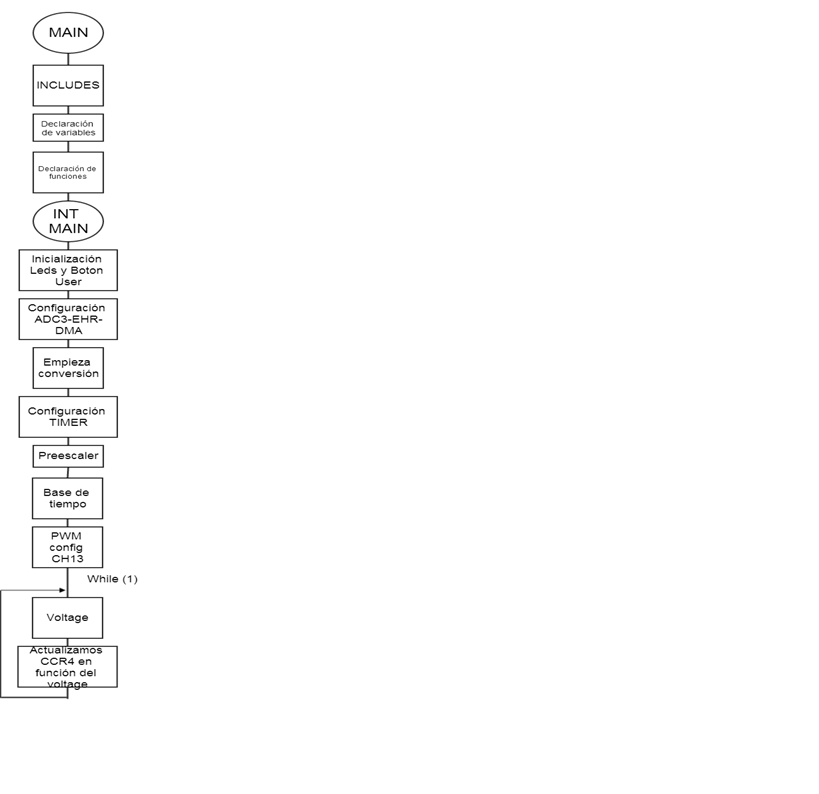
● El reloj del sistema tiene una frecuencia de 144 MHz, APB2 se fija en 72 MHz y el reloj para la conversión ADC toma el valor APB2/2.

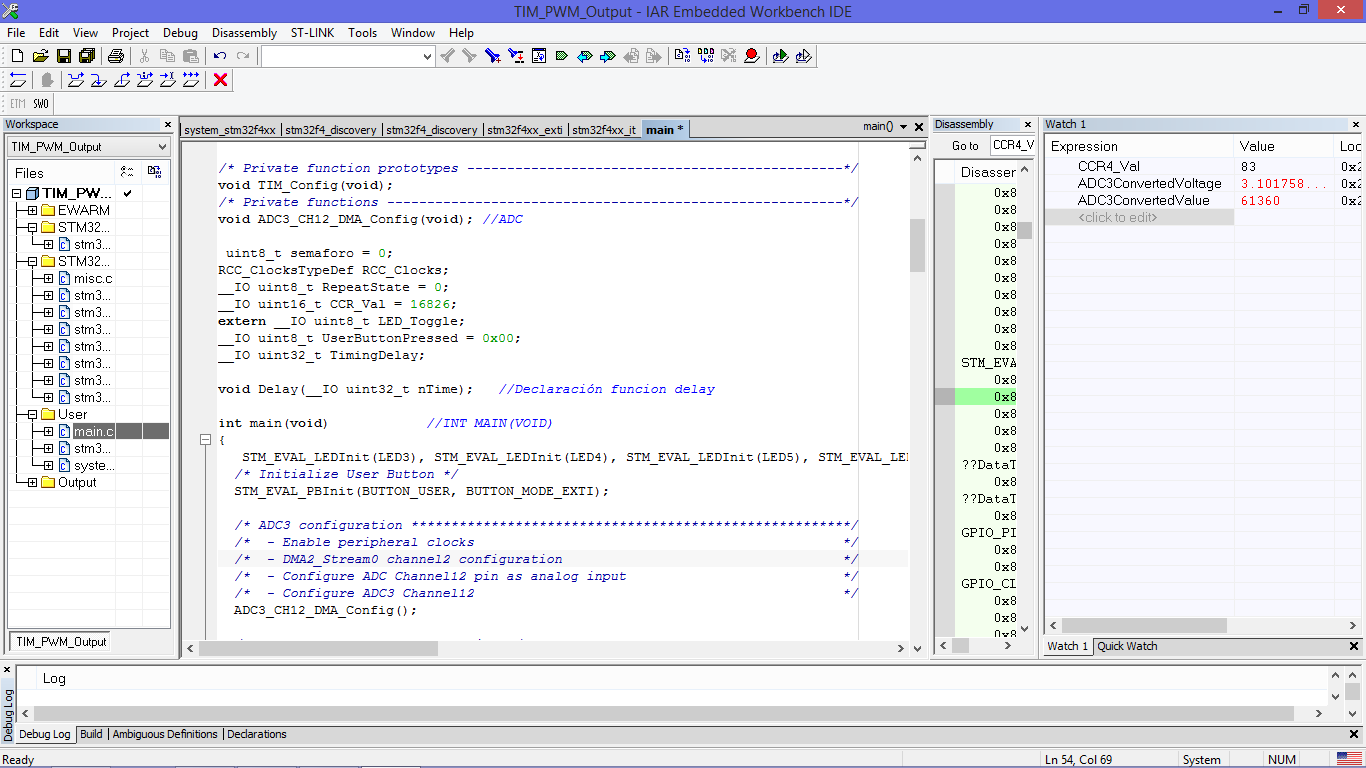
● Dado que el reloj ADC3 es de 36 MHz y el tiempo de muestreo se establece en 3 ciclos, el tiempo de conversión de datos de 12 bits es de 12 ciclos, por lo que el tiempo de conversión total es de (12 +3) / 36 = 0.41 us (2.4 Msps).

**GPIO** Configuramos PC0 como entrada ADC y PBO como salida PWM

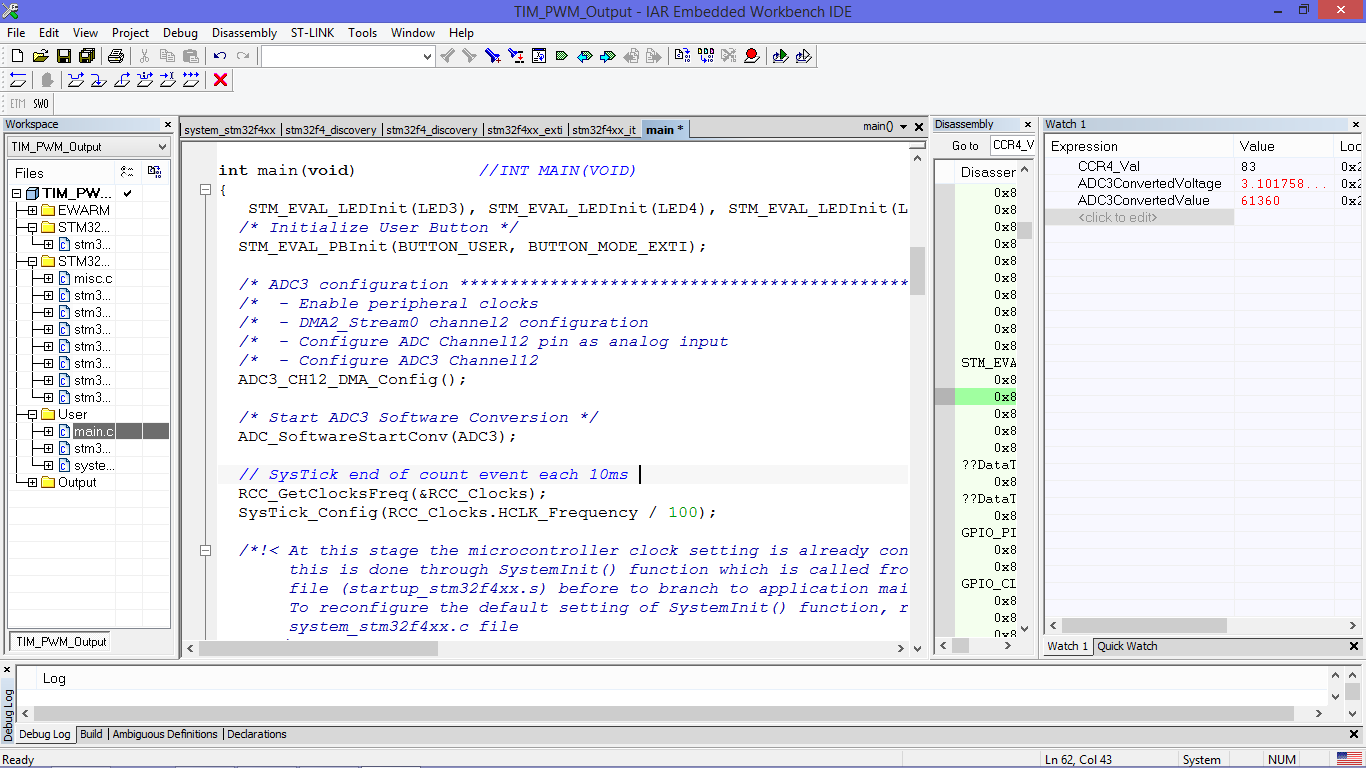
**EXTI** El Modo de control Manual mediante el botón USER produce una interrupción externa y en la rutina de atención a esta es donde se aumentará el ciclo de trabajo cíclicamente cada vez que se pulse el botón USER y la única forma de salir de la rutina de atención es pulsando el botón RESET que activaría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

**2.1-Implementación Software: Flujogramas**

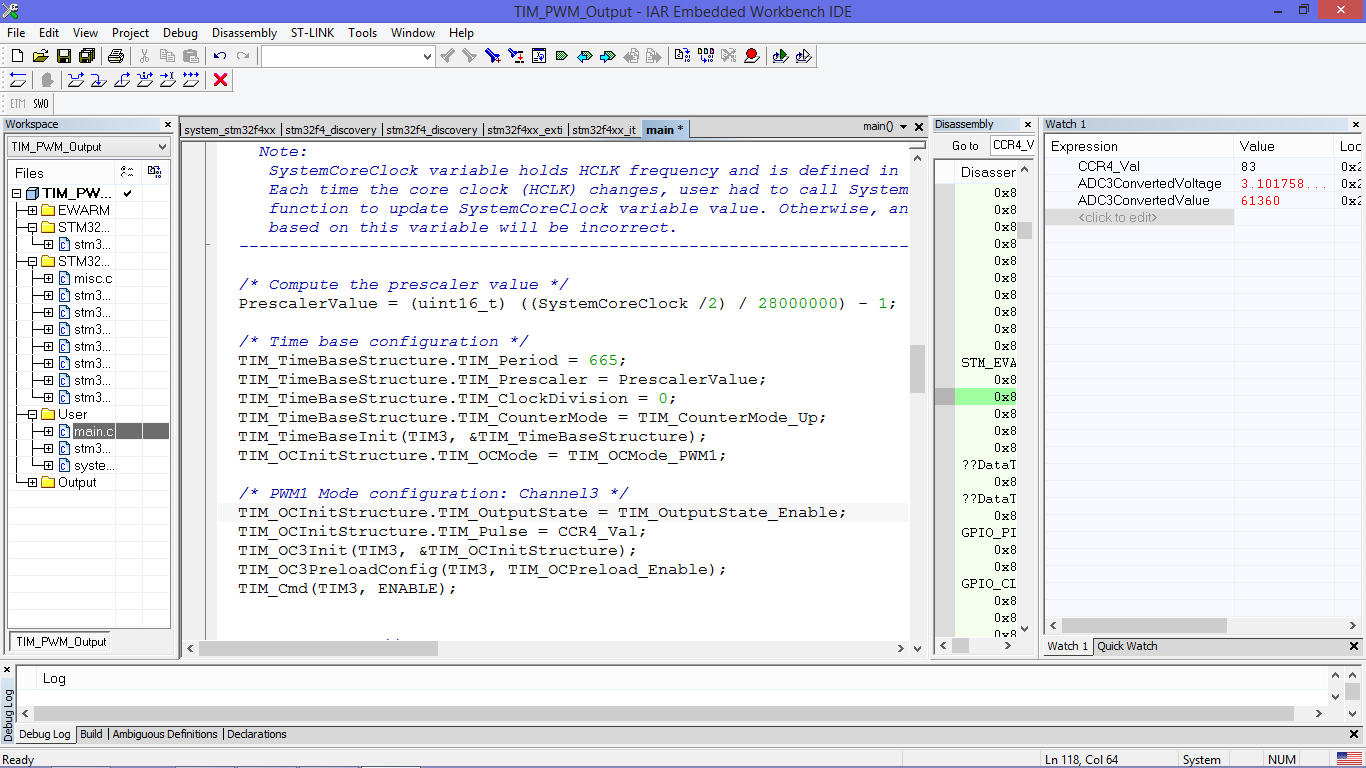
****

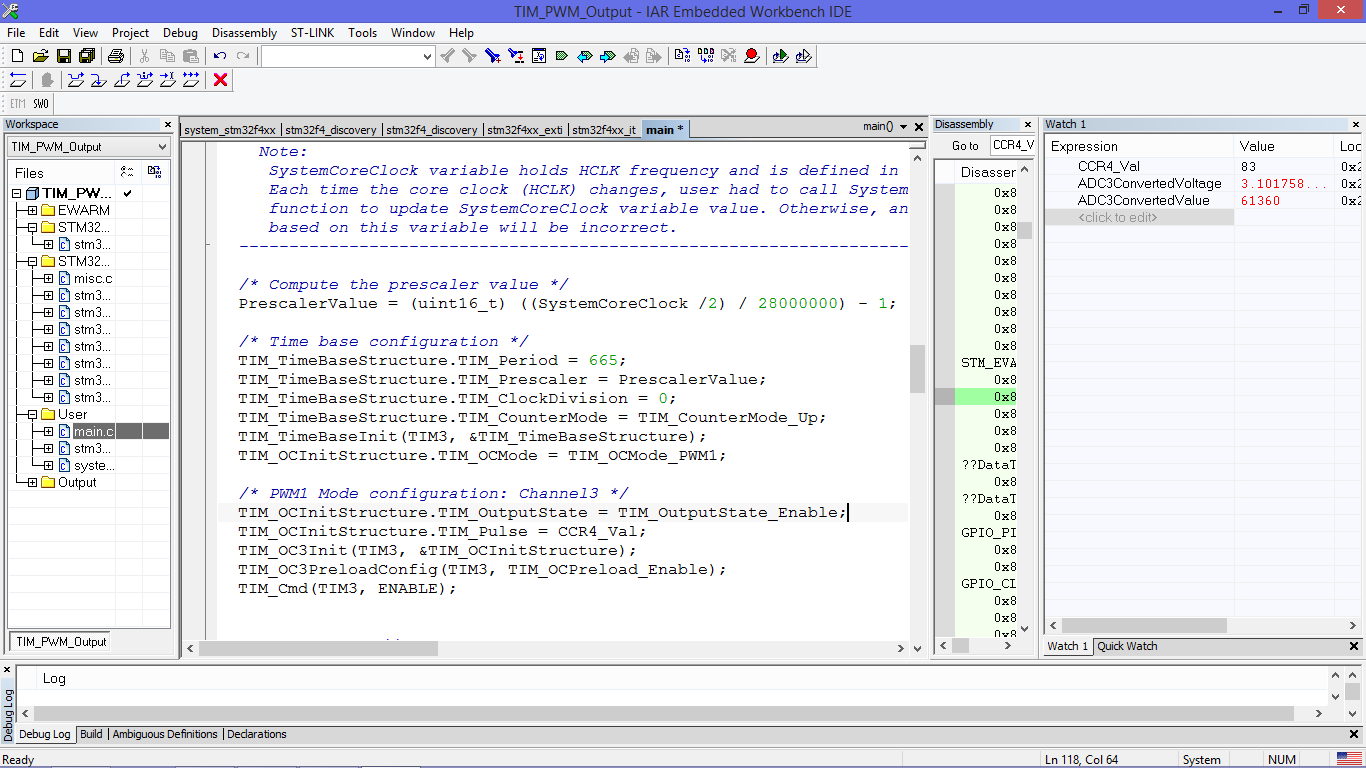


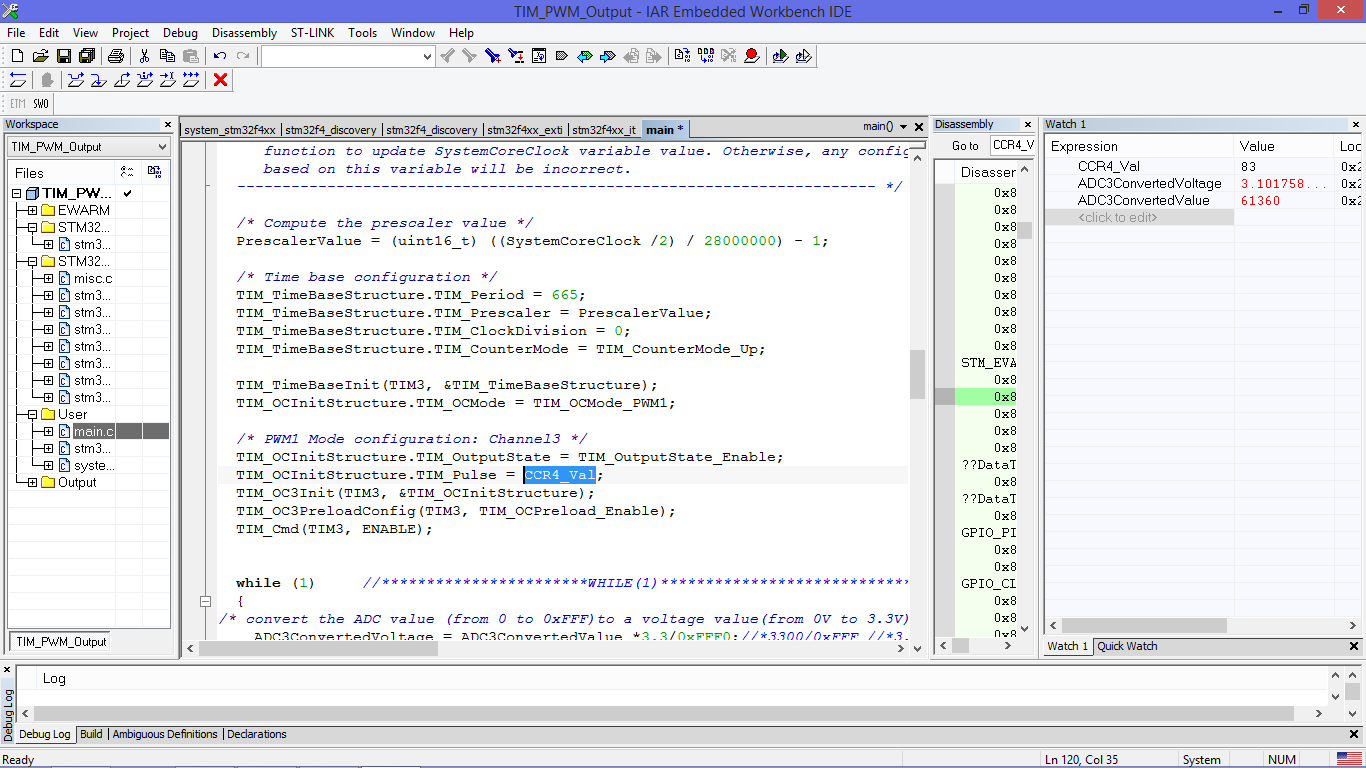
-Puerto C para GPI0 -Entrada ADC3->PC2 -Enable DMA, ADC3 DMA Y ADC3

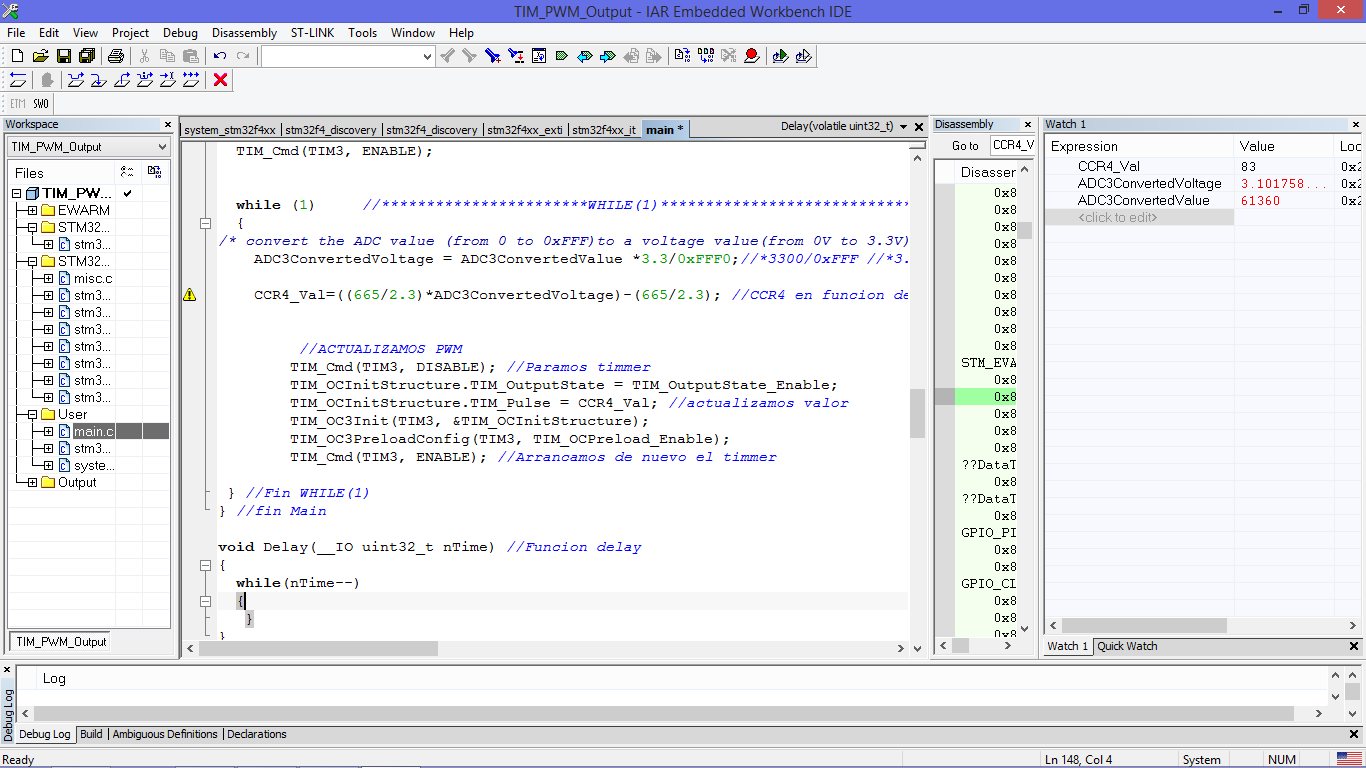


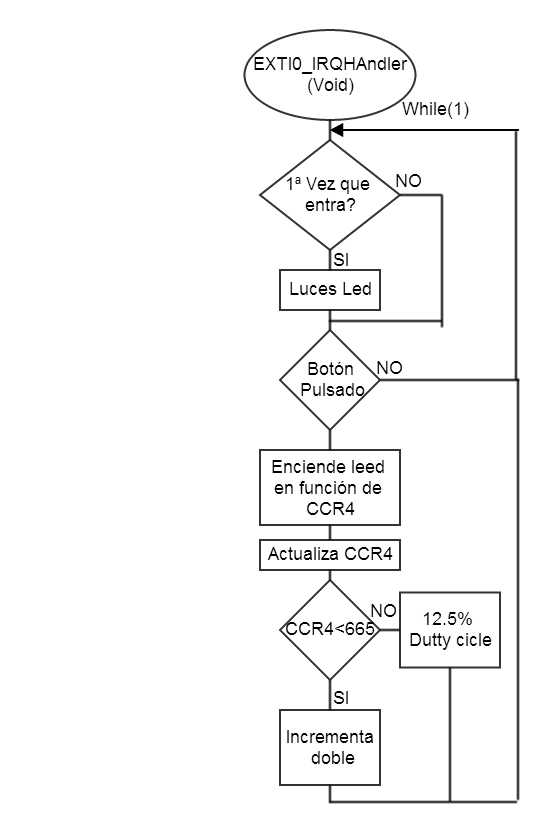
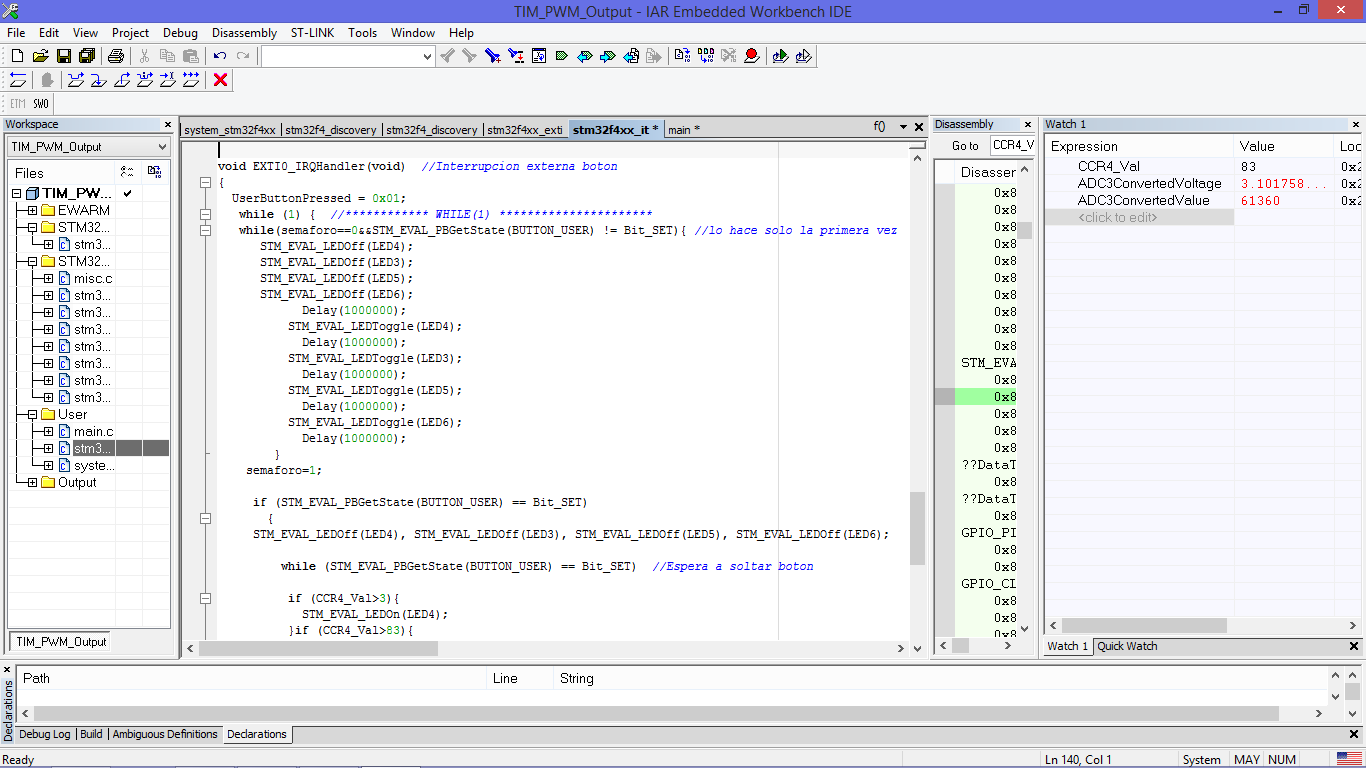
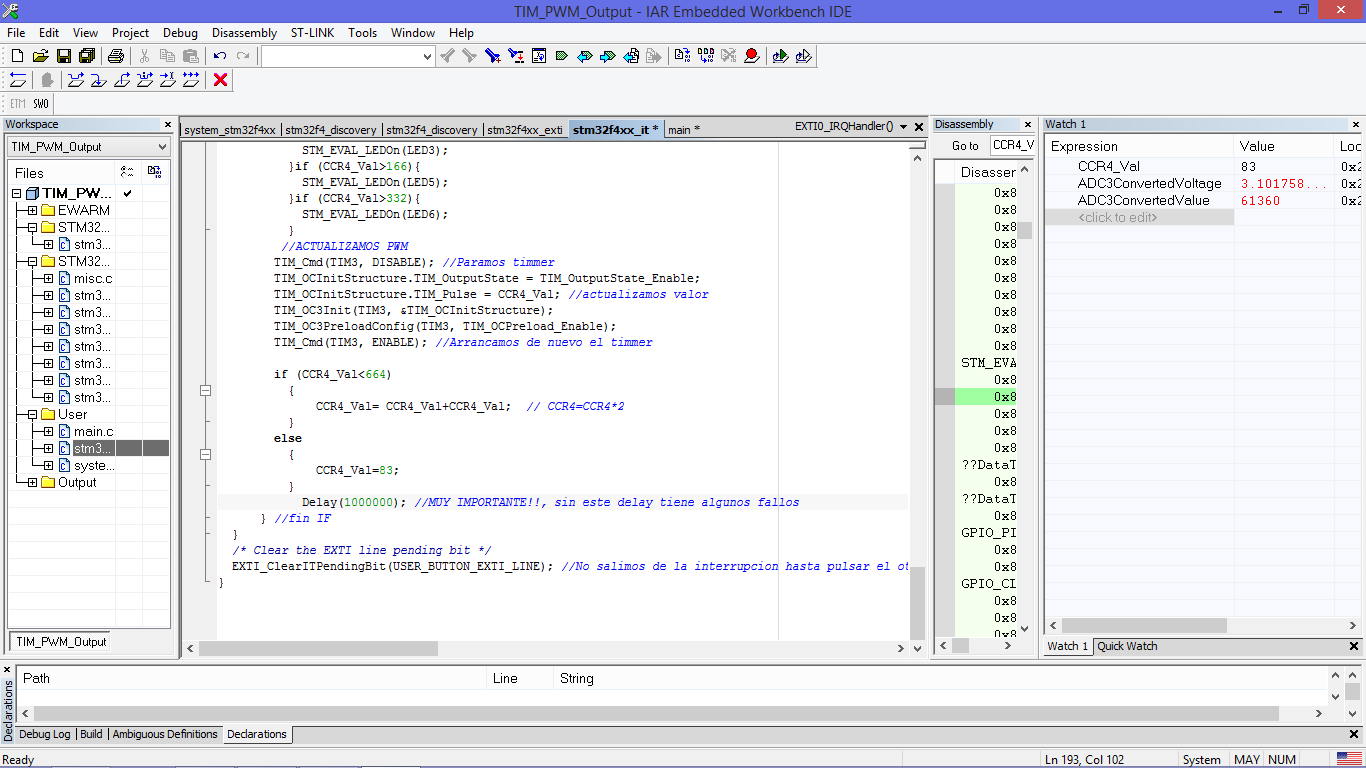
-PB0->Salida -TFM3 to AF2











**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)

}