```
#include "stm32l1xx.h"
#include "stm32_eval.h"
/* Private typedef -----------------------------------------------------------*/
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;
TIM_ICInitTypeDef TIM_ICInitStructure;
ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
/* Private define ------------------------------------------------------------*/
#define ADC1_DR_ADDRESS ((uint32_t)0x40012458)
/* Private variables ---------------------------------------------------------*/
uint16_t CCR1_Val = 125; /* for 25% DUTY CYCLE */
uint16_t CCR2_Val = 250; /* for 50% DUTY CYCLE */
uint16_t CCR3_Val = 400; /* for 80% DUTY CYCLE */
uint16_t CCR4_Val = 500; /* for 100% DUTY CYCLE */
uint16_t PrescalerValue = 0;
__IO uint16_t counter=0, ncount = 0, w=0;
__IO uint16_t i, n=0;
__IO uint32_t ADCdata[];
__IO uint32_t B[];
__IO uint16_t count;

/* Private function prototypes -----------------------------------------------*/
void Dutyset(uint16_t);
void ADC_Config(void);
int max_num(__IO uint32_t ADCdata[]);
void Delay(__IO uint32_t nCount);
/* Private functions ---------------------------------------------------------*/
/**
* @brief Main program
* @param None
* @retval None
*/
int main(void)
{
STM_EVAL_LEDInit(LED1);
STM_EVAL_LEDInit(LED2);
STM_EVAL_LEDInit(LED3);
STM_EVAL_LEDInit(LED4);

/* ---------------------------- System Clocks Configuration -----------------------*/
/* TIM4 clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 | RCC_APB1Periph_TIM4,
ENABLE);
/* GPIOD clock enable */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA | RCC_AHBPeriph_GPIOD,
ENABLE);

/*-------------------------------- GPIO Configuration ----------------------------*/
```

```
/* GPIOD Configuration: Pin PD.15 as PWM Output */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
GPIO_Init(GPIOD, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4); /* GPIO PD15
mapping to TIM4 */

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_40MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_TIM2); /* GPIO PA.01 mapping
to TIM2 */

/* Compute the prescaler value */
PrescalerValue = (uint16_t) (SystemCoreClock / 8000000) - 1; /* Prescaler Value = 3 */

/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 500;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
TIM_ARRPreloadConfig(TIM4, ENABLE);

/* TIM2 Input Capture Configuration */
TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
TIM_ICInitStructure.TIM_ICFilter = 0;
TIM_ICInit(TIM2, &TIM_ICInitStructure);

/* TIM2 Input trigger configuration: External Trigger connected to TI2 */
TIM_SelectInputTrigger(TIM2, TIM_TS_TI2FP2);
TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Gated);

/* Select the Master Slave Mode */
TIM_SelectMasterSlaveMode(TIM2, TIM_MasterSlaveMode_Enable);

/* Master Mode selection: TIM2 */
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Enable);

/* Slave Mode selection: TIM4 */
TIM_SelectInputTrigger(TIM4, TIM_TS_ITR1);
TIM_SelectSlaveMode(TIM4, TIM_SlaveMode_Gated);
```

```
/* TIM2 Enable Counter */
TIM_Cmd(TIM2, ENABLE);

/* TIM4 Enable Counter */
TIM_Cmd(TIM4, ENABLE);

/* Enable the CC2 Interrupt Request */
TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);

/* Enable the TIM4 global Interrupt */
NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

/* 25% Duty Cycle */
Dutyset(CCR1_Val); /* Software Trigger must be enabled only when ADC is ready to
convert */
//Delay(3000);

/* ADC1 channel3 configuration */
ADC_Config();

while(1)
   {
    if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET)
      {
                                        STM_EVAL_LEDToggle(LED2); /* Toggle LED1 */
       Delay(2833); /* Delay for ~1.7 ms */

                                        for(i=0;i<6;i++)
       {

          for(count=0; count < 16; count++) /* For the ADCdata[0] to ADCdata[15]
conversions and averaging into B[i] */
           {

                                                                /* Wait until ADC Channel 3
end of conversion */
              while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET)
               {}

                                                                    ADCdata[count] =
ADC_GetConversionValue(ADC1); /* Read ADC conversion result and clear EOC flag */

                                                                    counter++;

           }

                                                          B[n] = max_num(ADCdata);
```

```
                STM_EVAL_LEDToggle(LED3); /* Toggle LED1 */
                                                        n++;
        }


                                        Delay(2834); /* Delay for ~1.7 ms */
    }
/*
Dutyset(CCR2_Val);
Delay(3000);
Dutyset(CCR3_Val);
Delay(3000);
Dutyset(CCR4_Val);
Delay(3000);
Dutyset(CCR3_Val);
Delay(3000);
Dutyset(CCR2_Val);
Delay(3000);
Dutyset(CCR1_Val);
Delay(3000); */

   STM_EVAL_LEDToggle(LED4); /* Toggle LED1 */
   w++;
  } /* end of while */

 }

int max_num(__IO uint32_t ADCdata[])
 {
   __IO uint32_t j;
   __IO uint32_t m=0;

                for(j=0; j<16 ; j++)
  {
                        if(ADCdata[j] > m)
     m = ADCdata[j];
  }

                return m;
}


/* ------------------------------------ ADC Configuration --------------------------------- */
void ADC_Config(void)
{
/* Enable The HSI (16Mhz) */
RCC_HSICmd(ENABLE);
/* Enable the GPIOA Clock */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
/* Configure PA.3 (ADC Channel3) in analog mode */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
```

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
/* Check that HSI oscillator is ready */
while(RCC_GetFlagStatus(RCC_FLAG_HSIRDY) == RESET);
/* ------------------------------- ADC1 Configuration ------------------------------------------*/
/* Enable ADC1 clock */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
ADC_StructInit(&ADC_InitStructure);
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Falling;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T4_CC4;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC1, &ADC_InitStructure);
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div1;

/* ADC1 regular channel3 configuration */
ADC_RegularChannelConfig(ADC1, ADC_Channel_3, 1, ADC_SampleTime_96Cycles);

/* Enables the ADC1 Power Down during Delay */
ADC_PowerDownCmd(ADC1, ADC_PowerDown_Idle_Delay, ENABLE);

/* Enable ADC1 */
ADC_Cmd(ADC1, ENABLE);

/* Wait until ADC1 ON status */
while (ADC_GetFlagStatus(ADC1, ADC_FLAG_ADONS) == RESET)
{}



}

/* ------------------------------ DUTY CYCLE Configuration ------------------------------ */
void Dutyset(uint16_t CCRx_Val)
{
/* PWM1 Mode configuration: TIM4 Channel4 */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCRx_Val;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC4Init(TIM4, &TIM_OCInitStructure);
TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);
}
/* ------------------------------------DELAY in ms ------------------------------------------ */
void Delay(__IO uint32_t nCount) /* 0.607 µs Delay for nCount=1 */
{
__IO uint32_t index = 0;
```

```
for(index = (nCount); index != 0; index--)
{}
}
```

INTERRUPT SERVICE ROUTINE

```
/**
******************************************************************************
****
* @file TIM/InputCapture/stm32l1xx_it.c
* @author MCD Application Team
* @version V1.0.0
* @date 31-December-2010
* @brief Main Interrupt Service Routines.
* This file provides template for all exceptions handler and
* peripherals interrupt service routine.
******************************************************************************
****
* @attention
*
* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT
PROVIDING CUSTOMERS
* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER
FOR
THEM TO SAVE
* TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE
FOR ANY
* DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO
ANY
CLAIMS ARISING
* FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY
CUSTOMERS OF THE
* CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH
THEIR
PRODUCTS.
*
* <h2><center>&copy; COPYRIGHT 2010 STMicroelectronics</center></h2>
******************************************************************************
****
*/
/* Includes ------------------------------------------------------------*/
#include "stm32l1xx_it.h"
#include "stm32_eval.h"
/** @addtogroup STM32L1xx_StdPeriph_Examples
* @{
*/
```

```
/** @addtogroup TIM_Input_Capture
* @{
*/
__IO uint16_t tset=0;
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/


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


/*
*******************************************************************************
***/
/* Cortex-M3 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)
{}
/*
******************************************************************************
```

```
***/
/* STM32L1xx Peripherals Interrupt Handlers */
/*
******************************************************************************
***/
/**
* @brief This function handles TIM4 global interrupt request.
* @param None
* @retval None
*/
void TIM2_IRQHandler(void)
{
if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET)
{
STM_EVAL_LEDToggle(LED1); /* Toggle LED1 */

/* Clear TIM4 Capture compare interrupt pending bit */
TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
tset++;
}

}
/*
******************************************************************************
***/
/* STM32L1xx Peripherals Interrupt Handlers */
/* Add here the Interrupt Handler for the used peripheral(s) (PPP), for the */
/* available peripheral interrupt handler's name please refer to the startup */
/* file (startup_stm32l1xx_xx.s). */
/*
******************************************************************************
***/
/**
* @brief This function handles PPP interrupt request.
* @param None
* @retval None
*/
/*void PPP_IRQHandler(void)
{}
*/
/**
* @}
*/
/**
* @}
*/
/******************* (C) COPYRIGHT 2010 STMicroelectronics *****END OF
FILE****/
```