**定时器与中断实验报告**

**中英1702 菲华·帕兰斯 U201711458**

**一．实验目的**

1. 学习运用定时器及输入捕获功能，产生脉冲并进行脉冲宽度的测量；
2. 掌握定时器和GPIO有关寄存器的初始化和配置；

3、加深理解定时器和中断的概念，学习中断服务函数的编写。

**二．实验要求**

1、 选用GPIO某端口为输出，用中断方式产生一个周期性的高电平脉冲，脉冲宽度范围自定；

2、 采用定时器的输入捕获功能对该脉冲宽度进行测量，并利用串口调试助手确认测量结果；

3、 采用信号发生器输出相同幅值和对应宽度的高电平脉冲，接到同一输入端，改变脉冲宽度，测量3-5组数据，比较并分析测量结果；

4、 利用LCD显示测量所得的脉宽值；并确认可测脉冲宽度的范围。

**三．程序流程图**



**四．程序清单及注释**

（word排版较为困难，老师如果难以看清代码可以在这里看一下，比较清晰，

<https://github.com/OverActiveRX8/STM32-Projects/tree/master/TimerDemo/Core/Src>)

**1.main.c**

#include "main.h"

#include "tim.h"

#include "gpio.h"

//Personal Variables

int icValue = 0; //IC Value

uint8\_t olCounter = 0; //Timer Overload Counter

int pulsePeriod = 10; //Initial Output Pulse Width (\*10us)

//LVGL Input TextArea Object

extern lv\_obj\_t \*input\_ta;

//LVGL Driver Variables

static lv\_disp\_buf\_t disp\_buf;

static lv\_color\_t buf[LV\_HOR\_RES\_MAX \* 10];

lv\_disp\_drv\_t disp\_drv;

lv\_indev\_drv\_t indev\_drv;

//HAL CLK Function Declaration

void SystemClock\_Config(void);

//LVGL Driver Function Declarations

void my\_disp\_flush(lv\_disp\_t \* disp, const lv\_area\_t \* area, lv\_color\_t \* color\_p);

bool my\_touchpad\_read(lv\_indev\_drv\_t \* indev\_driver, lv\_indev\_data\_t \* data);

/\*\*

\* @brief: Main Function

\*/

int main(void)

{

//System Initializations

HAL\_Init();

SystemClock\_Config(); //CLK

MX\_GPIO\_Init(); //GPIO

MX\_TIM2\_Init(); //Timers

MX\_TIM7\_Init();

MX\_TIM6\_Init();

MX\_TIM14\_Init();

//TFTLCD Initializations

delay\_init(168); //Delay Functions

uart\_init(115200); //UART

usmart\_dev.init(84); //USMART

LCD\_Init(); //LCD

tp\_dev.init(); //Touch Screen

LCD\_Clear(WHITE); //Clear LCD

lv\_init(); //Lvgl

//LVGL Driver Initializations

lv\_disp\_buf\_init(&disp\_buf, buf, NULL, LV\_HOR\_RES\_MAX \* 10); /\*Initialize the display buffer\*/

lv\_disp\_drv\_init(&disp\_drv); /\*Basic initialization\*/

disp\_drv.flush\_cb = my\_disp\_flush; /\*Set your driver function\*/

disp\_drv.buffer = &disp\_buf;

/\*Assign the buffer to the display\*/

lv\_disp\_drv\_register(&disp\_drv);

/\*Finally register the driver\*/

lv\_indev\_drv\_init(&indev\_drv);

/\*Descriptor of a input device driver\*/

indev\_drv.type = LV\_INDEV\_TYPE\_POINTER;

/\*Touch pad is a pointer-like device\*/

indev\_drv.read\_cb = my\_touchpad\_read;

/\*Set your driver function\*/

lv\_indev\_drv\_register(&indev\_drv);

/\*Finally register the driver\*/

//Start Timer IT

HAL\_TIM\_Base\_Start\_IT(&htim7);

HAL\_TIM\_Base\_Start\_IT(&htim14);

HAL\_TIM\_IC\_Start\_IT(&htim2,TIM\_CHANNEL\_1);

//Display Screen

displayScr();

//Loop

while (1)

{

lv\_ta\_set\_text(input\_ta, int2String(icValue, intSize(icValue), "us")); //Print IC Value onto LCD (by LVGL)

printf("Width: %d us\r\n", icValue);

HAL\_Delay(500); //Delay 500ms

}

}

/\*\*

\*@brief System CLK Configuration

\*/

void SystemClock\_Config(void){

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 8;

RCC\_OscInitStruct.PLL.PLLN = 168;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief: LVGL Display Driver Function

\*/

void my\_disp\_flush(lv\_disp\_t \* disp, const lv\_area\_t \* area, lv\_color\_t \* color\_p)

{

int32\_t x, y;

for(y = area->y1; y <= area->y2; y++) {

for(x = area->x1; x <= area->x2; x++) {

LCD\_Fast\_DrawPoint(x,y,color\_p->full);

color\_p++;

}

}

lv\_disp\_flush\_ready((lv\_disp\_drv\_t \*)disp);

}

/\*\*

\* @brief: LVGL Input Driver Function

\*/

bool my\_touchpad\_read(lv\_indev\_drv\_t \* indev\_driver, lv\_indev\_data\_t \* data)

{

static lv\_coord\_t last\_x = 0;

static lv\_coord\_t last\_y = 0;

tp\_dev.scan(0);

data->state = tp\_dev.sta&TP\_PRES\_DOWN ? LV\_INDEV\_STATE\_PR : LV\_INDEV\_STATE\_REL;

if(data->state == LV\_INDEV\_STATE\_PR)

{

last\_x = tp\_dev.x[0];

last\_y = tp\_dev.y[0];

}

data->point.x = last\_x;

data->point.y = last\_y;

return false;

}

/\*\*

\* @brief: HAL Error Handles

\*/

void Error\_Handler(void){}

#ifdef USE\_FULL\_ASSERT

void assert\_failed(uint8\_t \*file, uint32\_t line){}

#endif /\* USE\_FULL\_ASSERT \*/

**2.main.h**

#ifndef \_\_MAIN\_H

#define \_\_MAIN\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include "stm32f4xx\_hal.h"

#include "sys.h"

#include "delay.h"

#include "lcd.h"

#include "touch.h"

#include "usmart.h"

#include "usart.h"

#include "stdbool.h"

#include "lvgl.h"

#include "screen.h"

void Error\_Handler(void);

#ifdef \_\_cplusplus

}

#endif

#endif /\* \_\_MAIN\_H \*/

**3.** **stm32f4xx\_it.c (只给出了定时器及IC中断回调函数部分代码)**

void TIM8\_TRG\_COM\_TIM14\_IRQHandler(void)

{

//LVGL Required Timer ITs

lv\_tick\_inc(10);

lv\_task\_handler();

HAL\_TIM\_IRQHandler(&htim14);

}

void TIM6\_DAC\_IRQHandler(void)

{

HAL\_TIM\_IRQHandler(&htim6);

olCounter ++; //Timer Overload Counter Add By 1 After IT

}

void TIM7\_IRQHandler(void)

{

HAL\_TIM\_IRQHandler(&htim7);

timerCounter ++;

if(timerCounter == pulsePeriod)

HAL\_GPIO\_TogglePin(GPIOE,GPIO\_PIN\_1);

timerCounter = 0;

}

}

void HAL\_TIM\_IC\_CaptureCallback(TIM\_HandleTypeDef \*htim)

{

if(htim->Channel == HAL\_TIM\_ACTIVE\_CHANNEL\_1)

{

if(!icState)

{

\_\_HAL\_TIM\_SetCounter(&htim2, 0); HAL\_TIM\_Base\_Start\_IT(&htim6); TIM\_RESET\_CAPTUREPOLARITY(&htim2,TIM\_CHANNEL\_1);

TIM\_SET\_CAPTUREPOLARITY(&htim2,TIM\_CHANNEL\_1,TIM\_ICPOLARITY\_FALLING);

icState = true; }

else

{

HAL\_TIM\_Base\_Stop\_IT(&htim6); \_\_HAL\_TIM\_SetCounter(&htim6, 0);

icValue = 0;

TIM\_RESET\_CAPTUREPOLARITY(&htim2,TIM\_CHANNEL\_1);

TIM\_SET\_CAPTUREPOLARITY(&htim2,TIM\_CHANNEL\_1,TIM\_ICPOLARITY\_RISING);

icValue += HAL\_TIM\_ReadCapturedValue(&htim2, TIM\_CHANNEL\_1);

icValue += olCounter \* 0xFFFF;

icValue \*= 2;

olCounter = 0; icState = false;

}

}

}

**五．实验结果及分析**

**实验结果：**

在本次试验中，我实现了GPIO输出高电平脉宽功能，同时可调节脉宽。其中计数器使用32位，定时器周期为10us，故理论上可产生10us-42949672950us脉宽，实际上发现超过1s之后脉宽已经不够稳定，故可认为实现了可产生10us-1s脉宽的功能。在测量方面，计时器采用1us周期，并利用双计时器的方式解决了计数溢出问题，理论上可测量相当长的脉宽，实际上宽度超过1s之后由于噪声等原因测量不够精准，可认为测量宽度为1us-1s。此外，我成功移植了LCD驱动及GUI库LVGL，实现了LCD显示脉宽和触摸调节输出宽度并让LCD显示了一个较为美观的界面。

**实验分析：**

1.输出脉宽部分

在输出脉宽时，选择正确的定时器周期比较重要，定时器周期决定了输出脉冲的可调范围 ，过小导致范围窄，过大导致调整间隔太大，故应综合多方面考虑选择合适的周期。

2.测量脉宽部分

输入捕获测量脉冲需要注意两个方面问题，一个是噪声处理，一个是计数溢出，噪声处理可以通过调节STM32自带的输入捕获滤波寄存器的值来实现，我使用的值是0100，目前看效果比不加滤波有很大提升，计数问题则可采用另加一个计数器记录溢出次数的方式来解决，好处是可测量周期长的信号，坏处是占用了更多资源。在比较测量外部和内部产生信号的过程中，发现外部信号的噪声更大一些，可能需要提高噪声过滤等级来保证计数值可靠性。

3显示部分

显示时应该采取一个合适的延时时间保证现实的结果清晰易读，我选择了500ms的延时，基本可以达到要求。

**六．对实验的建议**

无