Homework 3

Part 1

Write out the registers that are used (configurated or read) and related with the Basic TIM by the project `LED_BasicTImer` and their meanings.

```
void BASIC_TIM_Init(void)
{
    // 1 开启定时器时钟,即内部时钟CK_INT=72M
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,ENABLE);
    // 2 自动重装载寄存器的值
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_TimeBaseStructure.TIM_Period = BASIC_TIM_Period;
    // 时钟预分频数
    TIM_TimeBaseStructure.TIM_Prescaler= BASIC_TIM_Prescaler;
    // 初始化定时器
    TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);

    // 3 NVIC 中断配置   定时器相应中断配置
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM6_IRQn ;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    //清除计数器中断标志位
    TIM_ClearFlag(TIM6, TIM_FLAG_Update);
    //开启计数器中断
    TIM_ITConfig(TIM6,TIM_IT_Update,ENABLE);
    // 4 使能计数器
    TIM_Cmd(TIM6, ENABLE);
}
```

1. 计数器寄存器(`TIMx_CNT`)

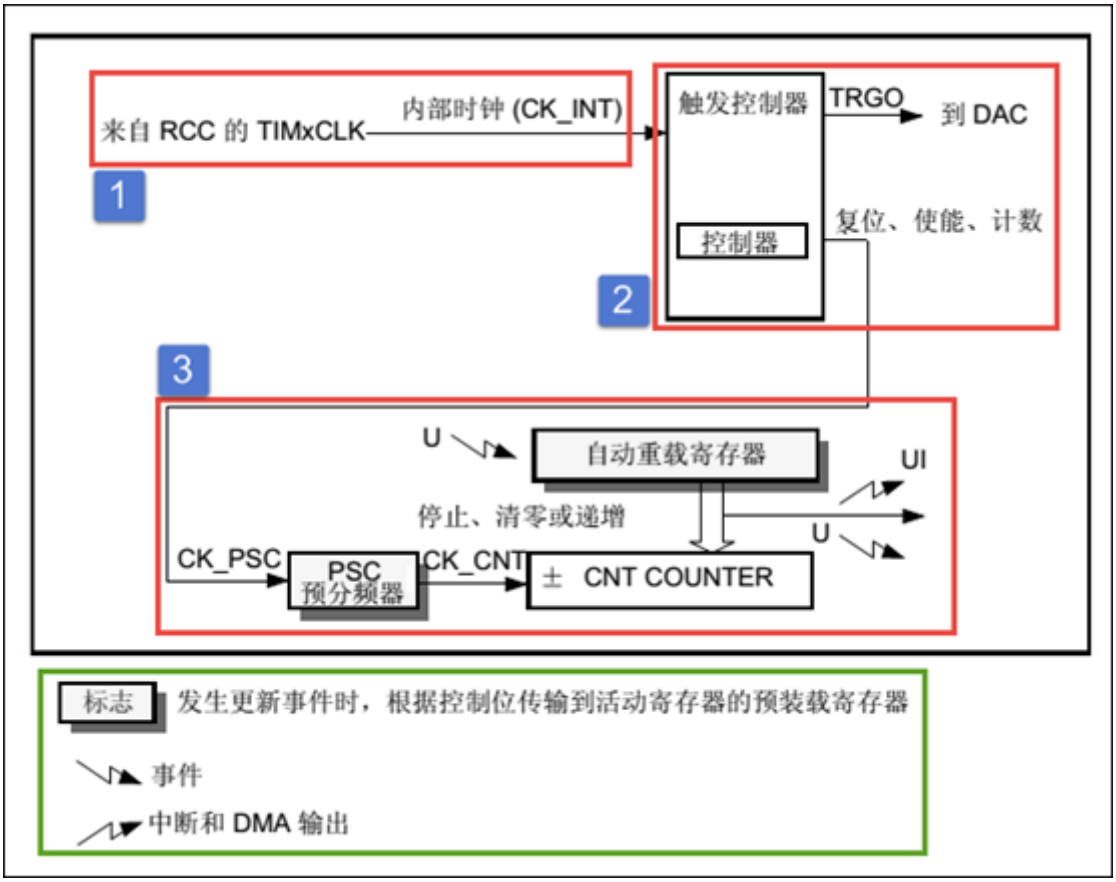
   在自动重载寄存器(`TIMx_ARR`)添加一个计数值后并使能`TIMx`，计数寄存器(`TIMx_CNT`)就会从0开始递增，当`TIMx_CNT`的数值与`TIMx_ARR`值相同时就会生成事件并把`TIMx_CNT`寄存器清0，完成一次循环过程。

2. 预分频器寄存器(`TIMx_PSC`)

   通过设置预分频器PSC的值可以得到不同的CK_CNT，计算公式：`fCK_CNT = fCK_PSC/(PSC[15:0]+1)`

3. 自动重载寄存器(`TIMx_ARR`)

   time for `cnt` from 0 -> ARR: `(ARR+1)*(psc+1)/fck_psc`



Part 2

Program to realize: PA4 output 100Hz square wave

`square.h`

```
#ifndef __SQUARE_H
#define __SQUARE_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32f10x.h"

#define WAVE_GPIO_CLK RCC_APB2Periph_GPIOA
#define WAVE_GPIO_Pin GPIO_Pin_4
#define WAVE_GPIO_PORT GPIOA

void WAVE_GPIO_Config(void);
void WAVE_0(void);
void WAVE_1(void);

#ifdef __cplusplus
}
#endif

#endif
```

square.c

```
#include "square.h"

void WAVE_GPIO_Config(void) {
    GPIO_InitTypeDef GPIO_InitStruct;
    RCC_APB2PeriphClockCmd(WAVE_GPIO_CLK, ENABLE);
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStruct.GPIO_Pin = WAVE_GPIO_Pin;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_Init(WAVE_GPIO_PORT, &GPIO_InitStruct);
}

void WAVE_0(void) {
    GPIO_ResetBits(WAVE_GPIO_PORT, WAVE_GPIO_Pin);
}

void WAVE_1(void) {
    GPIO_SetBits(WAVE_GPIO_PORT, WAVE_GPIO_Pin);
```

basic_timer.h

```
#ifndef __BASIC_TIMER_H
#define __BASIC_TIMER_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32f10x.h"
#include "square.h"
#include "led.h"

#define            BASIC_TIM_Period            (5000-1)
#define            BASIC_TIM_Prescaler         (72-1)
#define           BASIC_TIM                   TIM6

void BASIC_TIM_Init(void);

#ifdef __cplusplus
}
#endif

#endif
```

basic_timer.c

```
#include "basic_timer.h"

void BASIC_TIM_Init(void)
{

    // 1 开启定时器时钟,即内部时钟CK_INT=72M
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,ENABLE);
```

```
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    // 2 自动重装载寄存器的值，累计TIM_Period+1个频率后产生一个更新或者中断
    TIM_TimeBaseStructure.TIM_Period = BASIC_TIM_Period;

    // 时钟预分频数为
    TIM_TimeBaseStructure.TIM_Prescaler = BASIC_TIM_Prescaler;

    // 初始化定时器
    TIM_TimeBaseInit(BASIC_TIM, &TIM_TimeBaseStructure);

    // 3 NVIC 中断配置   定时器相应中断配置

    NVIC_InitTypeDef NVIC_InitStructure;
    // 设置中断来源
    NVIC_InitStructure.NVIC_IRQChannel = TIM6_IRQn ;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // 清除计数器中断标志位
    TIM_ClearFlag(BASIC_TIM, TIM_FLAG_Update);

    // 开启计数器中断
    TIM_ITConfig(BASIC_TIM,TIM_IT_Update,ENABLE);

    // 4 使能计数器
    TIM_Cmd(BASIC_TIM, ENABLE);
}

void TIM6_IRQHandler(void)
{
    static u8 i =0 ;
    if (TIM_GetITStatus(BASIC_TIM, TIM_IT_Update) != RESET)
    {
        switch (i)
        {
            case 0:GPIO_ResetBits(WAVE_GPIO_PORT, WAVE_GPIO_Pin);i++;break;
            case 1:GPIO_SetBits(WAVE_GPIO_PORT, WAVE_GPIO_Pin);i=0;break;
        }

        TIM_ClearITPendingBit(BASIC_TIM, TIM_IT_Update);

    }

}
```

main.cpp

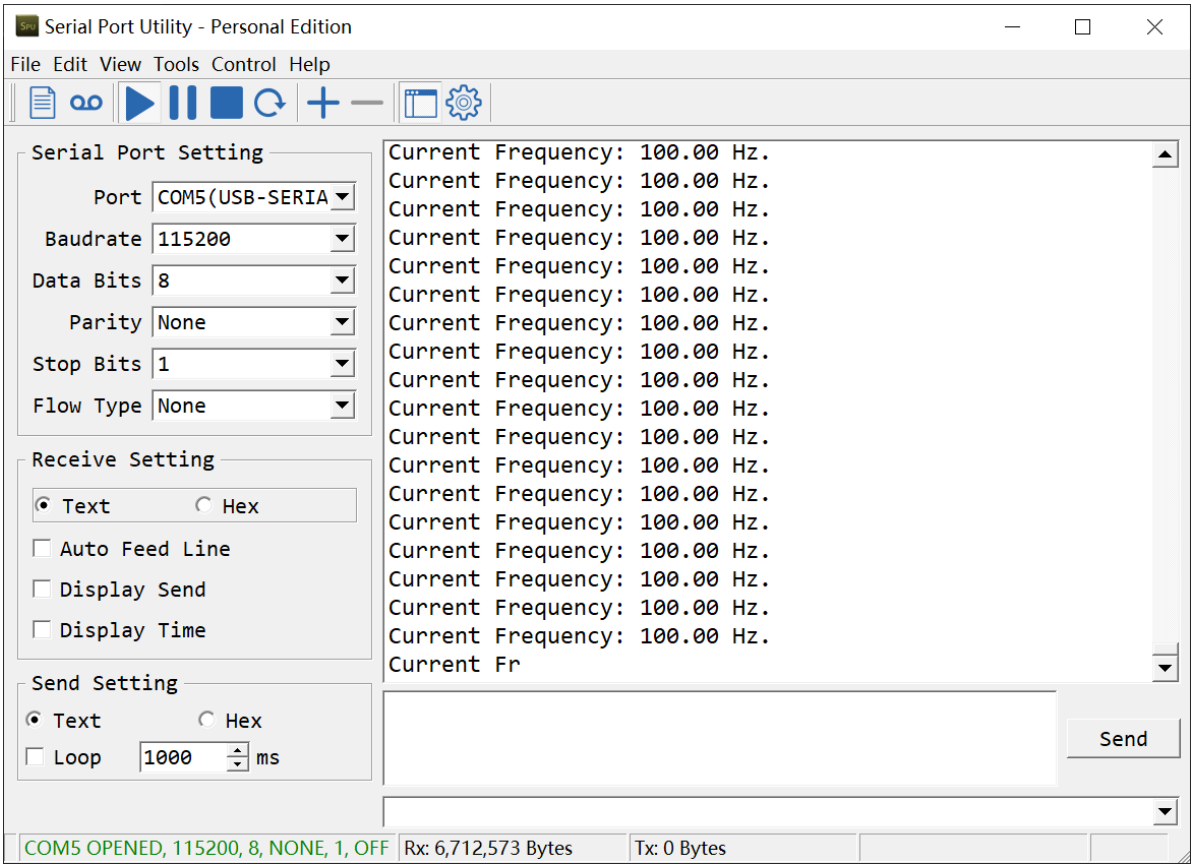
```
void Main() {
    LED_GPIO_Config();
    KEY_GPIO_Config();
    WAVE_GPIO_Config();
    BASIC_TIM_Init();
}
```

效果：把接口换到LED上测试没有问题，之后在下一题中也能够正常工作。

Part 3

Complete the project that is to measure the frequency of the input signal based on the unfinished project on BB. Sort out the code flow chart.

串口检测效果如图：



debug的时候发现了很多问题，一开始串口没有显示数据，大概有这样几个问题需要记录：

1. ```
   char msg[50] = {0};
   sprintf(msg, "Current Frequency: %.2lf Hz.\n", Frequency_value);
   Usart_SendString(DEBUG_USARTx, msg);
   ```

   使用交叉编译器时 `printf` 函数重定向有问题，不能在串口正常发送，所以需要使用 `sprintf`；

2. 因为在 `CLion` 上重新编写了代码，没有发现项目里老师原给了中断处理函数，通过猜测自己编写了中断处理函数，然后串口的中断忘加到 it 文件里了，导致串口没法用。要注意中断函数的完整性；

3. 连线时一次性插了一排线，然后不知道哪个GPIO口的复用在程序中被影响了，导致串口输出的Hz为0（已知输出方波的功能正常）。目前只连接PA4→PD2没有问题，其他端口在程序中的影响还没有排查。以后要注意最小系统问题；
4. 发现 `key` 函数没有被正常扫描跳转，而是用作开启 `while(1)` 循环，经过检查发现以前写的程序里按键的高低电位反了，flag起始标志位也反了，但由于main函数里没有其他内容，恰巧能够承担需
   ```

要的功能，key文件已经调整正常了，之前的作业应该也是反着用的，希望老师轻锤（×

Code flow chart:

在main程序中先进行各个GPIO端口、TIM、NVIC服务和串口服务的初始化→
进行循环扫描，同时PA4端口不断输出方波信号传递到PD2端口→
PD2端口作为TIM3的ETR端口，接收到的上升沿信号被记录到TIM3的计数器中，通过`TIM_GetCounter(TIM3)`调用→
TIM2每发生一次中断，启动TIM2服务函数，通过两次中断间的计数得出频率并储存到变量`Frequency_value`中→|
在main程序的输出msg中变量`Frequency_value`被调用，msg发送到串口→
其中，串口消息发送时调用串口的中断服务。

以下是主要代码：

```
float Frequency_value;

void Main() {

    LED_GPIO_Config();
    KEY_GPIO_Config();
    WAVE_GPIO_Config();
    BASIC_TIM_Init();
    LED_OFF();

    // 设置中断组为0
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    GENERAL_TIM_Init();
    USART_Config();
//    Usart_SendString( DEBUG_USARTx,"\nHello!\n");

    while (1) {
        char msg[50] = {0};
        sprintf(msg, "Current Frequency: %.2lf Hz.\n", Frequency_value);
        Usart_SendString(DEBUG_USARTx, msg);

        KEY_Scan();
        if (flag == 1) { LED_OFF(); }
        if (flag == 0) { LED_ON(); }

    }

    // TIM3-ETR-PD2 引脚配置
static void GENERAL_TIM_GPIO_Config()
{
    GPIO_InitTypeDef GPIO_InitStruct;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_Init(GPIOD, &GPIO_InitStruct);

}

    // TIM2相关配置
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
```

```
    TIM_TimeBaseStructure.TIM_Period = (20000 - 1);
    TIM_TimeBaseStructure.TIM_Prescaler = (720 - 1);
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    TIM_ETRClockMode2Config(TIM3, TIM_ExtTRGPSC_OFF,
TIM_ExtTRGPolarity_Inverted,0);


    // TIM3相关配置
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_TimeBaseStructure.TIM_Period = 0xffff-1;
    TIM_TimeBaseStructure.TIM_Prescaler= 0x00;
    TIM_TimeBaseStructure.TIM_ClockDivision=0x0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
```