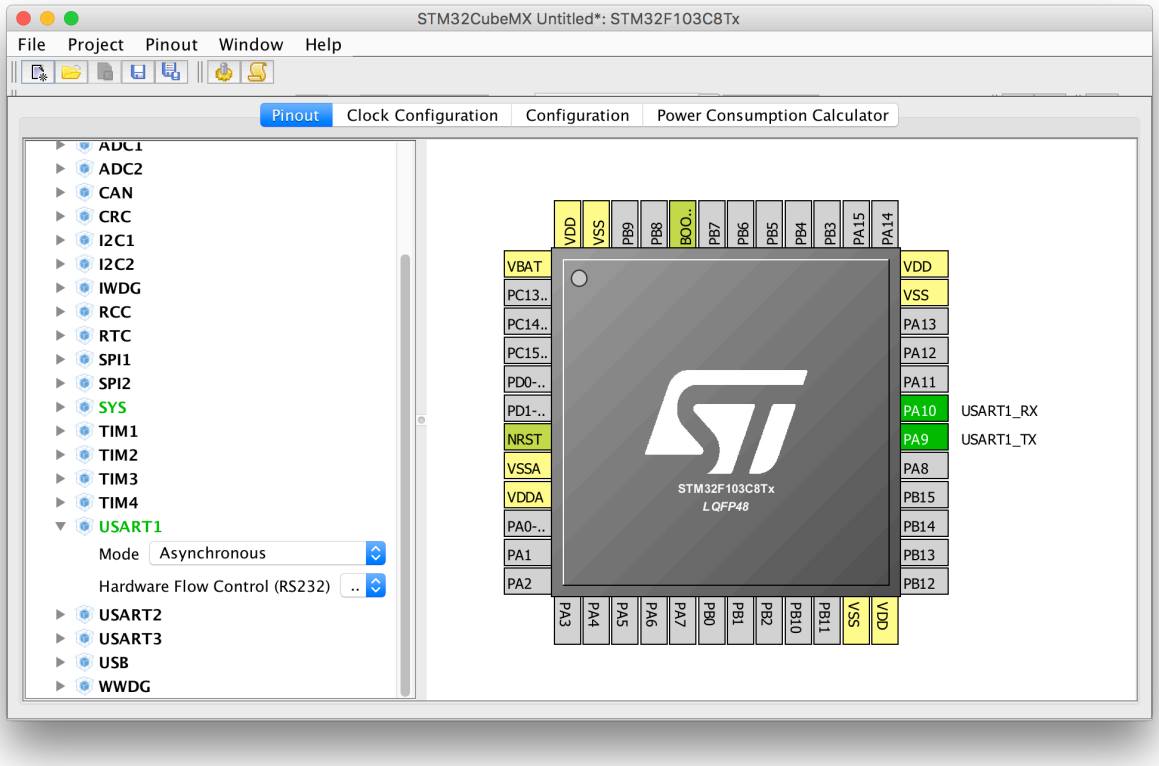
Lab 3：自行车码表

http://helloyym.me/2016/04/17/Lab3/

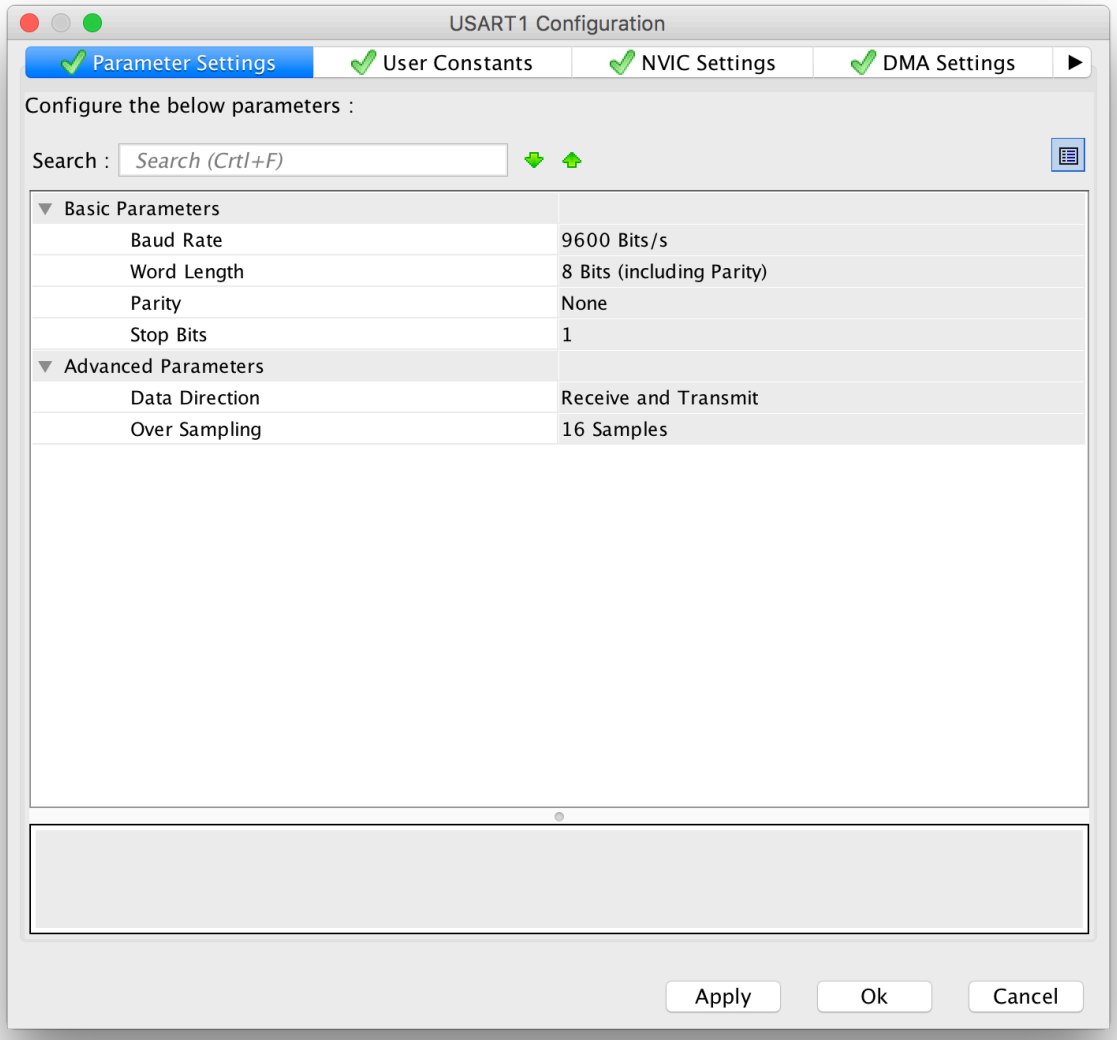
建立工程

打开 STM32CubeMX，选择对应的开发板型号，新建工程。

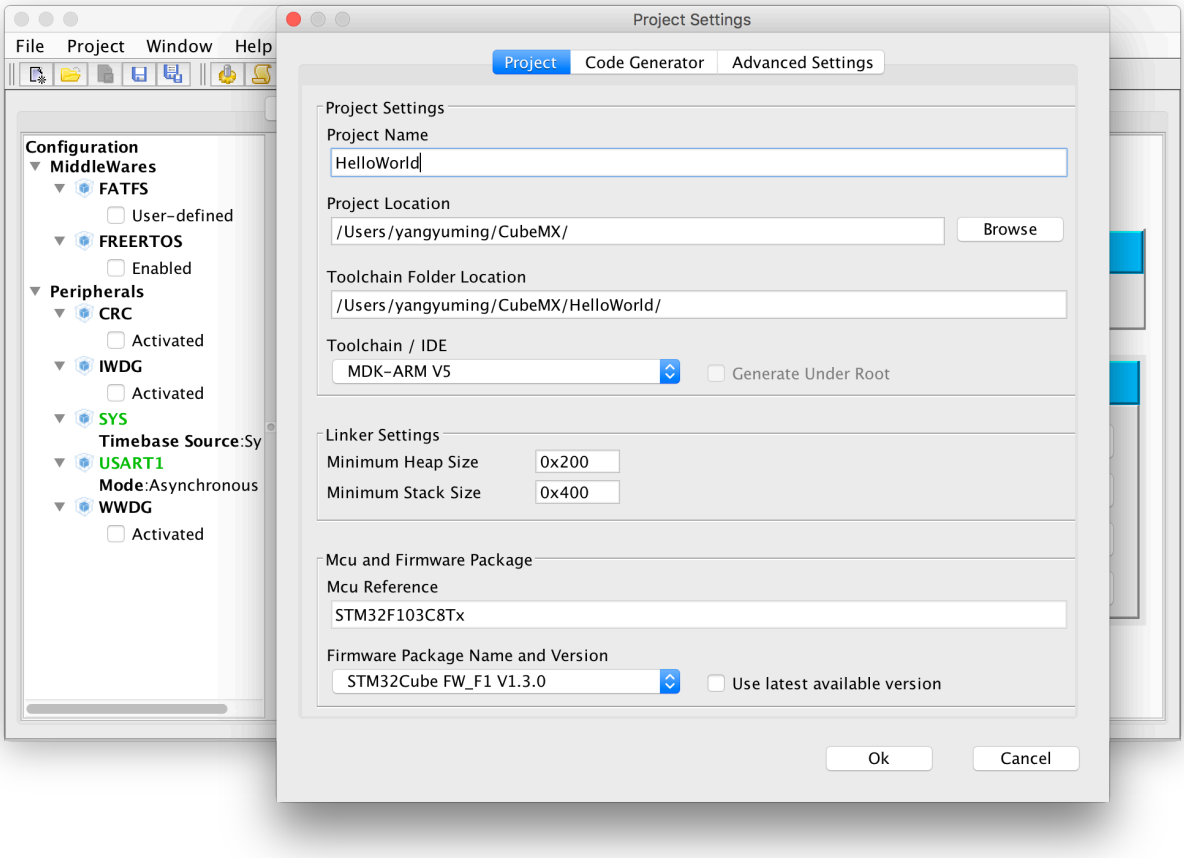
在 Pinout 中设置 USART1 为 Asyncronous。



Configuration 选项卡中可以设置具体功能。选择 Connnectivity 中的 USART1，设置波特率为 `9600 8n1`



选择 Project->Settings，设置 IDE 为 MDK-ARM，选择本地的 Cube Package，然后选择 Generate Code 自动生成工程模版。

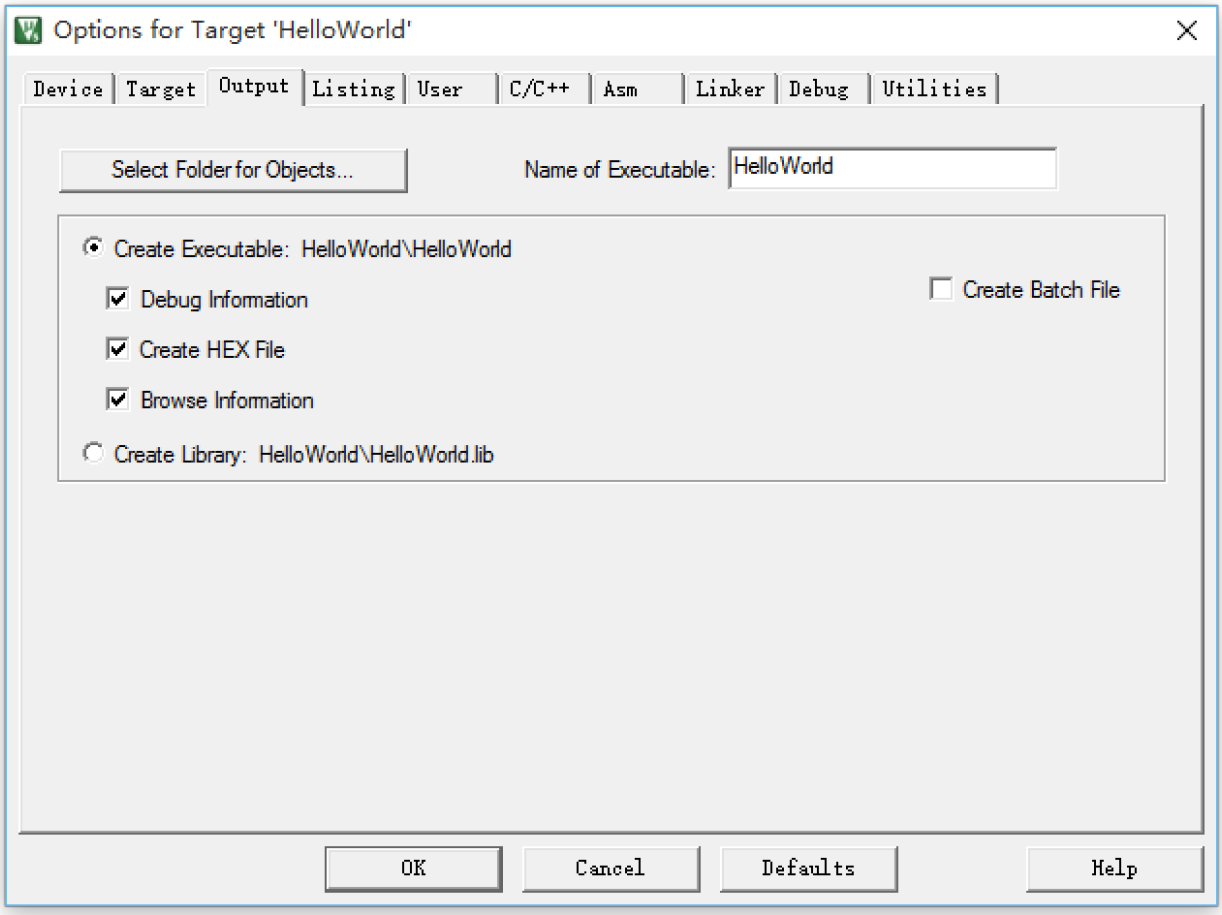


生成的模版工程可以用 Keil 打开，用户在 main.c 中编写代码。

编译&烧录

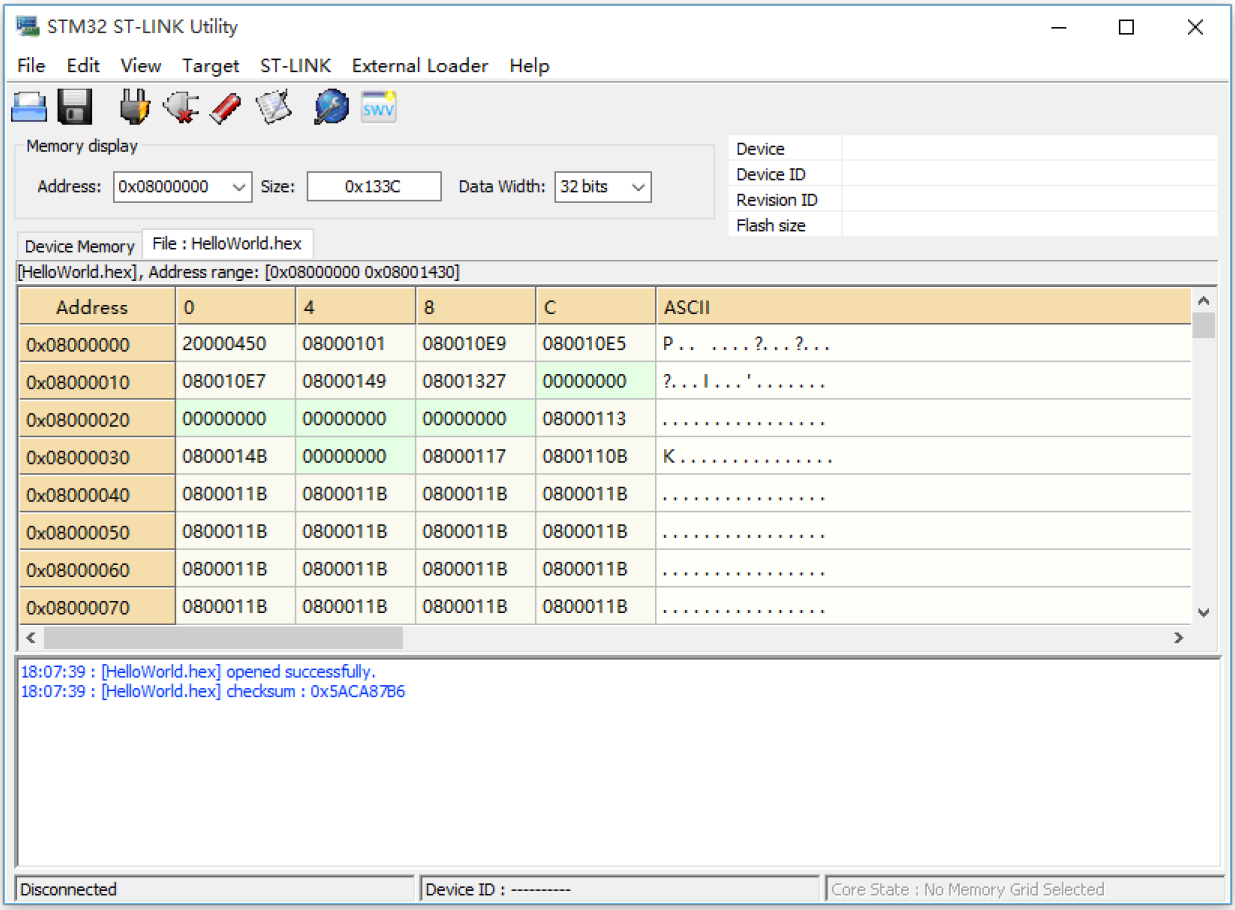
Keil 编译生成 hex 文件

在 Keil 工程中设置生成 hex 文件，并选择文件保存路径。



ST-LINK 烧录

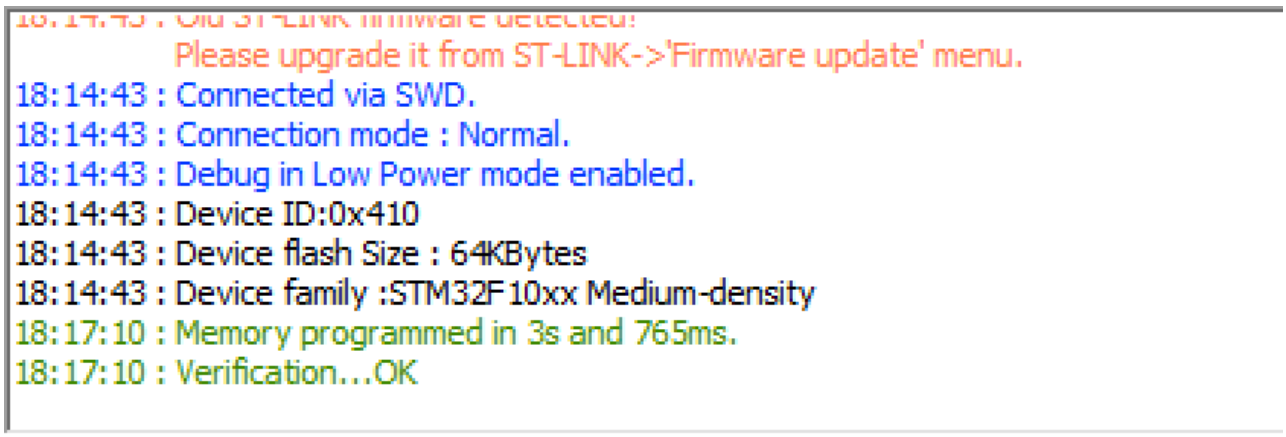
编译成功后，打开 ST-LINK 烧录程序，载入生成的 hex 文件。



将 STM32 开发板与 ST-LINK 连接，对应引脚如下。

ST-LINK	STM32
SWCLK	DCLK
SWDIO	DIO
3.3V	3.3V
GND	GND

连接成功后，点击 Target->Program 进行烧录，注意将 STM32 开发板上的 bt0 置为 1。 之后显示烧录成功，然后将 bt0 复位。



串口输出

USART设置

在 `MX_USART1_UART_Init()` 函数中是 CubeMX 自动生成的 USART 配置，初始化 UART。

```
/* USART1 init function */
void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  HAL_UART_Init(&huart1);

}
```

`HAL_UART_Init(&UartHandler)` 又调用 stm32f1xx_hal_msp.c 文件中的 `HAL_UART_MspInit(UART_HandleTypeDef* huart)`，这个函数里进行时钟和 GPIO 的初始化设置。

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspInit 0 */

  /* USER CODE END USART1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();

    /**USART1 GPIO Configuration
    PA9     ------> USART1_TX
    PA10     ------> USART1_RX
    */
    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_10;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN USART1_MspInit 1 */
    __HAL_RCC_USART1_FORCE_RESET();
    __HAL_RCC_USART1_RELEASE_RESET();
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_9);
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_10);
  /* USER CODE END USART1_MspInit 1 */
  }
}
```

重定向输出

为了更方便的发送数据，可以重定向 printf 函数，这样可以直接使用 printf 函数来向串口发送数据，这需要我们重写 fputc 函数，这个函数是声明为 weak 的,这就意味着我们的版本会替代系统的版本。这样我们可以在 main 函数中利用 printf 函数直接输出 “Hello” 到串口。

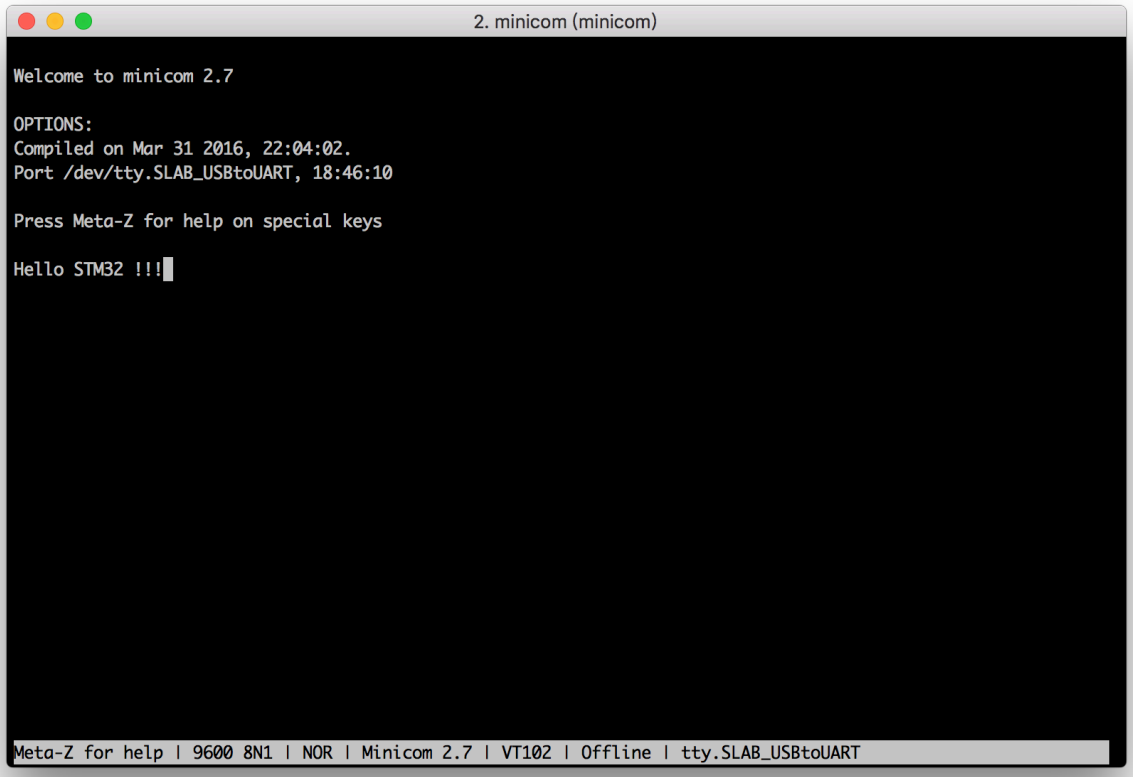
```
int fputc(int ch, FILE *f) {
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
```

将 STM32 开发板连接 usb-ttl 接头，对应引脚如下。

USB	STM32
3.3V	3.3V
GND	GND
TXD	A10
RXD	A9

PC 端接收

PC 端安装串口驱动程序，USB 连接串口后，显示设备 `/dev/tty.SLAB_USBtoUART`，设置minicom 程序波特率 `9600 8N1` 与串口通信。



轮询

GPIO 配置

配置 PA11 为内部上拉到输入模式，并通过面包板连接一个按钮。将 Speed 设置为 LOW 之后可以减少按钮抖动影响。

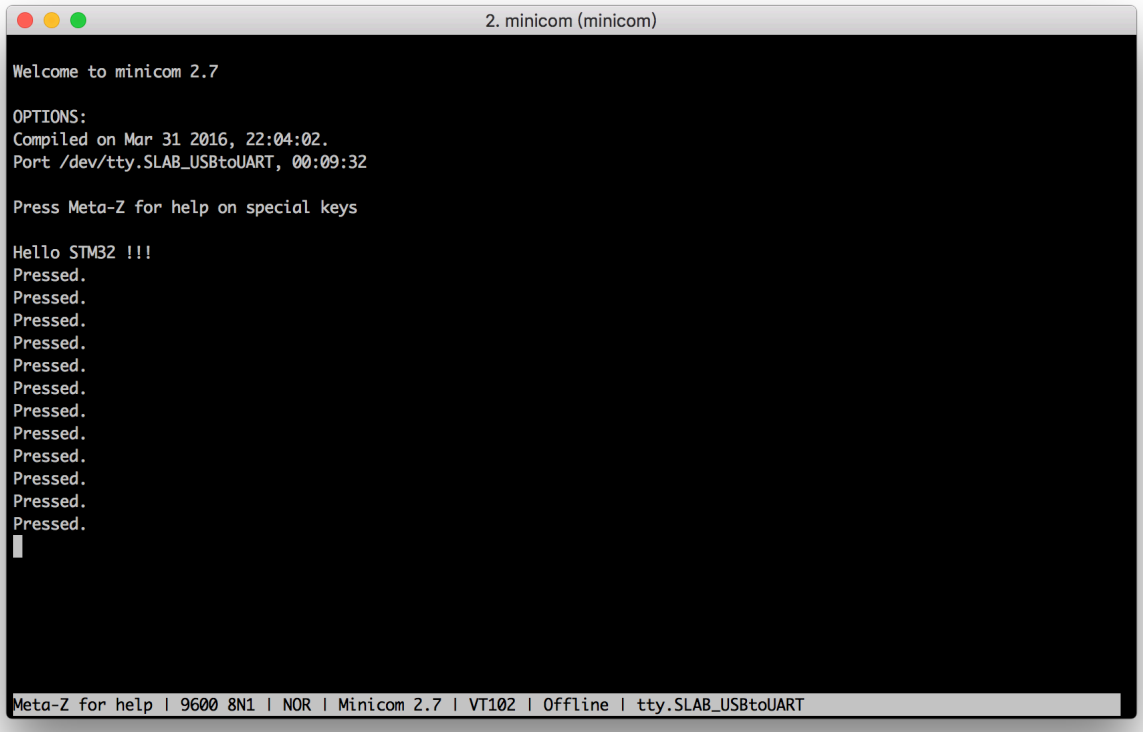
```
/*Configure GPIO pin : PA11 */
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

主循环检查

通过 HAL_GPIO_ReadPin 读取引脚输入，在 main() 函数中循环检查 PA11 按钮按下，并在按下时在串口输出 “Pressed”，延时去抖动。

```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
      if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11) == GPIO_PIN_RESET) {
          HAL_Delay(80);
          if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_11) == GPIO_PIN_RESET)
              printf("Pressed.\n\r");
      }
/* USER CODE END 3 */
```

建议自己添加代码时，在 `/ USER CODE BEGIN n /` 和 `/ USER CODE END n /` 之间添加，这样再次通过 CubeMX 修改之后就不会删除添加的代码，否则修改 CubeMX 工程文件重新生成代码会删除添加的代码。



中断

配置 GPIO 中断

配置 PA12 内部上拉到输入模式，并且下降沿触发中断。

```
  /*Configure GPIO pin : PA12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

设置中断优先级，enable 中断向量表处理。

```
  /* EXTI15_10_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

中断标识

程序中设置两个全局变量，一个为计数器，一个为标识。当中断触发时，计数器加1，并设置标识。

当 PA12 出现中断时，会调用 `EXTI15_10_IRQHandler()` 这个函数进行中断处理，在 stm32f1xx_it.c 中实现它。

这个函数中又要调用 `HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_12);` 判断中断标志位，以及调用中断返回函数 `HAL_GPIO_EXTI_Callback`，然后清除标志位。

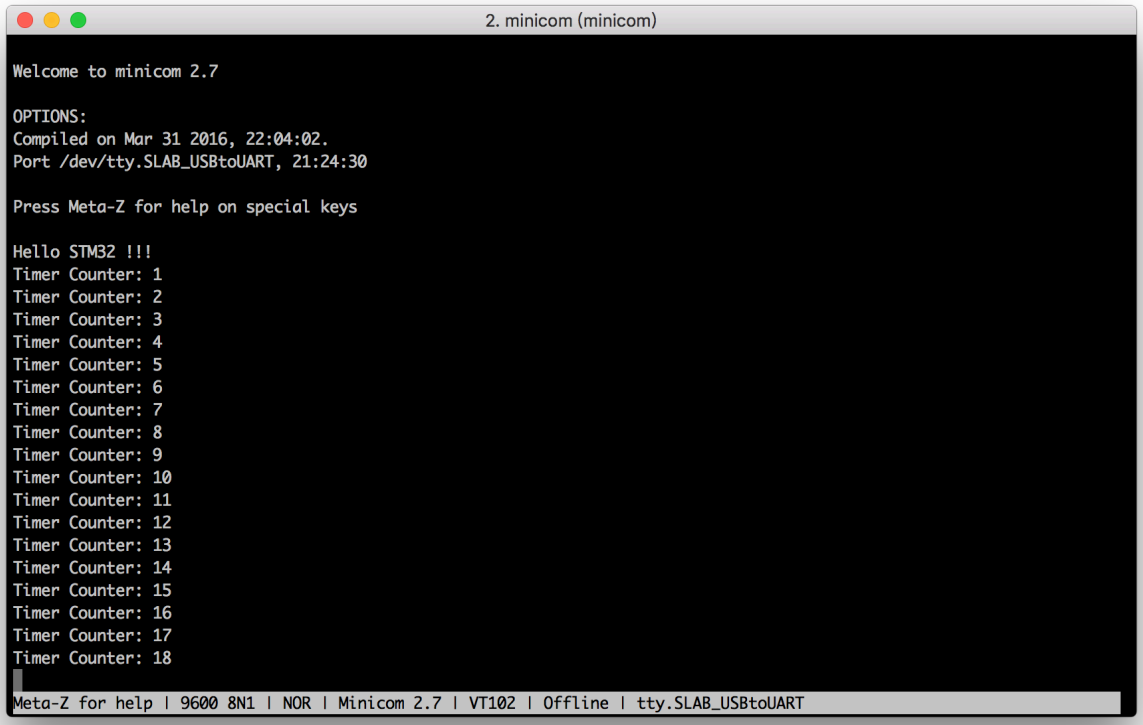
```
/**
* @brief This function handles EXTI line[15:10] interrupts.
*/
void EXTI15_10_IRQHandler(void) {
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_12);
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if (GPIO_Pin == GPIO_PIN_12){
        prsdCnt++;
        prsdFlag = 1;
    }else{
        UNUSED(GPIO_Pin);
    }
}
```

检查中断标识

在主循环中判断标识，如果标识置位则清除标识并通过串口输出计数值。

```
    /* PA12 INT */
    if (prsdFlag) {
        prsdFlag = 0;
        printf("Pressed Counter: %d\n", prsdCnt);
    }
```



定时器中断

配置定时器

配置 TIM3 向上计数，计数到 199，内部时钟频率为8MHz，Prescaler 分频值范围设置为 8000，这样定时器频率为 200ms

```
    htim3.Instance = TIM3;
    htim3.Init.Prescaler = 8000;
    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim3.Init.Period = 199;
    HAL_TIM_Base_Init(&htim3);
```

设置时钟源为内部时钟

```
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig);
```

设置复位模式,发生触发输入事件时计数器和预分频器能重新初始化

```
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig);
```

设置中断优先级，enable 中断向量表处理。

```
  /* TIM3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(TIM3_IRQn);
```

启动定时器

在 main() 函数中调用启动定时器。

```
HAL_TIM_Base_Start_IT(&TIM_Handle);
```

中断处理

与按钮中断类似，每次触发定时器中断将标识置为 1，在主循环检查标识，并输出到串口。

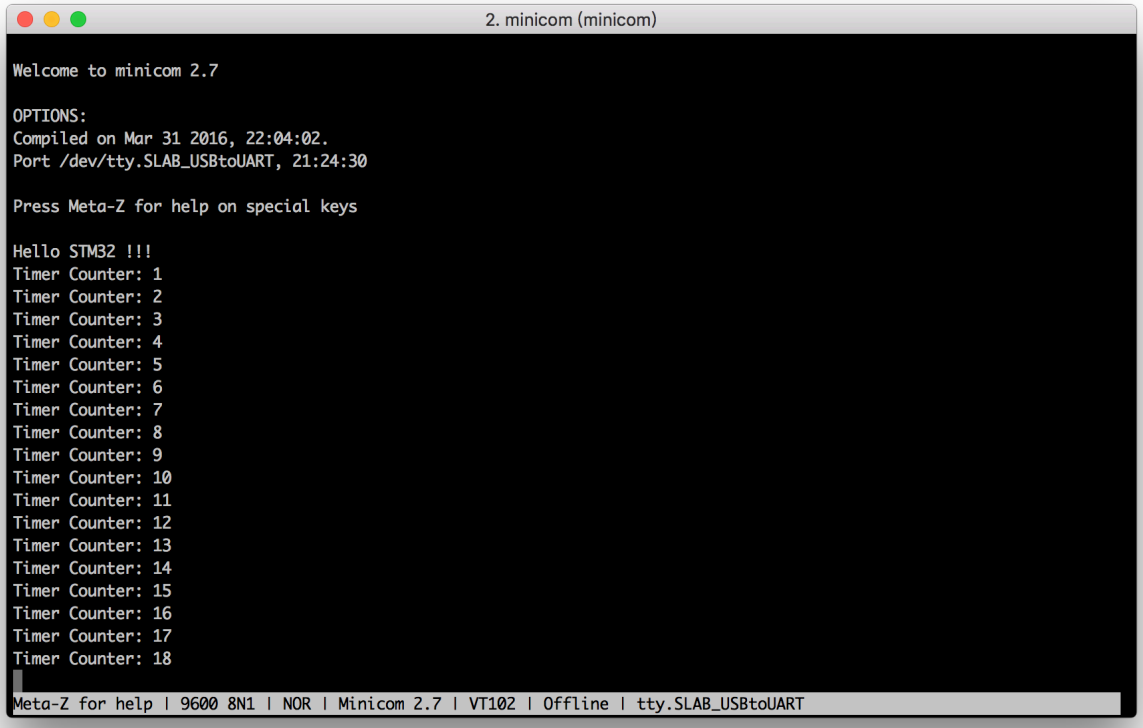
在 stm32f1xx_it.c 里面编写中断处理函数 `TIM3_IRQHandler()`，这个函数又调用 `HAL_TIM_IRQHandler(&htim3);`

最终调用回调函数 `HAL_TIM_PeriodElapsedCallback()` 进行中断处理。

```
/**
* @brief This function handles TIM3 global interrupt.
*/
void TIM3_IRQHandler(void) {
  HAL_TIM_IRQHandler(&htim3);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    timFlag = 1;
    timCnt++;
}
```

串口输出



码表程序

模式

码表采用两个模式，速度模式和里程模式，通过 PA3 按钮切换。

里程模式显示总里程，速度模式显示当前的近似速度。假设车轮周长为 2 米。

输出

输出采用定时器中断，每 200ms 输出串口一次，并根据当前模式输出对应的速度或里程。

速度

当前速度的计算采用最近 1s 的里程除以时间的近似。因此定时器计数为 5 时计算一次速度。

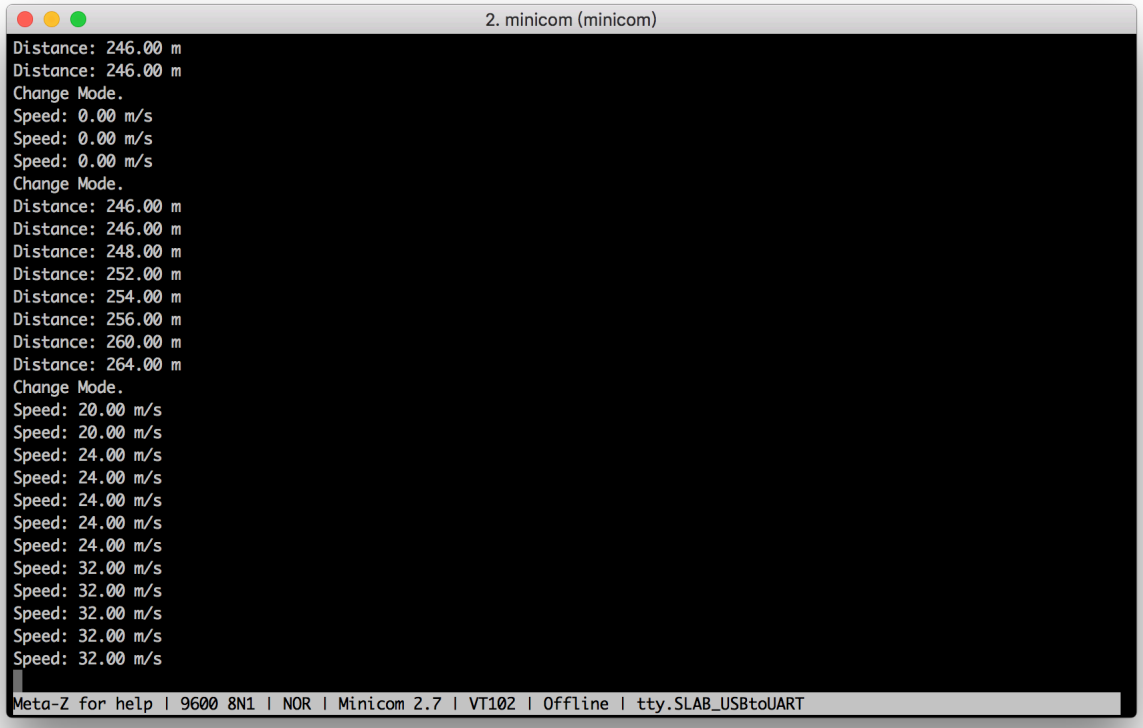
```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
        /* PA3 polling */
        if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3) == GPIO_PIN_RESET) {
            HAL_Delay(80);
            if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3) == GPIO_PIN_RESET) {
                printf("Change Mode.\n\r");
                dispMode = 1 - dispMode;
            }
        }

        /* PA12 INT */
        if (prsdFlag) {
            prsdFlag = 0;
            //printf("Pressed Counter: %d\n\r", prsdCnt);
        }

        /* TIMER INT */
        if (timFlag) {
            timFlag = 0;
            curDist = prsdCnt * 2.0;
            if (timCnt == 5) {
                speed = (curDist - preDist) * 2.0 / 1.0;
                preDist = curDist;
                timCnt = 0;
            }
            if (dispMode)
                sprintf(str, "Distance: %.2lf m \n\r", curDist);
            else
                sprintf(str, "Speed: %.2lf m/s \n\r", speed);
            printf(str);
        }
  }
  /* USER CODE END 3 */
```

结果验证

上电后串口输出 Hello STM32，并以 200ms 间隔输出，输出模式为速度或里程。PA3按钮改变速度，PA12按钮改变里程。



PA3 按钮也可采用中断模式，在 `HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)` 这个函数中判断引脚并分别处理。

中断驱动

休眠设置

暂停 SysTick，否则它会中断休眠。

```
HAL_SuspendTick();
```

然后进入休眠模式，PWR_SLEEPENTRY_WFI 表示可被中断唤醒。

```
HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
```

设置 CPU 中断后重新进入休眠模式。

```
HAL_PWR_EnableSleepOnExit();
```

中断处理程序

按钮中断，PA3 模式切换，PA12 旋转计数。

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
        if (GPIO_Pin == GPIO_PIN_12){
                prsdCnt++;
                prsdFlag = 1;
        }
        else if (GPIO_Pin == GPIO_PIN_3){
                printf("Change Mode.\n\r");
                dispMode = 1 - dispMode;
        }
        else{
                UNUSED(GPIO_Pin);
        }
}
```

计数器中断

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    curDist = prsdCnt * 2.0;
    if (timCnt == 5) {
        speed = (curDist - preDist) * 2.0 / 1.0;
        preDist = curDist;
        timCnt = 0;
    }
    if (dispMode)
        sprintf(str, "Distance: %.2lf m \n\r", curDist);
    else
        sprintf(str, "Speed: %.2lf m/s \n\r", speed);
    printf(str);
}
```

问题

在我建立的工程中，PA11 与 PA12 引脚同时使用时，相互的信号会发生重叠，当把 PA11 引脚改为 PA3 后解决了这个问题。原因可能是 PA12 的中断处理与 PA11 为同一组，所以发生冲突。