Robot Main Control Board (RHF407) User Manual

Ver： 1.1

Catalogue

Figure directory

1 Hardware description

This chapter will detail RHF407 development board of hardware resources, which gives users a detailed understanding of the functions and characteristics of the development board.

1.1 Robot main control board (RHF407) introduction

All the source code has compiled under MDK5.0.5 and verification through on the development board.
This development board is based on STM32F407VE processor, which mainly providing users with a demonstration platform.

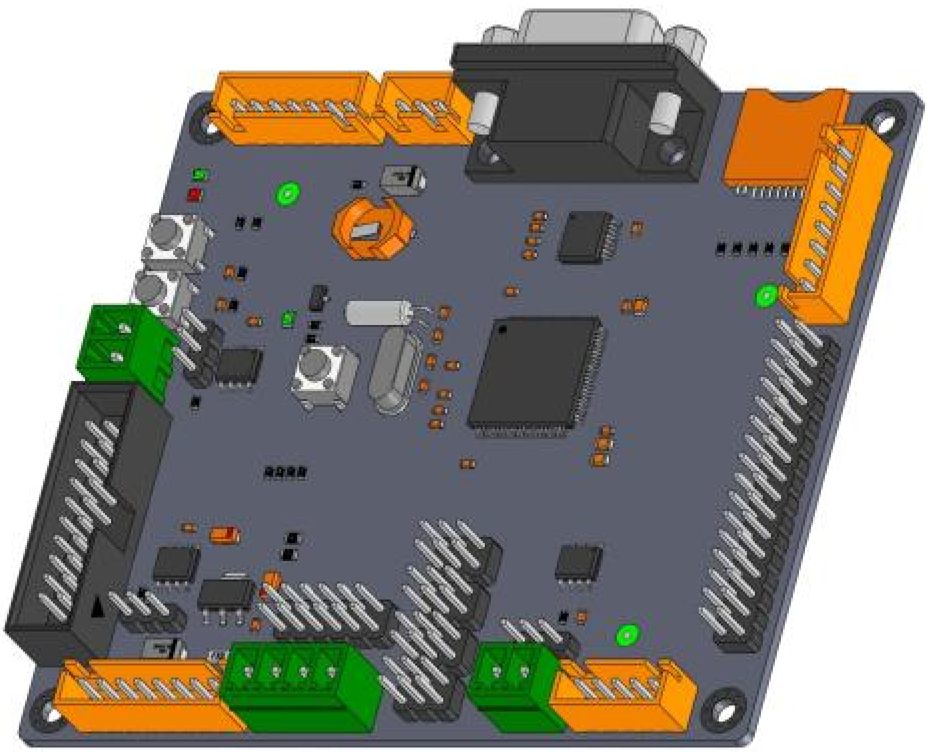


Figure 1. Development board external view

On-board resources are as follows:

- STM32F407VET6 is based on the high-performance ARM Cortex-M4 32-bit RISC core operating at a frequency of up to 168 MHz. incorporates high-speed embedded memories (Flash memory up to 1 M byte, up to 192 Kbytes of SRAM), up to 4 Kbytes of backup SRAM.
- Two RS232 level UART serial ports. One port can be directly connected to the PC, another can be connected to other equipment via 3 white pin terminal.
- A TTL level UART serial port with three general IO ports, which can be connected to the need to control other devices.
- A RS485 interface
- A reset signal status led
- Two programmed control led
- A reset switch
- Two normally open input keys
- A TF card interface, using the SPI bus
- A TFT LCD interface, you can use the FSMC, with SPI interface, sharing and TF card interface.
- Two CAN bus interfaces
- Two DAC output interfaces
- Eight ADC input interfaces
- Eight PWM output interfaces
- A I2C interface, which can be used as the PWM3 or PWM4 interface
- An RTC back-up battery interface
- A JTAG interface(20pin)

PCB size is 86 mm x 100 mm.

1.2 Circuit description

1.2.1 The processor circuit

Part of the processor circuit is as shown in the figure below.

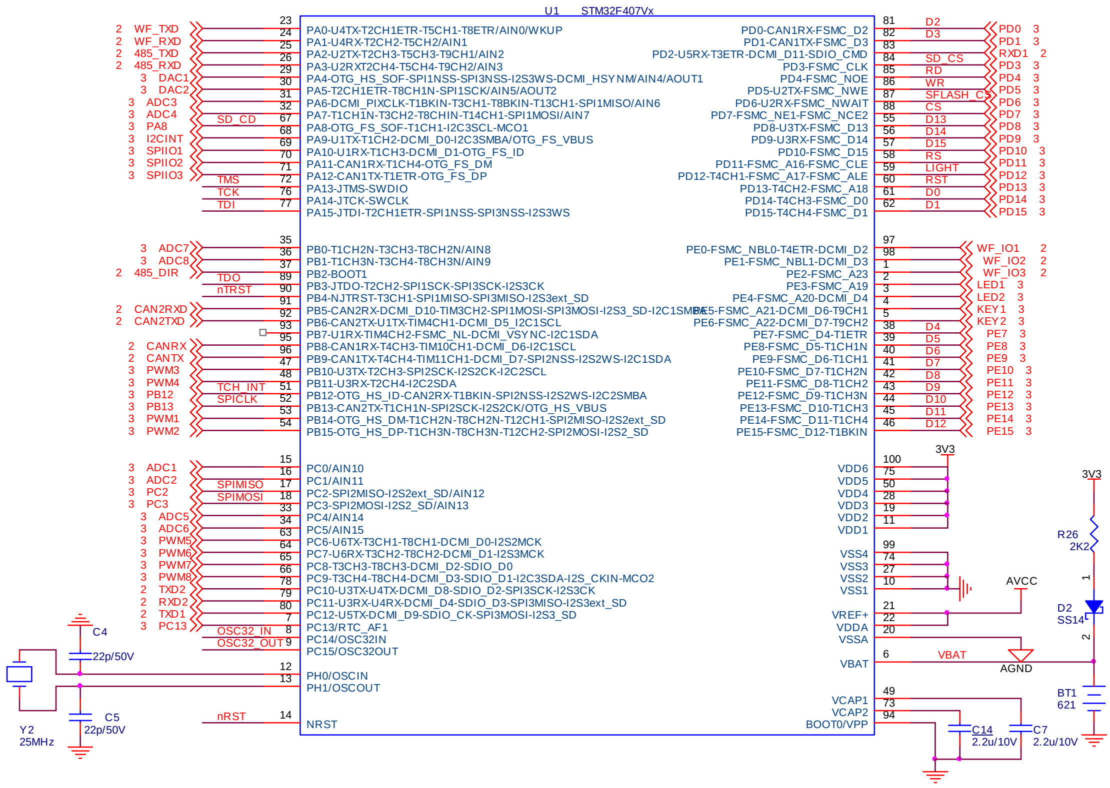


Figure 2. The processor circuit

- Master clock： It uses the 25MHz passive crystal, which is consistent with the official development board.
- Start-up mode： BOOT0 connect to GND and startup from the user flash.
- Backup power： Backup power is the rechargeable button lithium battery.

- External Capacitance： According to the official manual for use.

1.2.2 The reset circuit

The reset circuit uses the Special electrified SP809 reset chip. It instructions reset signal state through LED4.Reset switch SW1 is used to manually reset. The reset circuit is as shown in the figure below.

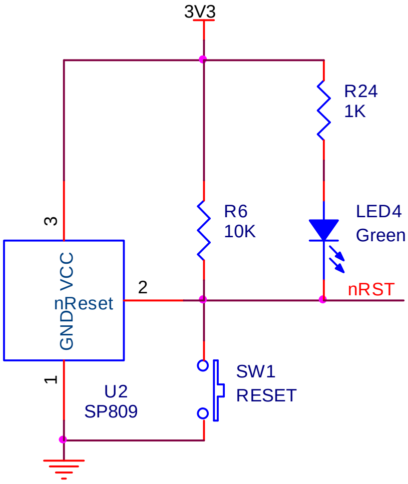


Figure 3. The reset circuit

1.2.3 Low frequency crystal circuit

Low frequency crystal circuit uses the 32.768MHz passive crystal as shown in the figure below.

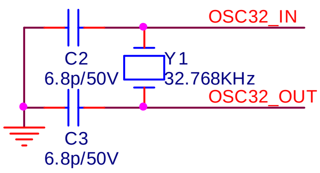


Figure 4. Low frequency crystal circuit

1.2.4 JTAG circuit

JTAG circuit use Standard 20 pins connector as shown in the figure below.

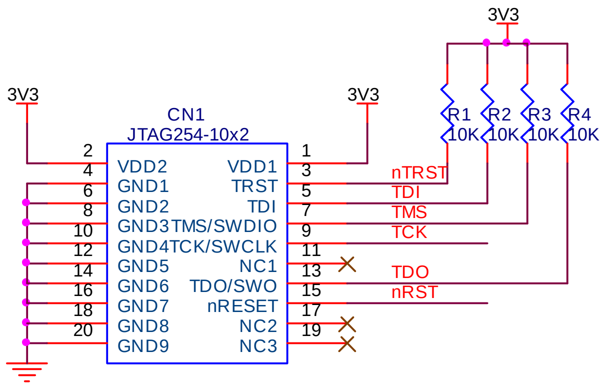


Figure 5. JTAG circuit

1.2.5 The power circuit

The power circuit is as shown in the figure below.

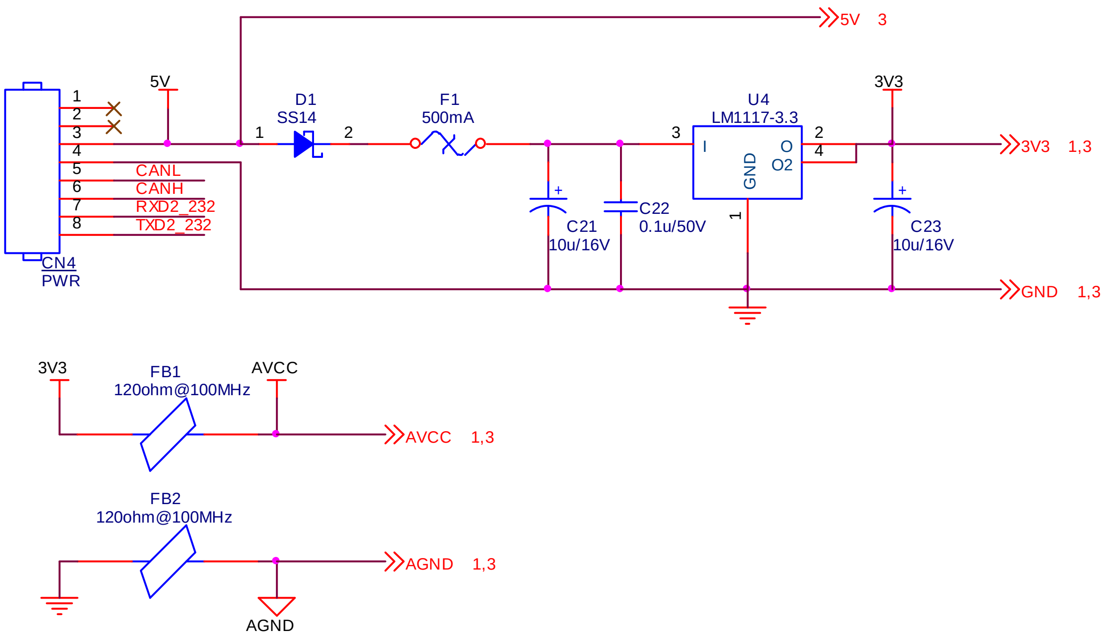


Figure 6. The power circuit

5V power supply from CN4 input, voltage to 3.3 V power supply to the system through D1 reverse voltage protection and LM1117-3.3.

3.3 V power supply electrical analog circuit after magnetic beads FB1 and FB2 filtering.

1.2.6 Indicator light circuit

The Indicator light circuit is as shown in the figure below.

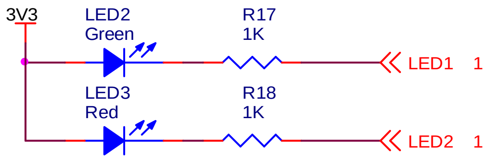


Figure 7. Indicator light circuit

Indicator light connects to the CPU I/O line. The control GPIO of LED2, LED3 respectively is PE3, PE4.

1.2.7 The button circuit

The button circuit is as shown in the figure below.

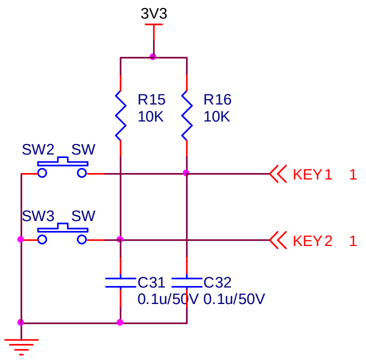


Figure 8. The button circuit

After filtering capacitor, it connects to the CPU I/O line by pulling up. The control line of SW2, SW3 respectively is PE5, PE6.

1.2.8 RS232 serial port circuit

RS232 serial port circuit is as shown in the figure below.

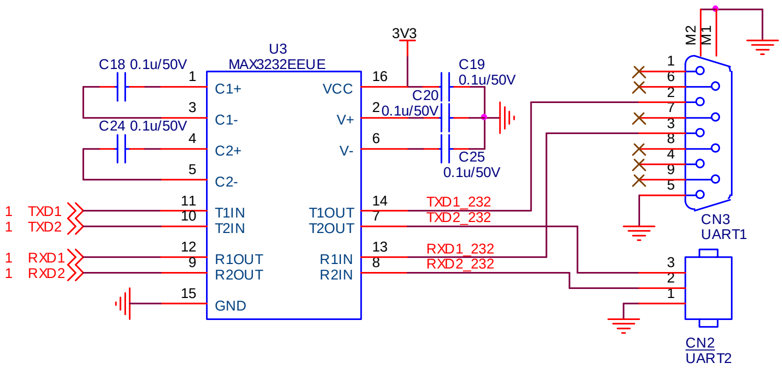


Figure 9. RS232 serial port circuit

The electric level conversion between TTL and RS232 can be realized by SP3232; one serial port connects to the DB9 socket which can be directly connected to the PC serial port, another can connect to the terminal socket. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
TXD1	PC12	
RXD1	PD2	
TXD2	PC10	
RXD2	PC11	

1.2.9 TTL serial portcircuit

TTL serial portcircuit is as shown in the figure below.

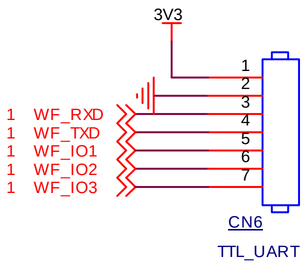


Figure 10. TTL serial port circuit

TTL serial port circuit is a circuit form the CPU to the terminal socket. The circuit includes 3 I/O control lines. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
WF_RXD	PA1	
WF_TXD	PA0	
WF_IO1	PE0	
WF_IO2	PE1	
WF_IO3	PE2	

1.2.10RS485 circuit

RS485 circuit is as shown in the figure below.

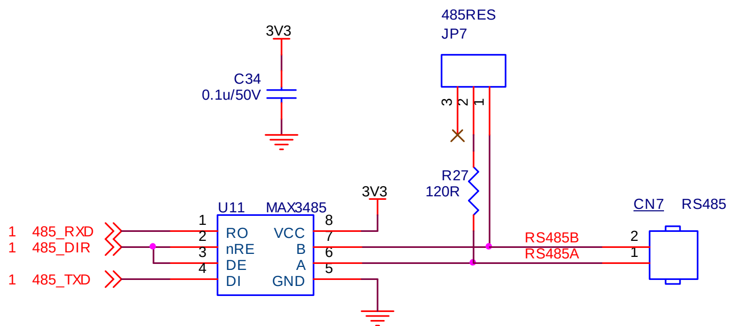


Figure 11. RS485 circuit

The electric level conversion between TTL and RS485 can be realized by SP3485. It can match impedance terminal resistance when JP7 1, 2 pin short circuit. The circuit driver chip can send and receive radio signal. The circuit driver chip is in a state of receiving signals at ordinary times, and in a state of sending signals only in need. After sending over, it comes to the receiving signal state immediately. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
485_RXD	PA3	
485_TXD	PA2	
485_DIR	PB2	

1.2.11CAN bus circuit

CAN bus circuit is as shown in the figure below.

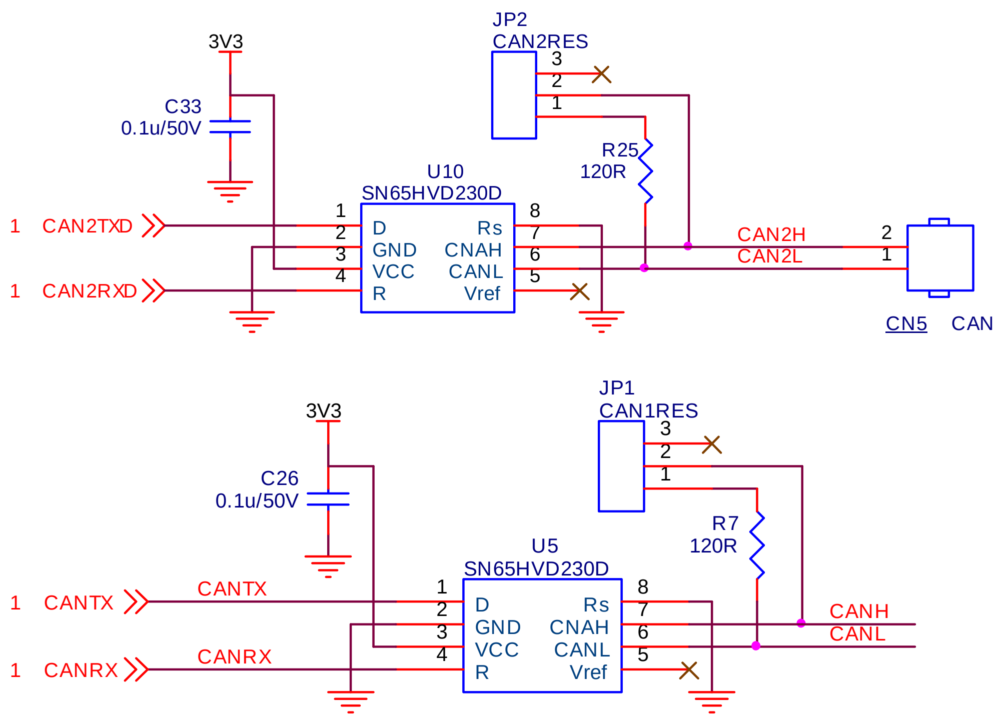


Figure 12. CAN bus circuit

This system provides two CAN bus circuits; one is along with the input power plug-in board, another can use a separate plug-in board.CAN1 can match the terminal resistors by JP1; CAN 2 can match the terminal resistors by JP2.CAN bus circuit driver chip is the TI SN65HVD230. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
CANTX	PB9	
CANRX	PB8	
CAN2TXD	PB6	
CAN2RXD	PB5	

1.2.12I2C interface circuit

I2C interface circuit is as shown in the figure below.

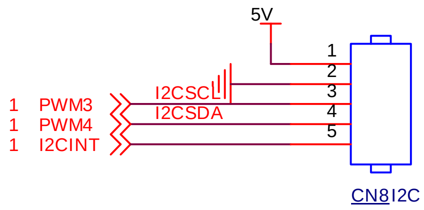


Figure 13. I2C interface circuit

I2C interface circuit connects to the CPU I/O line directly. I2C interface can be used as the PWM3 or PWM4 interface. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
SCL	PB10	
SDA	PB11	
INT	PA9	

1.2.13TF card circuit

TF card circuit is as shown in the figure below.

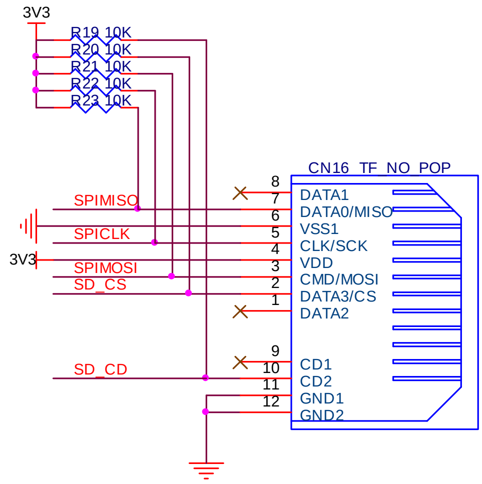


Figure 14. TF cardcircuit

TF card circuit communicates by SPI bus. Each control single has the external pull-up resistor. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
SPIMISO	PC2	
SPIMOSI	PC3	
SPICLK	PB13	
SD_CS	PD3	Select chip

SD_CD	PA8	Insert and test

1.2.14LCD module interface circuit

LCD module interface circuit is as shown in the figure below.

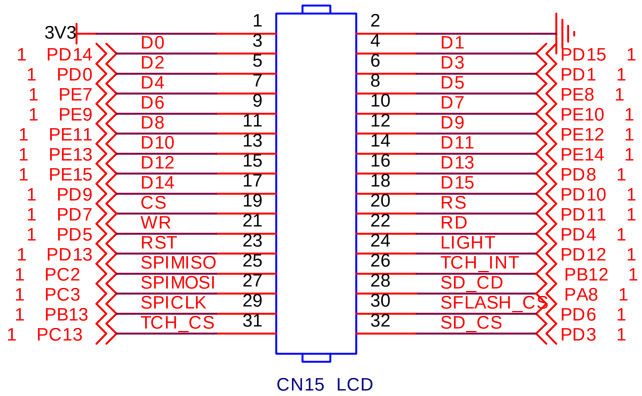


Figure 15. LCD module interface circuit

LCD module interface is compatible with 3.2 inch Shenzhou King TFT module and other development board modules. The CPU resources can be seen in the figure 15.

1.2.15ADC circuit

ADC circuit is as shown in the figure below.

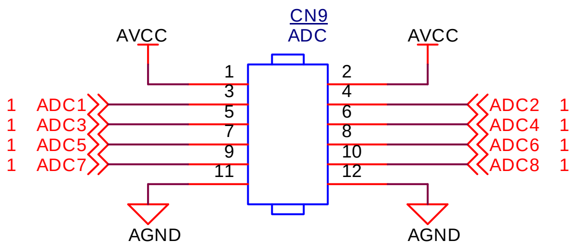


Figure 16. ADC circuit

ADC circuit can input to the CPU I/O line. When using, It cannot beyond the CPU limit condition to avoid damaging the processor, otherwise need to add protection before input circuit. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
ADC1	PC0/AIN10	
ADC2	PC1/AIN11	
ADC3	PA6/AIN6	
ADC4	PA7/AIN7	
ADC5	PC4/AIN14	
ADC6	PC5/AIN16	

ADC7	PB0/AIN8	
ADC8	PB1/AIN9	

1.2.16DAC circuit

DAC circuit is as shown in the figure below.

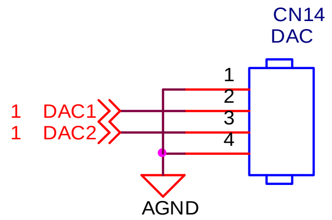


Figure 17. DAC circuit

The DAC circuit can output directly, which can also be used as the ADC input circuit. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
DAC1	PA4/AOUT1/AIN4	
DAC2	PA5/AOUT2/AIN5	

1.2.17PWM output circuit

PWM output circuit is as shown in the figure below.

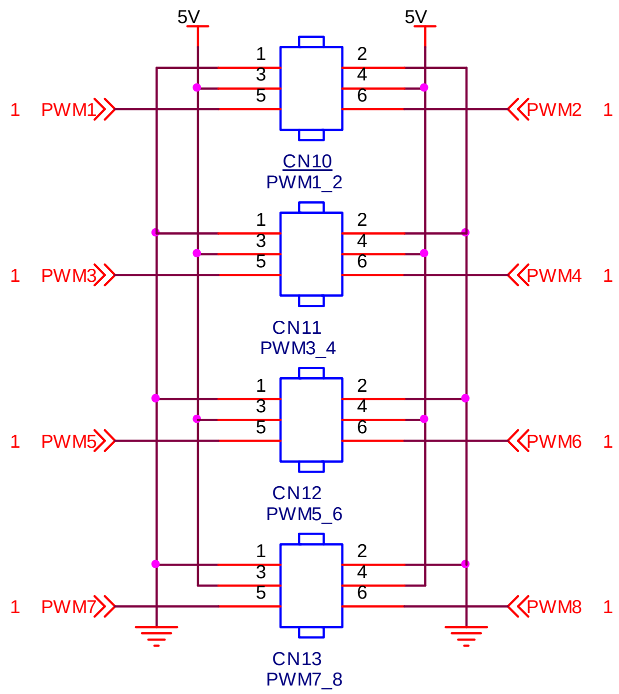


Figure 18. PWM output circuit

PWM output circuit can output directly, which can connect the steering-engine or other equipment. The CPU resources are as shown in the figure:

The signal name	The processor port	Note
PWM1	PB14/T12CH1	
PWM2	PB15/T12CH2	
PWM3	PB10/T2CH3	
PWM4	PB11/T2CH4	
PWM5	PC6/T3CH1	
PWM6	PC7/T3CH2	
PWM7	PC8/T3CH3	
PWM8	PC9/T3CH4	

2 The development environment introduction

Now there are so many ARM development environments. The MDK and IAR development environments are popular in china. The MDK development environment only supports the ARM7, ARM9 series and Cortex systems, while the IAR development environment can support the ARM11、Cortex-A9 systems. But the MDK development environment is more suitable for beginners.
There are some development environments based on GNU. Some of them need to pay.eg. TrueSTUDIO、RIDE. Some of them are free as emIDE.

2.1 MDK introduction

The RVMDK development environment is from KEIL in Germany. The name RVMDK is short for RealView MDK. RealView MDK integrates the industry's most advanced technology, including mu Vision4 integrated development environment and RealView compiler. The MDK development environment only supports the ARM7, ARM9 series and the latestCortex-M3、Cortex-M4.
The MDK development environment introduced is MDK ver4.5.

2.2 Establish a new project

Now what we introduced is to establish a new project named Demo in the RHF407_Examples folder in D disk. But you need to install MDK software and download to peripherals library stm32f4_dsp_stdperiph_lib. Zip from ST website and unzip to D disk root directory.

2.2.1 Establish a new project

The detailed step of establishing a new project: Start the' Keil uVision4', click the project menu 'New uVision Project' item, choose the directory 'D:\ RHF407_Examples\Demo' in the pop-up dialog box. If no directory in the dialog, you need to establish a new project named Demo in the dialog and click item 'save 'to close the dialog. As shown in the figure below:

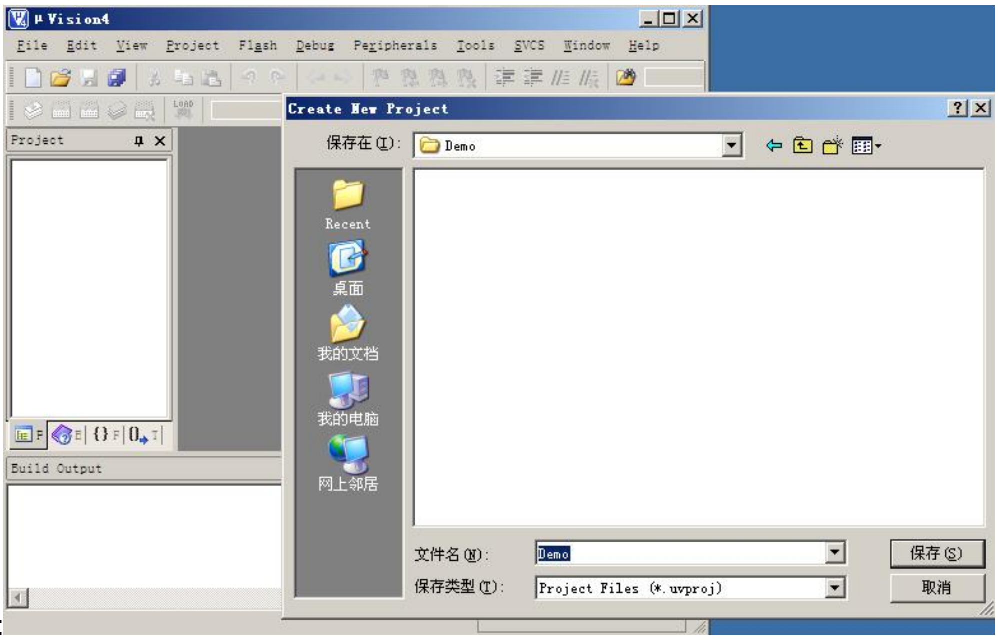


Figure 19. Establish a new MDK project

2.2.2 Select the device

Select the device' STM32F407VE ' in the pop-up dialog box, click 'OK 'to close the dialog.

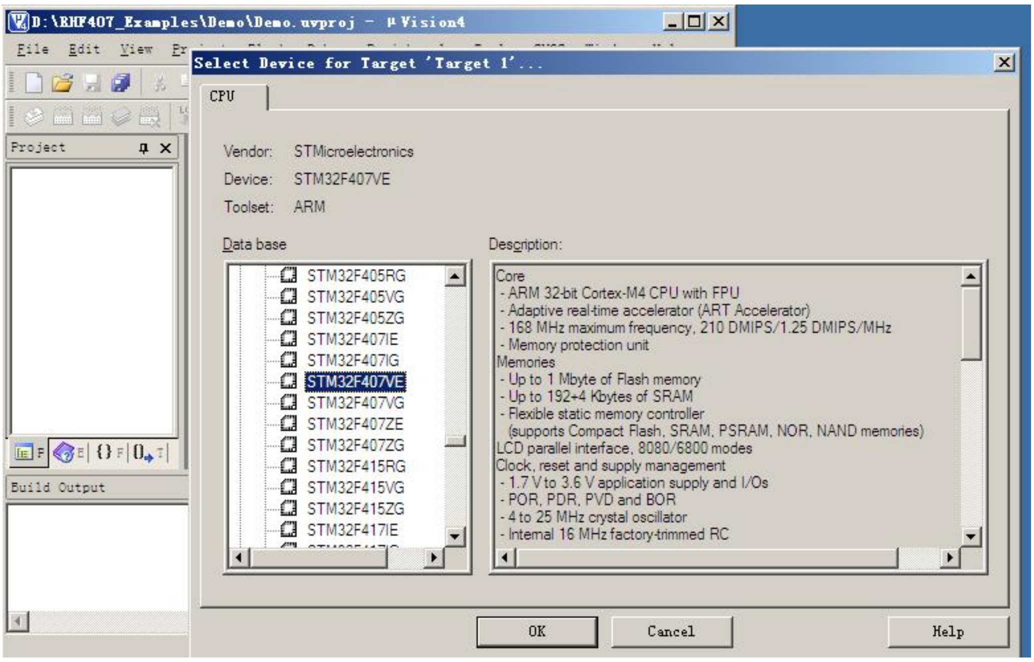


Figure 20. Select the MDK device

Next you will be asked if add files to the project automatically, you select the 'no' choice. An empty project will be established as shown in the figure below.

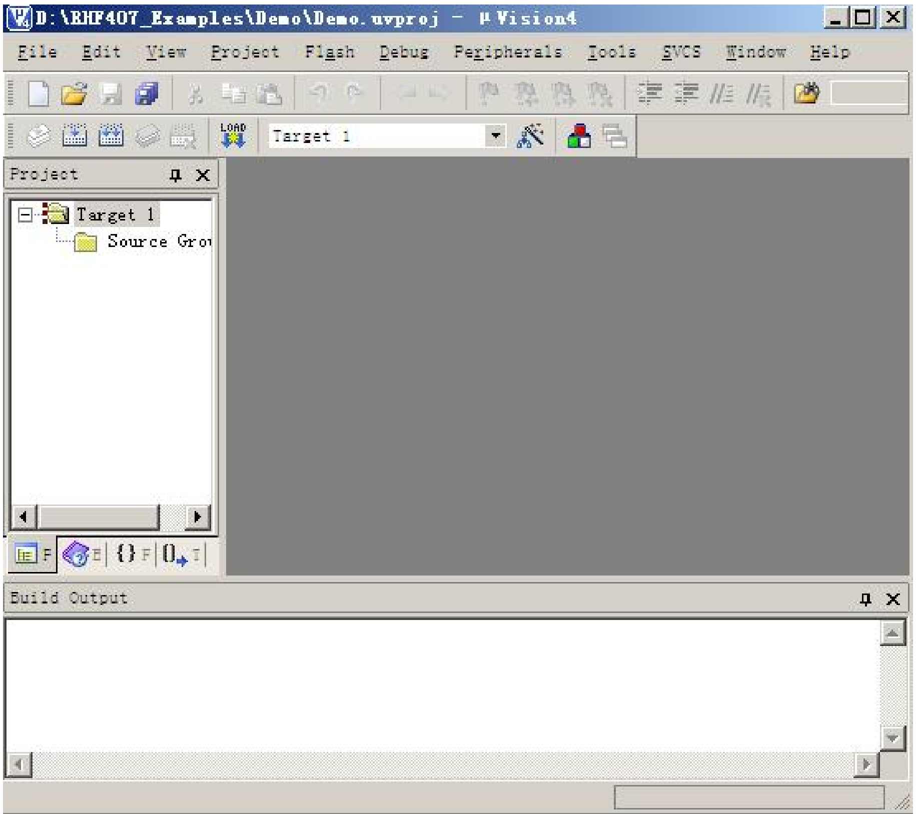


Figure 21. MDK empty project

2.2.3 Prepare the document

Copy the useful Libraries folder content to the Demo folder. You can delete unwanted files inside. Establish a new folder App in the Demo folder to hold the code files, and manually create the file App. C.

Click , you can manage the project files in the pop-up dialog box.

Double-click on the default entry in Project Targets and modify the name for the Debug.

Double-click on Source Group 1 in the groups and modify the name for the APP. Click the to set new CMSIS, Startup ,Library group in turn.

Choose the APP group, click Add files to add the file App.C in the APP folder.
Choose the CMSIS group, click Add files to add the file in the library
Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\system_stm32f4xx.c
Choose the Startup group, click Add files to add the file in the library
:Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm\startup_stm32f4xx.s
Choose the Library group, click Add files to add the standard peripherals driver file in the library
:Libraries\STM32F4xx_StdPeriph_Driver\src
Click 'OK 'to close the dialog.

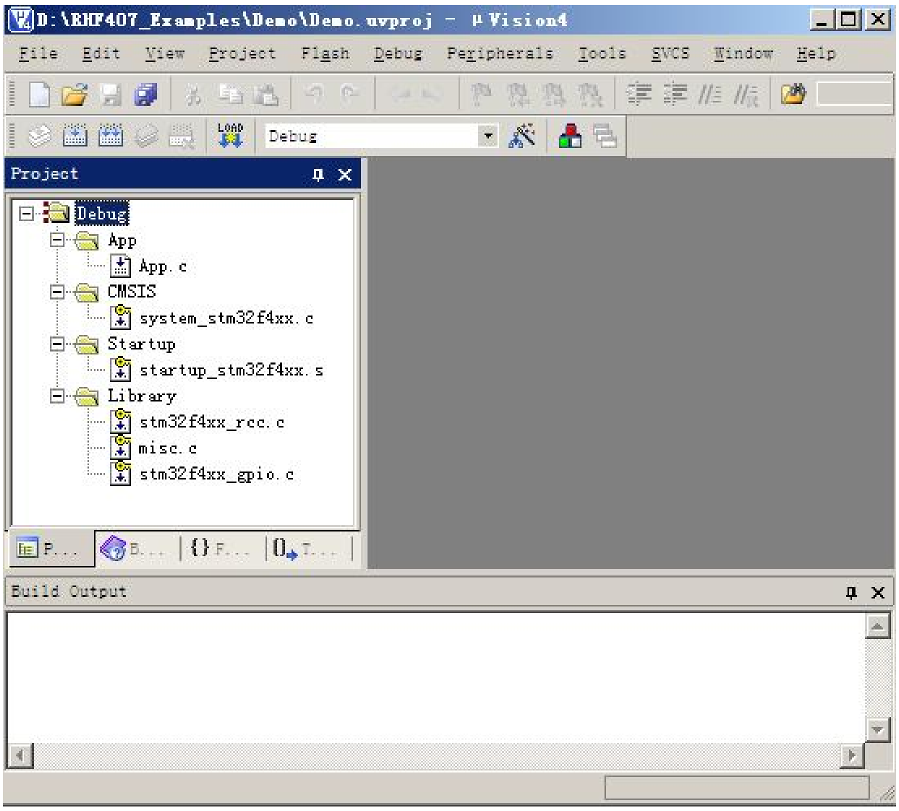


Figure 22. Add a file

2.2.4 Configure the project

Click , you can configure the project files in the pop-up dialog box.

Set the output file storage path in the Output page, click on Select Folder for Objects,add and select the new Folder bin.

Set the output file storage path in the Listing page, click on Select Folder for Listings,add and select the new Folder obj.

Setup compiler in C/C + + page, add macro definition USE_STDPERIPH_DRIVER,STM32F4XX after Define in order to use the official peripherals library. Add project header file path in the 'Includes Paths'.

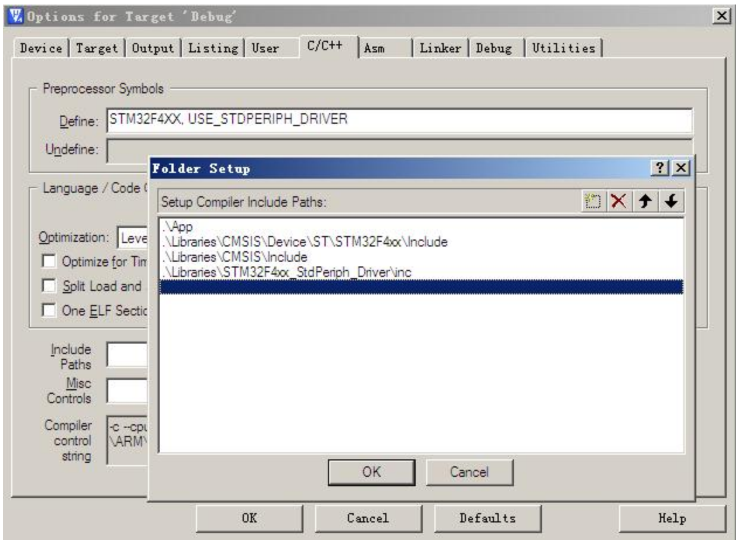


Figure 23. Configure the compiler

Configure the debugger in the Debug page, choose JLINK, Run to main, click Setting to set the JLINK debugger as shown in the figure below.

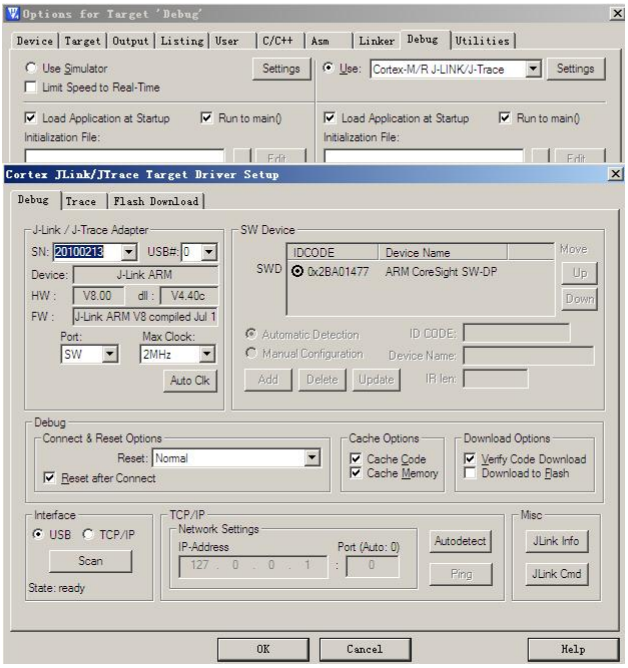


Figure 24. Configure the debugger

The dialog window to configure the 'JLINK' debugger in the figure above will be different when JLINK is not connected or target board has no electricity.

Configure the downloader in the Utilities page, choose 'JLINK', 'Update Target before Debugging'. Click 'Setting' to set the JLINK downloader as shown in the figure below.

You need to add the corresponding device if no devices in the 'Programming Algorithm'.

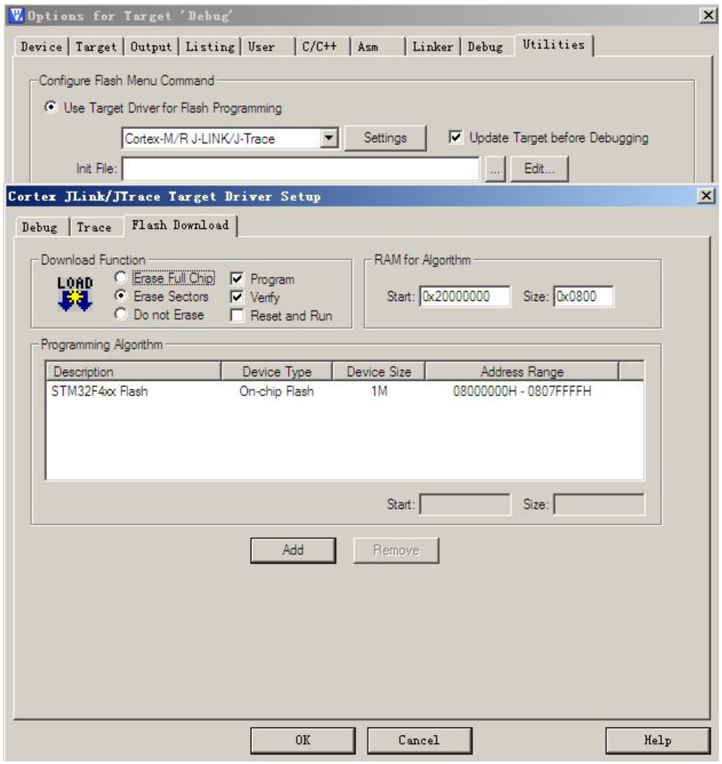


Figure 25. Configure the downloader

Now the project configuration is done.

2.2.5 Add the code

Although a basic project has been established, you need to add the codes because the file 'App.c' is empty. For example, you can open the file and add the code as shown in figure.

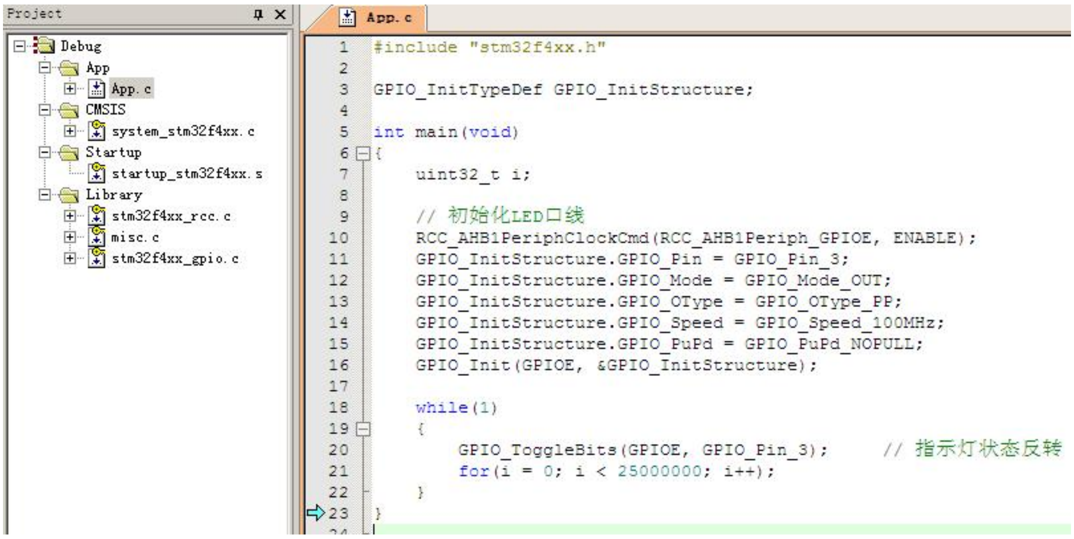
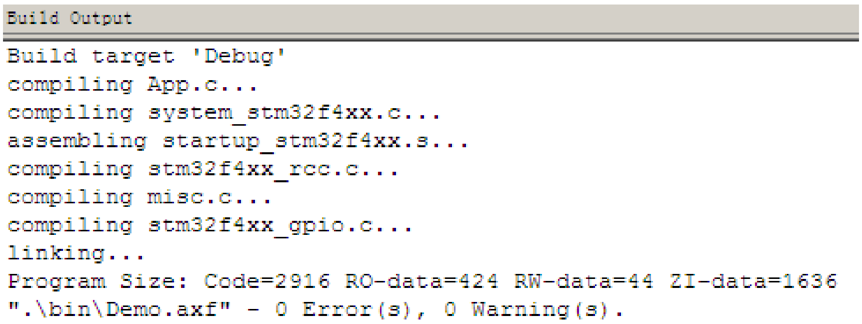


Figure 26. Add the code

2.2.6 Compile the project

Click , you can compile the project.

The output compiler information is as shown in the figure below.



```
Build Output
Build target 'Debug'
compiling App.c...
compiling system_stm32f4xx.c...
assembling startup_stm32f4xx.s...
compiling stm32f4xx_rcc.c...
compiling misc.c...
compiling stm32f4xx_gpio.c...
linking...
Program Size: Code=2916 RO-data=424 RW-data=44 ZI-data=1636
".\bin\Demo.axf" - 0 Error(s), 0 Warning(s).
```

Figure 27. Output compiler information

2.2.7 Simulation debugging

Click  to download the file into processor and start debugging. The program will execute to the entrance of the main function because of the settings 'Run to the main' in the debugger.

Click , the program will run at full speed. You can see that the green indicator light on the development board is flashing.

2.3 Starting process

It will execute start from the startup_stm32f4xx.s following code in the startup file when the system is reset.

```
Reset_Handler    PROC
                 EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP
```

Firstly you need to initialize System Init function in the executable file named 'system_stm32f4xx.c including enable floating-point unit, clock, external SRAM, and interrupt to scale positioning, etc. And then began to implement the main function in the file named main.c

3 Basic routines

We designed a few necessary routine which contains more practical functionat ordinary times .The operation is basically based on the library function to avoid directly operate the registers. The question can be solved by searching online information.

3.1 Button control indicator light

The function of this routine is the following.

It will interrupt when two buttons on the board are pressed.Each key control an LED.The LED state will reverse to the corresponding contrary state when you press the button again.

Set a new folder '1. LED and Key' in the routine catalogue.Copy the library to the folder.

Set a new folder Project\MDK-ARM,enter and set a new project named 'project'

The directory structure is as shown in the figure below.

Because user program files in the routineis not much, so they are in the Project folder.

Considering some routines clock configuration will be different, in order not to damage the library files, the file 'system_stm32f4xx. C' is copied to the project folder and added to the project. Program files in the project folder are main.c、stm32f4xx_conf.h、stm32f4xx_it.c、stm32f4xx_it.h、system_stm32f4xx.c

In this routine,other documents are the same as the official template in addition to the main c.

From the frontcircuit description, we know the followings:

The indicator light is on when the indicator light line is in low electricity ,the indicator light is out when the indicator light line is in high electricity .It is in the low level state when button is pressed; and it is in the high level state when button is released.

At the time of programming, people like to begin with the main function, and gradually add the programs after a good framework. Obviously, an indicator light and the button anti-fuzzy initial function are needed,then you can add an empty infinite loop programs,.The code is as following.

```
#include "stm32f4xx.h"

// main function
int main(void)
{
	LedConfig();
	KeyLineConfig();

	while(1)
	{
	}
}
```

Light control belongs to typical GPIO port output applications. You need initialize before using according to the following process.

GPIO port line portclock

set mouth line mode

Implemented to define LED hardware resources,which can be easy to maintain and transplant.

```
// LED light related definition
	#define GPIO_LED		GPIOE				// LED port
#define RCC_GPIO_LED	RCC_AHB1Periph_GPIOE	// GPIO clock used by LED

#define LED1_PIN		GPIO_Pin_3			// GPIO pin used by LED1
#define LED2_PIN		GPIO_Pin_4			// GPIO pin used by LED2
```

Realize initialize function

```
// configure LED
	void LedConfig(void)
	{
		GPIO_InitTypeDef GPIO_InitStructure;

		RCC_AHB1PeriphClockCmd(RCC_GPIO_LED, ENABLE);		// enable clock

		GPIO_InitStructure.GPIO_Pin = LED1_PIN|LED2_PIN;	// configure Pin pattern
		GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
		GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
		GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
		GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
		GPIO_Init(GPIO_LED, &GPIO_InitStructure);		// GPIO port initialize
}
```

LED state reverse， it is state reverse of GPIO output state

```
	// LED state reverse
	void Led1Toggle(void)
	{
		GPIO_ToggleBits(GPIO_LED,LED1_PIN);		//LED1 status change
```

```
}
void Led2Toggle(void)
{
	GPIO_ToggleBits(GPIO_LED,LED2_PIN);		//LED2 status change
}
```

Button is the typical GPIO input application, here it use break off, but it can check when it used on some occasion which needn't quick response.

- enable GPIO port clock
- enable configure control clock
- configure GPIO pattern
- connect external interruption to GPIO
- Configure interrupt lines
- Enable interrupt and set up interrupt priority

definite hardware resource first

```
//button definition
#define GPIO_KEY                GPIOE                       //KEY port
#define RCC_GPIO_KEY            RCC_AHB1Periph_GPIOE        // GPIO clock used by KEY

#define KEY1_PIN                GPIO_Pin_5                  // GPIO pin used by KEY1
#define KEY2_PIN                GPIO_Pin_6                  // GPIO pin used by KEY2

#define KEY1_PORT_SOURCE        EXTI_PortSourceGPIOE
#define KEY2_PORT_SOURCE        EXTI_PortSourceGPIOE

#define KEY1_PIN_SOURCE         EXTI_PinSource5
#define KEY2_PIN_SOURCE         EXTI_PinSource6

#define KEY1_EXTI_LINE          EXTI_Line5
#define KEY2_EXTI_LINE          EXTI_Line6

#define KEY1_EXTI_IRQN          EXTI9_5_IRQn
#define KEY2_EXTI_IRQN          EXTI9_5_IRQn
```

Write button initialize subprogram

```
// configure button
void KeyLineConfig(void)
{
	EXTI_InitTypeDef   EXTI_InitStructure;
	GPIO_InitTypeDef   GPIO_InitStructure;
	NVIC_InitTypeDef   NVIC_InitStructure;

	// enable port clock
	RCC_AHB1PeriphClockCmd(RCC_GPIO_KEY, ENABLE);

	// enable system configure controller clock, external GPIO interrupt need
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

	//
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStructure.GPIO_Pin = KEY1_PIN | KEY2_PIN;
	GPIO_Init(GPIO_KEY, &GPIO_InitStructure);

	// connect interrupt line to Pin
	SYSCFG_EXTILineConfig(KEY1_PORT_SOURCE, KEY1_PIN_SOURCE);
	SYSCFG_EXTILineConfig(KEY2_PORT_SOURCE, KEY2_PIN_SOURCE);

	// set up interrupt line
	EXTI_InitStructure.EXTI_Line = KEY1_EXTI_LINE;
	EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
```

```
        EXTI_InitStructure.EXTI_LineCmd = ENABLE;
        EXTI_Init(&EXTI_InitStructure);
        EXTI_InitStructure.EXTI_Line = KEY2_EXTI_LINE;
        EXTI_Init(&EXTI_InitStructure);

        // enable interrupt, set up interrupt priority
        NVIC_InitStructure.NVIC_IRQChannel = KEY1_EXTI_IRQN;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
        NVIC_InitStructure.NVIC_IRQChannel = KEY2_EXTI_IRQN;
        NVIC_Init(&NVIC_InitStructure);
}
```

attention， two keys trigger the same interrupt， and the design request that it can only control LED state after interrupt, this simple control can be done in the interrupt handling function. Interrupt response function is

```
//button interrupt handling function
void EXTI9_5_IRQHandler(void)
{
        // check interrupt trigger source is button 1 or not
        if( EXTI_GetITStatus(KEY1_EXTI_LINE) == SET )
        {
                Delay(10000);                          // delay disappear shake
                Led1Toggle();                          // indicator light 1 state reverse

                EXTI_ClearITPendingBit(KEY1_EXTI_LINE); // clear interrupt line hang up bit

        }
        // check interrupt trigger source is button 2 or not
        if( EXTI_GetITStatus(KEY2_EXTI_LINE) == SET )
        {
                Delay(10000);
                Led2Toggle();
                EXTI_ClearITPendingBit(KEY2_EXTI_LINE);
        }
}
```

We use a disappear shake function above, it can use simple software.

```
//delay function
void Delay(int   dly)
{
        while(-- dly);
}
```

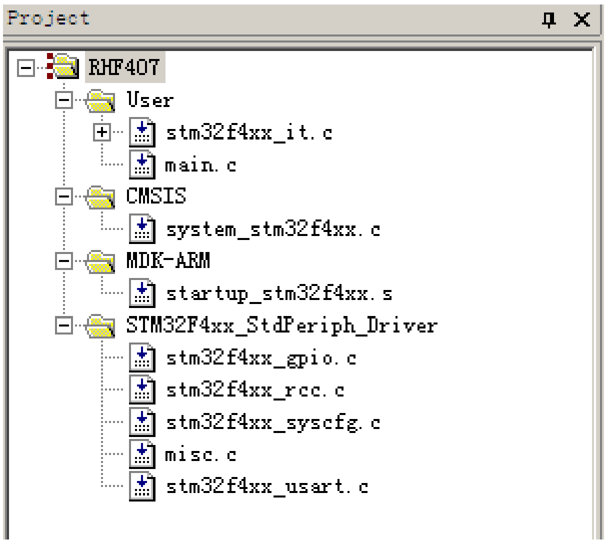
After compile and download, then run program，you can find two LED lights are all light up, because after initial because 口线 is on low voltage state after initialize, call below words, it is default out state when initialized.

```
GPIO_SetBits(GPIO_LED, LED2_PIN |LED2_PIN)
```

Then, every time push the button1, it will lead LED1 state reverse, every time push the button2, it will lead LED2 state reverse.

3.2 Serial port (checking receive way)

Serial port communication is used a lot, the checking receive way is used in many console application. Here we will introduce DB9 standard serial port, for examples, it is OK to use standard straight line connect to PC serial port. Similar to previous, prepare the files, it is quickly to copy project. Files as follows:



Add reference in the main.c file

```
#include "stm32f4xx.h"
#include <stdio.h>
```

Finish main function

```
// main function
int main(void)
{
	uint8  t rxchar:

	ScomConfig():
	printf("Please input char:\r\n"):

	while(1)
	{
		while(USART  GetFlagStatus(SCOM  BASE,USART  FLAG  RXNE) == RESET):
		rxchar = USART  ReceiveData(SCOM  BASE) & 0xFF:
		USART  SendData(SCOM  BASE, (uint8  t)rxchar):
	}
}
```

Use standard print function to output character string after call ScomConfig() initialize serial port. Main cycle is waiting for serial port receive data, when data come, call function USART_Receive Data to receive data to rxchar， at last call USART_Send Data send the data out. We need to check and familiar function in stm32f4xx_usart.c， it is easy to know the purpose from function name.

Do not configure interrupt in the processing of serial port initial, because there is no use interrupt.

- Enable receive and send clock where the Pin port
- Enable present serial port, peripheral clock , serial port 1 and 6 is locate in APB2, others locate in APB1.
- Connect GPIO to serial port receive and send Pin
- Configure interface line mode
- Configure serial port mode, baud rate, initial bit and stop bit
- Enable serial port

Definite hardware resource in front of file.

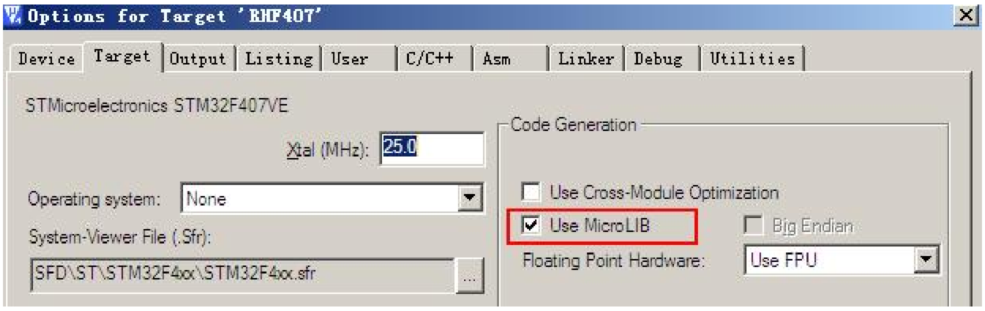
```
//serial definition
#define SCOM  BASE                          UART5
#define SCOM  CLK                           RCC  APB1Periph  UART5
#define SCOM  RCC  CMD                        RCC  APB1
```

```
#define SCOM_TX_PIN                  GPIO_Pin_12
#define SCOM_TX_GPIO_PORT            GPIOC
#define SCOM_TX_GPIO_CLK             RCC_AHB1Periph_GPIOC
#define SCOM_TX_SOURCE               GPIO_PinSource12
#define SCOM_TX_AF                   GPIO_AF_UART5

#define SCOM_RX_PIN                  GPIO_Pin_2
#define SCOM_RX_GPIO_PORT            GPIOD
#define SCOM_RX_GPIO_CLK             RCC_AHB1Periph_GPIOD
#define SCOM_RX_SOURCE               GPIO_PinSource2
#define SCOM_RX_AF                   GPIO_AF_UART5
```

Write serial port initial function

```
// configure serial port
Void ScomConfig(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    // enable RXD and TXD port peripheral clock
    RCC_AHB1PeriphClockCmd(SCOM_TX_GPIO_CLK | SCOM_RX_GPIO_CLK, ENABLE);
    // enable serial port peripheral clock
    RCC_APB1PeriphClockCmd(SCOM_CLK, ENABLE);

    // connect GPIO to serial port pin
    GPIO_PinAFConfig(SCOM_TX_GPIO_PORT, SCOM_TX_SOURCE, SCOM_TX_AF);
    //
    GPIO_PinAFConfig(SCOM_RX_GPIO_PORT, SCOM_RX_SOURCE, SCOM_RX_AF);

    // configure serial port TX as reuse function
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = SCOM_TX_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(SCOM_TX_GPIO_PORT, &GPIO_InitStructure);

    // configure serial port RX as reuse function
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = SCOM_RX_PIN;
    GPIO_Init(SCOM_RX_GPIO_PORT, &GPIO_InitStructure);

    // configure serial port
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl
                                  = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(SCOM_BASE, &USART_InitStructure);

    // enable serial port
    USART_Cmd(SCOM_BASE, ENABLE);
}
```

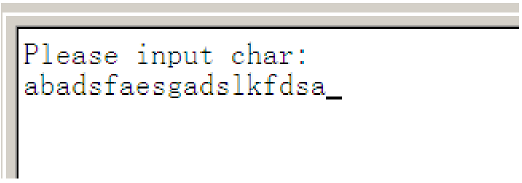
Now it can serial port receive and sent, but the former main function called standard print output function, it need to tell compiler how to output from serial port, and need to choose Use MicroLib in the Target page.



And realize below function

```
//printf realized
int fputc(int ch, FILE *f)
{
        USART_SendData(SCOM_BASE, (uint8_t)ch);
        While (USART_GetFlagStatus(SCOM_BASE, USART_FLAG_TXE) == RESET);

        Return ch;
}
```

Compile and download， you can see below after implement in the HyperTerminal procedures.



Below is input echo bit.

3.3 serial port （interrupt receiving）

Modify on the former project, main cycle of main function do not need code, delete it. Print bit change point to make difference. Main function is as follows:

```
// main function
int main(void)
{
        ScomConfig();
        printf("Uart in interrupt mode, please input char:\r\n");

        While (1)
        {
        }
}
```

Add the serial port interrupt number definition

```
#define SCOM_IRQn                                        UART5_IRQn
```

Add interrupt initialize part in front of enable serial port when serial initializing.

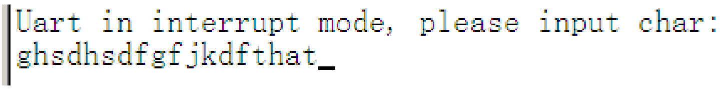
```
        // configure serial interrupt
        NVIC_InitStructure.NVIC_IRQChannel = SCOM_IRQn;
        NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStructure);
        // enable serial port receiving interrupt
        USART_ITConfig(SCOM_BASE, USART_IT_RXNE, ENABLE);
```

Now we need to have an interrupt service function, it will interrupt when receive a character every time, it will read this character in the function and then send it out.

```
// serial port interrupt service procedure serial interrupt service procedure
Void UART5_IRQHandler( void )
{
	if ( USART_GetITStatus(SCOM_BASE, USART_IT_RXNE) != RESET )
	{
		Uint8_t     rxchar;

		rxchar = (USART_ReceiveData(SCOM_BASE) & 0xFF);
		USART_SendData(SCOM_BASE, rxchar);
	}
}
```

attention：compared to checking，here the function is USART_GetITStatus，it is checking interrupt identification USART_IT_RXNE； while former function is USART_GetFlagStatus， it is checking state identification USART_FLAG_RXNE。

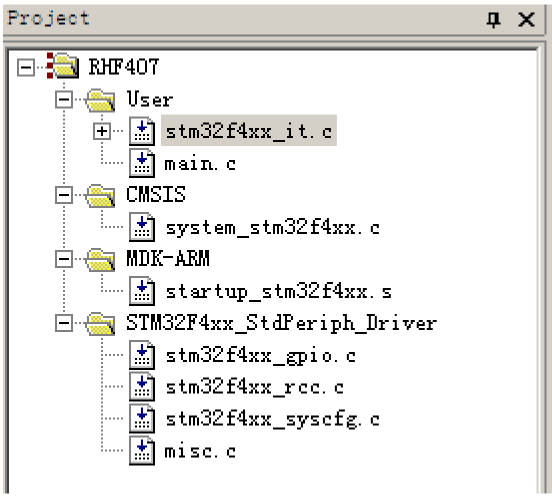
Compile and download， you can see below after running program in the HyperTerminal procedures.



3.4 Systick interrupt control LED

Systick aim to supply meter clock, many people use it to realize delay mode when have no control system. Because there are two 32 bite timer in the STM32F407, more flexible to delay, so here we use it as a common timer.

Files are as follows:



Main function in the main.c file is as follows:

```
// main function
int main(void)
{
	// initialize LED
	LedConfig();
	// initialize   Systick， beat is 1ms， if initialize fails, it will be an endless loop

	if (SysTick_Config(SystemCoreClock / 1000))
	{
		While (1);
```

```
    }

    While (1)
    {
    }
}
```

Definite LED hardware

```
//LED light definition
#define GPIO_LED                GPIOE                           //LED port
#define RCC_GPIO_LED            RCC_AHB1Periph_GPIOE            //GPIO clock used by LED

#define LED1_PIN                GPIO_Pin_3                      // GPIO pin used by LED1
```

Initialize LED

```
// configure LED
Void LedConfig(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_GPIO_LED, ENABLE);        // enable GPIO clock

    GPIO_InitStructure.GPIO_Pin = LED1_PIN;              // configure pin mode
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIO_LED, &GPIO_InitStructure);            // GPIO initialize
}
```

LED state reverse function

```
//LED state reverse
void Led1Toggle(void)
{
    GPIO_ToggleBits(GPIO_LED,LED1_PIN);                  //LED1 state reverse
}
```

Now it still lack a Systick interrupt service function, this function is in the stm32f4xx_it.c, we can modify in there.

```
static unsigned int cnt = 0;
Extern void Led1Toggle(void);
Void SysTick_Handler(void)
{
    cnt++;
    If (cnt >= 500)
    {
        cnt = 0;
        Led1Toggle ();
    }
}
```

Because here we use LED reverse function, so we must add an external declaration in front of function.

```
Extern void Led1Toggle (void);
```

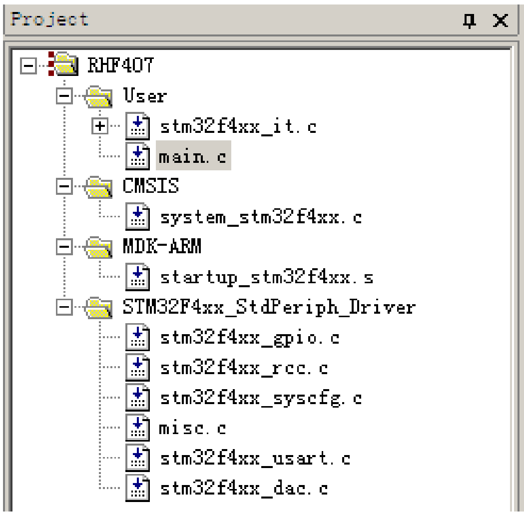
And because interrupts beat is 1 ms, if reversal LED state every time, frequency is 500 hz, the human eye can't be observed phenomenon, so add a global variable count, each interrupt accumulation, meter reset and inversion of the LED state after 500.

Compile and download, run program you can see LED2 will state revers every 1.5s.

3.5 DAC output

The project will configure output voltage by serial port, so it is simple to modify based on checking serial port.

Files need is as follows，Add the peripheral driver library file stm32f4xx_dac.c。



Add DAC initialize function in the main function. Configure the DAC voltage in the main cycle and trigger reverse, waiting serial port input “+”or“-” to modify output voltage.

```
// main function
int main(void)
{
	Uint8_t rxchar;
	Uint16_t dacdata = 0;

	ScomConfig();
	DacConfig();
	printf("\r\nPress '+' or '-' change DAC output voltage.");

	While (1)
	{
		// configure DAC1 and DAC2 output value, they are opposite
		DAC_SetChannel1Data(DAC_Align_12b_R, dacdata);
		DAC_SetChannel2Data(DAC_Align_12b_R, 4095 - dacdata);
		// trigger DAC reverse
		DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);
		DAC_SoftwareTriggerCmd(DAC_Channel_2, ENABLE);
		// print DAC1 output voltage
		printf("\r\nDAC1 voltage= %dmV",(uint16_t)(3300*dacdata/4095));

		// wait input serial bit to control output voltage
		while(USART_GetFlagStatus(SCOM_BASE,USART_FLAG_RXNE)==RESET );
		rxchar = USART_ReceiveData(SCOM_BASE) & 0xFF;
		if(rxchar == '+')
		{
			dacdata += 256;
		}
		else if(rxchar == '-')
		{
			dacdata -= 256;
		}
		dacdata %= 4096;
	}
}
```

Initialize DAC

```
//    configure DAC
void DacConfig(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
        DAC_InitTypeDef    DAC_InitStructure;

        // enable port clock
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
        // enable DAC peripheral clock
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

        // configure interface line
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
        GPIO_Init(GPIOA, &GPIO_InitStructure);

        // configure DAC channel
        DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software;
        DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
        DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
        DAC_Init(DAC_Channel_1, &DAC_InitStructure);
        DAC_Init(DAC_Channel_2, &DAC_InitStructure);

        // enable DAC channel
        DAC_Cmd(DAC_Channel_1, ENABLE);
        DAC_Cmd(DAC_Channel_2, ENABLE);
}
```

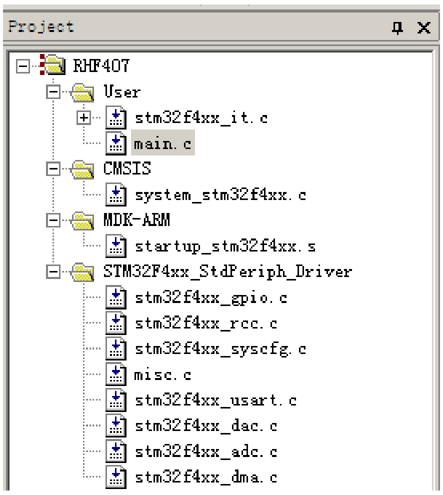
Choose trigger mode when configurate DAC channel, enable output buffer.

Compile and download, run program, we can use“+”or“-”to change DAC1 output voltage ， use multimeter to check DAC1 voltage on CN14.

```
Press '+' or '-' change DAC output voltage.
DAC1 voltage = 0mV
DAC1 voltage = 206mV
DAC1 voltage = 412mV
DAC1 voltage = 618mV
DAC1 voltage = 412mV
DAC1 voltage = 206mV
```

3.6 ADC input

This text is based on the DAC output, it use ADC to gather DAC output voltage and send out from serial port. The project will be modify based on DAC output.



In DAC initializing, delate some code about DAC channel 2.Add ADC initialize call in the main function，software trigger and start ADC reverse.configure DAC output in the main cycle, Print DAC and ADC valotage from serial port after delay, waiting to modify DAC voltage. Code is as follows:

```
// main function
int main(void)
{
    uint8_t rxchar;
    uint16_t dacdata = 0;
    uint32_t i;

    ScomConfig();
    DacConfig();
    AdcConfig();
    printf("\r\nPress '+' or '-' change DAC output voltage.");

    // start ADC reverse
    ADC _SoftwareStartConv(ADC1);

    while(1)
    {
        // configure DAC output voltage and trigger reverse
        DAC_SetChannel1Data(DAC_Align_12b_R, dacdata);
        DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);

        // delayed awaiting DAC input stable and ADC gathered well .
```

```
        for(i = 0; i < 10000; i++);
        //  show DAC and ADC value
        printf("\r\nVDAC = %dmV, VADC = %dmV",
                        (uint16_t)(3300*dacdata / 4095),
                        (uint16_t)(3300*adcvalue / 4095));

        // wait setial port configurate output voltage.
        while(USART_GetFlagStatus(SCOM_BASE,USART_FLAG_RXNE)==RESET );
        rxchar = USART_ReceiveData(SCOM_BASE) & 0xFF;
        if(rxchar == '+')
        {
            dacdata += 256;
        }
        else if(rxchar == '-')
        {
            dacdata -= 256;
        }
        dacdata %= 4096;
    }
}
```

ADC modules use DMA, configured to atomatic reverse mode, then it can work as design, and send sampling value assignment to global variable adcvalue.

Definite ADC1 data register address and sampling value variable

```
#define  ADC1_DR_ADDRESS    ((uint32_t)0x4001204C)  //ADC1 DR register address
uint16_t adcvalue = 0;
```

when initialize ADC, use ADC1, its DMA can use flow 0's channel 0 and channel 4 of DMA2， here we use channel 0. Code is as follows:

```
// configure ADC
void AdcConfig(void)
{
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    DMA_InitTypeDef DMA_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;

    // enable port peripheral and DMA2 clock
```

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2|RCC_AHB1Periph_GPIOC,ENABLE);
// enable ADC1 modual clock
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

// DMA2 flow 0 channel 0 configure
DMA_InitStructure.DMA_Channel = DMA_Channel_0;
DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&adcvalue;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_ADDRESS;
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_Enable;
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);

// enable DMA2 flow 0
DMA_Cmd(DMA2_Stream0, ENABLE);

// -------------------ADC relate configuration---------------------
// GPIO configuration
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOC, &GPIO_InitStructure);

// ADC common configuration
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);

        // ADC channel 10 configuration
        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(ADC1, &ADC_InitStructure);
        ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_3Cycles);

        // enable DMA request
        ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

        // enable ADC1's DMA
        ADC_DMACmd(ADC1, ENABLE);

        // enable  ADC1
        ADC_Cmd(ADC1, ENABLE);
}
```

Use dupont line to short-circuit DAC1 in the CN14 and ADC1 in the CN9.download and run program， use “+” or “-” to change DAC output voltage in the hyperTerminal ,you can see ADC voltage.as follows:

```
Press '+' or '-' change DAC output voltage.
VDAC = 0mV, VADC = 39mV
VDAC = 206mV, VADC = 188mV
VDAC = 412mV, VADC = 402mV
VDAC = 618mV, VADC = 615mV
VDAC = 825mV, VADC = 831mV
VDAC = 1031mV, VADC = 1042mV
VDAC = 825mV, VADC = 833mV
VDAC = 618mV, VADC = 614mV
VDAC = 412mV, VADC = 402mV
VDAC = 206mV, VADC = 187mV
VDAC = 0mV, VADC = 40mV_
```

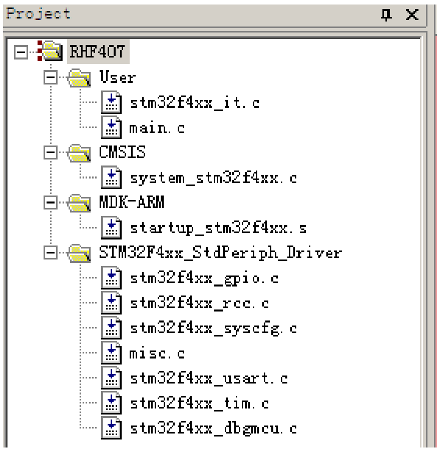
You can see ADC voltage will change as DAC voltage.

3.7 Basic timer interrupt

STM32 timer resource is rich and powerful, here we introduce a simple purpose,it is time interrupt, output the interrupt times in the interrupt service function. Compared to STM32F1xx ,STM32F4xx have two 32 bit timer, they are timer2 and timer5, here we use timer2.

Timer2 is locate in APB1 peripheral， when factional factor bigger than 2, input clock frenquency is two time APB1. Here the system clock frequency is 168MHz，APB1 factional is 4， so timer clock frequency is（168MHz/4）*2=84MHz。

Here we still use serial checking input to modify, files it need is



Add timer initial call in the main function, code is as follows:

```
// main function
int main(void)
{
        ScomConfig();
        TimerConfig();

        while(1)
        {
        }
}
```

Timer initial function is

```
// configure timer， use 32 bit timer 2
void TimerConfig(void)
{
        NVIC_InitTypeDef              NVIC_InitStructure;
        TIM_TimeBaseInitTypeDef       TIM_TimeBaseStructure;
```

```
    // enable timer peripheral clock
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    // timer interrupt vector configuration
    TIM_DeInit(TIM2);
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // timer parameter configuration
    TIM_TimeBaseStructure.TIM_Prescaler=(SystemCoreClock/2)/1000000-1;
//clock1MHz
    TIM_TimeBaseStructure.TIM_Period = 250000 - 1; //
250ms
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    // delate TIM2 overflow interrupt flag
    TIM_ClearFlag(TIM2, TIM_IT_Update);

    // timer overflow interrupt enable
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

    // when debugging, timer stop counting
    DBGMCU_APB1PeriphConfig(DBGMCU_TIM2_STOP, ENABLE);

    // start counting
    TIM_Cmd(TIM2, ENABLE);
}
```

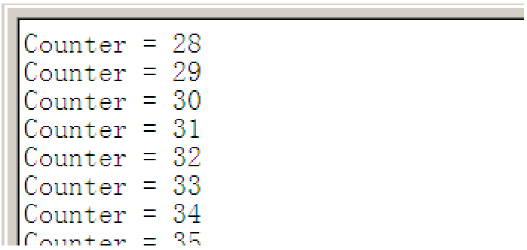
In the above code， SystemCoreClock is system clock frequency, it is definited as168MHz in the system_stm32f4xx.c. call DBGMCU_APB1PeriphConfig, it can stop counting while debugging, otherwise it will comulative, then we will not see our expection.

Now there is lack of movement when interrupt the reponse function. As follows:

```
// timer interrupt service function
```

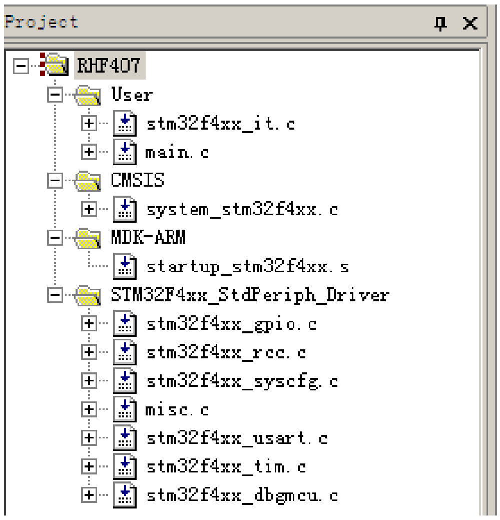
```
uint32  t cnt = 0;
void TIM2_IRQHandler(void)
{
    if( TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET )
    {
        // clear interrupt identification
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);

        // processing
        cnt++;
        printf("\r\nCounter = %d", cnt);
    }
}
```

Compile and download,run program, check it by HyperTerminal, you can find the intrrupt message will send out every 250ms.as follows:



3.8 Timer hardware delay

Here we will use timer2 to process hardware delay, for improve program running efficiency, byte operate register. Check project modify from serial port. The file project need is



Main function is

```
// main function
int main(void)
{
    uint32_t cnt = 0;

    ScomConfig();
    TimerConfig();
    printf("\r\nTimer delay example.");

    while(1)
    {
        printf("\r\nCounter = %d", cnt);
        cnt++;
        TimerDelayMs(1000);
    }
}
```

Initialize serial port and timer into main cycle, output delay time in the cycle, it will delay 1000ms every time.

Timer initialize function is

```
// configure function， use 32 bit timer2
void TimerConfig(void)
{
    RCC->APB1ENR  |=  RCC_APB1ENR_TIM2EN;                    //  enable  timer  2
peripheral clock
    RCC->AHB1RSTR |= RCC_APB1RSTR_TIM2RST;                   // reset timer 2
    DBGMCU_APB1PeriphConfig(DBGMCU_TIM2_STOP, ENABLE);  // timer stop counting
```

```
when debugging
    }
```

In the initialize, Enable timer and configure stop counting function when debugging.

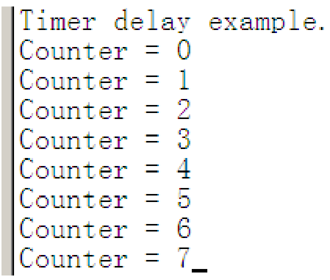
Millisecond delay procedure is

```
    // ms grade delay function
    void TimerDelayMs(uint32_t nms)
    {
        TIM2->PSC = 41999;                          //  timer  clock  frequency  is  84MHz ,
fractional frequency is 2KHz
        TIM2->CNT = 0;
        TIM2->ARR = nms*2;
        TIM2->CR1 = 0x01;                        // start counting
        while( 0 == (TIM2->SR) );
        TIM2->CR1 = 0;                           // stop counting
        TIM2->SR = 0;
    }
```

Because it is ms grade delay, so clock fractional frequency configured as clock frequency to solve 1Khz. Because fractional frequency coefficient is 0~65535， it is impossible to get 1KHz from 84MHz， but 2KHz is OK， so configure fractional frequency coefficient is（42000-1）。Because it is 0.5ms count pulse，so if want to get strong delay time, must delay microsecond multiply 2, assignment to timer automatic reset register, start counting. Then check state register all the time, until the state is not 0, it indicate that delay time is ready, stop counting and let state register return.

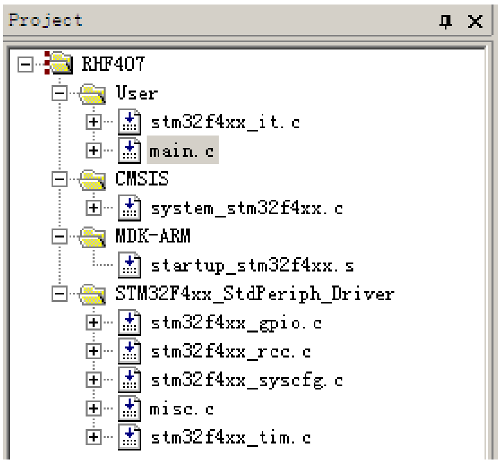
we can get microsecond delay subprogram after modify percale value. You can try if you have interest

After the procedure compiled and run, we can see plus 1 count value very second through HyperTerminal. As follows:


```
Timer delay example.
Counter = 0
Counter = 1
Counter = 2
Counter = 3
Counter = 4
Counter = 5
Counter = 6
Counter = 7_
```

3.9 Timer generate PWM waveform

Here we use timer 3 generate 4 PWM waveforms. They are have same frequency but different duty cycle. Find a former project, it need this follow file:



Main function is simple， it only need to initialize PWM and get into endless cycle.

```
// main function
int main(void)
{
    PwmConfig();

    while(1)
    {
    }
}
```

PWM initialize

```
uint16_t CCR1_Val = 139;
uint16_t CCR2_Val = 279;
uint16_t CCR3_Val = 419;
uint16_t CCR4_Val = 559;
uint16_t PrescalerValue = 0;
// configure PWM output
void PwmConfig(void)
{
    TIM_TimeBaseInitTypeDef     TIM_TimeBaseStructure;
    TIM_OCInitTypeDef           TIM_OCInitStructure;
    GPIO_InitTypeDef            GPIO_InitStructure;

    // timer clock enable
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
    // PWM function clock enable
```

```
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

    // configure PWM GPIO mode
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
    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(GPIOC, &GPIO_InitStructure);

    // connect GPIO to timer
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_TIM3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_TIM3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_TIM3);

    /*-----------------------------------------------------------------------
System clock frequency SystemCoreClock = 168MHz
    APB1 clock frequency PCLK1 = SystemCoreClock / 4 = 42MHz
Timer clock frequency TIM3CLK = 2 * PCLK1 = 84MHz

If timer count clock is 21MHz， prescale value is
    Prescaler = (TIM3CLK / TIM3 counter clock) - 1
              = ((SystemCoreClock /2) /21 MHz) - 1
              = 3

If PWM waveform frequency is 30KHz， Automatically reinstall values
    ARR = (timer count clock frequency / 30000) - 1 = 699;

Every channel duty cycle：
Channel 1：CCR1/700 = 20%
Channel 2：CCR2/700 = 40%
Channel 3：CCR3/700 = 60%
Channel 4：CCR4/700 = 80%
    -----------------------------------------------------------------------*/
```

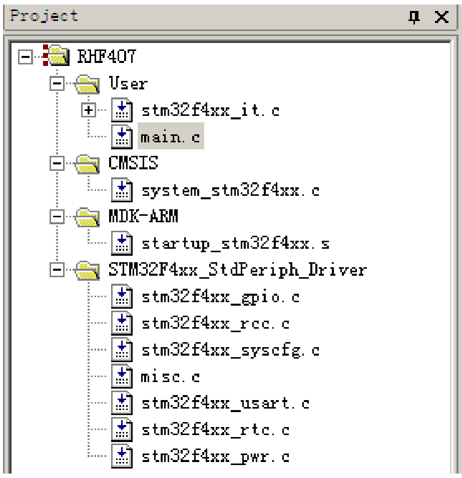
```
// calculate prescale value
PrescalerValue = (uint16_t) ((SystemCoreClock /2) / 21000000) - 1;

// timer configuration
TIM_TimeBaseStructure.TIM_Period = 699;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

// PWM1 mode channel 1 configuration
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR1_Val;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
TIM_OC1Init(TIM3, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);

// PWM1 mode channel 2 configuration
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR2_Val;
TIM_OC2Init(TIM3, &TIM_OCInitStructure);
TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);

// PWM1 mode channel 3 configuration
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR3_Val;
TIM_OC3Init(TIM3, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);

// PWM1 mode channel 4 configuration
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR4_Val;
TIM_OC4Init(TIM3, &TIM_OCInitStructure);
TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);
```

```
        // allow automatic reinstall
        TIM_ARRPreloadConfig(TIM3, ENABLE);

        // enable timer
        TIM_Cmd(TIM3, ENABLE);
}
```

Compile, download and run, we use oscilloscope , and we can see the 30Khz waveform from PWM5~PWM on the CN12 and CN13.

3.10RTC

RTC real time clock is a individual unit, it can still work even CPU have no charge when there is spare battery. RTC is a real clock in the STM32F4xxnot just a counter to read minute and second. In this example it involves how to configure time and how to read time. Check receiving project modify according to serial port, files the project need is:



Main function is

```
// main function
int main(void)
{
        uint32_t i;

        ScomConfig();
        RtcConfig();
        printf("Config RTC OK!\r\n");
```

```
    while(1)
    {
        for(i = 0; i < 25000000; i++);    // delay
        RTC_TimeShow();                              // output present time
    }
}
```

After initialize serial port and RTC, read delay some time and send RTC value.

RTC reserve is located in the area, so long as has the backup battery, can after configured from processor cores have been run.Should check whether the configuration after each processor reset it, if you have configured a reconfiguration is generally do not. This can be realized by using backup registers that set a backup to the specified register after writing a specific value, read the register after reset, and the specific value as ever configuration RTC.

To protect RTC register value， the core will not be allowed to process RTC after reset.

RTC configure subprogram is

```
// configure RTC
uint32_t AsynchPrediv = 0, SynchPrediv = 0;
void RtcConfig(void)
{
    // read backup region register 0, check if it is specific value 0x32F2
    if (RTC_ReadBackupRegister(RTC_BKP_DR0) != 0x32F2)
    {
        printf("\r\nNot set RTC, now set it......");

        // enable PWR clock
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

        //    allow access to RTC
        PWR_BackupAccessCmd(ENABLE);

        // enable external low frequency crystal
        RCC_LSEConfig(RCC_LSE_ON);

        // waiting external low frequency crystal stable
        while(RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET);

```

```
        // choose RTC clock is external low frequency crystal
        RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

        SynchPrediv = 0xFF;
        AsynchPrediv = 0x7F;

        // enable RTC clock
        RCC_RTCCLKCmd(ENABLE);

        // waiting    RTC APB register   synchronized
        RTC_WaitForSynchro();

        // configure RTC data register and presclar coefficient
        RTC_InitStructure.RTC_AsynchPrediv = AsynchPrediv;
        RTC_InitStructure.RTC_SynchPrediv = SynchPrediv;
        RTC_InitStructure.RTC_HourFormat = RTC_HourFormat_24;
        if (RTC_Init(&RTC_InitStructure) == ERROR)
        {
            printf("\n\r RTC Prescaler Config failed.\n\r");
        }

        // configure time register
        RTC_TimeRegulate();
    }
    else
    {
        printf("\n\r No need to configure RTC....\n\r");

        // enable PWR clock
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);

        // allow access RTC
        PWR_BackupAccessCmd(ENABLE);

        // waiting APB register   synchronized
```

```
        RTC_WaitForSynchro();
    }
}
```

When you do not have configured RTC before, Call RTC_TimeRegulate()to configurate present time， code is as follows:

```
// reture to the time that user input
void RTC_TimeRegulate(void)
{
    uint32_t tmp_hh = 0xFF, tmp_mm = 0xFF, tmp_ss = 0xFF;

    printf("\n\r==============Time Settings=====================================\n\r");
    RTC_TimeStructure.RTC_H12      = RTC_H12_AM;
    printf("    Please Set Hours:\n\r");
    while (tmp_hh == 0xFF)
    {
        tmp_hh = USART_Scanf(23);
        RTC_TimeStructure.RTC_Hours = tmp_hh;
    }
    printf("    %0.2d\n\r", tmp_hh);

    printf("    Please Set Minutes:\n\r");
    while (tmp_mm == 0xFF)
    {
        tmp_mm = USART_Scanf(59);
        RTC_TimeStructure.RTC_Minutes = tmp_mm;
    }
    printf("    %0.2d\n\r", tmp_mm);

    printf("    Please Set Seconds:\n\r");
    while (tmp_ss == 0xFF)
    {
        tmp_ss = USART_Scanf(59);
        RTC_TimeStructure.RTC_Seconds = tmp_ss;
    }
    printf("    %0.2d\n\r", tmp_ss);

```

```
        /* Configure the RTC time register */
        if(RTC_SetTime(RTC_Format_BIN, &RTC_TimeStructure) == ERROR)
        {
                printf("\n\r>> !! RTC Set Time failed. !! <<\n\r");
        }
        else
        {
                printf("\n\r>> !! RTC Set Time success. !! <<\n\r");
                RTC_TimeShow();
                /* Indicator for the RTC configuration */
                RTC_WriteBackupRegister(RTC_BKP_DR0, 0x32F2);
        }

        tmp_hh = 0xFF;
        tmp_mm = 0xFF;
        tmp_ss = 0xFF;
}
```

This subprogram just reminder user input time, USART_Scanf()is realize serial port input data receive and analysis.

In main function cycle call RTC_TimeShow() to read and send present time message, code is as follows:

```
// serial port shows present time
void RTC_TimeShow(void)
{
        // read present time
        RTC_GetTime(RTC_Format_BIN, &RTC_TimeStructure);
        // send time message
        printf("\n\r    Time is :    %0.2d:%0.2d:%0.2d",
                RTC_TimeStructure.RTC_Hours,
                RTC_TimeStructure.RTC_Minutes,
                RTC_TimeStructure.RTC_Seconds);
}
```

Compile and download， you can use HyperTerminal to configure time after the program run well, please attention, the date must be double-digit. Please see below:

```
Not set RTC, now set it......
==============Time Settings=====================
  Please Set Hours:
  13
  Please Set Minutes:
  18
  Please Set Seconds:
  20

>> !! RTC Set Time success. !! <<

  Time is :  13:18:20
Config RTC OK!
  Time is :  13:18:20
  Time is :  13:18:21
  Time is :  13:18:21
  Time is :  13:18:22
  Time is :  13:18:22
  Time is :  13:18:23
  Time is :  13:18:24_
```

3.11 CAN network

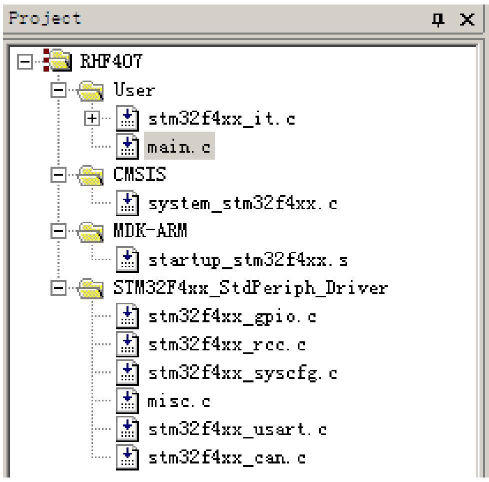
CAN is a popular bus, usually used in car and industry. This board have two individual CAN bus port, communicate with each other， this example is realize communicate in both CAN port, and send data from CAN port serial port.

Configure baud rate before use CAN, Especially the timing parameters can't be too casual, had better be in accordance with the standards。According to their own need to configure the acceptance filter, there can be multiple acceptance filter.

Data reception generally use the interrupt way, not so many requirements on send , only need to pay attention to the message ID, message type (standard frame or extended frame, data frame or remote frame), data length, and data will be send

In this example launched the first packet of data to send by CAN1, send data only one byte 0;CAN2 will add 1 and return data after receiving the message, CAN1will add 1 and return data after receiving the message, So keep going. Every time in the CAN interrupt handle function will send received message to the serial port and time delay.

This project can modify according to serial check receiving project. File need is as follows:



We need to definite a send packet variable and a receive packet for the two CAN port in the main.c file.

```
// main function
int main(void)
{
    // initialization serial port and CAN port
    ScomConfig();
    CanConfig();

    // ready to send text message
    TxMessage1.StdId = 0;
    TxMessage1.ExtId = SCAN2_MSG_ID;
    TxMessage1.RTR = CAN_RTR_DATA;
    TxMessage1.IDE = CAN_ID_EXT;
    TxMessage1.DLC = 1;
    TxMessage1.Data[0] = 0;

    TxMessage2.StdId = 0;
    TxMessage2.ExtId = SCAN1_MSG_ID;
    TxMessage2.RTR = CAN_RTR_DATA;
    TxMessage2.IDE = CAN_ID_EXT;
    TxMessage2.DLC = 1;
    TxMessage2.Data[0] = 0;

    // send initial text message, start to transmit
    CAN_Transmit(SCAN1_BASE, &TxMessage1);
```

```
    while(1)
    {
    }
}
```

define the message IDs received by CAN1 and CAN2

```
#define SCAN1_MSG_ID                    0x0001

#define SCAN2_MSG_ID                    0x0002
```

define CAN hardware resources

// CAN port defination

```
// SCAN1

#define SCAN1_BASE                      CAN1

#define SCAN1_CLK                       RCC_APB1Periph_CAN1

#define SCAN1_RX_PIN                    GPIO_Pin_8

#define SCAN1_TX_PIN                    GPIO_Pin_9

#define SCAN1_GPIO_PORT                 GPIOB

#define SCAN1_GPIO_CLK                  RCC_AHB1Periph_GPIOB

#define SCAN1_AF_PORT                   GPIO_AF_CAN1

#define SCAN1_RX_SOURCE                 GPIO_PinSource8

#define SCAN1_TX_SOURCE                 GPIO_PinSource9


#define SCAN1_RX0_IRQn                  CAN1_RX0_IRQn


// SCAN2

#define SCAN2_BASE                      CAN2

#define SCAN2_CLK                       RCC_APB1Periph_CAN2

#define SCAN2_RX_PIN                    GPIO_Pin_5

#define SCAN2_TX_PIN                    GPIO_Pin_6

#define SCAN2_GPIO_PORT                 GPIOB

#define SCAN2_GPIO_CLK                  RCC_AHB1Periph_GPIOB

#define SCAN2_AF_PORT                   GPIO_AF_CAN2

#define SCAN2_RX_SOURCE                 GPIO_PinSource5

#define SCAN2_TX_SOURCE                 GPIO_PinSource6


#define SCAN2_RX0_IRQn                  CAN2_RX0_IRQn
```

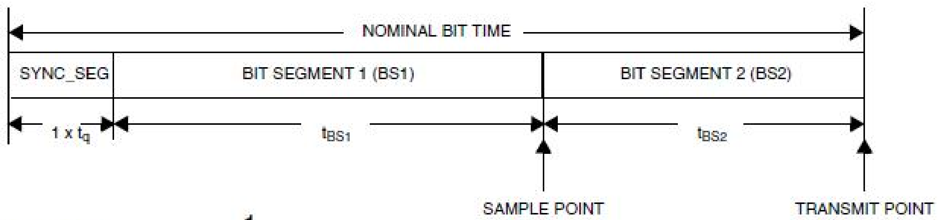
initialize the two CAN buses

```
//configure CAN

void CanConfig(void)

{

	NVIC_InitTypeDef	NVIC_InitStructure;

	GPIO_InitTypeDef	GPIO_InitStructure;

	CAN_InitTypeDef		CAN_InitStructure;

	CAN_FilterInitTypeDef	CAN_FilterInitStructure;


	// Configurate two CAN's interrupt channel

	NVIC_InitStructure.NVIC_IRQChannel = CAN1_RX0_IRQn;

	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0;

	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0;

	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

	NVIC_Init(&NVIC_InitStructure);

	NVIC_InitStructure.NVIC_IRQChannel = CAN2_RX0_IRQn;

	NVIC_Init(&NVIC_InitStructure);


// /Enable port GPIO Clock
	RCC_AHB1PeriphClockCmd(SCAN1_GPIO_CLK, ENABLE);


// link CAN with Pin

	GPIO_PinAFConfig(SCAN1_GPIO_PORT, SCAN1_RX_SOURCE, SCAN1_AF_PORT);

	GPIO_PinAFConfig(SCAN1_GPIO_PORT, SCAN1_TX_SOURCE, SCAN1_AF_PORT);

	GPIO_PinAFConfig(SCAN2_GPIO_PORT, SCAN2_RX_SOURCE, SCAN2_AF_PORT);

	GPIO_PinAFConfig(SCAN2_GPIO_PORT, SCAN2_TX_SOURCE, SCAN2_AF_PORT);


	// configure GPIO Pin

	GPIO_InitStructure.GPIO_Pin =
SCAN1_RX_PIN|SCAN1_TX_PIN|SCAN2_RX_PIN|SCAN2_TX_PIN;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;

	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;

	GPIO_InitStructure.GPIO_PuPd	= GPIO_PuPd_UP;

	GPIO_Init(SCAN1_GPIO_PORT, &GPIO_InitStructure);


//	enable	CAN peripheral clock
```

```
RCC _APB1PeriphClockCmd(SCAN1_CLK | SCAN2_CLK, ENABLE);

// CAN register restore defaults
CAN _DeInit(SCAN1_BASE);
CAN _DeInit(SCAN2_BASE);

//  configure CAN node cell
CAN _InitStructure.CAN_TTCM = DISABLE;
CAN _InitStructure.CAN_ABOM = DISABLE;
CAN _InitStructure.CAN_AWUM = DISABLE;
CAN _InitStructure.CAN_NART = DISABLE;
CAN _InitStructure.CAN_RFLM = DISABLE;
CAN _InitStructure.CAN_TXFP = DISABLE;
CAN _InitStructure.CAN_Mode = CAN_Mode_Normal;
CAN _InitStructure.CAN_SJW = CAN_SJW_1tq;

//   Set baud rate
CAN _InitStructure.CAN_BS1 = CAN_BS1_6tq;
CAN _InitStructure.CAN_BS2 = CAN_BS2_8tq;
CAN_InitStructure.CAN_Prescaler = 2;
CAN _Init(SCAN1_BASE, &CAN_InitStructure);
CAN _Init(SCAN2_BASE, &CAN_InitStructure);

//CAN1 acceptance filter structure
CAN_FilterInitStructure.CAN_FilterNumber = 0;
CAN_FilterInitStructure.CAN_FilterMode = CAN_FilterMode_IdMask;
CAN_FilterInitStructure.CAN_FilterScale = CAN_FilterScale_32bit;
CAN_FilterInitStructure.CAN_FilterIdHigh =
        (( ( uint32_t )SCAN1_MSG_ID << 3) & 0xFFFF0000) >> 16;
CAN_FilterInitStructure.CAN_FilterIdLow =
        (((  uint32_t)SCAN1_MSG_ID<<3)|CAN_ID_EXT|  CAN_RTR_DATA)  &  0xFFFF;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh =
        (( ( uint32_t )0xFF << 3) & 0xFFFF0000) >> 16; //0xFFFF;
CAN_FilterInitStructure.CAN_FilterMaskIdLow =
        (( ( uint32_t )0xFF << 3) | CAN_ID_EXT | CAN_RTR_DATA) & 0xFFFF; //0xFFFF;
CAN_FilterInitStructure.CAN_FilterFIFOAssignment = 0;
```

```
        CAN_FilterInitStructure.CAN_FilterActivation = ENABLE;
        CAN_FilterInit(&CAN_FilterInitStructure);

    //CAN2 acceptance filter structure
        CAN_FilterInitStructure.CAN_FilterNumber = 14;
        CAN_FilterInitStructure.CAN_FilterIdHigh =
                (( ( uint32_t )SCAN2_MSG_ID << 3) & 0xFFFF0000) >> 16;
        CAN_FilterInitStructure.CAN_FilterIdLow =
                (( ( uint32_t )SCAN2_MSG_ID << 3) | CAN_ID_EXT | CAN_RTR_DATA) &
0xFFFF;
        CAN_FilterInit(&CAN_FilterInitStructure);

    //  enable FIF00 message suspend and interrupt
        CAN _ITConfig(SCAN1_BASE, CAN_IT_FMP0, ENABLE);
        CAN _ITConfig(SCAN2_BASE, CAN_IT_FMP0, ENABLE);
    }
```

Attention shall be paid to baud rate and acceptance filter configuration during initialization. System Test describes the computational formula of baud rate as follows:



$$BaudRate = \frac{1}{NominalBitTime}$$

$$NominalBitTime = 1 \times t_q + t_{BS1} + t_{BS2}$$

with:

$t_{BS1} = t_q \times (TS1[3:0] + 1)$,

$t_{BS2} = t_q \times (TS2[2:0] + 1)$,

$t_q = (BRP[9:0] + 1) \times t_{PCLK}$

where t_q refers to the Time quantum

t_{PCLK} = time period of the APB clock,

BRP[9:0], TS1[3:0] and TS2[2:0] are defined in the CAN_BTR Register.

In a simple way:

In this case, the system clock is set as 120 MHz in file system_stm32f4xx.c, CANAPB1 peripheral clock shows four frequency of system clock, which equals to 30MHz. If the frequency factor is two, according to the formula above, baud rate can be calculated as:

Acceptance filters of CAN1 and CAN2 are set to receive previously defined ID message and extend teledata.

```
// CAN1 interrupt receive function
void CAN1_RX0_IRQHandler(void)
{
        uint32_t i;
//      read all FIFO data
        while (CAN_MessagePending(SCAN1_BASE, CAN_FIFO0))
        {
                CAN_Receive(SCAN1_BASE, CAN_FIFO0, &RxMessage1);
                for(i = 0; i < 25000000; i++);
                TxMessage1.Data[0] = RxMessage1.Data[0] + 1;
                printf("\r\n CAN1 recv 0x%02x", RxMessage1.Data[0]);
                CAN _Transmit(SCAN1_BASE, &TxMessage1);
        }
}

// CAN2 interrupt receive function
void CAN2_RX0_IRQHandler(void)
{
        uint32_t i;
        // read all FIFO data
        while (CAN_MessagePending(SCAN2_BASE, CAN_FIFO0))
        {
                CAN_Receive(SCAN2_BASE, CAN_FIFO0, &RxMessage2);
                for(i = 0; i < 25000000; i++);
                TxMessage2.Data[0] = RxMessage2.Data[0] + 1;
                printf("\r\n CAN2 recv 0x%02x", RxMessage2.Data[0]);
                CAN _Transmit(SCAN2_BASE, &TxMessage2);
        }
}
```

Short out pin 1, 2 of JP1 and JP2, and connect CAN buses of CN4 and CN5,then compile download and run, the following will show on the HyperTerminal:

```
CAN2 recv 0x00
CAN1 recv 0x01
CAN2 recv 0x02
CAN1 recv 0x03
CAN2 recv 0x04
CAN1 recv 0x05
CAN2 recv 0x06
CAN1 recv 0x07
```

The first data package was sent by CAN1, so the first line shows CAN2 receiving message.

4 Chassis control routine

4.1 Control protocol introduction

Platform motor drive can control 4 motors at most, while users need 2 commands only, which provide control command based on serial port and CAN bus.

4.1.1 CAN bus control command

CAN bus data package adopt extend date frame in unification, dividing ID into address, command, and sub command. When default of baud rate is 500Kbps, the address will be 0x40.

位 24~28	位 16~23	位 8~15	位 0~7
保留	子命令	功能码	设备地址

Each CAN data package could transmit 8 byte data at most, multibyte data(such as floating-point data) array in data area in form of little end structure(low-order bytes stay at low-order address).

4.1.1.1 User-defined motor speed

Subcommand	Function code	address	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
0	0x29	0x40	Motor 1 speed		Motor 2 speed		Motor 3 speed		Motor 4 speed	

Direct control the rotational linear speed of each wheel. Motor speed is a 16 bits integer, 0.1mm/s as the unit. For example, 1000 stands for 0.1m/s of motor speed.

4.1.1.2 triaxial speed control

Subcommand	Function code	address	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
0	0x2A	0x40	X directional speed		Y directional speed		autorotation rate		Reserve, 0	

Direct control the rotational linear speed of each wheel. Motor speed is a 16 bits integer, 0.1mm/s as the unit. For example, 1000 stands for 0.1m/s of motor speed.

X and Y directional speed that control platform’s movement and autorotation rate. Unit speed is 0.1mm/s for both X and Y directional speed, and autorotation rate is 0.0001rad/s, setting reserved bits to be zero. The speeds in three directions are 16 bits integers.

4.1.2 Serial port control command

Serial port command is transmitted in form of binary system.

Start character	Device type	Device address	Function code	Data length	Data area(N)	CRC check	End character
0xAA						（L，H）	0x0D

Explanation of each area:

Start character: One byte. the start of a data package. The fixed format is 0xAA.

Device type: 0x51.

Device address: 0x40。

Function code: one byte, expressing function command. Users get access to the device, with function code provided by the manufacturer, in accordance with specified format.

Data length: one byte, expressing number of bits followed in the valid data area. Number of bits ranging from 0 to 255.

Data area: N bytes, expressing valid data, length is same with number of bits defined by data length. There is no data area when the data length is zero. Multisystem data array in form of little end.

CRC check: 2 bytes. 16 bits CRC check value of all characters from start character to end character in the data area.

End character: one bit, expressing the end of data package. The fixed format is 0x0D

Serial port data package shall be continuously transmitted after all the data are ready. In this way, it can be avoided that receiver keep waiting for unfinished data package while sender is under abnormal condition. This transition process cannot pause for too long, otherwise the receiver would mistake for finished transition, thus lead to error during parsing the data package.

Serial port communication parameters are “9600, N, 8, 1”.

4.1.2.1 user-defined motor speed

Function code	Data length	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
0x29	8	Motor1 speed		Motor2 speed		Motor3 speed		Motor4 speed	

The description for above is same with command of CAN protocol.

4.1.2.2 three directional speed control

Function code	Data length	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
0x2A	8	X directional speed		Y directional speed		autorotation rate		Reserve, 0	

The description for above is same with command of CAN protocol.

4.2 Routine introduction

4.2.1 Routine function

In this routine, RHF407 development board will be used and together with CAN bus command and serial port command to complete two motions control. Users can choose communication interface according to the control system described later, then compile and download the project.

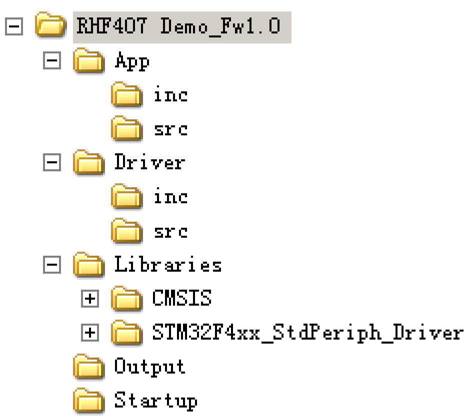
After power on and user pressed the button SW2, platform will conduct rectangle and diagonal movement sequence, which can be realized by calling three directional speed control command. When the button SW3 is pressed, platform will rotate in situ, which can be realized by callig user-defined motor speed.

If the parameters were examined to be error when calling command after SW2 or SW3 is pressed, LED 3 will be lightened. LED3 is in off status when parameters are right.

4.2.2 Set up project

According to the function description above, visible-to-user hardware resources are indicator lights, buttons, and serial ports and CAN bus. Delayer and timer will also be used in some normal situations.

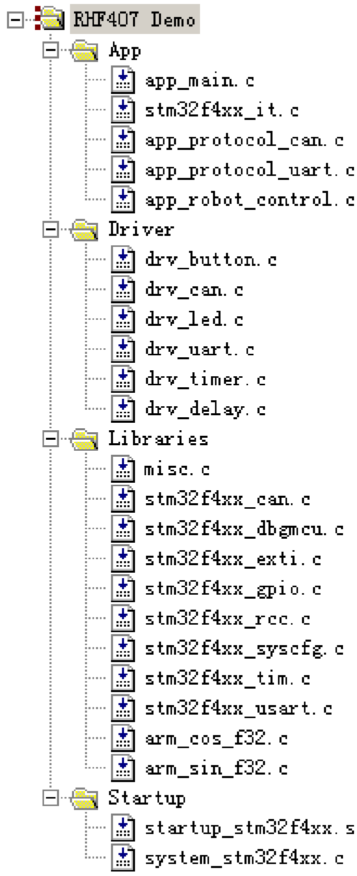
Firstly plan directory structure for project, and create some empty file(drv_ or app_ prefixal files).



Operation project folder named as RHF407 Demo, which includes all the codes needed for the whole firmware project.

- RHF407 Demo_Fw1.0: firmware source code project folder
  - App: user program, usually function code
    - Inc: user program header file
      - stm32f4xx_conf.h
      - stm32f4xx_it.h
      - app_protocal_uart.h
      - app_protocal_can.h
      - app_robot_control.h
    - Src:user program C file
      - app_main.c
      - app_robot_control.c
      - app_protocal_uart.c
      - app_protocal_can.c
      - stm32f4xx_it.c
  - Driver: drive program writed by user
    - Inc: drive program header file
      - drv_button.h
      - drv_can.h
      - drv_delay.h
      - drv_led.h
      - drv_timer.h
      - drv_uart.h
    - Src: drive program C file
      - drv_button.c
      - drv_can.c
      - drv_delay.c
      - drv_led.c
      - drv_timer.c
      - drv_uart.c
  - Libraries: office libraries file. Delete needless file for project, and move stm32f4xx.h and system_stm32f4xx.h to CMSIS\Include, delete empty folder. Sine and cosine calculation function in DSP file will be used, so copy arm_cos_f32.c and arm_sin_f32.c to Libraries\CMSIS.
  - Startup: start-up file, copy system_stm32f4xx.c and startup_stm32f4xx.s from official liabraries file.
    - startup_stm32f4xx.s
    - system_stm32f4xx.c
  - RHF407 Demo.uvproj and other project file
- Output: compile output file storage directory

Finally project needed files are：



Attention should be paid that, when the processor is working with the highest rated frequency 168MHz, it may be hard to reach CAN baud rate. To avoid too much performance reduction, it is suggested to adopt 160MHz. All the peripheral are in low speed, so the frequency do not need to be too high, but fractional frequency could be set larger. Alter file system_stm32f4xx.c.

```
#define PLL_M      25
#define PLL_N      320
```

```
  uint32_t SystemCoreClock = 160000000;
```

```
    /* PCLK2 = HCLK / 8*/
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV8;

    /* PCLK1 = HCLK / 4*/
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;
```

To start hardware floating point unit, macro definition should be also added at C/C++page for the project.

- USE_STDPERIPH_DRIVER
- STM32F4XX
- __FPU_PRESENT = 1
- __FPU_USED = 1
- ARM_MATH_CM4

4.2.3 Key file description

Realization of each drive program has been explained in previous basic routine, but only the frame is different. CAN bus driver is different, especially for baud rate and acceptance filter. Baud rate could be set as 500Kbps, acceptance filter would accept all IDs' surround, rather than select as before. Now introduction of other files:

4.2.3.1 app_main.c

Program subject. User control code start from main function.

```
//============================================================================
===
// Name: main
// Function: C main function
// parameter: none
// return:   none
// explaination: none
//============================================================================
===
int main( void )
{
    app_main_system_init();

    while ( 1 )
    {
        if( drv_button_get_status(Button2) )
        {
            //    recheck button status
            drv_delay_ms(200);
            if( 0 == drv_button_get_status(Button2) )
            {
                continue;
            }
            // if press buttion 2,move in a straight line, clockwise, 02m/s, 0 direction, rectangle, length of sides 08m and 1.6m separately
            if( app_robot_control_move_rectangle(1, 0.2, 0, 0.8, 1.6) )
            {
                //parameter error, LED3 light on
                drv_led_on(LED3);
            }
```

```
			else
			{
				drv_led_off(LED3);
			}
		}

		if( drv_button_get_status(Button3) )
		{
			// recheck button status
			drv_delay_ms(200);
			if( 0 == drv_button_get_status(Button3) )
			{
				continue;
			}
			// if press buttion3, rotate movement, linear speed of wheel 0.2m/s, rotate time 20s
			if( app_robot_control_move_rotate(0.2, 20000) )
			{
				// parameter error, LED3 light on
				drv_led_on(LED3);
			}
			else
			{
				drv_led_off(LED3);
			}
		}
	}
}
```

Enter into main circulation status after initialization, check button status during circulation. Select different buttons according to aimed movement. call initialization programs of each driver in system initialization function app_main_system_init.

```
//=============================================================================
// name: app_main_system_init
// function:	system initialization
// parameter: none
// return: none
// explain: none
```

```
//========================================================================
===
void app_main_system_init( void )
{
        drv  led_init();
        drv  button_init();    //    notice key initialization in query mode
        drv  delay_init();
        drv  can_init();
        drv_uart_init(UART_BAUDRATE_9600);
}
```

4.2.3.2 app_robot_control.c

This file could realize chassis control action. Header file shall be included at the top for calculation will use sine and cosine function in the DSP function library.

```
#include "arm_math.h"
```

Use the macro selective control command sending mode, serial port or CAN bus

```
// send command from UART, send command from CAN aferannotation
#define SEND  MSG_FROM_UART                                    1
```

define motor driver device type and device address

```
// define platform motor driver address
#define  MOTOR  DIRVER  ADDRESS                                0x51  //  configure  in
accordance with platform manual
// define motor driver device type and device address
#define MOTOR_DRIVER_DEVICE_TYPE                              0x40
```

radian unit is adopted ain angle calculation, in consideration of multiplication is faster than division operation, so add several constants' macro defination

```
// define floating-point calculation constant
#define PI_DIV2                                               (PI / 2)
#define PI_DIV180                                             (PI / 180)
#define   180_DIV_PI                                          (180 / PI)
```

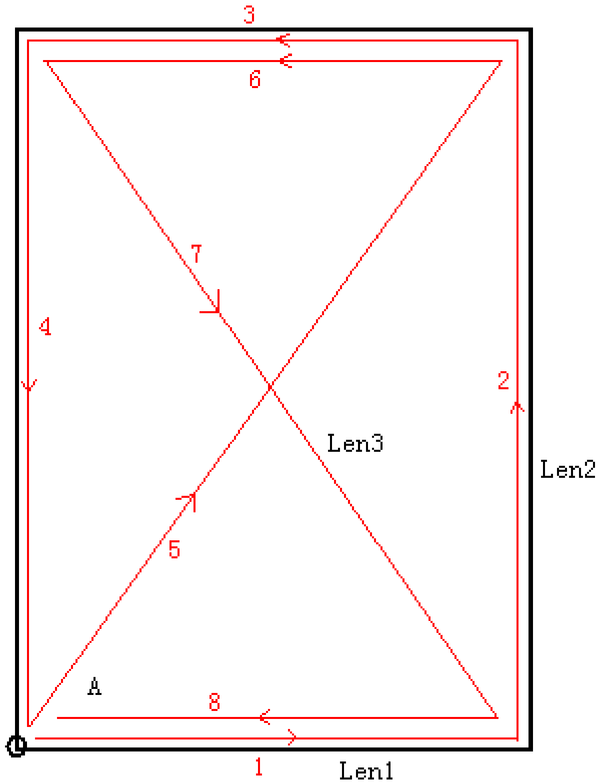
The platform will finally stop whatever the movement mode is.realize stop movement function, write function

```
//========================================================================
===
// name: app_robot_control_move_abort
// function: control robot to stop
// parameter: none
```

```
// return: none
// description: set all the motors' speed to be 0
//=====================================================================
===
void app_robot_control_move_abort( void )
{
    INT8U ZeroSpeed[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };

#ifdef SEND_MSG_FROM_UART
    app_protocol_uart_send_msg(MOTOR_DRIVER_DEVICE_TYPE,
            MOTOR_DIRVER_ADDRESS, UARTCMD_SetMotorSpeed, ZeroSpeed, 8 );
#else
    app_protocol_can_send_package(MOTOR_DIRVER_ADDRESS,
            CANCMD_SetMotorSpeed, 0, ZeroSpeed, 8);
#endif
    drv_delay_ms(500);
}
```

Send command to make all the speed of motors to be 0.through previously defined macro control from serial port CAN send command.

Write the first movement function to be realized rectangle and diagonal movement. For flexible control, it is suggest setting start direction. Set move linear speed, side lengths of rectangle and movement locus to be clockwise or anti-clockwise direction. This movement locus is formed by 8 tangential paths. Take anti-clockwise locus for example, see drawing below:



Command control could only control platform's move speed. Move time of each section could be controlled to control distance. Sudden turn will have applied force on part of motor. To avoid this, add delayer after stop of each movement.

```
//============================================================================
===
// name: app_robot_control_move_rectangle

// function: control platform to realize retangel command.

// parameter：RouteDir：locus，1 means anti-clockwise direction，0 means clockwise direction
// Speed: straight line movement speed, unit m/s,      nonnegative number

// StartDir：initial move direction, -180°~180°

// Len1：length of first side for movement，unit m，positive number

// Len2：length of second side for movement，unit m，positive number

// return: none

// description: can not rotate, rectangle movement then turnback by diagonal movement

//============================================================================
===
INT8U app_robot_control_move_rectangle( INT8U RouteDir, float Speed,

                                    float StartDir, float Len1, float Len2 )

{

    LineMoveDataType LineCmdBuffer[8];        // restore command parameters of 4 sides of rectangle and diagonal movement
    INT32U           DelayMs[8];               // restore delayer of each movement
    float StartDirRad;                         // radian value of start direction

    float SecondLineDirRad;                    // radian value of the second line of
```

```
rectangle
    float DiagonalRad;                              // radian value of diagonal
    float DiagonalLength;                           // length of diagonal
    INT8U i;

    // check validity of parameters

    // calculate radian angle and length of diagonal
    StartDirRad = StartDir * PI_DIV180;
    DiagonalRad = atan2(Len2, Len1);
    DiagonalLength = abs(Len2 / arm_sin_f32(DiagonalRad));

    // calculate the first line
    LineCmdBuffer[0].SpeedX = (INT16S)(Speed * arm_cos_f32(StartDirRad) * 10000);
    LineCmdBuffer[0].SpeedY = (INT16S)(Speed * arm_sin_f32(StartDirRad) * 10000);
    LineCmdBuffer[0].SpeedRotate = 0;
    LineCmdBuffer[0].Acceleration = 0;
    DelayMs[0] = ( INT32U )(Len1 * 1000 / Speed);

    // calculate the second line
    // judge angle viaration according to length value symbol
    if( RouteDir )
    {
        SecondLineDirRad = StartDirRad + PI_DIV2;
    }
    else
    {
        SecondLineDirRad = StartDirRad - PI_DIV2;
    }
    LineCmdBuffer[1].SpeedX   =   (INT16S)(Speed   *   arm_cos_f32(SecondLineDirRad)   *
10000);
    LineCmdBuffer[1].SpeedY = (INT16S)(Speed * arm_sin_f32(SecondLineDirRad) * 10000);
    LineCmdBuffer[1].SpeedRotate = 0;
    LineCmdBuffer[1].Acceleration = 0;
    DelayMs[1] = ( INT32U )(Len2 * 1000 / Speed);

    // speed of the third line and first line in reverse,same time
```

```
// speed of the third line and first line in reverse,same time

// diagonal line starting from original point
if( RouteDir )
{
    SecondLineDirRad = StartDirRad + DiagonalRad;
}
else
{
    SecondLineDirRad = StartDirRad - DiagonalRad;
}
LineCmdBuffer[4].SpeedX = (INT16S)(Speed * arm_cos_f32(SecondLineDirRad) *
10000);
LineCmdBuffer[4].SpeedY = (INT16S)(Speed * arm_sin_f32(SecondLineDirRad) * 10000);
LineCmdBuffer[4].SpeedRotate = 0;
LineCmdBuffer[4].Acceleration = 0;
DelayMs[4] = ( INT32U )(DiagonalLength * 1000 / Speed);

//  copy the third straight line

//  operate another diagonal route
if( RouteDir )
{
    SecondLineDirRad = StartDirRad - DiagonalRad;
}
else
{
    SecondLineDirRad = StartDirRad + DiagonalRad;
}
LineCmdBuffer[6].SpeedX = (INT16S)(Speed * arm_cos_f32(SecondLineDirRad) *
10000);
LineCmdBuffer[6].SpeedY = (INT16S)(Speed * arm_sin_f32(SecondLineDirRad) * 10000);
LineCmdBuffer[6].SpeedRotate = 0;
LineCmdBuffer[6].Acceleration = 0;
DelayMs[6] = DelayMs[4];
```

```
        // operate theirst line in reverse direction, samewith the third parammeter


        // send commands one by one to complete planed routine
        for( i = 0; i < 8; i++ )
        {
    #ifdef SEND  MSG_FROM_UART
app  protocol_uart_send_msg(MOTOR_DRIVER_DEVICE_TYPE,MOTOR_DIRVER_ADDRESS,
                                UARTCMD_SetPlatformSpeed,  (INT8U  *)(&LineCmdBuffer[i]),
8 );
    #else
            app  protocol  can_send_package(MOTOR_DIRVER_ADDRESS,
CANCMD  SetPlatformSpeed,
                                0, (INT8U *)(&LineCmdBuffer[i]), 8);
    #endif
            drv_delay_ms(DelayMs[i]);
            app  robot_control_move_abort();
        }

        return ( 0 );
    }
```

To complete 8 tangential paths, define an array to reserve speed parameters of 8 commands and define the processing time of the array reserving each command. Send in a sequence after calculating all the movements with delayers between each.

Confirm a rectangle according to parameters input, calculate the angle between diagonal and the first line, and calculate length of diagonal according to the angle and line length. In this way it could improve efficiency for narrowed calculation range.

The two functions used above are arm_cos_f32 and arm_sin_f32, which originates from arm_cos_f32 and arm_sin_f32, and their calculation efficiency is higher than that in MDK library.

Some line parameters that are same or reversed are here neglected.

The second movement to be realized is rotation in stubby using set motor speed command; distance from wheel to rotation center of chassis could be used to calculate rotate angular speed. The speeds of all the wheels are the same and wheels will not move. This is user-friendly for starter. Combinations of different motor speeds will create complex movements.

```
    //=================================================================
===
    // name：app_robot_control_move_rotate
    // function：control robot rotate
    // parameter：UsrSpeed：user –defined speed, unit m/s
```

```
//          MoveTimeMs：   time of movement Ms
// return: none
// description: user-defined speed
//=====================================================================
===
INT8U app_robot_control_move_rotate( float UsrSpeed, INT32U MoveTimeMs )
{
        INT8U SpeedDataBuffer[8];
        INT16S tmpSpeed;

        if( UsrSpeed <= 0 )
        {
                return 1;
        }

        if( MoveTimeMs <= 0 )
        {
                return 2;
        }

        tmpSpeed = (INT16S)(UsrSpeed * 10000);        // Linear speed of wheels needs to be
mutitiplied by 1000 for transmission
        SpeedDataBuffer[0] = tmpSpeed & 0xFF;
        SpeedDataBuffer[1] = tmpSpeed >> 8;
        SpeedDataBuffer[2] = SpeedDataBuffer[0];
        SpeedDataBuffer[3] = SpeedDataBuffer[1];
        SpeedDataBuffer[4] = SpeedDataBuffer[0];
        SpeedDataBuffer[5] = SpeedDataBuffer[1];
        SpeedDataBuffer[6] = SpeedDataBuffer[0];
        SpeedDataBuffer[7] = SpeedDataBuffer[1];

    #ifdef SEND_MSG_FROM_UART
        app_protocol_uart_send_msg(                    MOTOR_DRIVER_DEVICE_TYPE,
MOTOR_DIRVER_ADDRESS,
                                   UARTCMD_SetMotorSpeed, SpeedDataBuffer, 8 );
    #else
        app_protocol_can_send_package(MOTOR_DIRVER_ADDRESS,
CANCMD_SetMotorSpeed,
```

```
                                    0, SpeedDataBuffer, 8);
#endif
        drv_delay_ms(MoveTimeMs);
        app_robot_control_move_abort();

        return ( 0 );
}
```

4.2.3.3 app_protocol_can.c

CAN protocol processing file. When receiving a CAN data package in CAN interrupt receiving service program, call app_protocol_can_msg_process in this file to process CAN message.

```
//=====================================================================
===
// name： app_protocol_can_msg_process
// function:CAN message process
// parameter:none
// return: none
// description:none
//=====================================================================
===
void app_protocol_can_msg_process( CanRxMsg RxMsg )
{
        CanCommandType l_CanCmd;

        l_CanCmd.CmdAddress = app_protocol_can_extid_get_address(RxMsg.ExtId);
        l_CanCmd.CmdFunc = app_protocol_can_extid_get_funcode(RxMsg.ExtId);
        l_CanCmd.CmdIndex = app_protocol_can_extid_get_index(RxMsg.ExtId);
        l_CanCmd.pData = RxMsg.Data;
        l_CanCmd.MsgLen = RxMsg.DLC;


app_protocol_can_protocol_parse(l_CanCmd.CmdFunc,l_CanCmd.CmdIndex,l_CanCmd.pData);
}
```

app_protocol_can_extid_get_address extract device address information from message ID. app_protocol_can_extid_get_funcode extract command, app_protocol_can_extid_get_index extract subcommand. Their macro definition as follows:

```
#define app_protocol_can_extid_get_index(ID)        ((INT8U)((ID & 0X00FF0000) >> 16))
#define app_protocol_can_extid_get_funcode(ID)      ((INT8U)((ID & 0X0000FF00) >> 8))
#define app_protocol_can_extid_get_address(ID)      ((INT8U)(ID & 0X000000FF))
```

After extracting information from message process function, call app_protocol_can_protocol_parse to command prasing and execute commond.

```
//=========================================================================
===
// name:app_protocol_can_protocol_parse
// function:CAN protocol prasing
// parameter:Cmd：function code
//          Index：index, subcommand
//          Msg：data
// return：0 means success, others mean fail
// description: none
//=========================================================================
===
INT8U app_protocol_can_protocol_parse( INT8U Cmd, INT8U Index, INT8U *Data )
{
    switch ( Cmd )
    {
    case CANCMD_SetMotorSpeed:              // user defined motor speed
        {
        }
        break;
    case CANCMD_SetPlatformSpeed:           // user defined platform speed
        {
        }
        break;
    default:
        {
            return ( 1 );
        }
    }
    return ( 0 );
}
```

This is just a program frame, no process to the message returned by moter driver.

As for protocol layer, it is important to send packaging of data package according to the protocol. The code as follows:

```
//===========================================================================
===
// name:app_protocol_can_send_package
// function:    send data package through CAN
// parameter:Address：adress
//          Cmd：function code
//          Index：functional code subpackage transmission index
//          Data：data
//          Len：length of data
// return: none
// description: none
//===========================================================================
===
INT8U app_protocol_can_send_package( INT8U Address, INT8U Cmd, INT8U Index,
                                    INT8U *Data, INT8U Len )
{
    CanTxMsg    l_CanTxMsg;

    l_CanTxMsg.ExtId = app_protocol_can_extid_set(Index, Cmd, Address);
    l_CanTxMsg.StdId = 0;
    l_CanTxMsg.DLC = Len;
    l_CanTxMsg.RTR = CAN_RTR_DATA;
    l_CanTxMsg.IDE = CAN_ID_EXT;
    if( Len > 0 )
    {
        INT8U i;

        for( i = 0; i < Len; i++ )
        {
            l_CanTxMsg.Data[i] = Data[i];
        }
    }

    drv_can_send_msg(l_CanTxMsg);

```

```
        return ( 0 );
}
```

app_protocol_can_extid_set is used for settle address, command and subcommandinto message ID. The macro definition is as follows:

```
#define app_protocol_can_extid_set(index, cmd, address)          \
                                        ((INT32U)((index<<16) | (cmd<<8) | address))
```

Call drv_can_send_msg function in drive layer to send out message.

4.2.3.4 app_protocol_uart.c

Serial protocol process file. There is no message processed from serial port, just send out message in form of packaging in accordance with protocol.

```
//==========================================================================
===
// name:app_protocol_uart_send_msg
// function:serial port send fuction
// parameter:DeviceType
//           Address
//           DeviceId：device ID
//           Cmd：data package command code
//           Len：valid data length, one byte
//           Dat：valid data
// return: none
// description: continuously send when transmission, can not suspend, long time suspend willlead receivers to recognize end of transmission.
//==========================================================================
===
void app_protocol_uart_send_msg( INT8U DeviceType, INT8U Address,
                                 INT8U Cmd, INT8U *Dat, INT8U Len )
{
        INT8U     SendBuffer[100];
        INT8U     i = 0, j;
        INT16U    Crc16Value;

        SendBuffer[i++] = UART_MSG_PACKAGE_STARTCHAR;
        SendBuffer[i++] = DeviceType;
        SendBuffer[i++] = Address;
        SendBuffer[i++] = Cmd;
        SendBuffer[i++] = Len;
```

```
        for( j = 0; j < Len; j++ )
        {
            SendBuffer[i++] = Dat[j];
        }
        Crc16Value = app_protocol_uart_crc16(SendBuffer, i);
        SendBuffer[i++] = ( INT8U )Crc16Value;
        SendBuffer[i++] = ( INT8U )(Crc16Value >> 8);
        SendBuffer[i++] = UART_MSG_PACKAGE_ENDCHAR;

        // send out data in buffer area in form ofblocking from serial port
        drv_uart_put_multi_bytes_blocking(SendBuffer, i);
    }
```

Assign variate accordingly for buffer area in function, calculate CRC check value, then attach end mark, continuously send. Calculate CRC check code as follows:

```
    const INT16U crc_ta[256] = { /* CRC余式表 */
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
        0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
        0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
        0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
        0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
        0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
        0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
        0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
        0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
        0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
        0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
        0xff9f. 0xefbe. 0xdfdd. 0xcffc. 0xbf1b. 0xaf3a. 0x9f59. 0x8f78.
        0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
        0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
        0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
```

```
        0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
        0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
        0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
        0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
        0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
        0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
        0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
        0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
        0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};


//============================================================================
===
// name: app_protocol_uart_crc16
// function: calculate checksum
// parameter: *ptr： caculate charater string of checksum
//            len： length of bytes
// return: none
// description: none
//============================================================================
===
static INT16U app_protocol_uart_crc16( INT8U *ptr, INT16U len )
{
        INT16U crc;
        INT8U da;
        crc = 0;
        while ( len-- != 0 )
        {
                da = ( INT8U )(crc >> 8);
                crc <<= 8;
                crc ^= crc_ta[da ^ *ptr];
                ptr++;
        }
        return ( crc );
```