

megawin

MG32F10x *Porting To STM32F10x* *Manual*

Version 1.01
Date 2022/4/11

Index

1. Introduction	5
1.1. Document Using.....	5
2. Hardware difference comparison	6
2.1. Pin differences comparison	6
2.2. Resource comparison	6
3. Development environment setup	7
3.1. Development IDE for MG32F10x	7
3.2. Installation of the development package	7
3.3. Build a project	8
3.4. Peripheral library configuration.....	14
3.5. Debugger configuration	16
3.5.1. Use ST-Link to debug	16
3.5.2. Use J-Link to debug	19
4. Layout suggestion	23
4.1. Printed circuit board	23
4.2. Component location	23
4.3. Grounding and power supply(VSS/VDD)	23
4.4. Decoupling	23
4.5. Power supply scheme	24
4.6. Other signals	24
4.7. Unused IO and its properties.....	24
4.8. Clock	25
4.9. Analog signal.....	25
4.10. EMI	25
5. Peripheral porting	27
5.1. Preparation before porting.....	27
5.2. ADC.....	28
5.3. ANCTL(Analog controller)	30
5.3.1. CMP	30
5.3.2. DCSS	30
5.4. BKP.....	31
5.5. CRC	31
5.6. DMAC.....	31
5.7. EXTI	33
5.8. FMC(Flash memory controller).....	34
5.9. GPIO	34
5.10. I2C.....	35
5.11. I2S.....	37
5.12. IWDG(independent watchdog)	37
5.13. LED	39
5.14. NVIC.....	39
5.15. PWR(power control)	39
5.16. RCC	41
5.17. RNG(Random Number Generator).....	44
5.18. RTC.....	44
5.19. SFM(Special Function Macro)	46
5.20. SPI	46

5.21. SYSTICK.....	47
5.22. TIM.....	49
5.23. UART	52
5.24. USB.....	53
5.25. WWDG(window watchdog)	55
6. Reversion	56

1. Introduction

1.1. Document Using

MG32F10x series is the single-chip 32-bit microcontroller based on a high performance Core ARM 32-bit Cortex™-M3 CPU with embedded Nested Vectored Interrupt Controller (NVIC) launched in 2021 by megawin. MG32F10x series has the advantages of high performance and strong compatibility of software and hardware. This document is intended to help users of STM32F10x series chips port to MG32F10x series.

In general, the hardware porting is relatively simple, can be directly replaced from STM32F10x series chips to MG32F10x series, no need to modify the circuit. However, there are still some differences in the software, so it needs to be modified in the software. This document will also introduce the details of the software modification. Please understand if there are any mistakes or omissions.

2. Hardware difference comparison

2.1. Pin differences comparison

The following table shows the pin differences between MG32F10x series and STM32F10x series.

Package	Chip	
	MG32F10x	STM32F10x
LQFP48	Fully compatible with pin position and function	
LQFP64	Fully compatible with pin position and function	

2.2. Resource comparison

The following table shows the differences of hardware resources between MG32F10x series and STM32F10x series.

	MG32F103	MG32F104	STM32F103
Core	Cortex-M3	Cortex-M3	Cortex-M3
Flash	96K~128K	256K	16K~1M
RAM	28K	36K	6K~96K
Frequency	72MHz	96MHz	72MHz
Access Flash wait state	Cache,no wait cycle	Cache,no wait cycle	2 cycle
Flash write/erase cycle	100k cycle	100k cycle	10k cycle
TIMER	4	4	4/5/8
U(S)ART	3	3	2/3/5
I2C	2	2	1/2
SPI	2	2	1/2/3
I2S	0~1	0~1	2
CAN	--	--	1
USB	Device	Device	Device
SDIO	--	--	1
ADC	1(10~16)	1(10~16)	2(10)/2(16)/3(21)
CMP	2	2	--
LED Driver	8 Segment	8 Segment	--
Active Power	100uA/MHz @3.3V	100uA/MHz @3.3V	292uA/MHz @3.3V
Sleep	5mA	5mA	5.5mA
Stop	30uA	30uA	24uA
Standby	4.5uA	4.5uA	2uA
Vbat	1.2uA	1.2uA	1.4uA

<Note> Flash write/erase cycle: When MCU is running, operations involving Flash access, such as reading instructions and variables, need the wait cycle. The longer wait cycle, the lower the actual operation efficiency.

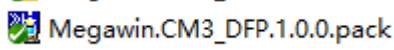
3. Development environment setup

3.1. Development IDE for MG32F10x

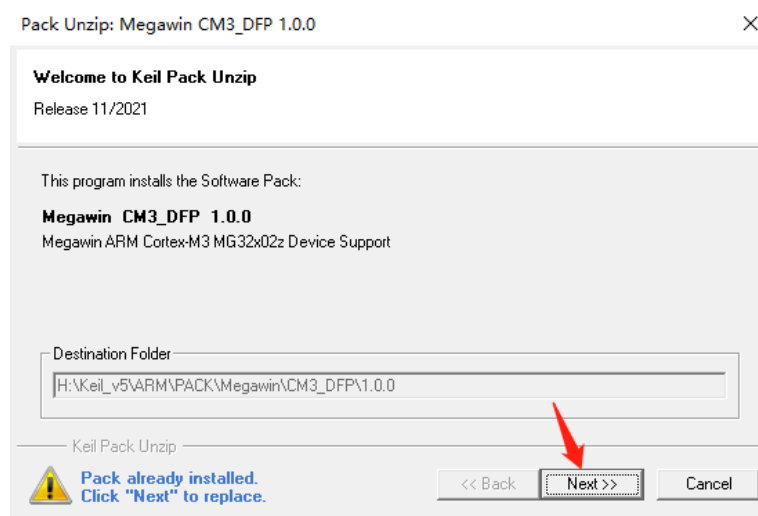
The development of MG32F10x series needs to use the Keil 5 MDK for ARM version. The Keil 5.26 and above version must be installed. **If the version is too low, it will be impossible to identify the development package installation program.**

3.2. Installation of the development package

Open Megawin.CM3.DFP.1.0.0.pack.

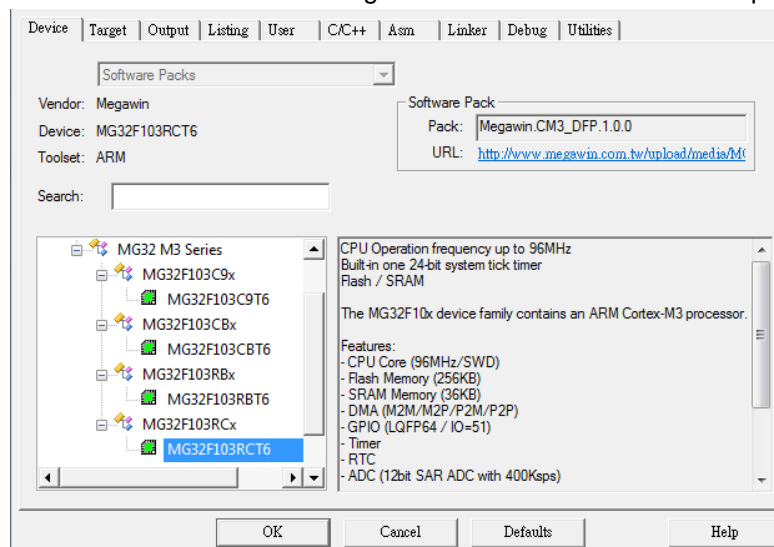


Click Next, and wait for finish.



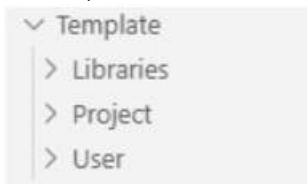
[Notify]: *If the installation package shown in the figure above cannot be opened, the Keil version is too early to run. Use a newer version of Keil.*

MG32F10x series MCU of megawin can be selected for development after successful installation.

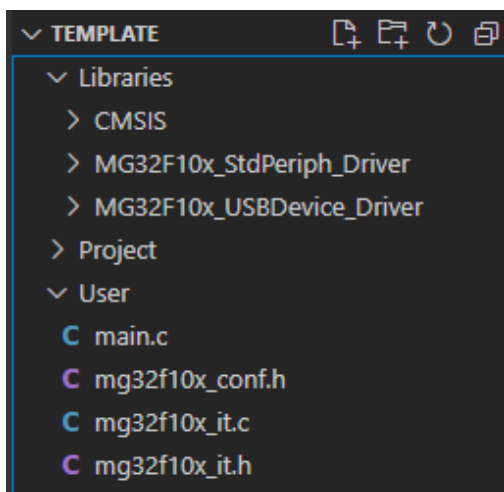


3.3. Build a project

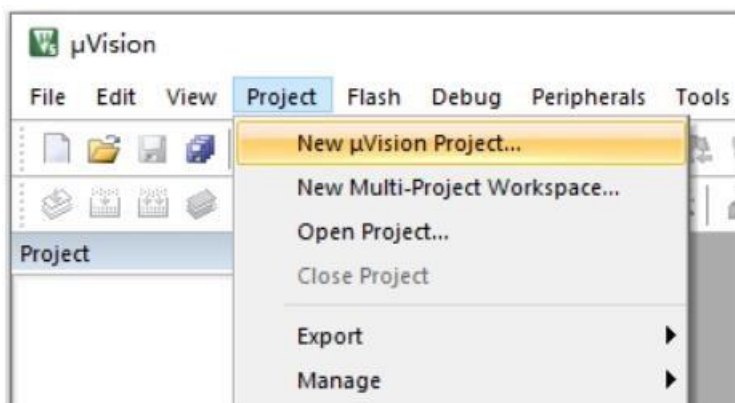
- 1) New a folder named "Template" to contain project.
- 2) New Libraries, Project and User subfolder in Template folder. (User can also customize their own folder structure)



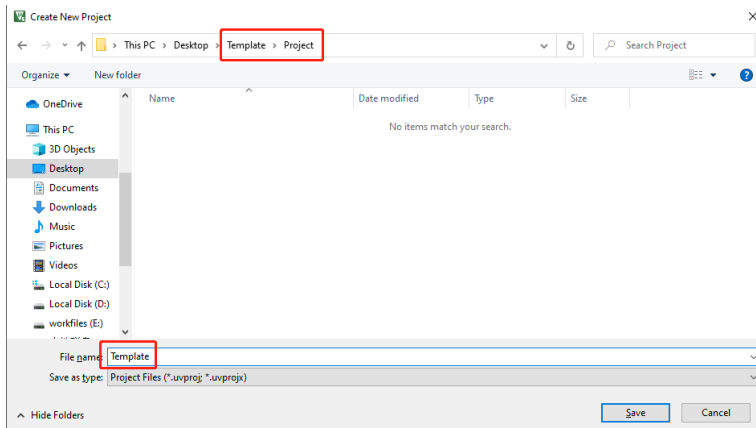
- 3) Copy MG32F10x standard peripheral library's Libraries folder's content to Template\Libraries.
- 4) Copy MG32F10x standard peripheral library's Project\MG32F10x_StdPeriph_Template folder's content to Template\User.



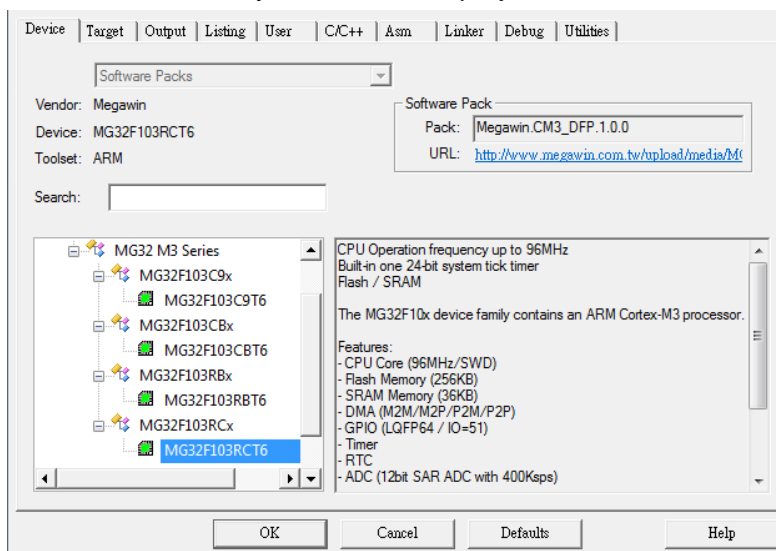
- 5) Open Keil MDK, New uVision project.



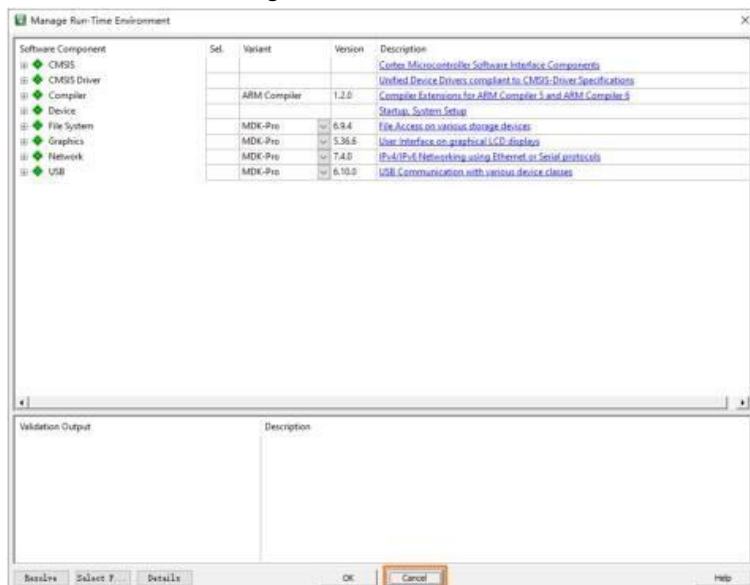
6) New a project named Template at Template\Project path.



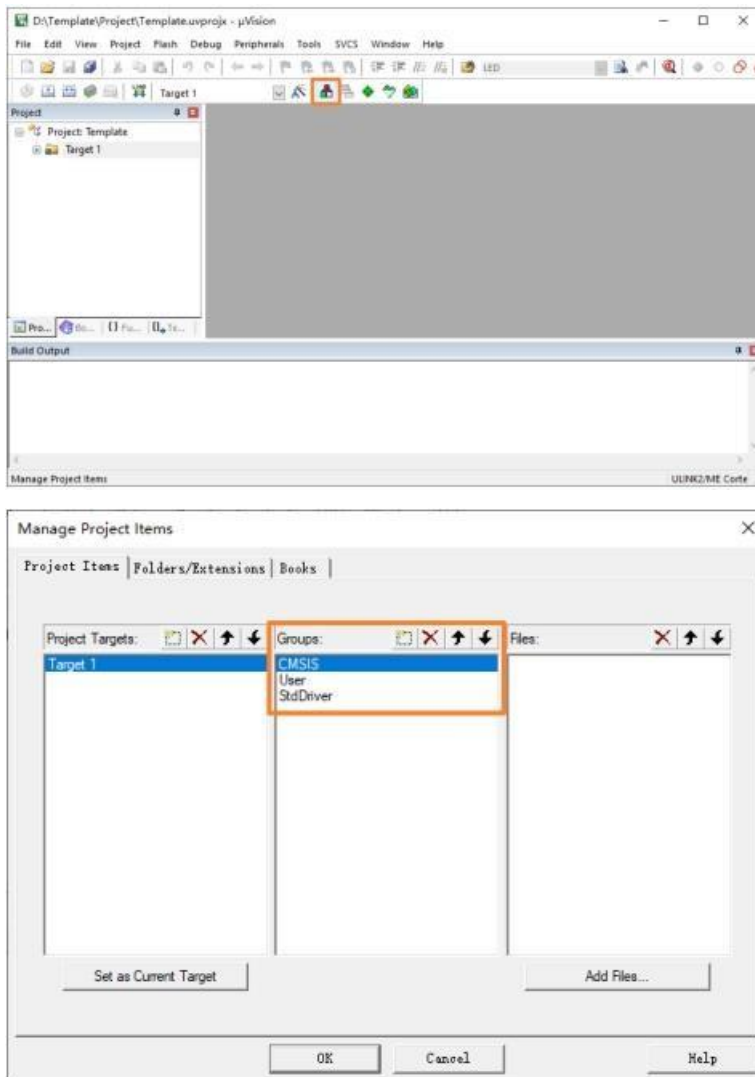
7) Select device which you need in this project, then click OK.



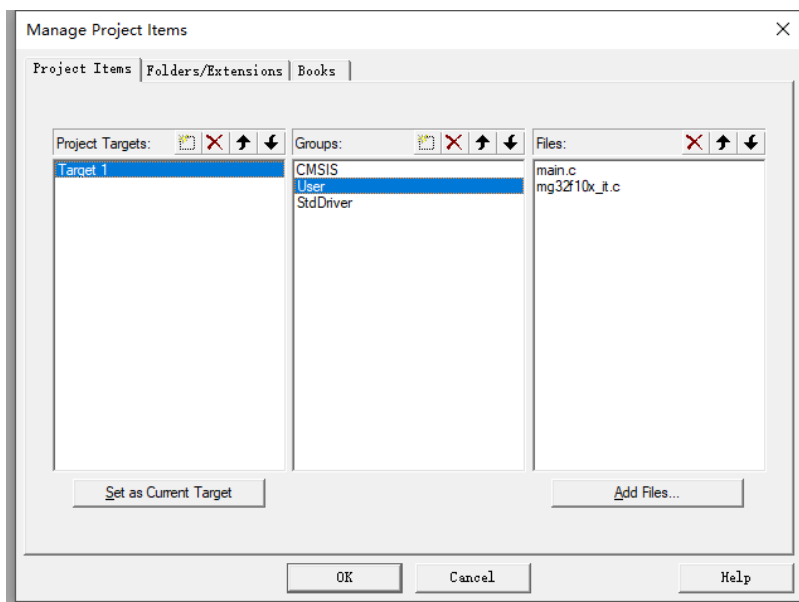
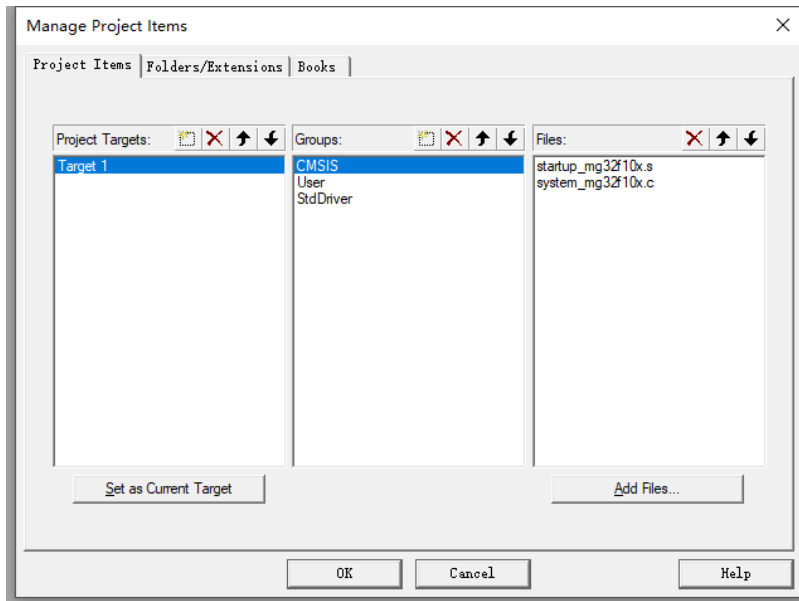
8) Then it will show Manage Run-Time Environment window. Click Cancel.

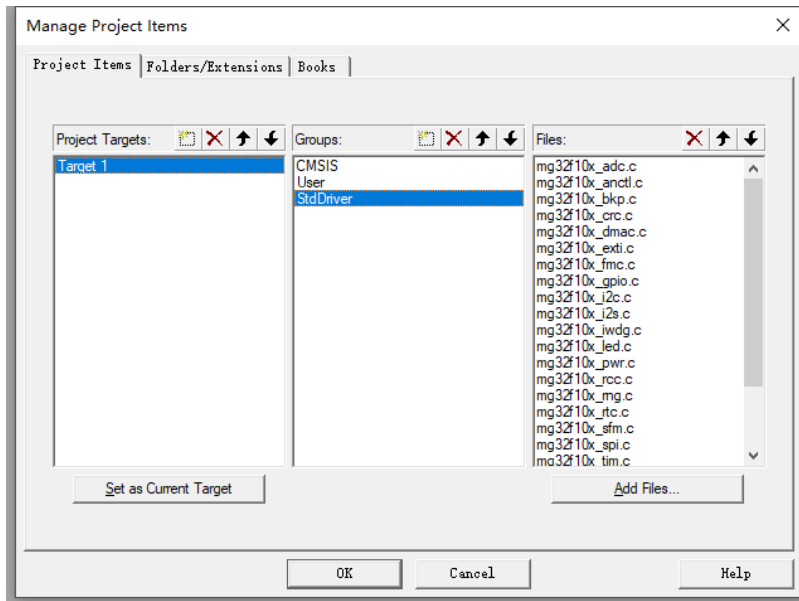


- 9) New 3 groups: CMSIS, User and StdDriver.

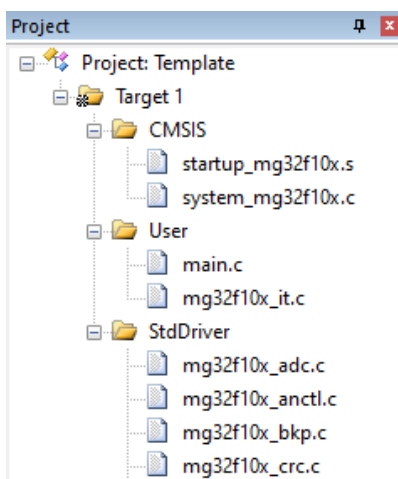


- 10) Add peripheral library file into Group
Add in CMSIS Group:
Template\Libraries\CMSIS\Device\MG\MG32F10x\startup\arm\startup_mg32f10x.s
Template\Libraries\CMSIS\Device\MG\MG32F10x\system_mg32f10x.c
Add in User Group:
Template\User\main.c
Template\User\mg32f10x_it.c
Add in StdDriver Group Group:
Template\Libraries\MG32F10x_StdPeriph_Driver\src folder's every .c file.

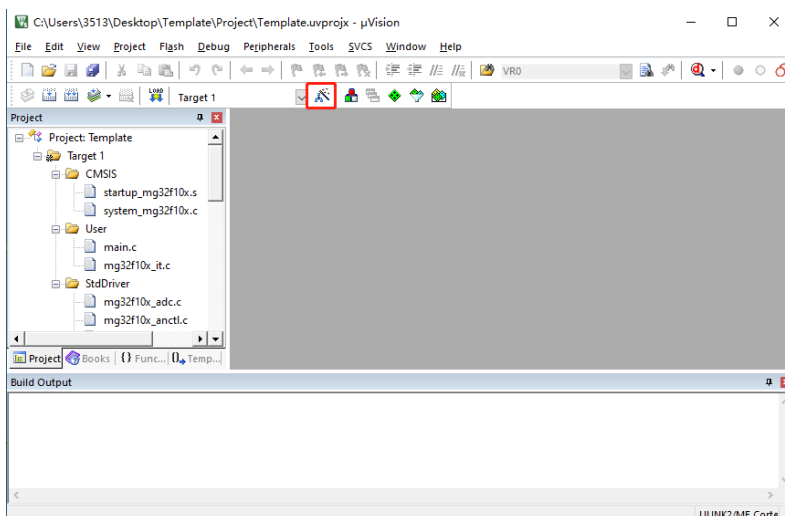




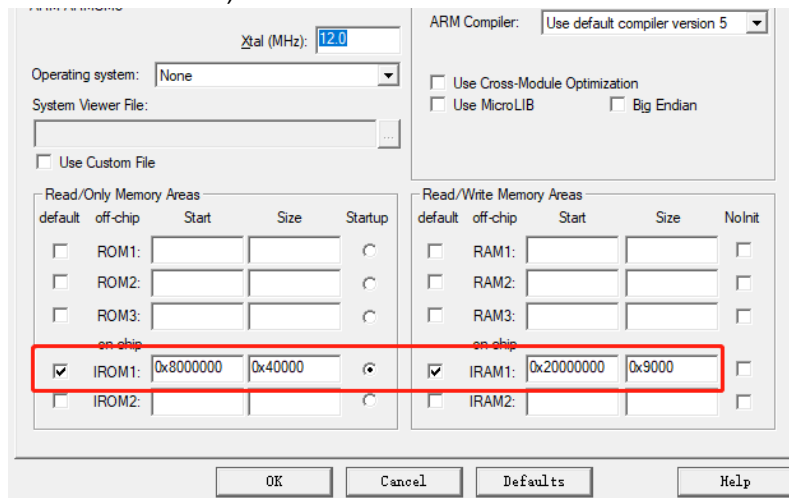
11) Final project structure as below:



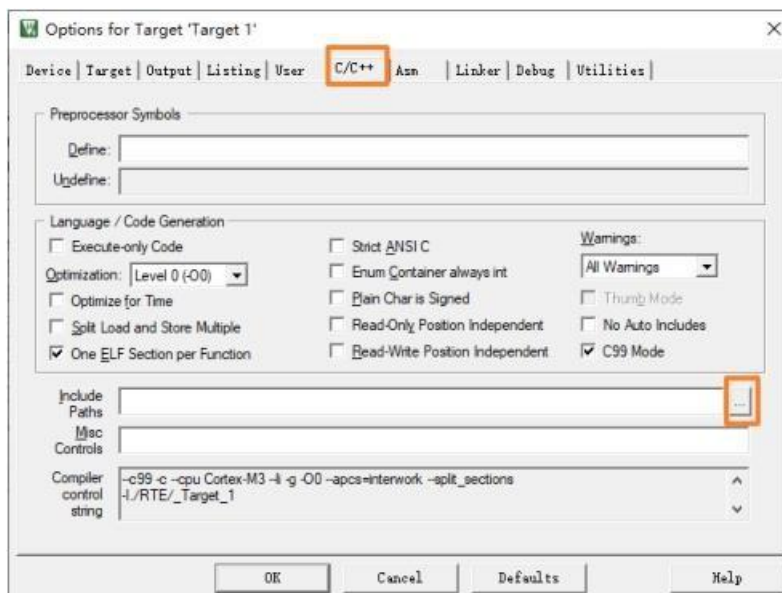
12) Open Options for Target window.



- 13) Configure Read/Only Memory Areas and Read/Write Memory Areas (Configure Flash and SRAM start address and size).

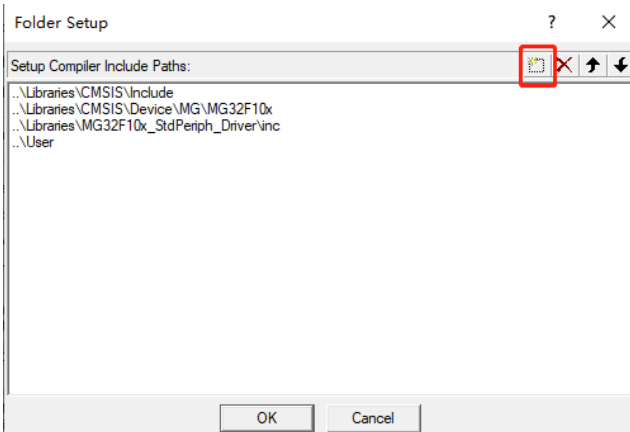


- 14) Configure project head file path in C/C++ tab.

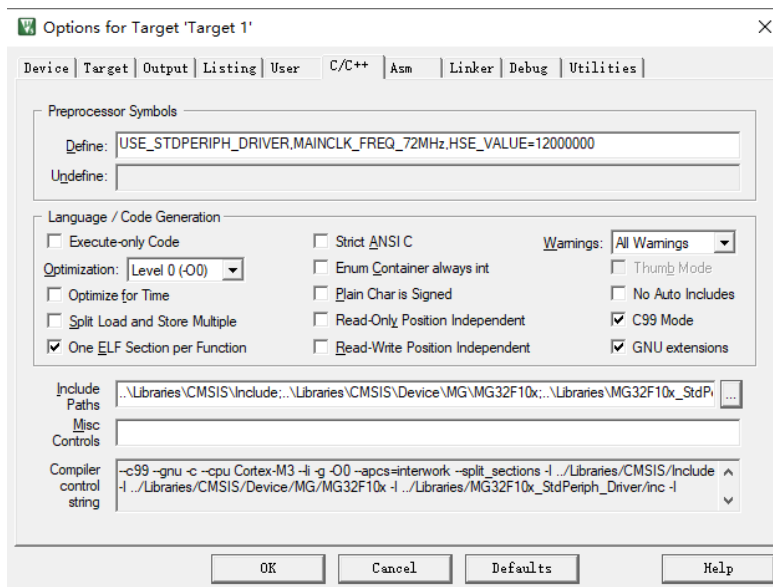


Add 4 path as below:

..\Libraries\CMSIS\Include
..\Libraries\CMSIS\Device\MG\MG32F10x
..\Libraries\MG32F10x_StdPeriph_Driver\inc
..\User



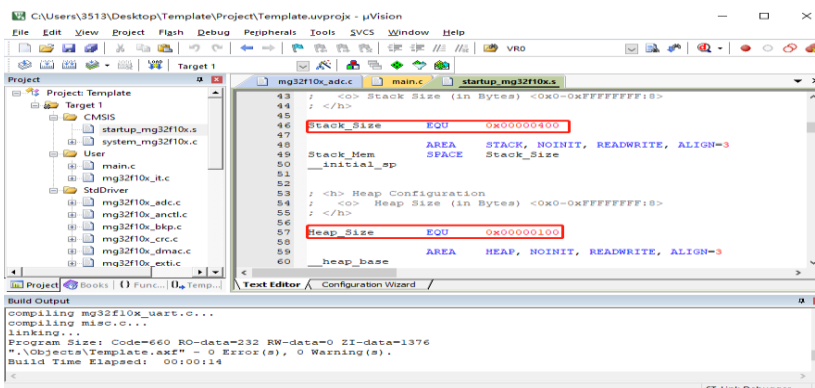
15) Add define in Preprocessor Symbols: **USE_STDPERIPH_DRIVER, MAINCLK_FREQ_72MHz, HSE_VALUE=12000000**



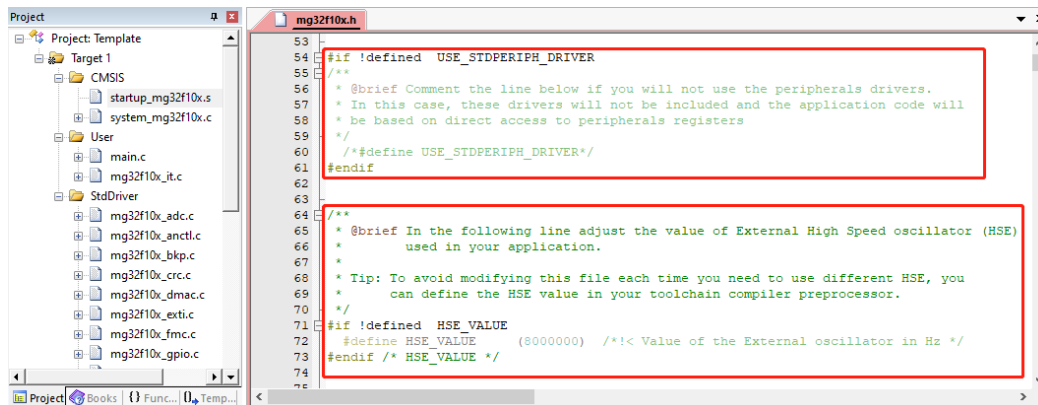
Click OK. So far, the project configuration is completed.

3.4. Peripheral library configuration

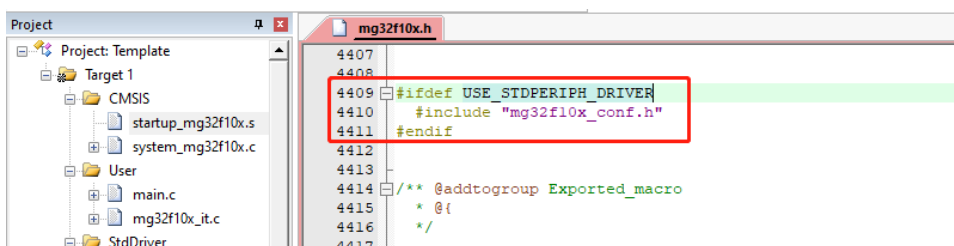
1) startup_mg32f10x.s could be used for configure application stack and heap size as below.



2) Two macro definition in mg32f10x.h should be noticed.



```
53
54 #if !defined USE_STDPERIPH_DRIVER
55 /**
56  * @brief Comment the line below if you will not use the peripherals drivers.
57  * In this case, these drivers will not be included and the application code will
58  * be based on direct access to peripherals registers
59  */
60 /**#define USE_STDPERIPH_DRIVER*/
61 #endif
62
63
64 /**
65  * @brief In the following line adjust the value of External High Speed oscillator (HSE)
66  * used in your application.
67  *
68  * Tip: To avoid modifying this file each time you need to use different HSE, you
69  * can define the HSE value in your toolchain compiler preprocessor.
70  */
71 #if !defined HSE_VALUE
72 #define HSE_VALUE (8000000) /*!< Value of the External oscillator in Hz */
73 #endif /* HSE_VALUE */
74
75
```

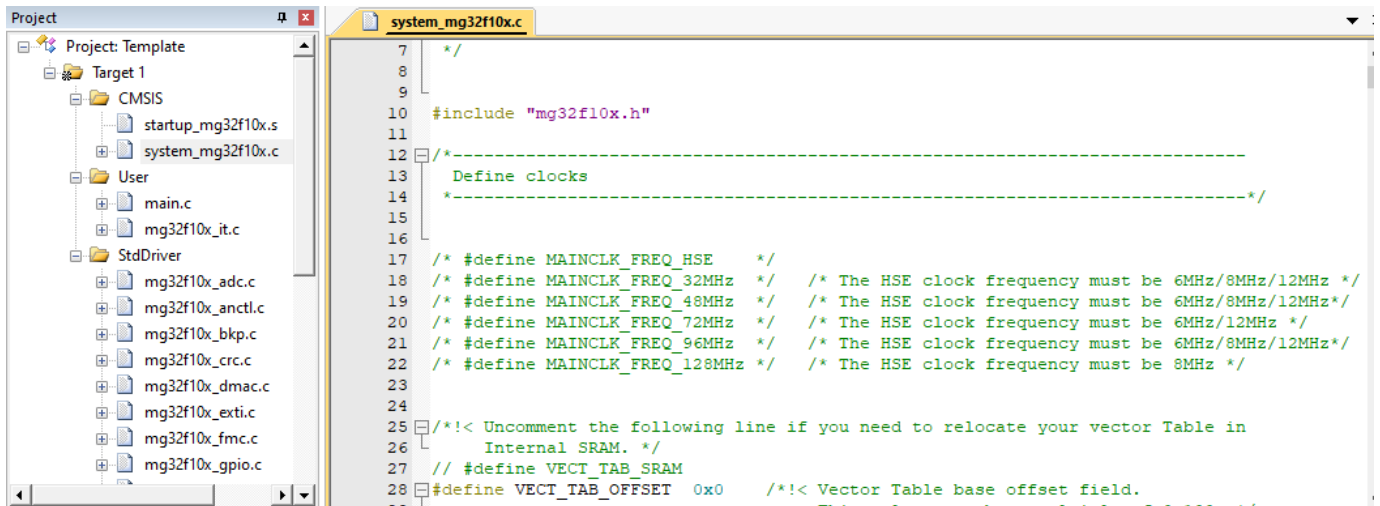


```
4407
4408
4409 #ifndef USE_STDPERIPH_DRIVER
4410 #include "mg32f10x_conf.h"
4411 #endif
4412
4413
4414 /** @addtogroup Exported_macro
4415  * @{
4416  */
4417
```

USE_STDPERIPH_DRIVER This macro definition represents that this application will use standard driver, and this project will include mg32f10x_conf.h head file.

HSE_VALUE This macro definition is used for define MG32F10x Xtal's frequency. Peripheral library will set external HSE Xtal frequency is **8MHz** by default. User should modify this definition here or at the preprocessor symbol if the Xtal which is using is not **8MHz**.

3) Some macro definition in system_mg32f10x.c should be noticed.



MAINCLK_FREQ_* This macro definition is used for configure the main clock frequency of MCU. Only one of the definitions can be selected at the preprocessor symbol (If no definition is selected, the chip main clock would be MHSI). Should be noticed that all these frequency definitions have requirements for the Xtal of MCU. For example, the Xtal frequency must be 6/12MHz when user select **MAINCLK_FREQ_72MHz**(Notify: macro definition **HSE_VALUE** should also be modified).

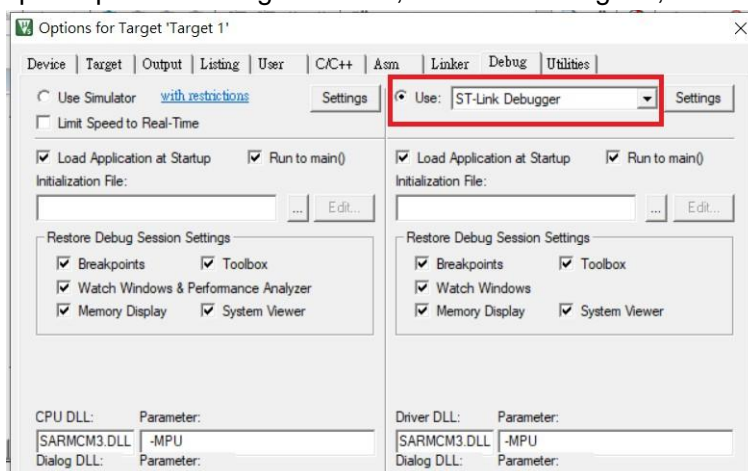
VECT_TAB_SRAM This macro definition represents that interrupt vector will be mapping to SRAM. (Only while the project needs to be run in SRAM)

VECT_TAB_OFFSET This macro definition is used for configure interrupt vector table start address offset. (related to Flash or SRAM start address)

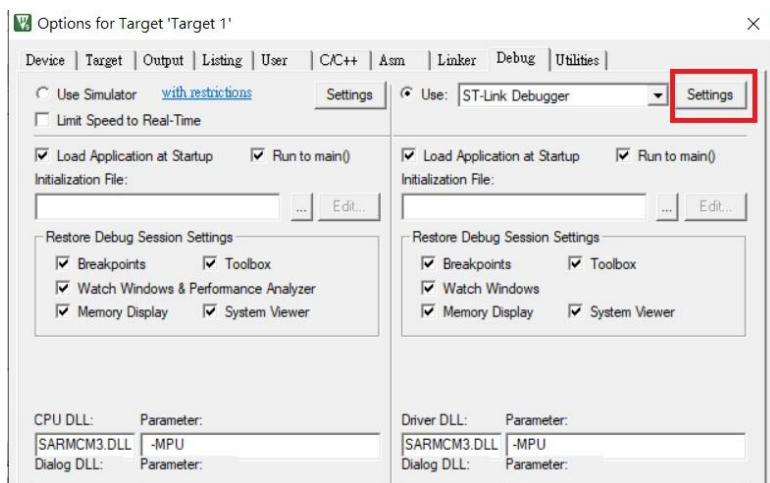
3.5. Debugger configuration

3.5.1. Use ST-Link to debug

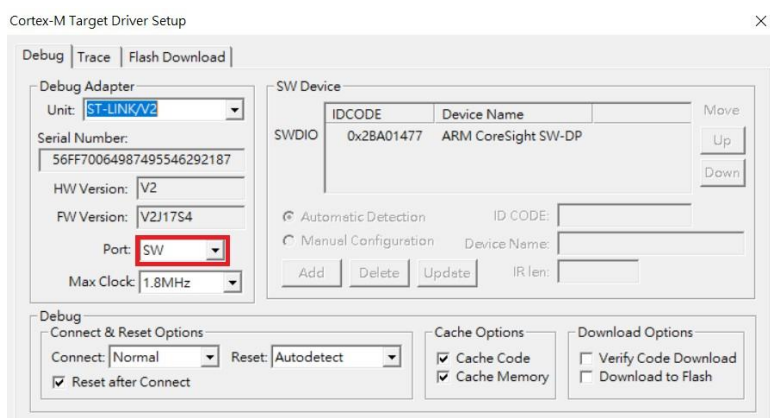
- 1) MG32F10x is embedded a CPU core of ARM Cortex-M3 processor. So MG32F10x supports kinds of debugger which support Cortex-M3 core MCU (such as: JLink, ULink, STLink and CMSIS-DAP). Take ST-link as example to demonstrate the MG32F10x debugging configuration.
- 2) Connect ST-Link to PC, connect ST-Link and MG32F10x through SWD interface. Power on the MCU.
- 3) Open Options for Target window, switch to Debug tab, select ST-Link debger.



- 4) Setting debugger configuration.

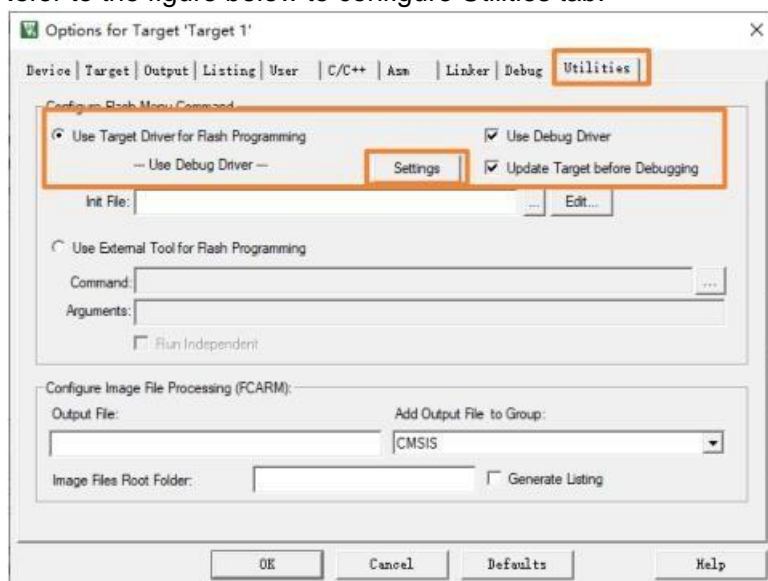


Select SW Port, then MG32F10x should be detected by ST-Link and be seen on SW Device.

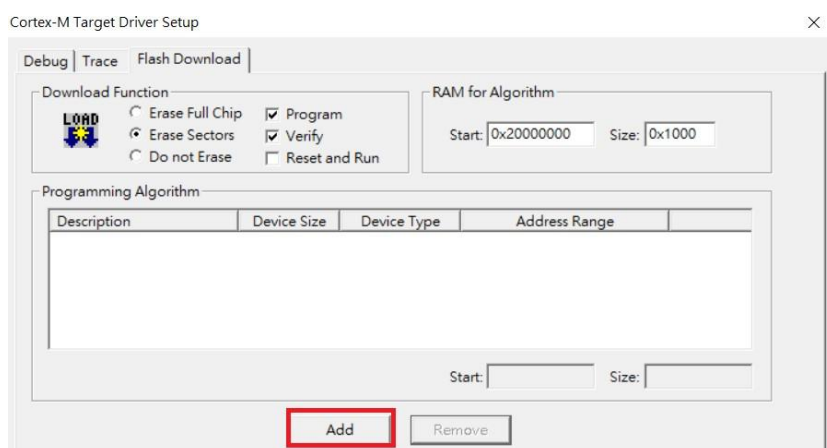
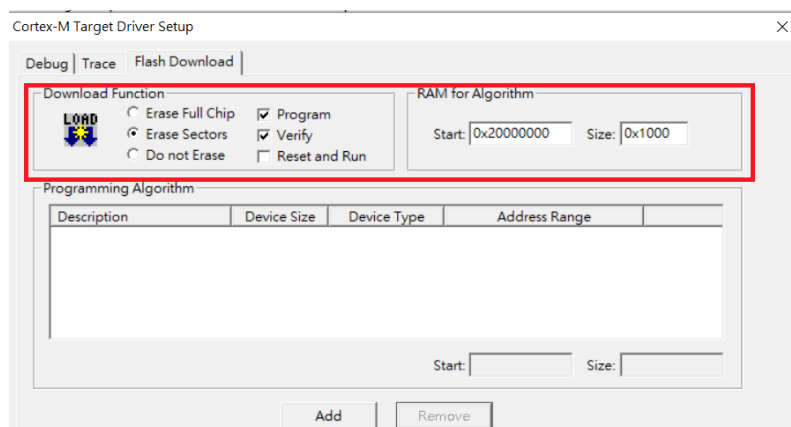


Then click OK.

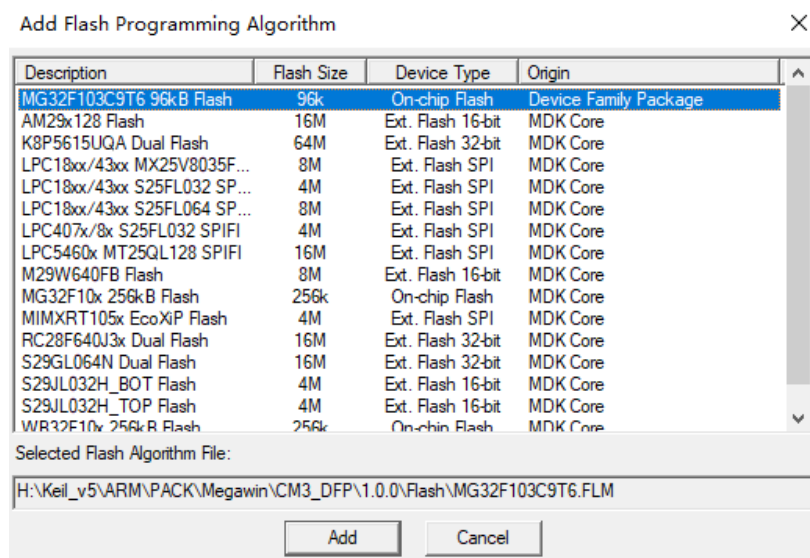
5) Refer to the figure below to configure Utilities tab.

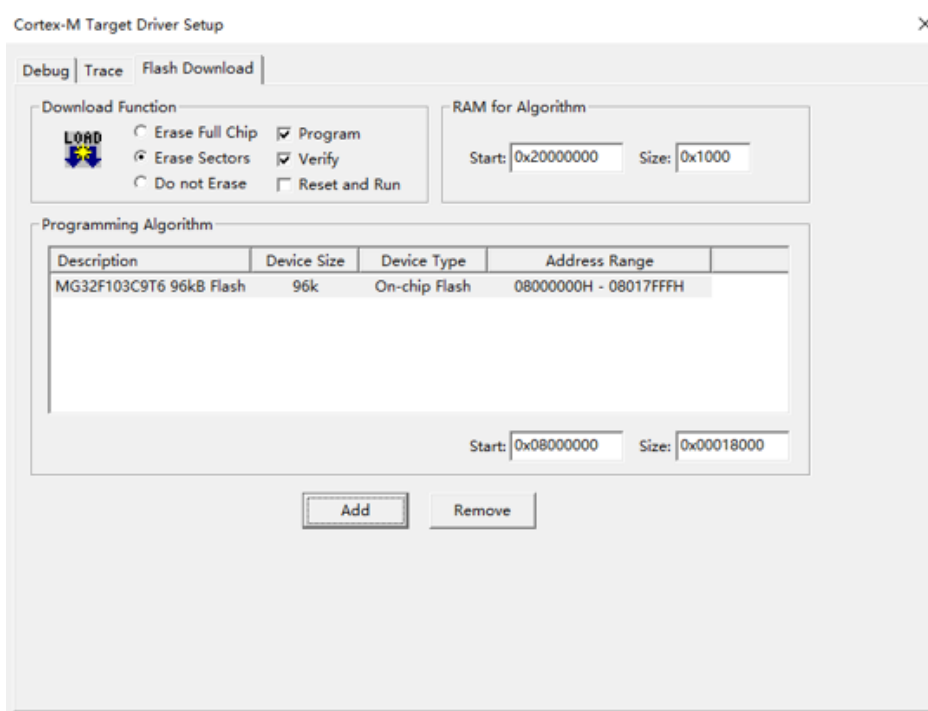


Then click Settings button, switch to Flash Download tab, and configure as below.



Find the device you need and click Add.





Click OK. Now user should be able to compile, download and debug their own code.

3.5.2. Use J-Link to debug

- 1) New a folder named "Megawin" under J-Link installed path H:\Program Files (x86)\SEGGER\JLink_V635g\Devices (J-link installed path could be modified, and the demo machine of this document is installed on disk H). Then new a folder named "MG32F10x" under "Megawin" folder.
- 2) Copy MG32F10x Flash algorithm to J-Link installed path H:\Program Files (x86)\SEGGER\JLink_V635g\Devices\Megawin\MG32F10x. Flash algorithm would be located under users' Keil installed path Keil_v5\ARM\PACK\Megawin\CM3_DFP\1.0.0\Flash

(H:) > Keil_v5 > ARM > PACK > Megawin > CM3_DFP > 1.0.0 > Flash

Name	Date modified	Type	Size
MG32F103C9T6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F103CBT6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F103RBT6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F104RCT6.FLM	2021/11/2 10:54	FLM File	15 KB

(H:) > Program Files (x86) > SEGGER > JLink_V635g > Devices > Megawin > MG32F10x

Name	Date modified	Type	Size
MG32F103C9T6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F103CBT6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F103RBT6.FLM	2021/11/2 10:54	FLM File	15 KB
MG32F104RCT6.FLM	2021/11/2 10:54	FLM File	15 KB

- 3) Open J-link devices file on H:\Program Files (x86)\SEGGER\JLink_V635g\JLinkDevices.xml and add MG32F10x information as below, then save.

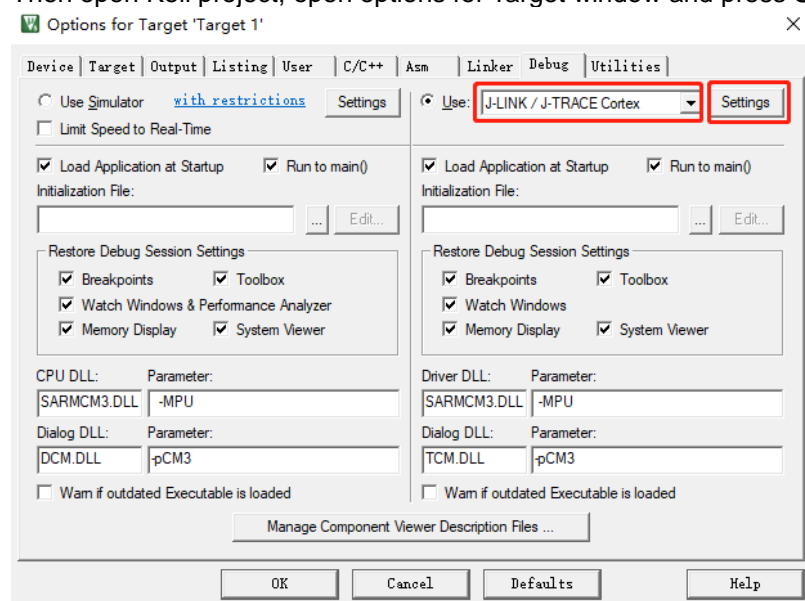
```
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103C9T6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x18000" Loader="Devices\Megawin\MG32F10x\MG32F103C9T6.FLM"
LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103CBT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x20000" Loader="Devices\Megawin\MG32F10x\MG32F103CBT6.FLM"
LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
```

```

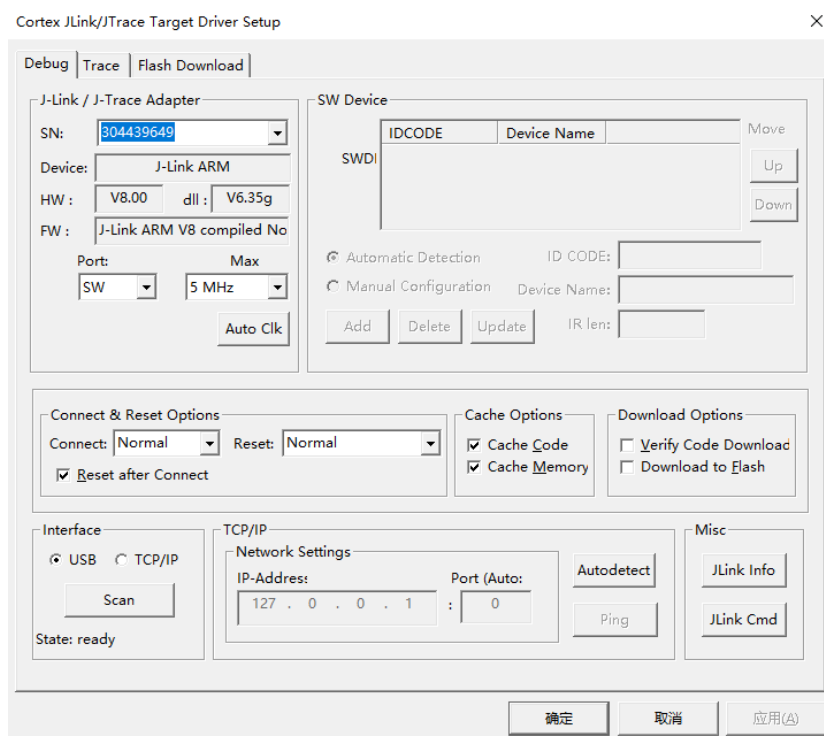
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103RBT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x20000" Loader="Devices\Megawin\MG32F10x\MG32F103RBT6.FLM"
LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F104RCT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x40000" Loader="Devices\Megawin\MG32F10x\MG32F104RCT6.FLM"
LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103C9T6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x10000" Loader="Devices\Megawin\MG32F10x\MG32F103C9T6.FLM" LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103CBT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x20000" Loader="Devices\Megawin\MG32F10x\MG32F103CBT6.FLM" LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F103RBT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x20000" Loader="Devices\Megawin\MG32F10x\MG32F103RBT6.FLM" LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>
<Device>
  <ChipInfo Vendor="Megawin" Name="MG32F104RCT6" Core="JLINK_CORE_CORTEX_M3" WorkRAMAddr="0x20000000" WorkRAMSize="0x1000" />
  <FlashBankInfo Name="Internal Flash" BaseAddr="0x08000000" MaxSize="0x40000" Loader="Devices\Megawin\MG32F10x\MG32F104RCT6.FLM" LoaderType="FLASH_ALGO_TYPE_CMSIS" AlwaysPresent="1" />
</Device>

```

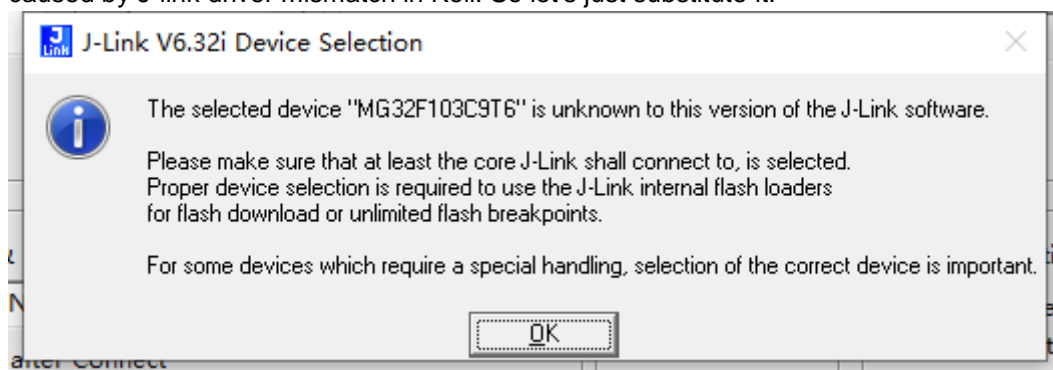
Then open Keil project, open options for Target window and press Settings button after select J-Link on Debug tab.



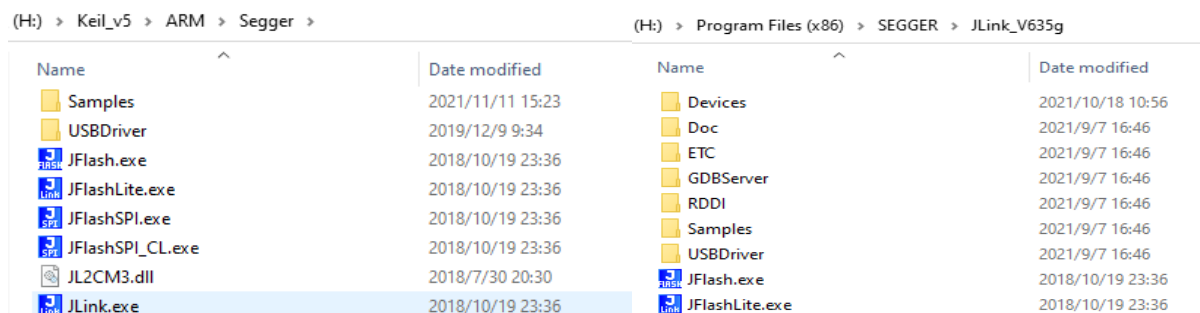
If J-link is connected to PC and J-Link driver is installed successfully, you shall see the window as below.



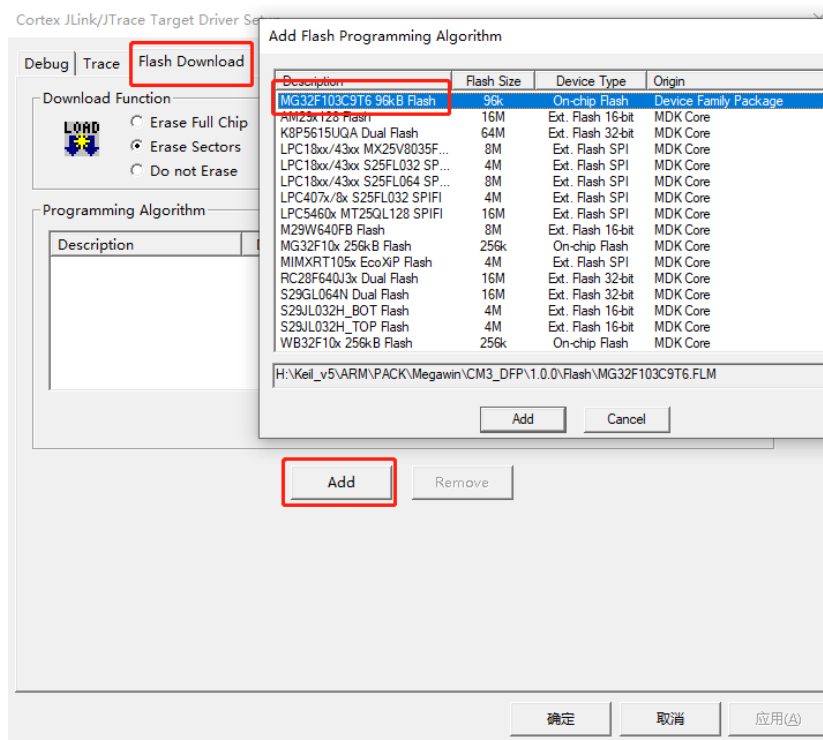
If you click Settings and keil prompts something like The selected device xxx is unknown, this problem should be caused by J-link driver mismatch in Keil. So let's just substitute it.



Picture below on the left shows the J-Link driver path which is installed with Keil, and the picture on the right is the J-Link driver path installed and configured according to chapter 1 by ourself. **Now We cover the J-Link driver which is installed with Keil with everything in our installed J-link directory.** After cover completed, reopen Settings in Debug tab, you shall see the window normally.



Select Flash Download tab, select the chip you need, you shall be able to debug.



4. Layout suggestion

4.1. Printed circuit board

For technical reasons, it is preferable to use a multilayer PCB with a dedicated independent grounding layer (VSS) and dedicated independent power supply layer (VDD) to provide better coupling and shielding effect. In many applications, limited by economic conditions that cannot use such printed circuit boards, so it is necessary to ensure a good grounding and power supply structure.

4.2. Component location

In order to reduce cross-coupling on the PCB, different circuits need to be separated according to their impact on EMI when designing the layout. For example, high current circuits, low voltage circuits and digital devices.

4.3. Grounding and power supply(VSS/VDD)

Each module (noisy circuits, low sensitivity circuits, digital circuits) should be grounded individually, and all the ground should eventually be connected at one point. Avoid or minimize loop areas. To reduce the area of the power supply loop, the power supply should be as close to the ground as possible. This is because the power supply loop acts as an antenna, acting as a transmitter and receiver of EMI. Areas of the PCB without components need to be filled to provide good shielding (especially for single-layer PCB).

4.4. Decoupling

All pins need to be properly connected to the power supply. These connections, including pads, wires and Vias, shall have as little impedance as possible. A common approach is to increase the width of the line, including the use of a separate supply layer in a multi-layer PCB. At the same time, each power pin on the MG32F10x should be paralleled with a decoupled filter ceramic capacitor C(100nF) and chemical capacitor C(10μF). These capacitors should be as close to the power/ground pins as possible; Or on another layer of the PCB, under the power/ground pin. Typical values range from 10nF to 100nF, depending on the needs of practical applications. Figure 1 shows a typical layout of such power/ground pins.

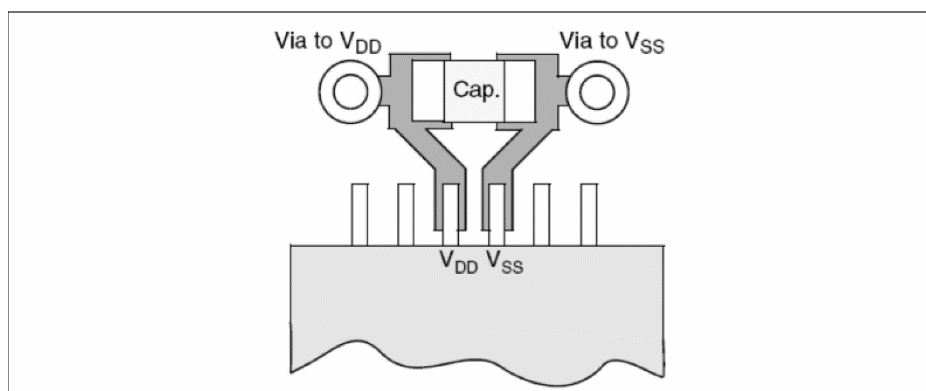


Figure 1 typical layout of VDD /VSS pins

4.5. Power supply scheme

The circuit is powered by a stable power supply VDD.

- Notify:

- If ADC is used, the VDD range must be between 2.4V and 3.6V
- If ADC is not used, the VDD ranges from 2V to 3.6V

- The VDD pins must be connected to a VDD power supply with external stabilized capacitors (11 100nF ceramic capacitors and one tantalum capacitor (4.7μF minimum, 10μF typical)).

- The VBAT pin must be connected to an external battery ($1.8V < V_{BAT} < 3.6V$). If no external battery is available, this pin must be connected to the VDD power supply along with a 100nF ceramic capacitor

- VDDA pins must be connected to two external stabilized capacitors (10nF ceramic capacitor + 1μF tantalum capacitor).

- VREF+ pins can be connected to VDDA external power supplies. If a separate external reference voltage is used on VREF+, a 10nF and a 1μF capacitor must be connected to the pins. In all cases, VREF+ must be between 2.4V and VDDA.

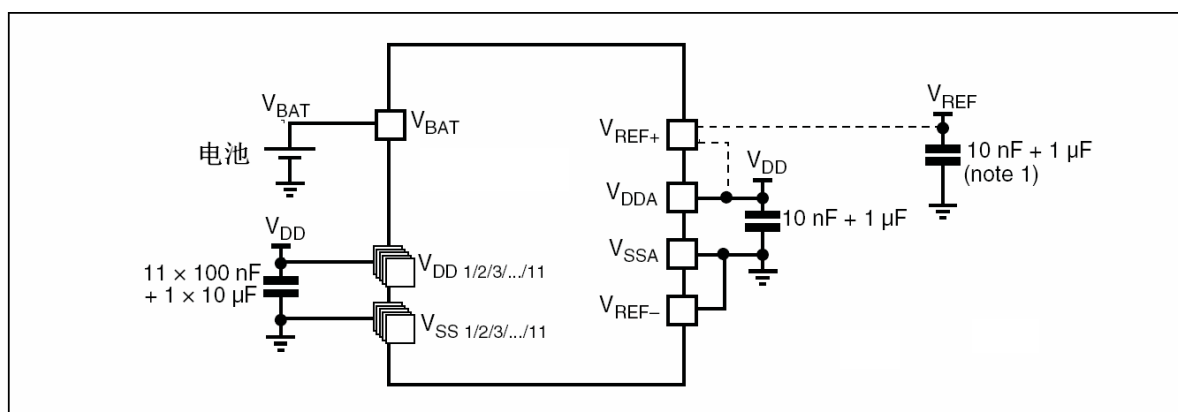


Figure 2 power supply scheme

Note 1. Optional. If a separate external reference voltage is used on VREF+, two capacitors (10nF and 1uF) must be connected.

Note 2. VREF+ connect to VDDA or VREF .

4.6. Other signals

In practical applications, EMC performance can be improved by paying attention to the following points:

- Signals that are affected by temporary disturbances (such as interrupt or shaking signals, rather than LED commands)

For these signals, laying the ground around the signal line, shortening the line distance, eliminating adjacent noise and sensitive wiring can improve EMC performance.

For digital signals, the best possible signal-property margin must be achieved to effectively distinguish between the two logical states (Raise logic '1' as high as possible and lower logic '0' as low as possible). A slow Schmidt trigger is recommended to eliminate the parasitic state.

- When wiring, the 3W principle should be met as far as possible, and the wiring should be kept away from adjacent lines as far as possible to reduce coupling and interference. If ADC and CMP require high precision, the wire of ADC and CMP should be around with ground.

- Noise signal (clock, etc.)
- Sensitive signal (high resistance, etc.)

4.7. Unused IO and its properties

All microcontrollers are designed for a variety of applications, and common applications do not use all microcontroller resources.

To improve EMC performance, unused clocks, timers, or I/O pins need to be handled accordingly. For example,

I/O ports should be set to '0' or '1' (pull-up or pull-down for unused I/O pins). Modules that are not used should be disabled.

4.8. Clock

To minimize parallel wiring between LSE and HSE. In figure 3, LSE and HSE are directly separated when they are led out from the pad.

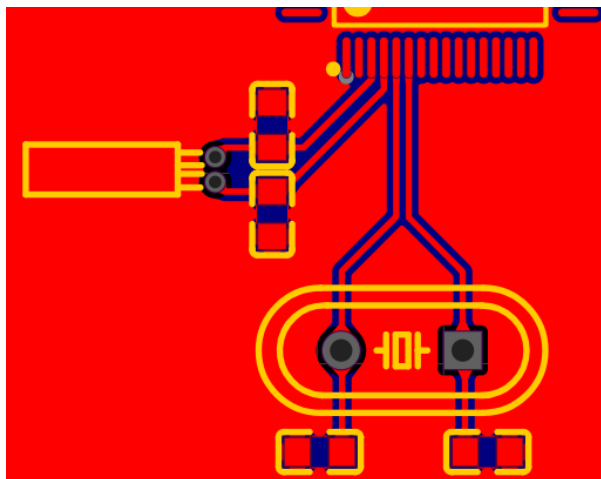


Figure 3 LSE and HSE layout

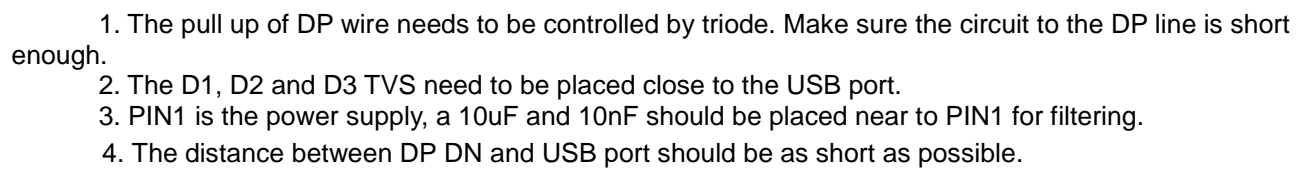
4.9. Analog signal

The analog signal is separated from the digital signal, and the analog signal needs to be shielded by the ground line, so that the sampling accuracy can be guaranteed as much as possible.

4.10. EMI

1. Make sure the power rating is suitable for the application and is optimized using a decoupling capacitor.
2. Provide sufficient filter capacitors on the power supply. High capacity/bypass and decoupling capacitors shall have low equivalent series inductance (ESL).
3. Create a ground plane if space is available on the wiring layer. These ground areas are connected to the ground plane through Vias.
4. Keep the current loop as small as possible. Add as many decoupling capacitors as possible.
5. The differential line pair must match the line length, otherwise it will result in timing offset, reduced signal quality, and increased EMI.
6. Differential routing is required to be on the same plate layer, because the impedance and hole difference between different layers will reduce the effect of differential mode transmission and introduce common mode noise.
7. Do not have Vias for high-speed signal routing. Ensure that the ground plane at the rear is complete, and shorten the wires routing distance away from adjacent wires. If the USB interface chip needs a series end resistor or D line is connected to a pull resistor. Be sure to place these resistors as close to the chip as possible.
8. The power pin of each VDD in MCU should leave 2 capacitor: 1uF and 0.1uF as far as possible
9. SPI or IIC communication line, each signal line string a resistance of about 10R, reserved a 120PF capacitor. The signal line should be as short as possible.
10. Crystal wiring should be short enough, do not route the signal line on the back of the crystal, to ensure that the crystal ground plane is complete. If the layout allows, more ground Vias should be drilled around the crystal oscillator.

Schematic diagram design reference.



5. Peripheral porting

5.1. Preparation before porting

Users can use MG32F10x to replace STM32F10x chips of the same type for learning or product program transplantation, which greatly reduces user cost.

On the other hand, MG32F10x series are independently designed chips, and the underlying design and library function package are bound to have many differences with STM32F10x. When writing programs, the programs realized on STM32F10x can not be simply and brutally transplanted directly to MG32F10x, and simple modifications should be made according to the MG driver library. Requires users to pay a low cost of learning and time. But don't worry, only need to modify the part involves the MCU register access related part of the code, the software algorithm, the upper structure, variables such as the main content of the code is not need to change, that is to say, you need to modify, contains only use to replace the peripheral initialization, interrupt function structure and the code involves driving function of peripheral data reading and writing, It's not going to be a big change.

Users who familiar with STM32F10x can be according to the MG32F10x reference manual, firmware library guide and firmware library to the routine quickly start.









When migrating software, refer to [Build a project](#) section and associate header files.

[Notify]: The sample code provided in the development kit almost all use XTAL as the clock source by default, so if you want to develop with XTAL, please refer to the [RCC](#) section of the Software porting chapter to change this.

5.2. ADC

As for the ADC initialization part, we can refer to the relevant ADC sample code to replace the underlying driver of the related ADC initialization and ADC value acquisition of STM32F10x as a whole. The driver used is located at mg32f10x_pwr.c, mg32f10x_adc.c, mg32f10x_rcc.c, mg32f10x_anctl.c and mg32f10x_gpio.c. Please note to add driver files.

```

 ADC_AnalogWatchdog
 ADC_ChipTemperature
 ADC_DMA
 ADC_DMA_Injected
 ADC_ExtLinesTrigger
 ADC_GetVDD
 ADC_Interrupt
 ADC_TIMTrigger_AutoInjection

/* ADC configuration -----*/
PWR_UnlockANA();
ANCTL_SARADCCmd(ENABLE);
PWR_LockANA();
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 1;
ADC_Init(&ADC_InitStructure);
/* ADC regular channel3 configuration */
ADC_RegularChannelConfig(ADC_Channel_3, 1, ADC_SampleTime_7Cycles5);
/* Enable EOC interrupt */
ADC_ITConfig(ADC_IT_EOC, ENABLE);
/* Enable ADC external trigger conversion */
ADC_ExternalTrigConvCmd(ENABLE);
/* Enable ADC */
ADC_Cmd(ENABLE);
/* Start ADC calibration */
ADC_StartCalibration();
/* Check the end of ADC calibration */
while(ADC_GetCalibrationStatus());
/* Enable ADC reset calibration register */
ADC_ResetCalibration();
/* Check the end of ADC reset calibration register */
while(ADC_GetResetCalibrationStatus());

/* Start ADC Software Conversion */
ADC_SoftwareStartConvCmd(ENABLE);

/* Waiting for EOC */
while(ADC_GetFlagStatus(ADC_FLAG_EOC) != SET){}
X= ADC_GetADValue(ADC->DR); // Read ADC value

```

[Notify]: Do not read ADC values by directly reading the ADC data register. This will result in incorrect values. Please use driver ADC_GetADValue(uint16_t data); to read ADC value.

Interrupt function structure template:

```
void ADC_IRQHandler(void)
{
    if(ADC_GetITStatus(ADC_IT_EOC) != RESET)
    {
        ADC_ClearITPendingBit(ADC_IT_EOC);
        printf("\rADC Channel 3: %-5d", ADC_GetADValue(ADC->DR));
    }
}
```

In addition, for the wiring of analog signals, it is recommended to separate analog signals from digital signals, and analog signals need to be shielded by ground lines, as mentioned in the previous section of hardware layout suggestions, so as to ensure the sampling accuracy as much as possible.

5.3. ANCTL(Analog controller)

5.3.1. CMP

Analog comparator is a very useful peripheral that is not included in the STM32F10x series. The sample code is located in ANCTL\CMP. User can call the function in it. The driver used is located at mg32f10x_pwr.c, mg32f10x_rcc.c, mg32f10x_anctl.c and mg32f10x_gpio.c. Please note to add driver files.

```
PWR_UnlockANA();
ANCTL_CMPAConfig(CMPA_PSEL_PB4, CMPA_NSEL_PB6);
ANCTL_CMPACmd(ENABLE);
PWR_LockANA();
```

```
CMP_Result = ANCTL_CMPAGetOutputLevel();//Read output level
```

The code above sets the positive and negative input of the comparator and completes the initialization of CMP.

[Notify]: There is no interrupt function in CMP.

5.3.2. DCSS

DCSS is a clock safety system that starts to work when HSE is stable and stops working when HSE stops. It can be used as a detector of XTAL failed event. So if there is a requirement of detect XTAL failed event from user, refer to the code below and replace the underlying driver of the related initialization of STM32F10x as a whole. The driver used is located at mg32f10x_pwr.c, mg32f10x_rcc.c, mg32f10x_anctl.c and mg32f10x_gpio.c. Please note to add driver files.

```
RCC_DCSSCLKCmd(ENABLE);
ANCTL_ClockSecuritySystemCmd(ENABLE);
```

Interrupt function structure template:

```
void NMI_Handler(void)
{
    if (ANCTL_GetITStatus(ANCTL_IT_DCSS) != RESET)
    {
        ANCTL_ClearITPendingBit(ANCTL_IT_DCSS);

        while (1)
        {
            /* LED2 blink */
            GPIO_ToggleBits(GPIOB, GPIO_Pin_13);
            Delay(100000);
        }
    }
}
```

Through the code above to enable DCSS.

5.4. BKP

BKP is the data backup register which can store some backup data. Since the backup domain (BKP) is still powered by VBAT when VDD is turned off, its contents are not lost if there is a battery is connected to the VBAT pins. It's simple to use by directly store value after enable the register. The driver used is located at mg32f10x_pwr.c, mg32f10x_rcc.c and mg32f10x_bkp.c. Please note to add driver files.

```
PWR_BackupAccessCmd(ENABLE);
BKP->DR1 = 0x55AA;    //Store value 0x55AA into BKP
```

5.5. CRC







It's simple to use CRC by directly use the function in driver after enable CRC. User can use the Deinit() below to replace the underlying driver of the related initialization of STM32F10x as a whole. Use the CRC check function when you need to calculate. The driver used is located at mg32f10x_crc.c, mg32f10x_rcc.c, mg32f10x_sfm.c, computeBytes.c, computeHalfWords.c and computeWords.c. Please note to add driver files.

```
CRC_SFM_DeInit();
result1 = CRC8_ComputeBytes(bytes, _countof(bytes));
```

5.6. DMAC

Direct memory access (DMA) is used to provide high-speed data transfer between peripherals, between peripherals and memory, or between memory and memory without CPU process. That allows CPU to finish other mission while DMA is working. User can refer actual situation to replace the underlying driver of the related initialization of STM32F10x as a whole. The driver used is located at mg32f10x_rcc.c and mg32f10x_dmac.c. Please note to add driver files.

The actual porting code can refer to the DMAC sample code.

-  DMAC_MemoryToMemory
-  DMAC_MemoryToUart
-  DMAC_MemoryToUart_MultiBlock
-  DMAC_UartToMemory
-  DMAC_UartToMemory_MultiBlock
-  DMAC_UartToUart_MultiBlock

```
DMAC_ChannelCmd(DMAC1, DMAC_Channel_0, ENABLE);
```

Call the function above after finish the initialization of DMA. Modify the channel and module, then the DMA 初始 transmission should be started.

Interrupt function structure template:

```
void DMAC1_IRQHandler(void)
{
    if(DMAC_GetITStatus(DMAC1, DMAC_Channel_0, DMAC_IT_BLOCK) != RESET)
    {
        DMAC_ClearITPendingBit(DMAC1, DMAC_Channel_0, DMAC_IT_BLOCK);
        printf("DMA block transfer complete.\r\n");
    }

    if(DMAC_GetITStatus(DMAC1, DMAC_Channel_0, DMAC_IT_TFR) != RESET)
    {
        DMAC_ClearITPendingBit(DMAC1, DMAC_Channel_0, DMAC_IT_TFR);
        printf("DMA transfer complete.\r\n");
    }
}
```

```
    if(memcmp(memDst, memSrc, sizeof(memSrc)) == 0) {
        printf("DMA transfer success!!!\r\n");
    }
    else {
        printf("DMA transfer failed!!!\r\n");
    }
}

if(DMAC_GetITStatus(DMAC1, DMAC_Channel_0, DMAC_IT_ERR) != RESET)
{
    DMAC_ClearITPendingBit(DMAC1, DMAC_Channel_0, DMAC_IT_ERR);
    printf("DMA transfer error!!!\r\n");
}
}
```

[Notify]: Strongly suggest not to use DMAC to do the ADC, and ADC is only supported in single DMA mode.

5.7. EXTI

External interrupt EXTI support different interrupt or event, and support rising, falling edge or both edge trigger. Code below shows the configuration of EXTI. User can refer actual situation to replace the underlying driver of the related initialization of STM32F10x as a whole. The driver used is located at mg32f10x_rcc.c, mg32f10x_exti.c and mg32f10x_gpio.c. Please note to add driver files.

```
/* Connect EXTI0 Line to PA0 pin */
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);

/* Configure EXTI0 line */
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);

/* Configure and enable EXTI0 interrupt */
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Replace the external interrupt initialization of the code and refer to actual situation to select the event trigger and pin.

Interrupt function structure template:

```
void EXTI0_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        /* Toggle LED1 */
        GPIO_ToggleBits(GPIOB, GPIO_Pin_14);

        /* Clear the EXTI line 0 pending bit */
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}
```

Another interrupt function structure template:

```
void EXTI9_5_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line6) != RESET)
    {
        /* Toggle LED2 */
        GPIO_ToggleBits(GPIOB, GPIO_Pin_13);

        /* Clear the EXTI line 6 pending bit */
        EXTI_ClearITPendingBit(EXTI_Line6);
    }
}
```

5.8. FMC(Flash memory controller)

FMC is Flash memory controller which is used to modify flash data. There are two sample code in SDK as below. The first one is calculate the flash CRC while the second is programming flash data. User can refer actual situation to replace the programming flash code. The driver used is located at mg32f10x_pwr.c, mg32f10x_rcc.c, mg32f10x_anctl.c and mg32f10x_fmc.c. Please note to add driver files.

FMC_CalculateCRC

FMC_Program

```
/* Before flash operation, FHSI must be enabled */
PWR_UnlockANA();
ANCTL_FHSICmd(ENABLE);
PWR_LockANA();

/* Erase the specified FLASH page */
FMC_ErasePage(TEST_PAGE_ADDR);
/* Clear page latch */
FMC_ClearPageLatch();
/* Write data to page latch */
for(iter = 0; iter < 64; iter++) {
    FMC->BUF[iter] = 0x12345678 + iter;
}
/* Program data in page latch to the specified FLASH page */
FMC_ProgramPage(TEST_PAGE_ADDR);
```

[Notify]: User should enable FHSI clock and don't disable it before performing flash operations.

As the code above, erase the corresponding area of flash, then clear page latch, then user could program the flash. After that, should execute FMC_ProgramPage(TEST_PAGE_ADDR); to apply.

5.9. GPIO

There are 4 sample code about GPIO as below. User can find them in GPIO directory. The driver used is located at mg32f10x_rcc.c and mg32f10x_gpio.c. Please note to add driver files.

GPIO_BitBand

GPIO_I2C_Master

GPIO_InputOutput

GPIO_IOToggle

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_BMX1 |RCC_APB1Periph_GPIOA
|RCC_APB1Periph_QSPI, ENABLE);
GPIO_Init(GPIOA, GPIO_Pin_4 |GPIO_Pin_5 |GPIO_Pin_6 |GPIO_Pin_7, GPIO_MODE_AF
|GPIO_OTYPE_PP |GPIO_PUPD_NOPULL |GPIO_SPEED_HIGH |GPIO_AF5);
```

The initialization of GPIO is simple. Just call GPIO_Init() after enable port clock to enable GPIO. The first parameter of the function is select port while the second parameter is setting pin, alternative function and mode.

5.10. I2C

The MG32F10x has two I2C, which can control all I2C bus-specific sequences, protocols, arbitrations and timing, and can support standard mode, fast mode and high-speed mode. I2C1 provides multi-master mode function and supports SMBUS(System Management Bus) protocol.

There are 2 sample code about I2C as below. User can find them in I2C directory. The driver used is located at mg32f10x_rcc.c, mg32f10x_i2c.c and mg32f10x_gpio.c. Please note to add driver files. If user need to use 24C02, file drv_eeprom_24c02.c also should be added.

- I2C_24C02
- I2C_24C02_Interrupt
- I2C_Master_HighSpeed
- I2C_MasterDMARx_SlaveDMATx
- I2C_MasterDMATx_SlaveDMARx
- I2C_Simulate_24C02
- I2C_SMBus_Master

It can be seen that the sample code type is more, but also contains the I2C memory 24C02 sample code, convenient for users to use. User can configure the STM32F10x based on the actual usage frequency, and then replace the I2C initialization code of the STM32F10x.

```
eeprom_24c02_init();
```

```
result = eeprom_24c02_random_read(0x06, &rdata1);
printf("Read from [0x06] is 0x%02X\r\n", rdata1);
printf("eeprom_24c02_random_read() - TX_ABORT_SOURCE = %08X\r\n", result); // I2C read data
if(result != 0) {
    errCode = 0x11;
    goto finish;
}
```

```
result = eeprom_24c02_byte_write(0x06, rdata1 + 1); //I2C write data
```

User can use driver functions to read and write data from 24C02 after initialization.

Interrupt function structure template:

```
// I2C2 Interrupt Routine
void I2C2_IRQHandler(void)
{
    uint32_t cmd;
    uint32_t tx_limit, rx_limit;

    if(I2C_GetITStatus(I2C2, I2C_IT_TX_ABORT) != RESET)
    {
        g_i2c_xfer_info.tx_abrt_source = I2C_GetTxAbortSource(I2C2);
        I2C2->INTR_MASK = I2C_INTR_STOP_DET; // Disable all interrupt except STOP_DET interrupt
        goto tx_aborted;
    }

    if(I2C_GetITStatus(I2C2, I2C_IT_RX_FULL) != RESET)
    {
        while((I2C_GetFlagStatus(I2C2, I2C_FLAG_RFNE) != RESET) && (g_i2c_xfer_info.rx_len))
        {
            *g_i2c_xfer_info.rx_buf = I2C_ReadData(I2C2);
            g_i2c_xfer_info.rx_buf++;
            g_i2c_xfer_info.rx_len--;
        }
    }
}
```

```

    }

    if(I2C_GetITStatus(I2C2, I2C_IT_TX_EMPTY) != RESET)
    {
        tx_limit = 8 - I2C_GetTxFIFOLevel(I2C2);
        rx_limit = 8 - I2C_GetRxFIFOLevel(I2C2);
        while((tx_limit > 0) && (rx_limit > 0))
        {
            if((g_i2c_xfer_info.tx_len + g_i2c_xfer_info.rx_cmd_len) == 0) {
                I2C_ITConfig(I2C2, I2C_IT_TX_EMPTY, DISABLE);    // Disable TX Empty Interrupt
                break;
            }

            cmd = 0;
            if((g_i2c_xfer_info.tx_len + g_i2c_xfer_info.rx_cmd_len) == 1) {
                cmd |= I2C_DATA_CMD_STOP;
            }

            if(g_i2c_xfer_info.tx_len != 0)
            {
                I2C_WriteDataCmd(I2C2, cmd | *g_i2c_xfer_info.tx_buf);
                g_i2c_xfer_info.tx_buf++;
                g_i2c_xfer_info.tx_len--;
            }
            else if(g_i2c_xfer_info.rx_cmd_len != 0)
            {
                I2C_WriteDataCmd(I2C2, cmd | I2C_DATA_CMD_READ);
                g_i2c_xfer_info.rx_cmd_len--;
                rx_limit--;
            }
            tx_limit--;
        }
    }

    tx_aborted:
    if(I2C_GetITStatus(I2C2, I2C_IT_STOP_DET) != RESET) {
        I2C_ClearITPendingBit(I2C2, 0xFFFF);    // Clear all interrupt flag
        I2C_ITConfig(I2C2, 0xFFFF, DISABLE);    // Disable all interrupt
        g_i2c_xfer_info.flag_complete = 1;
    }
}


```


If user need to use DMA, the sample code provides a sample that uses DMA to send and receive data. When porting, copy the DMA and I2C related functions to the project you want to port.

5.11. I2S

The MG32F10x has a built-in I2S bus interface, supporting a variety of audio transmission protocols, working in the master mode, supporting dual channel input and output. Provide master clock (MCLK) and serial clock (SCLK) as well as frame clock (WS) and serial data (SD0/SD1).

I2S is generally used for playing audio, so it also provides the example of recording audio code, refer to the example code in I2S path. User can refer actual situation to replace the underlying driver of the related initialization of STM32F10x as a whole. The driver used is located at mg32f10x_gpio.c, mg32f10x_rcc.c, mg32f10x_i2c.c and mg32f10x_i2s.c. Please note to add driver files. If user need to use es8316, files drv_es8316.c and wav_data.c also should be added.

 I2S_PlayAudio

 I2S_RecordPlayAudio

```
BSP_I2S_Init();
```

```
StartPlay();
```

According to the two functions above, adjust the required functions and packet size, and you can communicate with the I2S device and play audio.

Interrupt function structure template:

```
void I2S_IRQHandler(void)
{
    if (I2S_Channel_GetITStatus(1, I2S_IT_TXFE) != RESET)
    {
        I2S_Channel_WriteLeftData(1, (audio_data[audio_index + 1] << 8) | audio_data[audio_index]);
        I2S_Channel_WriteRightData(1, (audio_data[audio_index + 3] << 8) | audio_data[audio_index + 2]);

        audio_index += 4;
        if(audio_index >= audio_data_length)
        {
            audio_index = 0;
        }
    }
}
```

[Notify]: Strongly suggest not to use MG32F10x as USB audio, cause USB Buffer is 128Bytes which is not enough for audio.

5.12. IWDG(independent watchdog)

Refer to the IWDG sample code to enable the watchdog, the watchdog configuration is relatively simple. User can refer actual underflow frequency to replace the underlying driver of the related initialization of STM32F10x as a whole. The driver used is located at mg32f10x_pwr.c, mg32f10x_anctl.c, mg32f10x_rcc.c, mg32f10x_iwdg.c and mg32f10x_gpio.c. Please note to add driver files.

```
/* Enable IWDG clock */
```

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_IWDG, ENABLE);
```

```
/* Enable write access to IWDG_PR and IWDG_RLR registers */
```

```
IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);
```

```
/* IWDG counter clock: LSI/32 */
```

```
IWDG_SetPrescaler(IWDG_Prescaler_32);
```

```
while(IWDG_GetFlagStatus(IWDG_FLAG_PVU) != RESET);
```

```
/* IWDG timeout is about 250ms */
IWDG_SetReload(LSI_FREQ / 32 * 0.250);
while(IWDG_GetFlagStatus(IWDG_FLAG_RVU) != RESET);

/* ATTENTION: It is best to reload IWDG counter when the RVU bit is 0. */
while(IWDG_GetFlagStatus(IWDG_FLAG_RVU) != RESET);
IWDG_ReloadCounter();

/* Enable IWDG */
IWDG_Enable();

/* ATTENTION: It is best to reload IWDG counter when the RVU bit is 0. */
if (IWDG_GetFlagStatus(IWDG_FLAG_RVU) == RESET) {
    IWDG_ReloadCounter();
}
```

There are some limitations to the IWDG that need to be noted:

1. Once IWDG is enabled, it cannot be disabled even if a reset occurs.

Solution: Set the IWDG timeout setting to maximum before the code runs, and reload the IWDG counter constantly.

2. If a reset occurs after the IWDG is started, it takes three LSI clock cycles for the IWDG domain to be ready.

Solution: After the LSI is ready, delay about 1 millisecond before configuring the IWDG.

3. There is a chance that IWDG reload counter cannot be reloaded even execute reload IWDG counter repeatedly.

Solution: Make sure the RVU bit is 0 before performing the reload IWDG counter operation. (RVU bit indicates whether the reload counter operation is complete)

4. After the watchdog is enabled, the Debug function is out of control. If you still need the debug function, you are advised to enable DBG_IWDG_STOP in the DBGMCU_CR register to stop the watchdog in debugging.

5. The independent watchdog cannot generate interrupt. To generate interrupt, use the window watchdog.

At present, the sample code has been provided in accordance with the above standards to do, users transplant please pay attention to copy. In addition, the LSI clock has some deviation. Therefore, you need to reserve more time for timing.

5.13. LED

LED driver controller contains hardware 8-segment LED driver serial output circuit, and the module clock is the internal system clock by default. LED driver peripherals also have a corresponding example code, imitate the example of LED path under the sample code. The driver used is located at mg32f10x_rcc.c, mg32f10x_led.c and mg32f10x_gpio.c. Please note to add driver files.

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_BMX2 | RCC_APB2Periph_LED, ENABLE);

/* Reset LED module */
LED_DeInit();




/* LED configuration */
LED->CYC = 200;
LED->ECO = 180;
LED->CON = (0x00 << 4);
LED->CON |= 0x01;

/* Infinite loop */
while (1)
{
    for(iter = 0; iter < 16; iter++)
    {
        LED_SetSegmentCode(0, table[iter]);
    }
}
```

After initializing the peripherals, call the driver function LED_SetSegmentCode() to output. The actual display array needs to be adjusted according to the actual LED.

5.14. NVIC





NVIC mainly involves the control and shielding of interrupt priority, which will not be used in general migration, but if necessary, you can refer to the following program of NVIC path under the sample code.

-  NVIC_DMA_WFIMode
-  NVIC_IRQ_Mask
-  NVIC_IRQ_Priority

See readme.txt under each sample code for the functions implemented. It's explained in detail.

5.15. PWR(power control)

Setting power mode is a common setup, and the MG32F10x series offers SLEEP, STANDBY and STOP low-power modes which has sample code. The driver used is located at mg32f10x_pwr.c, mg32f10x_anctl.c and mg32f10x_rcc.c. Please note to add driver files.

-  PWR_PVD
-  PWR_SLEEP
-  PWR_STANDBY
-  PWR_STOP

```

/* Enter SLEEP Mode */
PWR_EnterSLEEPMode(PWR_FCLK_Div2, PWR_EntryMode_WFI);
/* Enter STANDBY Mode */
PWR_EnterSTANDBYMode();
/* Enter STOP Mode */
PWR_EnterSTOPMode(PWR_STOPMode_LP4_S32KOFF, PWR_EntryMode_WFI);

```

Select a mode based on actual requirements. User can refer actual using mode to replace the underlying driver of the related initialization of STM32F10x as a whole.

PVD is power detector to detect VDD voltage, user can set the detect voltage to detect VDD voltage. Copy void PVD_Config(void) into user project, and modify the voltage to use the PVD. The driver used is located at mg32f10x_pwr.c, mg32f10x_anctl.c and mg32f10x_rcc.c. Please note to add driver files.

```

/* Configure the PVD Level to 5 (refer to the electrical characteristics of
you device datasheet for more details) */
ANCTL_PVDLevelConfig(ANCTL_PVDLevel_5);

```

Table 12-3. PVD voltage threshold selection

PLS[2:0]	Voltage detect level on falling edge	Voltage detect level on rising edge
3'b000	2.14	2.25
3'b001	2.24	2.35
3'b010	2.34	2.45
3'b011	2.44	2.55
3'b100	2.54	2.65
3'b101	2.64	2.75
3'b110	2.74	2.85
3'b111	2.84	2.95

Refer to MG32F10x_RM.pdf, select voltage detect level.

Interrupt function structure template:

```

void PVD_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line16) != RESET)
    {
        /* Clear the EXTI line 16 pending bit */
        EXTI_ClearITPendingBit(EXTI_Line16);



        /* Change LED2 status */
        GPIO_ToggleBits(GPIOB, GPIO_Pin_13);
    }
}

```

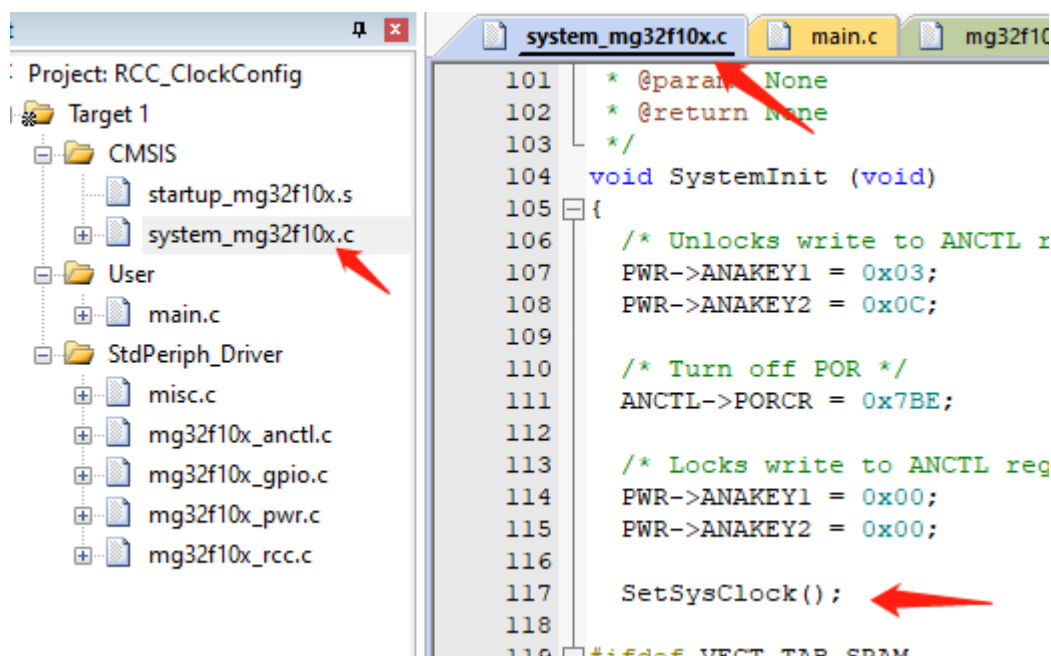

5.16. RCC

There are four clock sources in MG32F10x can be used to drive the system clock: MHSI(8MHz)internal OSC, FHSI(48MHz) internal OSC, PLL clock and HSE external OSC. In addition, There are two subclock sources in device: LSI(32KHz)internal low-speed OSC LSI(32KHz) internal lowspeed OSC, used to drive IWDG and LSE(32.768KHz) LSE(32.768KHz) external low-speed OSC clock to drive RTC.

There are two sample code is provided. The driver used is located at mg32f10x_pwr.c, mg32f10x_anctl.c, mg32f10x_rcc.c and mg32f10x_gpio.c. Please note to add driver files.

-  RCC_ClockConfig
-  RCC_ClockConfig2

The sample code under RCC_ClockConfig path is using XTAL to initial system clock while RCC_ClockConfig2 is using internal OSC. User may not find the place of clock initial. The function is located at the place as below instead of main.c. Of course, users can adjust the location of the clock initialization according to custom, such as putting it back in main.c.



```
static void SetSysClockTo72(void)
{
    __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

    /* Unlocks write to ANCTL registers */
    PWR->ANAKEY1 = 0x03;
    PWR->ANAKEY2 = 0x0C;

    /* APB1CLK = MAINCLK */
    RCC->APB1PRE = RCC_APB1PRE_SRCEN;
    RCC->APB1PRE |= 0x00;

    /* Configure PD0 and PD1 to analog mode */
    RCC->APB1ENR = RCC_APB1ENR_BMX1EN | RCC_APB1ENR_GPIODEN;
    GPIOD->CFGMSK = 0xFFFFC;
    GPIOD->MODER = 0x0F;

    /* Enable HSE */
    ANCTL->HSECR1 = ANCTL_HSECR1_PADOEN;
    ANCTL->HSECR0 = ANCTL_HSECR0_HSEON;

    /* Wait till HSE is ready and if Time out is reached exit */
    do
    {
        HSEStatus = ANCTL->HSESR & ANCTL_HSESR_HSERDY;
        StartUpCounter++;
    } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

    if (HSEStatus != 0)
    {
        /* Configure Flash prefetch, Cache and wait state */
        CACHE->CR = CACHE_CR_CHEEN | CACHE_CR_PREFEN_ON | CACHE_CR_LATENCY_2WS;

        /* AHBCLK = MAINCLK */
        RCC->AHBPRE = 0x00;

        /* APB2CLK = MAINCLK */
        RCC->APB2PRE = RCC_APB2PRE_SRCEN;
        RCC->APB2PRE |= 0x00;

        #if (HSE_VALUE == 6000000)
            /* PLL configuration: PLLCLK = 6MHz * 12 = 72 MHz */
            RCC->PLLSRC = RCC_PLLSRC_HSE;
            RCC->PLLPRE = RCC_PLLPRE_SRCEN;
            RCC->PLLPRE |= 0x00;
            ANCTL->PLLCR = ANCTL_PLLCR_PLLMUL_12;
        #elif (HSE_VALUE == 12000000)
            /* PLL configuration: PLLCLK = 12MHz / 2 * 12 = 72 MHz */
            RCC->PLLSRC = RCC_PLLSRC_HSE;
            RCC->PLLPRE = RCC_PLLPRE_SRCEN;
            RCC->PLLPRE |= RCC_PLLPRE_RATIO_2;
            RCC->PLLPRE |= RCC_PLLPRE_DIVEN;
            ANCTL->PLLCR = ANCTL_PLLCR_PLLMUL_12;
```

```
#endif
```

```

/* Enable PLL */
ANCTL->PLENR = ANCTL_PLENR_PLLON;

/* Wait till PLL is ready */
while(ANCTL->PLLSR != 0x03)
{
}

/* Select PLL as system clock source */
RCC->MAINCLKSRC = RCC_MAINCLKSRC_PLLCLK;
RCC->MAINCLKUEN = RCC_MAINCLKUEN_ENA;
}
else
{ /* If HSE fails to start-up, the application will have wrong clock
   configuration. User can add here some code to deal with this error */
while (1);
}

/* Locks write to ANCTL registers */
PWR->ANAKEY1 = 0x00;
PWR->ANAKEY2 = 0x00;
}

```

The code above is the code of clock initialization. User can copy and modify the code above refer to actual frequency you need and replace the underlying driver of the related initialization of STM32F10x as a whole. Please include `#include "mg32f10x.h"`.

Among them, the most critical part is frequency multiply, select PLL source.

```

RCC->PLLSRC = RCC_PLLSRC_HSE;
RCC->PLLPRE = RCC_PLLPRE_SRCEN;
RCC->PLLPRE |= RCC_PLLPRE_RATIO_2;
RCC->PLLPRE |= RCC_PLLPRE_DIVEN;
ANCTL->PLLCR = ANCTL_PLLCR_PLLMUL_12;

```

PLLSRC is select PLL source. If user don't use XTAL, the SRC should not select HSE, can set to `RCC_PLLSRC_MHSI`. The internal OSC is 8MHz. Then delete the code about HSE, and select correct frequency multiple and division multiple. After that, user can output correct PLL clock.

```

/* Select PLL as system clock source */
RCC->MAINCLKSRC = RCC_MAINCLKSRC_PLLCLK;
RCC->MAINCLKUEN = RCC_MAINCLKUEN_ENA;

```

Finally, select PLLCLK when select MAINCLKSRC.

5.17. RNG(Random Number Generator)

Random Number Generator (RNG) uses a 24bit LFSR to generate an 8bit random number. The RNG SFRs can be accessed through APB2. The driver used is located at mg32f10x_rng.c and mg32f10x_rcc.c. Please note to add driver files.

It's very simple to use RNG, just to enable it.

```
/* Reset RNG module */
RNG_DeInit();

/* Enable RNG generation */
RNG_Cmd(ENABLE);
printf("Generated random number is %d\r\n", RNG_RandByte());
```

As the code above, using RNG_RandByte() to catch a random number.

5.18. RTC

RTC migration is relatively simple, refer to the sample code under RTC path. The driver used is located at mg32f10x_pwr.c, mg32f10x_rcc.c, mg32f10x_rtc.c and mg32f10x_bkp.c. Please note to add driver files.

```
void RTC_Configuration(void)
/* Reset Backup Domain */
BKP_DeInit();

/* Enable LSE */
BKP_LSEConfig(BKP_LSE_ON);
/* Wait till LSE is ready */
while (BKP_GetLSEReadyFlagStatus() == RESET)
{}

/* Select LSE as RTC Clock Source */
BKP_RTCCLKConfig(BKP_RTCCLKSource_LSE);

/* Enable RTC Clock */
BKP_RTCCLKCmd(ENABLE);

/* Wait for RTC registers synchronization */
RTC_WaitForSynchro();

/* Wait until last write operation on RTC registers has finished */
RTC_WaitForLastTask();

/* Enable the RTC Second */
RTC_ITConfig(RTC_IT_SEC, ENABLE);

/* Wait until last write operation on RTC registers has finished */
RTC_WaitForLastTask();

/* Set RTC prescaler: set RTC period to 1sec */
RTC_SetPrescaler(32767); /* RTC period = RTCCLK/RTC_PR = (32.768 KHz)/(32767+1) */

/* Wait until last write operation on RTC registers has finished */
```

```
RTC_WaitForLastTask();
```

```
/* Sets the RTC counter */
```

```
RTC_SetCounter(40271);
```

As above, modify prescaler and period to make RTC working functionally. User can copy the code above and replace the underlying driver of the related initialization of STM32F10x as a whole.

```
void Time_Display(uint32_t TimeVar)
{
    uint32_t THH = 0, TMM = 0, TSS = 0;

    /* Reset RTC Counter when Time is 23:59:59 */
    if (RTC_GetCounter() >= 0x0001517F)
    {
        RTC_SetCounter(0x0);
        /* Wait until last write operation on RTC registers has finished */
        RTC_WaitForLastTask();
    }

    /* Compute  hours */
    THH = TimeVar / 3600;
    /* Compute minutes */
    TMM = (TimeVar % 3600) / 60;
    /* Compute seconds */
    TSS = (TimeVar % 3600) % 60;

    printf("Time: %0.2d:%0.2d:%0.2d\r", THH, TMM, TSS);
}
```

The sample code also provides Hour, Minute and second unit to convert.


Interrupt function structure template:


```
void RTC_IRQHandler(void)
{
    if (RTC_GetITStatus(RTC_IT_SEC) != RESET)
    {
        /* Clear the RTC Second interrupt */
        RTC_ClearITPendingBit(RTC_IT_SEC);

        /* Enable time update */
        TimeDisplay = 1;
    }
}
```

5.19. SFM(Special Function Macro)

SFM is simply used to Count the number of "1" in a WORD (32bit), expand all the bits in a WORD (32bit) by defined rate. Refer to the sample code under SFM path. The driver used is located at mg32f10x_sfm.c and mg32f10x_rcc.c. Please note to add driver files.

 SFM_ComputeBit1

 SFM_ExpandBits


```
CRC_SFM_DeInit();
```


```
printf("The number of bit 1 in 0xAAAAAAAA is %d\r\n", SFM_ComputeBit1Number(0xAAAAAAAA));
printf("The number of bit 1 in 0x55555555 is %d\r\n", SFM_ComputeBit1Number(0x55555555));
printf("The number of bit 1 in 0xFFFFFFFF is %d\r\n", SFM_ComputeBit1Number(0xFFFFFFFF));
printf("The number of bit 1 in 0x7FFFFFFF is %d\r\n", SFM_ComputeBit1Number(0x7FFFFFFF));
printf("The number of bit 1 in 0x00000000 is %d\r\n", SFM_ComputeBit1Number(0x00000000));
printf("The number of bit 1 in 0x1BC4D029 is %d\r\n", SFM_ComputeBit1Number(0x1BC4D029));
printf("The number of bit 1 in 0xFFFF0000 is %d\r\n", SFM_ComputeBit1Number(0xFFFF0000));
printf("The number of bit 1 in 0x0000F0FF is %d\r\n", SFM_ComputeBit1Number(0x0000F0FF));
printf("The number of bit 1 in 0x5503AAFF is %d\r\n", SFM_ComputeBit1Number(0x5503AAFF));
```


User can directly call the driver function to compute after simply initial SFM module.


5.20. SPI


Sample code about SPI is more, user can use them refer to their needs. All code is under SPI path. The driver used is located at mg32f10x_gpio.c, mg32f10x_rcc.c and mg32f10x_spi.c. Please note to add driver files.


 QSPI_Master_DMA


 QSPI_Master_Interrupt


 QSPI_QuadSPI_FLASH


 QSPI_SPI_FLASH


 SPIM2_Master_DMA


 SPIM2_Master_Interrupt

 SPIM2_SPI_FLASH

 SPIS1_Slave_DMA

 SPIS1_Slave_Interrupt

 SPIS2_Slave_DMA

 SPIS2_Slave_Interrupt

```
/* SPI configuration */
SPI_DeInit(SPIM2);
SPI_InitStructure.SPI_TransferMode = SPI_TransferMode_TxAndRx;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_BaudRatePrescaler = 8;
SPI_InitStructure.SPI_FrameFormat = SPI_FrameFormat_SPI;
SPI_Init(SPIM2, &SPI_InitStructure);
SPI_ITConfig(SPIM2, 0xFF, DISABLE);

SPI_NSSConfig(SPIM2, SPI_NSS_0, ENABLE);
```

As above to initial SPI, user can modify SPI clock according to prescaler and SPI data format to replace the underlying driver of the related initialization of STM32F10x as a whole.

```
while (SPI_GetFlagStatus(QSPI, SPI_FLAG_TFE) == RESET);
SPI_WriteData(QSPI, x);    //write value x through SPI

while (SPI_GetFlagStatus(QSPI, SPI_FLAG_RFNE) == RESET);
x = SPI_ReadData(QSPI);    //read SPI data to x
```

Interrupt function structure template:

```
void QSPI_IRQHandler(void)
{
    if(SPI_GetITStatus(QSPI, SPI_IT_RXF) != RESET)
    {
        while(SPI_GetFlagStatus(QSPI, SPI_FLAG_RFNE) != RESET)
        {
            master_rx_buf[rx_index] = SPI_ReadData(QSPI);
            rx_index++;
            if(rx_index >= 20) {
                SPI_ITConfig(QSPI, SPI_IT_RXF, DISABLE);
                break;
            }
        }
    }

    if(SPI_GetITStatus(QSPI, SPI_IT_TXE) != RESET)
    {
        while(SPI_GetFlagStatus(QSPI, SPI_FLAG_TFNF) != RESET)
        {
            SPI_WriteData(QSPI, master_tx_data[tx_index]);
            tx_index++;
            if(tx_index >= 20) {
                SPI_ITConfig(QSPI, SPI_IT_TXE, DISABLE);
                break;
            }
        }
    }
}
```

5.21. SYSTICK

Systick always be used to do millisecond delay. Refer to the sample code under SysTick path to configure SYSTICK. The driver used is located at mg32f10x_rcc.c. Please note to add driver files.

```
SystemCoreClockUpdate();
if (SysTick_Config(SystemCoreClock / 1000))
{
    /* Capture error */
    while (1);
}
```

Through code above to finish systick initialization. Parameter systemCoreClock should be modify to actual system clock. User can replace the underlying driver of the related initialization of STM32F10x as a whole.

```
void Delay(__IO uint32_t nTime)
{
    TimingDelay = nTime;

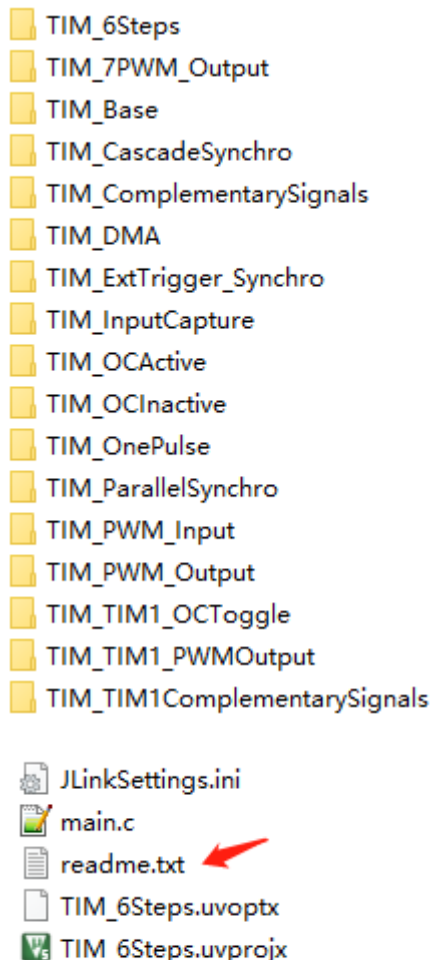
    while(TimingDelay != 0);
}

/**
 * @brief This function handles SysTick Handler.
 * @param None
 * @return None
 */
void SysTick_Handler(void)
{
    if (TimingDelay != 0x00)
    {
        TimingDelay--;
    }
}
```

Finally, add the interrupt function to call the Delay function. This Delay should also be replaced from STM32F10x Delay.

5.22. TIM

Timer is very commonly used peripherals, functions are very much, in order to facilitate the user transplant, we also provide a large number of timer sample code. Users can according to the sample code under TIM path of each different code in the readme.TXT, understand the implementation of each program function, and according to the actual use frequency and mode of configuration, and then replace the related TIM initialization function code of STM32F10x. The driver used is located at mg32f10x_tim.c and mg32f10x_rcc.c. Please note to add driver files.



[Notify]: If user need PWM with Dead time, please use TIM1.

TIM1ComplementarySignals code is the PWMComplementary output. User can refer to this sample in applications like BLDC and so on.

```
/* Time Base configuration */
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Period = TimerPeriod;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;

TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

/* Channel 1, 2 and 3 Configuration in PWM mode */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
TIM_OCInitStructure.TIM_Pulse = Channel1Pulse;
```

```

TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;

TIM_OC1Init(TIM1, &TIM_OCInitStructure);

TIM_OCInitStructure.TIM_Pulse = Channel2Pulse;
TIM_OC2Init(TIM1, &TIM_OCInitStructure);

TIM_OCInitStructure.TIM_Pulse = Channel3Pulse;
TIM_OC3Init(TIM1, &TIM_OCInitStructure);

/* Automatic Output enable, Break, dead time and lock configuration*/
TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;
TIM_BDTRInitStructure.TIM_DeadTime = 11;
TIM_BDTRInitStructure.TIM_Break = TIM_Break_Enable;
TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;

TIM_BDTRConfig(TIM1, &TIM_BDTRInitStructure);

/* TIM1 counter enable */
TIM_Cmd(TIM1, ENABLE);

/* Main Output Enable */
TIM_CtrlPWMOutputs(TIM1, ENABLE);

```

If user need a simply timing function, TIM_Base sample should be a good sample to refer. The driver used is located at mg32f10x_tim.c and mg32f10x_rcc.c. Please note to add driver files. The most difference from PWM sample is the TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing; PWM is PWMMode.

```

/* -----
TIM2 Configuration: Output Compare Timing Mode:
TIM2 counter clock at 12 MHz
CC1 update rate = TIM2 counter clock / CCR1_Val = 286.72 Hz
CC2 update rate = TIM2 counter clock / CCR2_Val = 523.97 Hz
CC3 update rate = TIM2 counter clock / CCR3_Val = 811.08 Hz
CC4 update rate = TIM2 counter clock / CCR4_Val = 1389.85 Hz
----- */

/* Compute the prescaler value */
PrescalerValue = (uint16_t) (SystemCoreClock / 12000000) - 1;

/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 0xFFFFF;
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

```

```
/* Prescaler configuration */
TIM_PrescalerConfig(TIM2, PrescalerValue, TIM_PSCReloadMode_Immediate);

/* Output Compare Timing Mode configuration: Channel1 */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR1_Val;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OC1Init(TIM2, &TIM_OCInitStructure);

TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);

/* Output Compare Timing Mode configuration: Channel2 */
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR2_Val;

TIM_OC2Init(TIM2, &TIM_OCInitStructure);

TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Disable);

/* Output Compare Timing Mode configuration: Channel3 */
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR3_Val;

TIM_OC3Init(TIM2, &TIM_OCInitStructure);

TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Disable);

/* Output Compare Timing Mode configuration: Channel4 */
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCR4_Val;

TIM_OC4Init(TIM2, &TIM_OCInitStructure);

TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Disable);

/* TIM IT enable */
TIM_ITConfig(TIM2, TIM_IT_CC1 | TIM_IT_CC2 | TIM_IT_CC3 | TIM_IT_CC4, ENABLE);

/* TIM2 enable counter */
TIM_Cmd(TIM2, ENABLE);
```

5.23. UART

Serial port is also commonly used. We also provide a large number of timer sample code about it. Users can according to the sample code under UART path of each different code in the readme.TXT, understand the implementation of each program function. Mostly UART_Printf sample is enough. The driver used is located at mg32f10x_uart.c, mg32f10x_rcc.c and mg32f10x_gpio.c. Please note to add driver files.

```
/* UART1 configuration */
UART_DelInit(UART1);
UART_InitStructure.UART_BaudRate = 115200;
UART_InitStructure.UART_WordLength = UART_WordLength_8b;
UART_InitStructure.UART_StopBits = UART_StopBits_One;
UART_InitStructure.UART_Parity = UART_Parity_None;
UART_InitStructure.UART_AutoFlowControl = UART_AutoFlowControl_None;
UART_Init(UART1, &UART_InitStructure);
UART_FIFOCmd(UART1, ENABLE);
```

Through code above, modify baud rate and data format you need to initial UART. Using pin can be changed 实 in GPIO initialization.

After initialization is finished, then user should be able to output data through UART by using printf function. Be attention, user should include # include <stdio.h> before using printf.

```
scanf("%d", &value);    //read uart data
printf("You enter is %d\r\n", value);    //write uart data
```

或者

```
while(!(UART_GetLineStatus(UART1) & UART_LINE_STATUS_DR));
value = UART_ReadData(UART1);    // read uart data
UART_WriteData(UART1, value);    // write uart data
while(!(UART_GetLineStatus(UART1) & UART_LINE_STATUS_THRE));
```





Interrupt function structure template:

```
void UART1_IRQHandler(void)
{
    uint8_t rbyte;
    uint8_t int_id;
    int_id = UART_GetIntID(UART1);
    if(int_id == UART_INTID_RDA)
    {
        rbyte = UART_ReadData(UART1);

        rxBuffer[rxIndex++] = rbyte;
        if (rxIndex >= 100) {
            flag = 1;
            rxIndex = 0;
        }
    }
    else if (int_id == UART_INTID_THRE)
    {
        if (txIndex < sizeof(txBuffer)) {
            UART_WriteData(UART1, txBuffer[txIndex]);
            txIndex++;
        }
        else {
            UART_ITConfig(UART1, UART_IT_THRE, DISABLE);
        }
    }
}
```

5.24. USB

We provide 4 different sample code about USB. User can do porting according to different applications. The specific implementation can also be seen in the readme.txt of various programs under the USB path of the sample code. The driver used is located at mg32f10x_anctl.c, mg32f10x_rcc.c, mg32f10x_pwr.c, mg32f10x_gpio.c and usbd_user.c. Please note to add driver files.

-  USB_CDC_Echo
-  USB_HID_Mouse
-  USB_Mass_Storage_SPI_FLASH
-  USB_Mass_Storage_SRAM

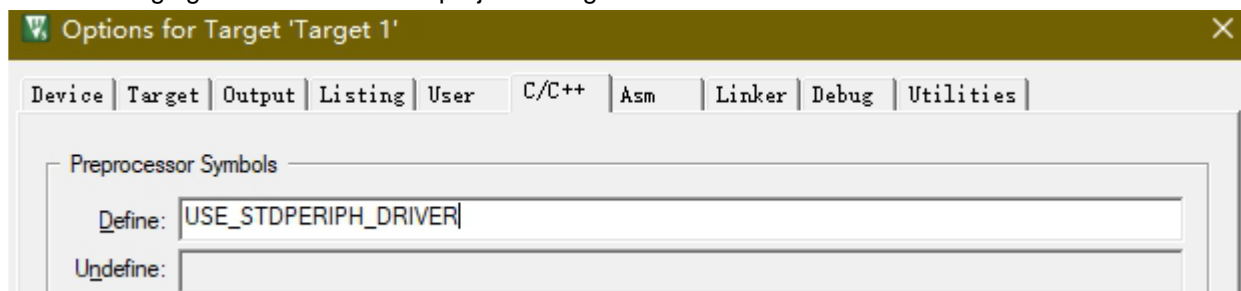
[Notify]: If user use **MHSI internal OSC** to be USB clock source, additional code need to added into project. The code has already stored in /Documents/USB_Do_Not_Use_Crystal_Code/

The following is the configuration process of the MHSI to be USB source. This method need to use SysTick or TIM4. Select a peripheral file you need to perform the following configuration.

一、Keil configuration

[Notify]: This configuration is only valid while using USB Function.

The following figure shows the KEIL project configuration.

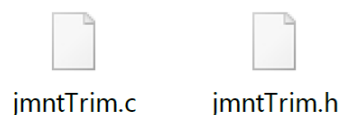


二、Peripheral requirement

This method need to use SysTick **or** TIM4.

三、Using course

1. Insert MHSI Trim code.



Insert jmntTrim.c and jmntTrim.h file into project.

2. Configure USB

a. Enable USB SOF interrupt.

```

/**
 * @brief Connects the device to the USB host.
 * @return None
 */
void USBD_User_Connect(void)
{
    /* Enable BMX1, GPIOA clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_BMX1 | RCC_APB1Periph_GPIOA, ENABLE);
    /* Configure the drive current of PA11 and PA12 */
    GPIO_DriveCurrentConfig(GPIOA, GPIO_Pin_11 | GPIO_Pin_12, 0x03);
    /* Configure PA11 and PA12 as Alternate function mode */
    GPIO_Init(GPIOA, GPIO_Pin_11 | GPIO_Pin_12, GPIO_MODE_AF | GPIO_AF3);

    USB->INTRUSBE = USB_INTRUSBE_RSTIE | USB_INTRUSBE_RSUIE | USB_INTRUSBE_SOFIE;
}

```

b. Include JmntTrim.h head file in usbd_user.c

```
#include "jmntTrim.h"
```

c. Call CheckTune function in USBD_User_SOF function.

```

void USBD_User_SOF(void)
{
    CheckTune();
}

```

d. The interrupt priority of USB should be configured to highest priority.

e. Other configurations can be configured based on user definition.

3. Call jmntTrimInit function in main function.

```

int main(void)
{
    MAINCLKConfig_MHSI_48MHz();
    jmntTrimInit();
}

```

Please note that jmntTrim.h head file must be included.

4. Configure Trim parameter refer to different system clock.

The default main frequency is 48 MHz. If the user's main frequency is 48 MHz, do not change the parameter. If the user's main frequency is not 48 MHz, user can modify the value.

```

/* The range of sof frame interval TIMER counts. */
#define INR_HEAD 44500
#define INR_TAIL 51500

```

5.25. WWDG(window watchdog)

The window watchdog timer (WWDG) is used to detect system failures due to software malfunctions. After the window watchdog timer starts, the value of downcounter reduces progressively. The watchdog timer causes a reset when the counter reached 0x3F (the CNT[6] bit becomes cleared). The watchdog timer also causes a reset if the counter is refreshed before the counter reached the window register value. So the software should refresh the counter in a limited window.

We also provide sample code about WWDG for user to port. User can configure the underflow period refer to they need and replace the related initialization function code of STM32F10x. The driver used is located at mg32f10x_wwdg.c and mg32f10x_rcc.c. Please note to add driver files.

```
/* WWDG clock counter = (PCLK(96MHz)/4096)/8 = 2929.6875 Hz (~0.341 ms) */
WWDG_SetPrescaler(WWDG_Prescaler_8);

/*
    Enable WWDG and set counter value to 127, WWDG timeout = ~0.341 ms * 64 = 21.8 ms
    In this case the refresh window is: ~0.341 ms * (127-80) = 16.027 ms < refresh window < ~0.341 ms *
    64 = 21.8ms
*/
WWDG_SetWindowValue(80);
WWDG_Enable(127);

WWDG_SetCounter(127);
```

The window watchdog is setting to allow to reload watchdog between counter value is window value to 64. Any other window to reload the counter or the counter value is less than 64, will cause a reset.

It's easy to reload the counter. Just execute WWDG_SetCounter(x); function in right time.

Interrupt function structure template:

```
void WWDG_IRQHandler(void)
{
    WWDG_ClearFlag();
}
```

6. Reversion

Version 1.01 (2022_0411)		Chapter
1	Change "LCD Driver" in hardware resource difference table to "LED Driver"	2.2
2	ADC software port added access data function and interrupt function template	5.2
3	CMP software port added read compare value function and notify	5.3.1
4	DCSS software port added interrupt function template	5.3.2
5	DMAC software port added interrupt function template	5.6
6	EXTI software port added interrupt function template	5.7
7	I2C software port added interrupt function template	5.10
8	I2S software port added interrupt function template	5.11
9	IWDG software port added notify	5.12
10	PVD software port added interrupt function template	5.15
11	RTC software port added access data function and interrupt function template	5.18
12	SPI software port added access data function and interrupt function template	5.20
13	UART software port added access data function and interrupt function template	5.23
14	WWDG software port added interrupt function template	5.25
Version 1.0 (2022_0214)		Chapter
1	Initial version	