SINO WEALTH

SH30F9/SB0 Series User Guide

-32-bit MCU Based on ARM® Cortex^{TM}-M0+ Core

- Version 1.0

Specification change record

Version	Record	Date
V1.0	Initial version	2024-10

Documentation Instructions

This user guide is guidance manual for the application of SH30F9010/30F9011/30F9810 (hereinafter referred to as SH30F9/SB0) microcontroller products. It mainly contains basic information and special attentions on how to useSH30F9/SB0 series products, including the configuration and use method of typical working mode of functional modules, and provides basic operation methods in the form of sample code.

This user guide does not contain a detailed description of product functions and technical features, including module internal structure, complete function introduction, pin number and distribution, electrical characteristics, package information, etc., which are provided in the product data manual.

This sample code of user guide uses direct register operation method, and does not use encapsulated peripheral library function operation method. For product development based on peripheral library, please refer to "SH30F9XX_SB0-STD_LIB_MANUAL.chm" instruction document in the SH30F9/SB0 series demo.The structure used in the routine is defined in the header file "sh30f9xx_sb0.h ".

Contents

1. RAM

1.1. Overview

The SH30F9/SB0 series chip has only one SRAM. SRAM is located in the 0x20000000 area. It is mainly used to store program variables, and can also run program code. It has a write protection function, which can protect the code in SRAM from being tampered with accidentally.

1.2. Programming Setting

The SRAM write protection of SH30F9/SB0 series chip takes 1K Bytes as a sector, and one protection bit corresponds to one sector, which is controlled by the SRAMLOCK register in the SYSCFG module.

【Note】 Writing to the write-protected SRAM will trigger a Hardfault exception.

【Note】 To run a specific program in SRAM, you need to identify the program segment in the program, and specify the address of the identified segment in the Linker configuration file to SRAM. Please refer to Appendix 1 for further information.

2. NVIC

2.1. Overview

The NVIC is the main component of the Cortex-M0+, which is tightly integrated with the CPU, reducing interrupt latency and allowing incoming interrupts to be handled efficiently. In addition to the exceptions provided by Cortex-M0+,The SH30F9/SB0 series chip also has 32 maskable interrupts and 4 programmable interrupt priorities (using 2-bit interrupt priority).

2.2. Programming Setting

2.2.1. *Interrupt configuration basis*

Each interrupt has an interrupt number, which is the identity of all interrupt operations and is defined in the chip header file. The initial address of the interrupt vector table is determined by the register VTOR, and this register is 0 when reset. The interrupt vector table is defined in startup_sh30f9xx_sb0_keil.s, and a default processing function is defined for each interrupt. When you need to replace this processing function, you only need to define an interrupt service function with the same name in the interrupt vector table in the code. There is no need to make any modification to startup_sh30f9xx_sb0_keil.s, the compiler will automatically replace it with a custom function.

2.2.2. *Enable and disable interrupt*

When enabling an interrupt, first set the interrupt enable bit inside the module to 1, and then set the interrupt enable bit in the NVIC to 1. When disabling, as long as any one of them is disabled, the interrupt service routine will not be executed. In addition, the special function register PRIMASK can more conveniently screen and interrupt. This register can be accessed by MSR, MRS instructions.

PRIMASK: interrupt mask register, only 1 bit. When set to 1, all maskable interrupts are masked.

2.2.3. *Interrupt pending and releasing*

The state of the interrupt can be controlled by the interrupt pending register (SETPEND) and the interrupt pending clear register (CLRPEND). So as to realize the function of triggering an interrupt by the program.

【Note】 The operation of writing the interrupt pending register has no effect on the suspended or disabled interrupts.

【Note】 The PEND register in NVIC will be automatically set and cleared to 0, so there is no need to modify it under non-special circumstances.

2.2.4. *Interrupt priority*

The external interrupt of SH30F9/SB0 series chip has a corresponding priority register, each register has 8 bits, and the Cortex-M0+ series can only use the highest 2 bits.

2.2.5. *Interrupt / fault response sequence*

Push: Push 8 registers into the stack (xPSR, PC, LR, R12, R3, R2, R1, R0)

Get vector: find out the corresponding service program entry address from the vector table

Update registers: select stack pointer MSP/PSP, update stack pointer SP, link register LR, program counter PC

SP: The stack pointer (PSP/MSP) will be updated to the new position when it is pushed into the stack. After the execution of the service routine, the access to the stack will be taken care of by the MSP.

PSR: The IPSR field (located in the lowest part of the PSR) will be updated with the exception number of the new response.

PC: After the vector is fetched, the PC will point to the entry address of the service routine.

LR: The usage of LR will be reinterpreted, and its value will be updated to a special value called "EXC_RETURN", and used when interrupt/abnormal return.

Update NVIC register: interrupt pending bit is cleared, active bit is set.

2.2.6. *Interrupt / fault return*

A return sequence is required after the interrupt/fault service program is executed. There are three ways to trigger the return sequence

BX <reg> when LR stores EXC_RETURN, use BX LR to return.

POP {PC} / POP {…,PC} EXC_RETURN is stored in the stack when out of the PC stack.

LDR / LDM loads PC as target register into EXC_RETURN.

Return sequence

POP: Restore the register of the previous stack

Update NVIC register: Interrupt active bit is cleared by hardware. For EXTI, if the interrupt input is set to be valid again, the pending bit will be set again, and a new interrupt response sequence will start again.

2.2.7. *Nested interrupts*

The NVIC of the CM0+ core supports interrupt nesting. It is necessary to configure the interrupt priority to control the nesting of interrupts.

【**Note**】 The capacity of the main stack should maintain a safe value, and each nesting level needs at least 8 words of space.

【**Note**】 The same interrupt does not allow re-entry.

2.2.8. *Tail interrupt*

If the priority of the pending interrupt is higher than that of all the push stack interrupts, the processor performs tail chaining. Tail-chaining means that when exiting ISR and entering another interrupt, the processor skips the push stack and pop stack operations of 8 registers, because it has no effect on the contents of the stack. Therefore, it is possible to implement back-to-back processing without redundant state saving and recovery instructions between two interrupts.

2.2.9. *Late interrupt*

If the previous ISR has not entered the execution phase, and the priority of the late interrupt is higher than that of the previous interrupt, the late interrupt can preempt the previous interrupt. The response to late interrupt makes it necessary to perform a new vector address and ISR prefetch operation. Late interrupt does not save the state because the state has been executed by the original interrupt.

3. Power, clock and reset

3.1. Overview

Power, clock and reset circuit are the core of industrial control chip design, and also the guarantee of reliability and anti-interference ability.

In terms of power, it supports 2.0V-5.5V power supply range, which is a real wide voltage power supply MCU. Besides, low voltage protection (LVR) and brownout detection (BOD) conventional protection circuits are added.

In terms of clock, it supports HSI, HSE, LSI, LSE and PLL clock sources. HSI clock can reach 0.3% accuracy at room temperature, which can replace crystal oscillator as high-precision clock source in most control applications.

In terms of reset circuit, it supports power on reset, independent pin reset, low voltage reset, watchdog reset and internal software reset, with various and perfect reset methods.

3.2. Programming Setting

3.2.1. *Power*

【**Note**】There will be a warm-up (stabilization) time for the power supply, so it is recommended to reserve 15ms (a certain margin on the basis of 11ms). The power supply can be preheated to about 1ms when it wakes up or reset from shutdown mode, because it does not involve power on again.

【**Note**】BOD is a flexible brownout detection circuit, which supports down detection, rise detection and bidirectional detection. It can also query the high and low status of current voltage and set voltage. After BOD is generated, it can apply for BOD interruption or NMI interruption. The latter is set by IEN_ BOD@SYSCFG_ SAFR control bit.

【**Note**】 BOD interrupt cannot wake up STOP mode, it is recommended not to try to use BOD interrupt to wake up STOP.

【**Note**】 In order to improve the anti-interference ability, the BOD is designed with a debouncing function to prevent BOD errors from occurring due to interference and glitches on VDD.

【**Note**】 There are four options for LVR. The LVR setting should refer to the VDD voltage and be higher than the minimum working voltage of the chip. For example, VDD=5V, you can set LVR=4.1V, VDD=3.3V, you can set LVR=2.8V. The LVR is designed with hysteresis and debouncing functions to prevent LVR errors from occurring due to interference and burrs on VDD.

【**Note**】 BOD is turned on and set the BOD threshold voltage by the internal register, and LVR is also turned on and set the LVR voltage by the internal register.

3.2.2. *Clock*

【**Note**】In addition to external crystal oscillator, HSE can also be filled with clock from XTAL1 pin, which can be filled with square wave or sine wave.

【**Note**】All clock sources should be switched after the corresponding READY bit is set.

【**Note**】HCLK、PCLK0 and PCLK1 have the maximum frequency limit, which should be noted when configuring the bus clock.

【**Note**】CSM is disabled by default and can be enabled through the configuration register. After CSM occurs, the system automatically switches to internal clock operation in 8 frequency division, and applies for NMI interrupt (enabled by the IEN_CSM@SYSCFG_SAFR control bit), and some system protection work can be done during NMI interrupt.

【**Note**】After CSM occurs, except for HSECSMF being set, some control bits of RCC will not be cleared, such as HSEON, LSION, SW, SWS all keep the original value, that is, the system clock has been switched at this time to HSI/8clock, but SW still maintains the original setting.

【**Note**】The system clock selects HSE, and when the CSM module is turned on, HSI must not be turned off, otherwise the CSM will work abnormally.

【**Note**】After CSM occurs, although the system clock has automatically switched to HSI/8, in order to maintain system stability, the AHB bus division factor must be set to 8 and the clock must be switched to HSI. After the failed clock is restored (HSERDY flag is set), clear the CSM flag, and the system clock will switch back to the abnormal pre clock. Then, change the AHB bus division factor back to the original set value.

【**Note**】SysTick uses HCLK/8 as the clock source by default, and the period constant is fixed at 60000. When HCLK=48MHz, the SysTick period is 10ms. If HCLK is other main frequency, the corresponding period will also change automatically. The calculation method is 60000*8/ HCLK.

【**Note**】The reset of each module is set by the software, and the hardware is automatically cleared after the reset is completed. After starting the reset, it is required to wait for a HCLK clock before reading the reset parameters.

3.2.3. *Reset*

【**Note**】The SH30F9/SB0 series has a complete power-on reset circuit, the key (pin) reset is not necessary, the key (pin) reset is reserved for flexibility and functional considerations, for example, after all the SWD debugging interfaces are multiplexed as GPIO ports To restore the debugging function, the key (pin) reset intervention is required, which is described in detail in the SYSCFG module.

【**Note**】 There is a power warm-up time after power-on reset and low-voltage reset, which is calculated as 18ms, while button (pin) reset, watchdog reset, and software reset only need to reserve about 1ms warm-up time.

4. GPIO

4.1. Overview

4.1.1. *GPIO mode*

GPIO mode is the basic working mode of I/O. After power-on, all I/O ports are in full-off mode (AF = 0) except the debug interface (SWD port). If you need to use the GPIO function, please set it to The relevant AF is set to 1, and it is recommended to set AF to 0 for unused GPIO ports.

GPIO mode has three basic working modes: GPIO input, GPIO output (push-pull), GPIO output (open drain).

GPIO input: MODER=0b, PHDRx=0b/1b, the read access to the input register IDR can get the level status of the I/O port.

【**Note**】 The I/O port level sampling is triggered only by the read operation, and the IDR keeps the last sampling value at other times.

GPIO output (push-pull): MODER=1b, OTYPE=0b, ODRVR_SINKx=0b/1b, the write operation to the output register ODR will change the I/O port level, write 1 to activate PMOS, output high level, write 0 to activate NMOS, output low level;

GPIO output (open drain): MODER=1b, OTYPE=1b, ODRVR_SINKx=0b/1b, writing 0 to the output register ODR can activate NMOS, output low level, write 1 cannot activate PMOS, cannot output high level, in High resistance state;

【Note】 In the open-drain mode, high-level output can be obtained through external pull-up, but the external pull-up should not exceed the working voltage of the chip, and in extreme cases, it should not exceed VDD+0.3V.

【Note】 Adjusting ODRVR_SINKx in output mode can change the capability of sinking current. Among them, the 70mA sinking current capability is only provided by some 8 pins, which are PB2-PB9/PC3~PC10. Limited by the chip GND maximum current limit of 200mA, only one channel of 200mA sink current can be started at a time in the application. If there are multiple channels, it can only be started in time-sharing by scanning.

4.1.2. *Multiplexing for digital peripherals*

Digital peripherals can select the mapped I/O port through the AF register. The SH30F9/SB0 series chip supports powerful pin remapping. The same peripheral can be mapped to I/O ports in different positions, which is convenient for system wiring and function combination.

The digital peripheral will control the input and output of the I/O port, that is, after an I/O port is mapped to a digital peripheral, the user does not need to configure the MODER register. In addition, other function options of the I/O port need to be configured by user software, such as pull-up, push-pull/open drain, and output drive capability.

【Note】 The configuration of digital peripherals and I/O port configuration does not have a strict order. Generally speaking, it is recommended to configure peripherals first for output ports and then I/O ports, and for input ports or bidirectional ports to configure I/O ports first and then configure peripherals. The former is because the output port of some peripherals will show a high-impedance state or a level before it is turned on, and it will show a level after it is turned on. If the I/O port is mapped in advance, the level change may have an impact on the external circuit. The latter is for the peripheral input port, which requires a deterministic level before it is turned on, so the I/O port needs to be pre-configured to avoid peripheral misoperation caused by level changes.

The configuration sequence can be simply summarized as:

Output peripherals: first configure the peripherals, turn on the peripherals, then configure the I/O port pull-down, output mode, and finally switch to the peripherals with AF.

Input peripherals: first configure the pull-down and output mode of the I/O port, then AF switches to the peripheral, and finally configure and turn on the peripheral (if the peripheral has an enable switch, it can be used as a peripheral synchronously during I/O configuration configuration, the last step is to turn on the peripherals).

When mapped as a digital peripheral, the GPIO output will be disconnected (actually only the ODR register output will be disconnected), but the GPIO input will not be closed, that is, the I/O port level can always be sampled through the IDR.

The mapping between peripherals and I/O ports can be easily determined through the "Multiplex Function Mapping Table".

【**Note**】The SH30F9/SB0 series chip does not prohibit overlapping mapping. For example, RXD0 can be mapped to PA4, PC3, PC4, PC5, PD0, and PD1 at the same time. Once this is done, the received signals of these 6 pins will be sent to RXD0, causing signal disorder. Application This configuration should be avoided. In the same way, TXD0 hardware also allows this kind of overlapping mapping, which should be avoided consciously by the user.

4.1.3. *Multiplexing for analog peripherals*

GPIO input and output channels are closed when mapped to analog peripherals. At this time, if the user reads the IDR register, it will read 0.

It is meaningless to configure the MODER, OTYPE, and ODRVR registers for analog peripherals, but it is allowed to configure the PUPDR register to construct a pull-up on the I/O port.

【**Note**】 The analog peripheral requires the I/O port to be switched to the relevant AF first, and then the analog peripheral is turned on.

Note: This is different from digital peripherals. Digital peripherals are only recommended, and analog peripherals are required.

4.1.4. *Full close mode*

The full close mode will cut off the input and output channels, which can completely eliminate the leakage problem that exists when the I/O port is configured as an input floating. For simplicity, unused I/O ports can be configured as full close during application.

During power-on reset, all I/O ports except the SWD port are in the fully closed state, and enter the default state (GPIO input and output are all closed) after the reset is completed.

4.1.5. *Debug interface*

Two-wire SW debug interface (SWD interface): Both SWDIO and SWCLK can be multiplexed as GPIO mode.

After the system is powered on and reset, it is used as the debug interface by default. To reuse it as GPIO mode, it needs to be modified through the SWJCFG@SYSCFG_SAFR register.

【**Note**】 The GPIO mode here includes the digital peripheral mode, collectively refers to the "non-debugging interface" mode, which can be used as a digital peripheral mapping port.

【**Note**】 Please refer to the definition of the OSCCFG@SYSCFG_SAFR register for the multiplexing of the oscillator interface XTAL1/XTAL2. The default state is that the input and output are all off when powered on.

【**Note**】 The two control bits SWJCFG and OSCCFG in the SYSCFG_SAFR register can only be set once after reset, and once set in the program, they cannot be changed.

4.1.6. *Other notes*

【**Note**】 The I/O configuration operation has a corresponding lock/unlock function, and the configuration information cannot be modified after being locked.

```
GPIOB ->LCKR.V32 = 0x5AA5FFFF;          // lock PB config
GPIOB ->LCKR.V32 = 0x5AA50000;          // unlock PB config
```

【**Note**】 There are two ways to set and clear the port: BSRR and ODR, and the set and clear operation of ODR is realized through BSRR. When BSRR is valid for setting and clearing a certain bit at the same time, setting has high priority.

```
GPIOB->BSRR.BIT.BS = 0x8000;        // set PB15 = 1
GPIOB->BSRR.BIT.BR = 0x8000;        // set PB15 = 0
GPIOB->ODR |= 0x8000;               // set PB15 = 1
GPIOB->ODR &= 0x7FFF;               // set PB15 = 0
```

【**Note**】 When using the GPIOx->ODR register, you should pay attention to turn off the interrupt before use, and turn on the interrupt after the setting is completed. It is recommended that customers use the GPIOx->BSRR register to set or clear the IO

【**Note**】 TTL level input function. This function is mainly aimed at the level matching problem when the external port working at TTL level is connected to this chip. The TTL level input option changes the input high and low voltage thresholds (V_{IH3}, V_{IL3}).

【**Note**】 The TTL level input function is still valid in STOP mode.

5. Flash&EEPROM

5.1. Overview

The SH30F9/SB0 series chip has a built-in programmable Flash main program storage area of up to 256K, For detailed information, please refer to the product information section in the specification sheet. each sector is 1024 bytes, and 256 sectors can be erased individually. After the erase operation of the main storage area, each 16-bit or 32-bit can only be programmed once, that is, if each bit of the programmed 16-bit or 32-bit data is not 0, it cannot be programmed again, otherwise the corresponding error flag set to 1.

The 4KB built-in EEPROM-like storage area can store data, each sector is 1024 bytes, and each sector can be erased individually.

The 1KB OTP area is used to store factory initialization data. It should be noted that once the data in the OTP area is written, it cannot be erased.

The main program area in Flash can be prevented from being illegally read by setting read protection; it is also possible to set write protection for each sector of the Flash storage area to prevent it from being accidentally changed when the program runs away. The basic unit of write protection is 8 sectors as a protection unit.

5.2. Programming Setting

The operation procedures of Flash, EEPROM, and OTP areas are basically the same, roughly divided into the following steps:

- Unlock the corresponding operation register
- Set the Flash operation timer
- Flash (E2\OTP) erase programming configuration
- Lock the corresponding operation register

【**Note**】 Before erasing and programming the Flash (E2\OTP) memory, please feed the WDT to the dog, and the LVR function is turned off to avoid reset during the operation. and turn off the total interrupt. After the operation is completed, turn on the total interrupt.

5.2.1. *Flash Control Register Unlocking*

After the system is reset, the Flash memory operation register enters the locked state, so the Flash memory operation register needs to be unlocked before erasing and programming the Flash. Different Flash storage areas have different unlock registers, and operations in different areas need to unlock the corresponding registers. To unlock Flash, you need to write two unlock values in sequence to the corresponding unlock register. The first unlock value is the initial unlock value, and the second unlock value is the unlock value of a single or multiple operations; the first key value is written incorrectly or The second key value write error will not unlock the corresponding area of the Flash and will generate a HardFault; if any area is in the unlocked state, unlocking the unlock register will also generate a HardFault. There are also two situations for unlocking the Flash. One is that a single operation can be performed on the Flash after unlocking, and the other is that multiple operations can be performed on the Flash after unlocking. If the Flash memory operation register is in the unlocked state, the flash memory area can be erased and programmed by controlling FLASH_CR.

Unlock value and description of each unlock register

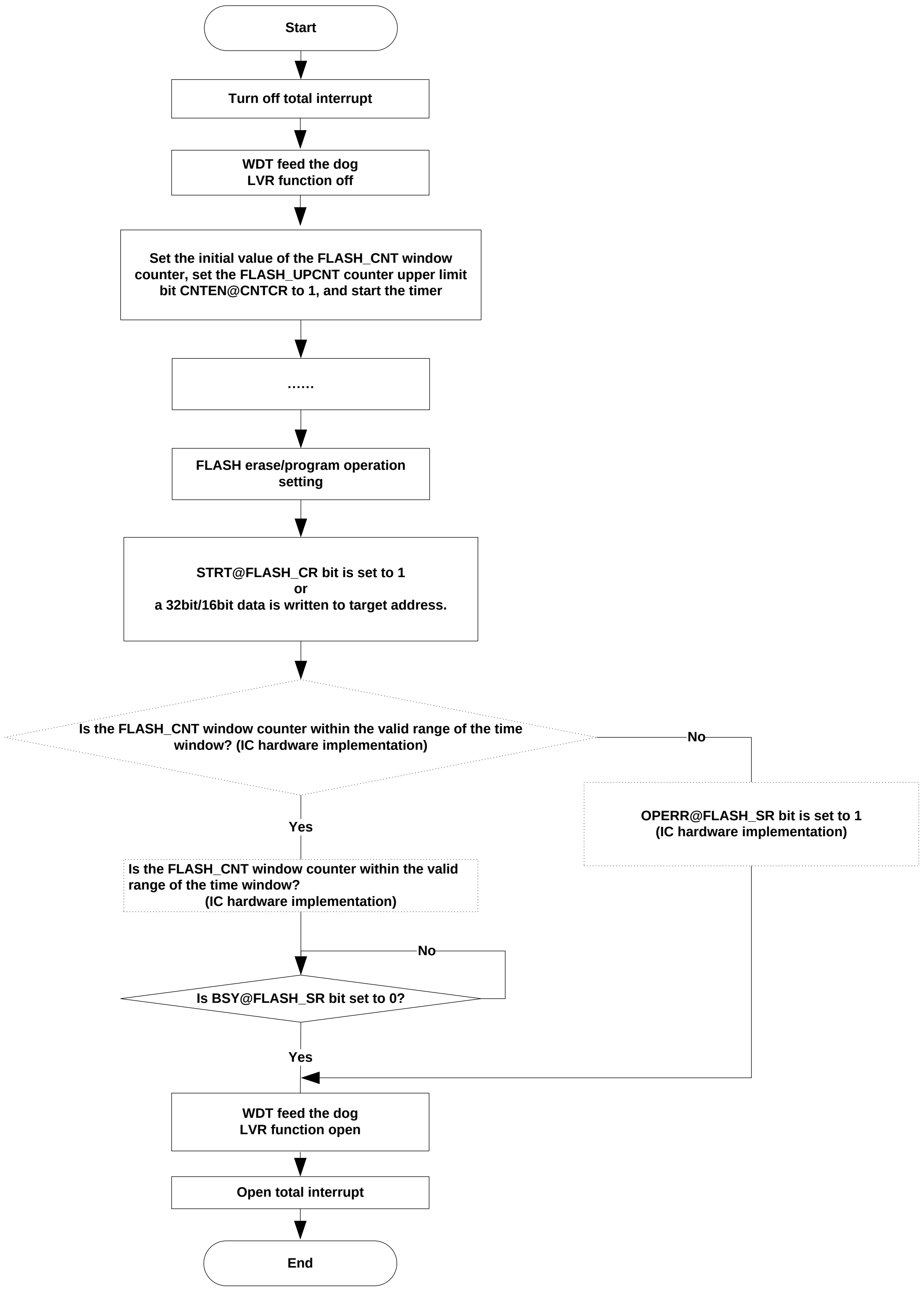
Register	Function	Flash Initial unlock value	Single operation unlock value	Multiple operations unlock value	Instructions
FLASH_MKYR	Unlock main program area	0x8ACE 0246	0xC3C3 C3C3	0xB4B4 B4B4	Directly access through programs running in RAM, programs in the main program area, programs in the Boot area or using debugging tools
FLASH_E2KYR	Unlock EEPROM like area of	0x9BDF 1357	0xC3C3 C3C3	0xB4B4 B4B4	Directly access through programs running in RAM,

	special information area				programs in the main program area, programs in the Boot area or using debugging tools
FLASH_ IKYR	Unlock the code protection block, customer block, and OTP area of the special information area	0xABCD 5678	0xC3C3 C3C3	0xB4B4 B4B4	The code protection area, customer information area and OTP area can be implemented by programs running in RAM, programs in the main program area, programs in the Boot area, or direct access to the Flash operation registers using a debugging tool; while the Boot area can only be implemented by using debugging The tool directly accesses the Flash operation register to achieve this.

5.2.2. *Flash operation timer function*

The Flash control module has an operation timer. The count value of the Flash operation timer is based on the system clock as the clock source. The count value of the Flash operation timer must be less than the upper limit value of the Flash operation timer and greater than zero before storing data in the Flash. Area to operate, or write a 32bit/16bit data to the destination address. If a Flash operation is started, that is, when bit STRT@FLASH_CR is set to 1 or a write operation to Flash is started, the count value of the Flash operation timer is not within the valid time window, the Flash operation is not executed, and the bit PGWERR@FLASH_SR is set to 1. The specific usage of the Flash operation timer can refer to the figure below.

It should be noted that the CNTEN@FLASH_CNTCR bit controls whether the operation timer starts counting down. When the CNTEN@FLASH_CNTCR bit is 1, the timer starts counting down; when the CNTEN@FLASH_CNTCR bit is 0, the Flash operation timer counts unchanged, so , as long as the count value of the Flash operation timer is not greater than the upper limit of the Flash operation timer, the Flash operation can be performed normally.


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Set the initial value of the FLASH_CNT window counter, set the FLASH_UPCNT counter upper limit bit CNTEN@CNTCR to 1, and start the timer
……
FLASH erase/program operation setting
STRT@FLASH_CR bit is set to 1
or
a 32bit/16bit data is written to target address.
Is the FLASH_CNT window counter within the valid range of the time window? (IC hardware implementation)
No
OPERR@FLASH_SR bit is set to 1
(IC hardware implementation)
Yes
Is the FLASH_CNT window counter within the valid range of the time window?
(IC hardware implementation)
No
Is BSY@FLASH_SR bit set to 0?
Yes
WDT feed the dog
LVR function open
Open total interrupt
End

Flash Operation Timer Flowchart

5.2.3. *Flash(E2\OTP) Erasing and Programming*

After the Flash, EEPROM, OTP area are operated, and then the flash controller is unlocked and the flash operation timer is set, the flash control register needs to be configured. The options to be configured include:

- Flash operation commandword: CMD@FLASH_CR
  0xE619: main block sector erase (MSE)
  0x6E91: main block programming (MPG)
  0xB44B: E2Prom block programming (E2PG)
  0x4BB4: E2Prom block sector erase (E2Prom-like) (E2SE)
  0xF00F: OTP block programming (OPG)
- Program operation bit width selection：PSIZE@FLASH_CR
  0: 32-bit programming simultaneously
  1: 16-bit programming simultaneously
- Sector erase selection: SNB@FLASH_CR1
  00000000: Sector 0
  00000001: Sector 1
  00000010: Sector 2
  ……..
  11111110: Sector 254
  11111111: Sector 255

Note: This bit is only valid when the main program area is erased by sector or the EEPROM-like area is erased by sector. When the main program area is erased by sector, the size of each sector is 1024 bytes; the EEPROM-like area is During sector erasing, the size of each sector is 1024 bytes.

- Starting flag bit: STRT@FLASH_CR

When the operation command word is erase, an operation will be triggered when the bit is '1'; when the operation command word is programming, an operation will be triggered when a 32-bit word/16-bit half word is written to the destination address. This bit can only be set to '1' by software and cleared to '0' when BSY@FLASH_SR becomes '1'. When BSY@FLASH_SR is cleared '0', it means the operation is over. At this time, you need to query the status bits in the FLASH_SR register to determine whether the operation is performed normally. If there is an error, the corresponding error flag will be set to 1.

5.2.4. *Flashcontrol register locking*

If the unlocking mode of flash control register allows multiple operations, the corresponding flash control register must be relocked to prevent the flash from being misoperated.

5.3. Flash (E2\OTP) erase programming flow chart

5.3.1. *Mass erase of main program area*

The Flash operation provides a full-chip erase function, which can erase the contents of the main memory block area. Specific steps are as follows:

(1) Turn off the total interrupt;

(2) Feed the WDT to the dog, and the LVR function is turned off to avoid reset during operation;

(3) FLASH_MKYR sequentially writes the Flash unlock value and single operation unlock value to ensure that the Flash main program area is not locked;

(4) Check the bit BSY@FLASH_SR to determine whether the Flash memory is running;

(5) When the bit BSY@FLASH_SR is 0, write the mass erase command word in the Flash main program area to the FLASH_CR register;

(6) Start the chip erase operation by setting bit STRT@FLASH_CR to 1;

(7) Check the bit BSY@FLASH_SR to determine whether the erase command has been executed;

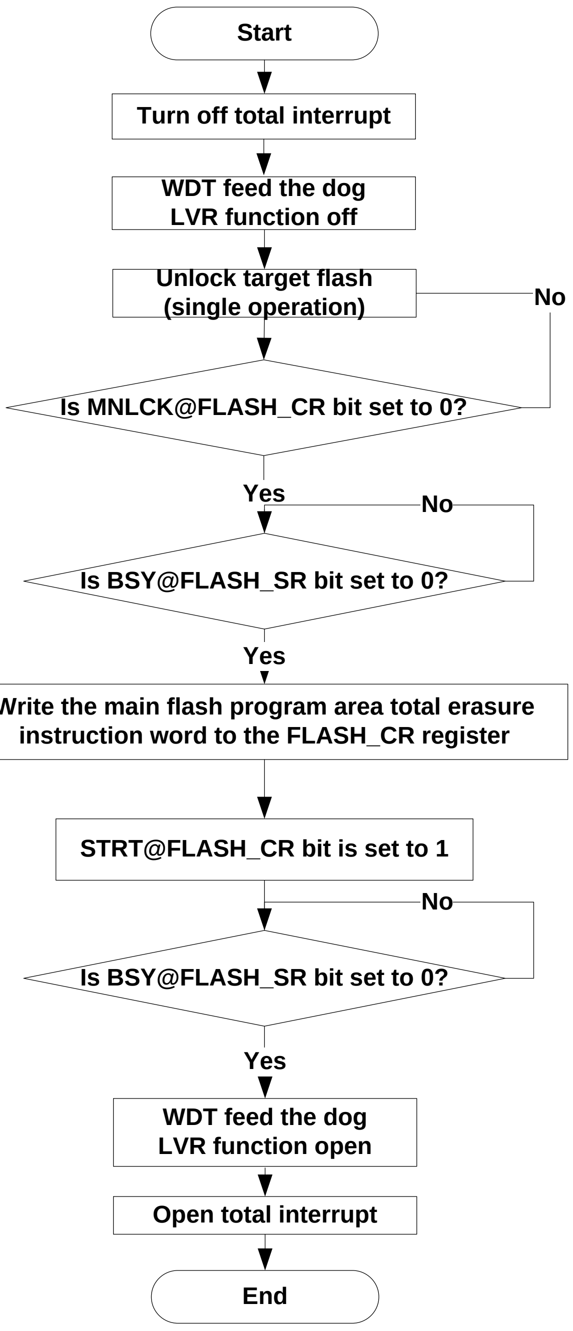
(8) When the bit BSY@FLASH_SR is 0, the operation is completed;

(9) Feed the WDT to the dog, and the LVR function is turned on;

(10) Turn on the total interrupt.

The bit EOP@FLASH_SR indicates the end of the operation, and this bit is 1, indicating that the operation is completed. All Flash data is reset to 0x0000 0000 after erasing.

The flow chart is as follows:


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Unlock target flash
(single operation)
No
Is MNLCK@FLASH_CR bit set to 0?
Yes
No
Is BSY@FLASH_SR bit set to 0?
Yes
Write the main flash program area total erasure instruction word to the FLASH_CR register
STRT@FLASH_CR bit is set to 1
No
Is BSY@FLASH_SR bit set to 0?
Yes
WDT feed the dog
LVR function open
Open total interrupt
End

Mass erase of the main program area

Note: Before running the program in the main program area of Flash to execute the mass erase, the mass erase protection byte (Addr: 0x0FFF8020) in the main program area needs to be programmed to 0x01, and then the erase operation can be performed.

5.3.2. *Main program area sector erase*

Each sector of the Flash memory can be erased independently without affecting the contents of other sectors. The steps of flash operation to erase sectors are as follows:

(A) Erase a certain sector individually

(1) Turn off the total interrupt;

(2) Feed the WDT to the dog, and the LVR function is turned off to avoid reset during operation;

(3) FLASH_MKYR sequentially writes the Flash unlock value and single operation unlock value to ensure that the Flash main program area is not locked;

(4) Check the bit BSY@FLASH_SR to determine whether no Flash memory is running;

(5) When the bit BSY@FLASH_SR is 0, write the sector erase command word in the main program area and the sector number to be erased to the FLASH_CR and FLASH_CR1 registers respectively;

(6) Start the sector erase operation by setting the bit STRT@FLASH_CR to 1;

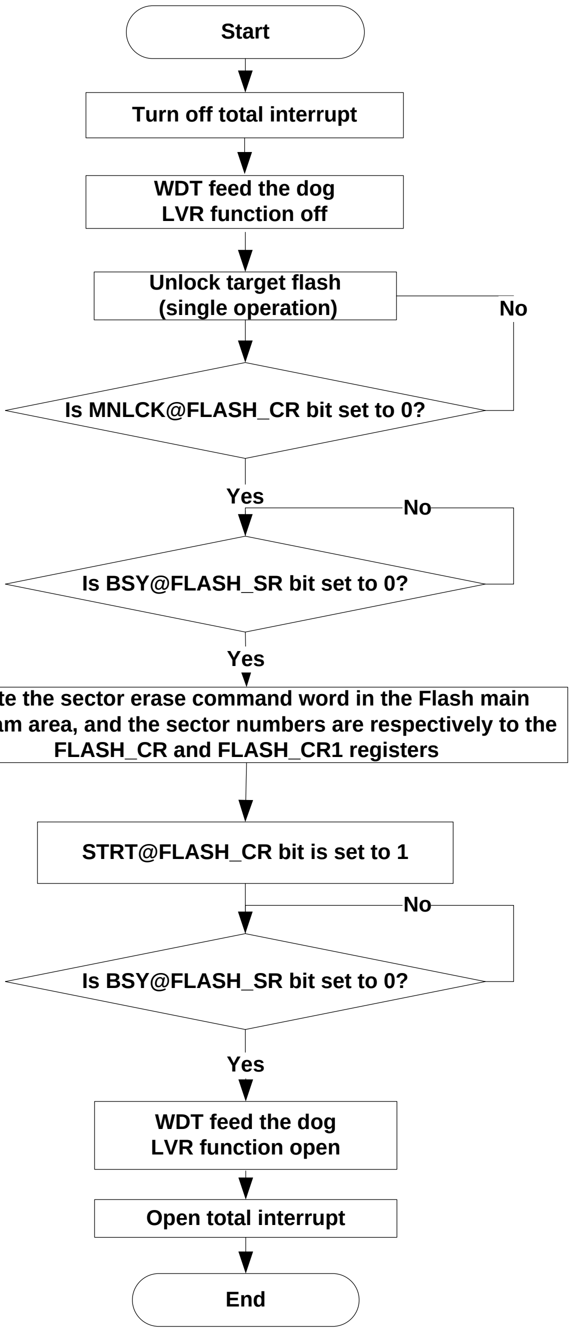
(7) Check the bit BSY@FLASH_SR to determine whether the erase command has been executed;

(8) When the bit BSY@FLASH_SR is 0, the operation is completed;

(9) Feed the WDT to the dog, and the LVR function is turned on;

(10) Turn on the total interrupt.

The flow chart is as follows:


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Unlock target flash
(single operation)
No
Is MNLCK@FLASH_CR bit set to 0?
Yes
No
Is BSY@FLASH_SR bit set to 0?
Yes
Write the sector erase command word in the Flash main program area, and the sector numbers are respectively to the FLASH_CR and FLASH_CR1 registers
STRT@FLASH_CR bit is set to 1
No
Is BSY@FLASH_SR bit set to 0?
Yes
WDT feed the dog
LVR function open
Open total interrupt
End

The main program area is erased by sector (single sector)

(B) Erase multiple sectors

(1) Turn off the total interrupt;

(2) Feed the WDT to the dog, and the LVR function is turned off to avoid reset during operation;

(3) FLASH_MKYR sequentially writes the Flash unlock value and multiple operation unlock values to ensure that the Flash main program area is not locked;

(4) Check the bit BSY@FLASH_SR to determine whether no Flash memory is running;

(5) When the bit BSY@FLASH_SR is 0, write the sector erase command word and sector code in the main program area to FLASH_CR and FLASH_CR1 respectively

device;

(6) Start the sector erase operation by setting the bit STRT@FLASH_CR to 1;

(7) Check the bit BSY@FLASH_SR to determine whether the erase command has been executed;

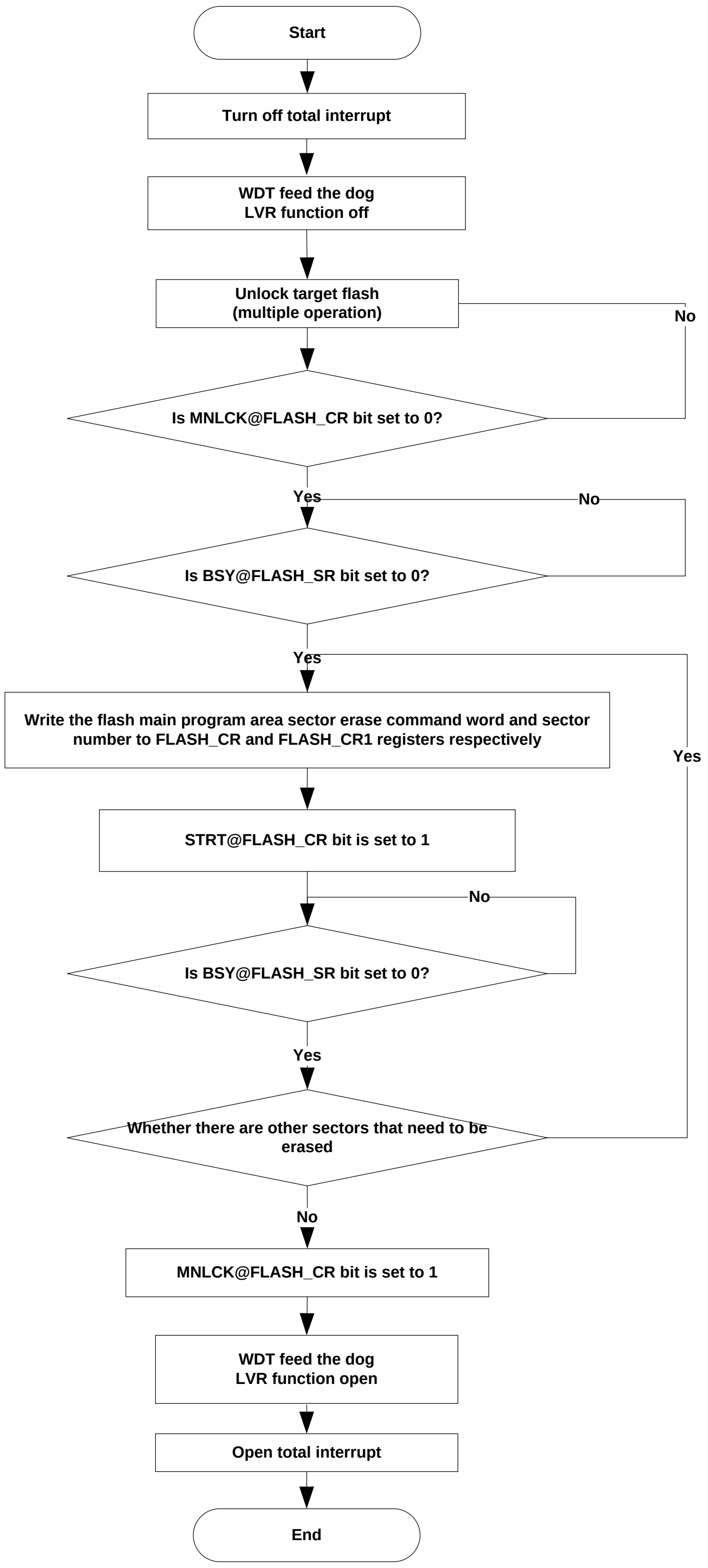
(8) Repeat (5)-(7) to erase other sectors;

(9) Bit MNLCK@FLASH_CR is set to 1, and the operation is completed;

(10) Feed the WDT to the dog, and the LVR function is turned on;

(11) Turn on the total interrupt.

The operation flow chart is as follows:


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Unlock target flash
(multiple operation)
Is MNLCK@FLASH_CR bit set to 0?
No
Yes
Is BSY@FLASH_SR bit set to 0?
No
Yes
Write the flash main program area sector erase command word and sector number to FLASH_CR and FLASH_CR1 registers respectively
Yes
STRT@FLASH_CR bit is set to 1
Is BSY@FLASH_SR bit set to 0?
No
Yes
Whether there are other sectors that need to be erased
No
MNLCK@FLASH_CR bit is set to 1
WDT feed the dog
LVR function open
Open total interrupt
End

The main program area is erased by sector (multiple sectors)

When the target erased sector is used to fetch instructions or access data, the corresponding erase operation is invalid, but the Flash operation will not provide any notification, so it is necessary to ensure the correctness of the target erased sector address. Bit EOP@FLASH_SR heralds the end of operation.

5.3.3. *Main program area programming*

Flash operation provides a 32-bit word/16-bit half-word programming function to modify the content of the Flash main memory block. 32-bit/16-bit can be written into the Flash main program area each time, and the address must be aligned according to 32-bit/16-bit. If it is not aligned, the corresponding operation is invalid and the relevant error flag is set to 1. When the FLASH_CR register operation command word is set to program in the main program area, writing a word or half word at a Flash address will start a program. During the programming process (the bit BSY@FLASH_SR is '1'), any operation of reading and writing Flash will cause the CPU to suspend until the end of this Flash programming.

The following steps show the word program operation register procedure.

(A) Single word or half word programming

(1) Turn off the total interrupt;

(2) Feed the WDT to the dog, and the LVR function is turned off to avoid reset during operation;

(3) FLASH_MKYR sequentially writes the Flash unlock value and single operation unlock value to ensure that the Flash main program area is not locked;

(4) Check the bit BSY@FLASH_SR to determine whether no Flash memory is running;

(5) When the bit BSY@FLASH_SR is 0, write the programming operation command word in the Flash main program area to the FLASH_CR register;

(6) Write a 32-bit word/16-bit half word to the destination address;

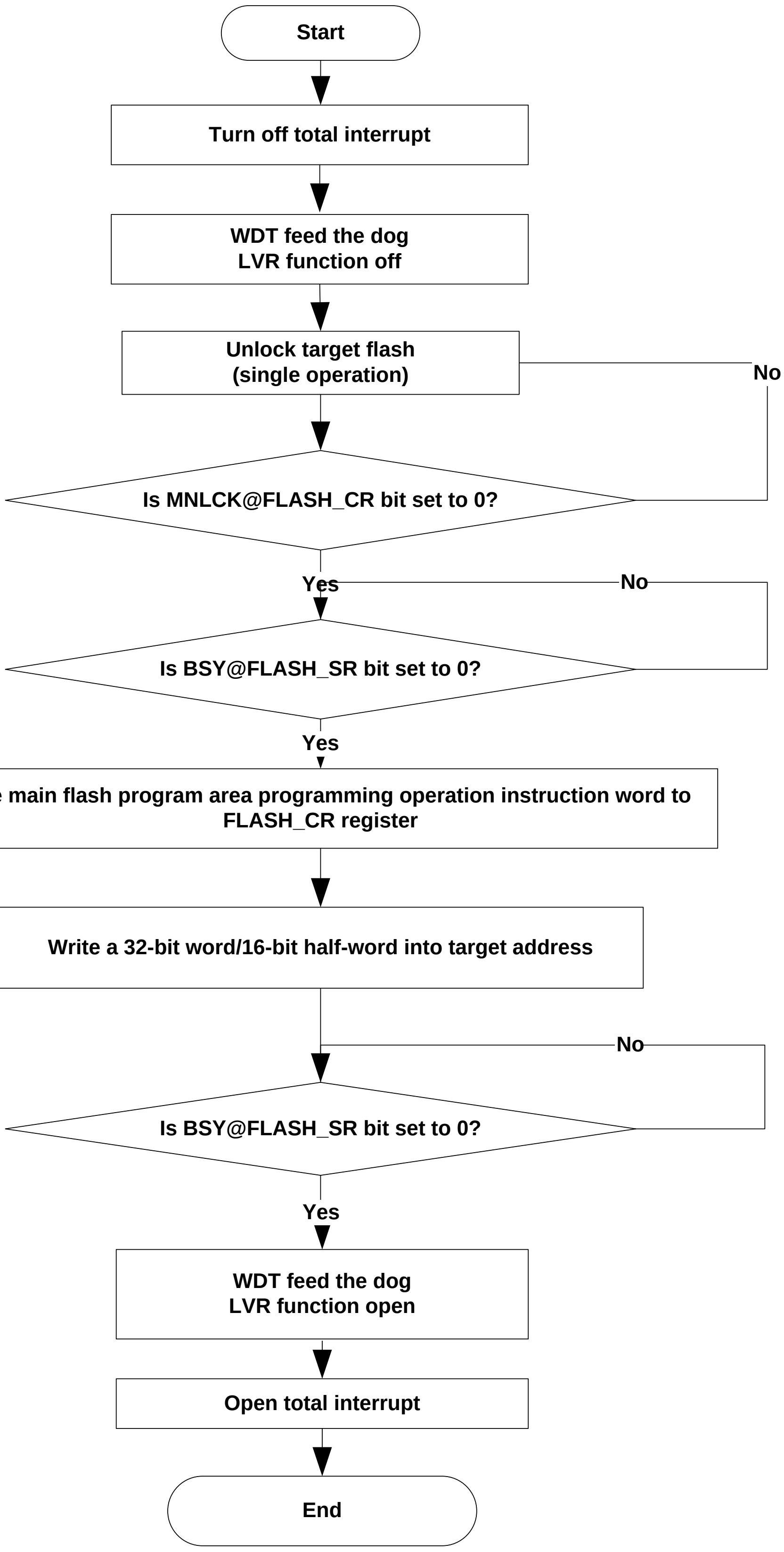
(7) Check the bit BSY@FLASH_SR to judge whether the programming instruction has been executed;

(8) When the bit BSY@FLASH_SR is 0, the operation is completed;

(9) Feed the WDT to the dog, and the LVR function is turned on;

(10) Turn on the total interrupt.

The operation flow chart is as follows:


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Unlock target flash
(single operation)
No
Is MNLCK@FLASH_CR bit set to 0?
Yes
No
Is BSY@FLASH_SR bit set to 0?
Yes
Write main flash program area programming operation instruction word to FLASH_CR register
Write a 32-bit word/16-bit half-word into target address
No
Is BSY@FLASH_SR bit set to 0?
Yes
WDT feed the dog
LVR function open
Open total interrupt
End

Main program area programming (single word or half word)

(B) Continuously write multi-word programming

(1) Turn off the total interrupt

(2) Feed the WDT to the dog, and the LVR function is turned off to avoid reset during operation;

(3) FLASH_MKYR sequentially writes the Flash unlock value and multiple operation unlock values to ensure that the Flash main program area is not locked;

(4) Check the bit BSY@FLASH_SR to determine whether no Flash memory is running;

(5) Write the programming operation command word in the Flash main program area to the FLASH_CR register;

(6) Write a 32-bit word/16-bit half word to the destination address;

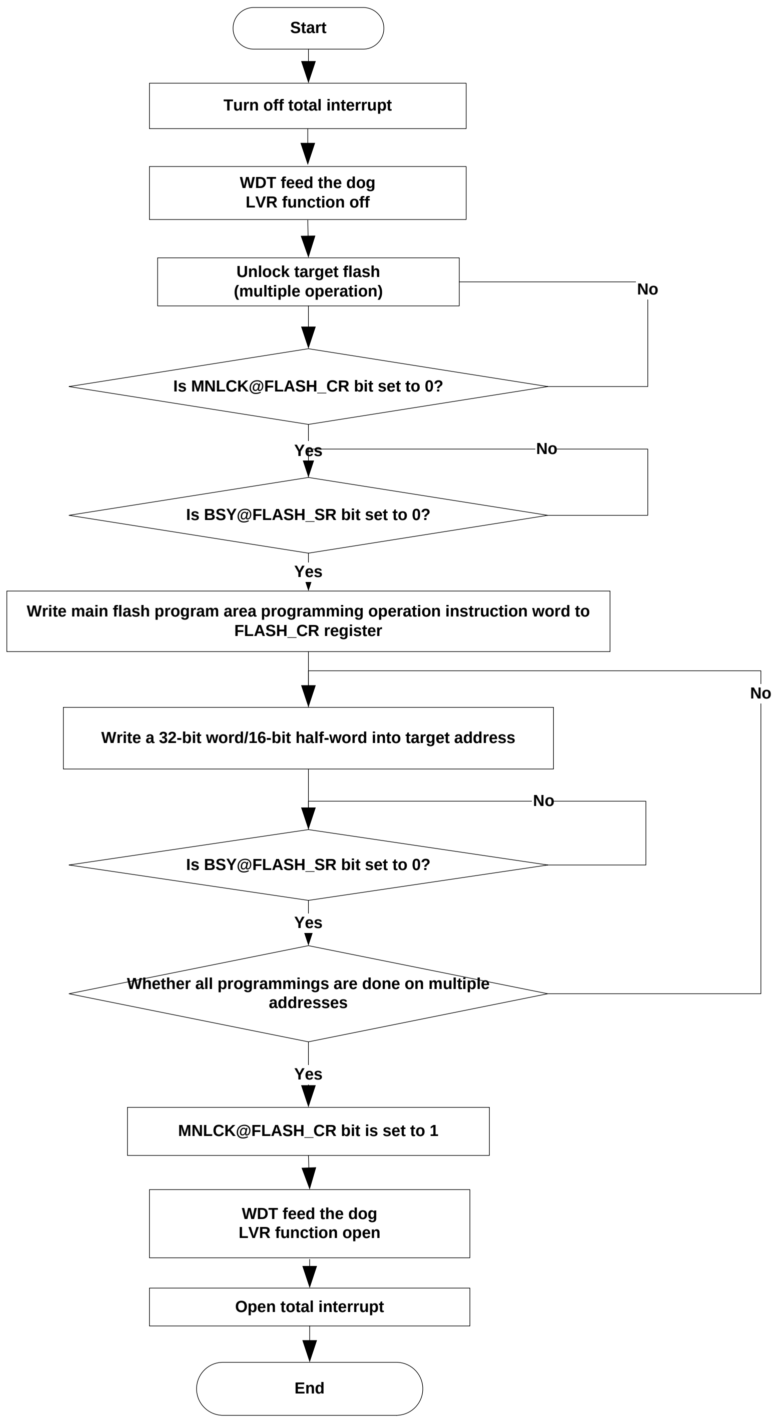
(7) Check the bit BSY@FLASH_SR to judge whether the programming instruction has been executed;

(8) Repeat (6)-(7) until the multi-address programming ends;

(9) The MNLCK@FLASH_CR bit is set to 1, and the operation is completed;

(10) Feed the WDT to the dog, and the LVR function is turned on;

(11) Turn on the total interrupt.


Start
Turn off total interrupt
WDT feed the dog
LVR function off
Unlock target flash
(multiple operation)
No
Is MNLCK@FLASH_CR bit set to 0?
Yes
No
Is BSY@FLASH_SR bit set to 0?
Yes
Write main flash program area programming operation instruction word to FLASH_CR register
No
Write a 32-bit word/16-bit half-word into target address
No
Is BSY@FLASH_SR bit set to 0?
Yes
Whether all programmings are done on multiple addresses
Yes
MNLCK@FLASH_CR bit is set to 1
WDT feed the dog
LVR function open
Open total interrupt
End

Main program area programming (continuously write multi-word programming)

For Flash memory, 16-bit or 32-bit data can be written each time. After the erase operation of the main storage area, each 16-bit or 32-bit can only be programmed once, that is, if one of the programmed 16-bit or 32-bit data is not 0, it cannot be programmed again, otherwise the programming error flag Bit PGERR@FLASH_SR is set to 1.

If a sector in the Flash main program area has been protected by the write protection bit, but the sector is still being written, the write protection error flag bit WRPRTERR @FLASH_SR is set to 1.

5.3.4. *EEPROM-like memory area erase programming*

The EEPROM-like storage area erasing programming is similar to the main program area erasing and programming by sector, the difference is that the Flash unlock register is different, the operation command word written into the FLASH_CR register is different, and the flag bit of the locked storage area is different.

Note: Before erasing and programming the EEPROM-like storage area, please feed the WDT to the dog, and the LVR function is turned off to avoid reset during the operation.

5.3.5. *OTP area programming*

The OTP area cannot be erased. The programming of the OTP area is similar to the programming of the main program area. The difference is that the Flash unlock register is different, the operation command word written into the FLASH_CR register is different, and the flag bit of the locked storage area is different.

Note: Before erasing and programming the OTP area, please feed the WDT to the dog, and the LVR function is turned off to avoid reset during the operation.

5.4. Anti-interference measures for programming/erasing

Since the program code includes programs for erasing and programming functions in the main program area or E2PROM area, when the system is strongly disturbed, the erasing and programming operation has certain risks. For example, when doing an ESD/EFT test, the program may run away. If it runs directly from the outside to the place where the erased code is written, it may rewrite or erase the non-program intended, causing the system to run abnormally.

For this reason, it is recommended to add anti-interference measures when operating the Flash main program area, EEPROM area or other areas:

(1) Since only when the start flag STRT@FLASH_CR is set to 1, the erasing/programming of Flash will be started or waited for, so add a software flag at the beginning of the operation, and then set the start flag STRT@FLASH_CR before Check whether the software flag matches. If the flag bit does not match, the operation will exit directly, so the programming or erasing action will not be started.

(2) To make full use of the function of the Flash operation timer, you can start the Flash operation timer at the beginning of the operation, calculate and configure the window time for the allowed operation, so that the program runs normally until the erasing or programming operation is performed, and the Flash operation is timed The count value of the counter is just within the executable window range, which can avoid misoperation caused by program runaway.

(3) Try to avoid calling the programming and erasing subfunctions in the main program area or EEPROM area with parameters. Because when the program runs away, the parameters are random values, which are easy to misuse. Therefore, it is recommended that the main program area or EEPROM area erase sub-function, without parameters, assign a constant value to the address in the function, and erase a fixed sector.

(4) When operating the Flash main program area, EEPROM area or other areas, put the

Flash unlock operation outside the erase and write functions.

【**Note**】 When writing to Flash (main program area, EEPROM area, OTP area or other areas), it can only be operated by word (32 bits) or half word (16 bits).

SH30F9/SB0 series chip .

6. CRC

6.1. Overview

For the convenience of data verification, The SH30F9/SB0 series chip provides CRC module to calculate cyclic redundancy code. This module provides 4 kinds of generating polynomials and various data pre- and post-processing to meet different requirements for CRC algorithms.

6.2. Programming Setting

6.2.1. *Generating polynomials*

Support 4 general generator polynomials: CRC-32 bits (0x04C11DB7), CRC-16 bits (0x8005), CRC-CCITT (0x1021), CRC-8 (0x7). Generate 32-bit, 16-bit, and 8-bit CRC codes respectively.

6.2.2. *Input data processing*

Support format conversion of input data and then perform CRC operation. The supported conversion methods are bitwise inversion, byte order inversion, and byte order inversion.

6.2.3. *Result processing*

Supports format conversion of CRC operation results. The supported conversion methods are bitwise inversion, byte order inversion, and byte order inversion.

6.2.4. *Operation trigger*

After the initial value is imported into the internal operation register through the reload control bit, as long as the value is written to the DR register, a CRC operation will be triggered.

6.2.5. *Result obtaining*

Read the DR register to get the CRC operation result without special waiting. If the operation is not completed, the read instruction will not be executed.

【**Note**】 Reload the initial value before starting the continuous operation. Otherwise, the initial value of the internal register is unknown, which will affect the calculation result. In the same way, the initial value cannot be reloaded during continuous operation, otherwise the intermediate result will be changed and the final result will be affected

【**Note**】 The Reload function is to load the initial register directly into the internal register without any format conversion. When the CRC value of a certain area is calculated by dividing into blocks, if there is a format conversion process for the output result at this time, the result obtained is not the value of the internal register. Therefore, when performing the next block operation, the result obtained last time cannot be directly loaded into the internal register, but the correct intermediate result needs to be obtained through the inverse operation of the format conversion and then loaded into the internal register. The simple method is: do not set the result format processing for the first few blocks, and then set the result format processing after the calculation reaches the last block.

【**Note**】 When writing data to the DR register for calculation, it supports 8-bit, 16-bit, and 32-bit writing. However, it is required that the register address must be word-aligned when writing/reading (4-byte alignment, that is, it must be the first address of the DR register). Otherwise the result is indeterminate.

【**Note**】 When the generator polynomial is 32 bits, on the premise that the format of the input data is not converted, the result of writing 32-bit data once for calculation is the same as writing 8-bit data four times (low byte first) .

【**Note**】 When the generator polynomial is 8 bits, if one word is written at a time, only one byte is calculated, and the high bits are ignored. Similarly, when the generator polynomial is 16 bits, if one word is written at a time, only one half word is calculated, and the high bits are ignored.

7. WDT

7.1. HWDT

7.1.1. *Overview*

The independent watchdog of SH30F9/SB0 series chip has a 16-bit down counter, using the built-in 128kHz RC as the clock source.
The independent watchdog cannot be turned off after power on.
HWDT can work in shutdown mode, so it can be used as a wake-up timer. By setting HWDTPD@HWDT_CR, you can choose to turn on or off the watchdog when the shutdown is disabled (the default shutdown mode is off). This bit cannot be changed after 1, only after Reset Change.
When the power is turned on, the watchdog works with a longer overflow time, and the overflow time is about 4096*32/128 = 1024ms.

7.1.2. *HWDT Programming Setting*

Before the timer overflows, write 0xAAAA to the feed dog register IWDT_CLR to update the counter to restart counting to avoid overflow.
When the timer overflows, the chip will be reset and the HWDTRSTF@RCC_RSTSTR flag will be set.

7.1.3. *HWDT Application examples*

```
//Configure independent watchdog overflow time, T= 2.048s
HWDT->CR.V32 = ((0X5AA5<<HWDT_CR_LOCK_Pos)|0x00);
HWDT->CLR =0xAAAA; //feed the dog
```

7.2. WWDT

7.2.1. *Overview*

Window watchdogs are usually used to monitor software failures caused by external disturbances or unforeseen logic conditions that cause applications to deviate from the normal operating sequence. The window watchdog is an 8-bit decrement counter. The dog feeding interval is a window area. If feeding the dog earlier than the window feeding area, an advanced exception event will be generated, and if it is later than the window feeding dog area, a delayed exception event will be generated. Both of them will cause a reset. . In addition, when the counter reaches the lower boundary of the window dog feeding area, it is the latest dog feeding time, and WWDT interrupt can be applied for, which can be used as the last remedial time for window dog feeding.

7.2.2. *WWDT Programming Setting*

If RL represents the reload value of the window watchdog, WT represents the upper limit value of the window watchdog. If RL>=WT is set, it is a normal window monitoring function with advanced abnormal monitoring, that is, the window watchdog can feed the dog only when the counter of the window watchdog is decremented below the upper limit value of the dog feeding, otherwise it will be reset. If RL<WT is set, there is no advanced abnormality monitoring, only delayed abnormality can be monitored. If the delayed feeding interrupt is enabled, when the counter reaches the lower boundary of the window feeding area, it is the latest feeding time, and the WWDT interrupt can be applied for, which can be used as the last remedial moment of the window feeding; if the delayed feeding interrupt is disabled, the counter When it reaches the lower boundary of the dog feeding area of the window, it is found to be reset directly. Therefore, the setting of dog feeding time and window watchdog must be calculated accurately according to the program.

Note: If RL=WT=0xFF is set, the window is the largest, without advanced monitoring, and the function is similar to ordinary watchdog.

7.2.3. *WWDT application examples*

```
//Configure window watchdog overflow time

WWDT->CR.V32 = ((0x5AA5 << WWDT_CR_LOCK_Pos) | //Register unlock bit
                        (1 << WWDT_CR_WWDTON_Pos ) | // Start WWDT
                        (1 << WWDT_CR_WWDTIE_Pos ) | //Enable the latest dog feeding
interrupt of the WWDT
                        (WWDT_Prescaler_32 << WWDT_CR_WWDTPR_Pos) |//Clock
frequency division
                        (0xFF<< WWDT_CR_WWDTRLR_Pos) ); //Reload value

//Configure the upper limit of the WWDT
WWDT->WTR.V32 = ((0x5AA5 << WWDT_WTR_LOCK_Pos) | //Register unlock value
                         (0x7F<< WWDT_WTR_WWDTWTR_Pos) ); //The upper limit of the
```

feeding window

... (user code)
WWDT->CLR =0x5555; //feed the dog

8. EXTI

8.1. Overview

The external interrupt is composed of 16 edge detectors that generate event/interrupt requests, which can be used to detect the edge signal and level signal of the external input. All GPIO pins on the chip can be configured as external interrupt lines, and each interrupt line can be independently configured with an input type, and can also be independently shielded.

8.2. Programming Setting

8.2.1. *Edge trigger mode selection*

The FTSR and RTSR registers select whether the corresponding external interrupt line can be triggered by an external signal. For example, bit0 of RTSR is 1, indicating that the external interrupt line EXTI0 can be triggered by an external high level/rising edge, and other bits are similar. Once the corresponding bits of FTSR and RTSR are set to 1, as long as there is an external trigger signal, the corresponding bit of the PR register will be set to 1, even if the enable bit such as IMR/EMR/DMR is 0.

8.2.2. *Software triggered interrupt*

The register SWIER is a software trigger register. Writing 1 to the corresponding bit of SWIER is equivalent to an external signal trigger on the corresponding external interrupt line. It is often used when there is no external trigger signal, but the program needs to enter the external interrupt function to execute. This bit will be automatically cleared after writing 1, and the software does not need to operate.

8.2.3. *External interrupt line selection*

Each GPIO pin can be configured as an external interrupt line, so multiple GPIO pins must share an external interrupt line. For example, PIN0 of GPIOA/GPIOB/GPIOC/GPIOD is connected to EXTI0, and so on, EXTI0 ~15 are respectively connected to PIN0~15 of different ports, and which GPIO port is selected by the specific EXTI line is determined by the CFGL and CFGH registers in the EXTI module. The following figure lists the value of the EXTI0 bit field:

EXTI0[2:0]	Description
000	PA0 is connected to EXTI0
001	PB0 is connected to EXTI0

010	PC0 is connected to EXTI0
011	PD0 is connected to EXTI0

Therefore, in practical applications, try to avoid using the ports of the same external interrupt line such as PA0/PB0...PD0 at the same time. If there are three buttons that need to be connected to external interrupt lines, it is recommended to connect them to PA1, PB2, PA3 and other pins that occupy different EXTI ports, instead of PA0, PB0, and PC0, otherwise it needs to be screened in the program .

8.2.4. *Matters needing attention*

(1) Since the registers that control trigger interrupts, trigger events are independent of each other, the same external pulse can trigger the above twosituations at the same time.

(2) Since the Cortex-M0+ core uses a 2-stage pipeline, it is necessary to pay attention to the local execution timing. There will be this requirement in the interrupt service program. Specifically, do not put the clear interrupt flag in the last instruction before the interrupt return, otherwise it may cause a secondary response to the interrupt. You can also use a pipeline to isolate the instruction (such as "DSB") to ensure that the interrupt flag is cleared before the interrupt returns. For details, see the "routine" description.

9. System configuration module (SYSCFG)

9.1. Overview

The system configuration module is mainly used to set some system parameters, such as SRAM lock, DEBUG environment, etc.

9.2. Programming Setting

9.2.1. *BOD*

The BODMD bit field can set the mode of detecting V_{DD}. When BODMD=10b, it can detect the rise and fall of V_{DD}. When V_{DD} rises, the BODF bit is 0, and when V_{DD} drops, BODF is 1.

9.2.2. *LVR*

VLVR[1:0] can select three LVR voltages, and the value of this LVR related register remains unchanged after reset.

9.2.3. *NMI interrupt switch*

The NMI interrupt is a non-maskable interrupt with the highest priority in the entire chip, and the three interrupts CSM, EXTI0, and BOD can be connected to the NMI interrupt by enabling the corresponding bit segment of the SAFR register.

Once the above three interrupts are connected to the NMI interrupt, when an interrupt occurs, it will directly enter the NMI interrupt service function. Although the original interrupt can also be triggered, but because the priority is lower than the NMI interrupt, the service function of the original interrupt will not be implemented, which requires special attention during the application design process.

9.2.4. *Crystal oscillator pins are used as GPIO*

The crystal oscillator pins XTAL1/XTAL2 default input and output are all off. If you want to use an external crystal oscillator, you need to set the OSCCFG bit of the SAFR register, and it cannot be modified until the next reset.

OSCCFG[1:0]	Description
00	XTAL1 and XTAL2 input and output are all closed (Default)
01	XTAL1 and XTAL2 are used as external oscillator interface (Crystal oscillator and Ceramic oscillator)
10	XTAL1 is used as an external clock source input, and XTAL2 is used as a GPIO
11	XTAL1 and XTAL2 input and output are all closed (Default)

9.2.5. *Simulation pins are used as GPIO*

When a large number of IO pins need to be used in an application, the emulation pin can be used as a normal GPIO by setting the SWJCFG bit field of SAFR.

When SWJCFG!=101b, SW-DP is off, but there are special cases for SWDIO and SWCLK. That is, when the chip is in the normal operation mode, the program not sets SWJCFG to 101b, then all the emulation pins are regarded as GPIO pins at this time, and cannot be modified until the next RESET, and cannot enter the emulation mode. When the chip is single-step running in the emulation mode, even when it runs to the statement of setting SWJCFG!=101b, SWDIO and SWCLK are still used as emulation pins, and the emulation mode will not be forced to exit.

9.2.6. *The peripherals run in debug mode*

When the chip is in the emulation mode, the peripherals will be in a stop state under normal circumstances, such as the TIM counter will not continue to count, etc., by setting the corresponding bit of the DBGCR register, the peripheral can continue to run, such as setting BIT9 of the register to 1 Then the TIM can run normally.

The upper 16 bits of the DBGCR register are unlock bits, which must be 0x5AA5. When setting, 32 bits of data need to be written to the register at the same time to take effect.

【Note】 When debugging, it is forbidden to configure the emulation port as an open-drain function, otherwise it will affect the emulation debugging. In actual project application, if the emulation port is reused as TWI, it is recommended to close the TWI initialization (IO, TWI module) in the emulation mode.

9.2.7. *Low-power consumption mode*

There are two low-power modes, sleep mode and stop mode. The difference is that the latter not only stops the core, but also stops the clocks of the peripherals, which is equivalent to deep sleep.

In terms of software processing, the stop mode needs to set the bit SLEEPDEEP of the core system control register (0XE000ED10) to 1, and then execute the WFE or WFI instruction.

An example of configuring sleep mode is as follows:

Turn off peripherals that do not need to run in sleep mode

Configure wake-up sources, such as external interrupts or other interrupts that can run in sleep mode

Use the command WFI or WFE to enter sleep mode (if WFE is used and there is an interrupt before, it is recommended to call it twice)

Wait for the MCU to wake up

An example of configuring stop mode is as follows:

Configure wake-up sources, such as external interrupts, etc. (most peripheral interrupts in stop mode cannot wake up the MCU)

Turn off the clock source that does not use peripherals, and the analog module also needs to turn off the enable bit of the module

Unused GPIO pins can be configured as input and output all off or analog input to save power consumption

Set the bit SLEEPDEEP of the system control register to 1

Call the command WFI or WFE to enter stop mode (if WFE is used and an interrupt has been generated before, it is recommended to call it twice)

Wait for the MCU to wake up

Among the above two methods, the stop mode has the lowest power consumption. After entering this mode, all peripheral clocks are turned off, so ordinary peripheral interrupts cannot wake up the MCU. For example, the TIM using the APB0 clock, because the clock is turned off , the counter has been unable to overflow and cannot generate interrupts, so it cannot be used as an interrupt source.

WFE and WFI are two ways for the kernel to enter low power consumption. WFE is event wake-up, and WFI is interrupt wake-up. Usually, the interrupt that can wake up WFI can wake up WFE, and vice versa.

【Note】 When the system wakes up from PD, the system clock runs at internal 128K and needs to be manually switched back to the clock before entering PD.

10. PCA

10.1. Overview

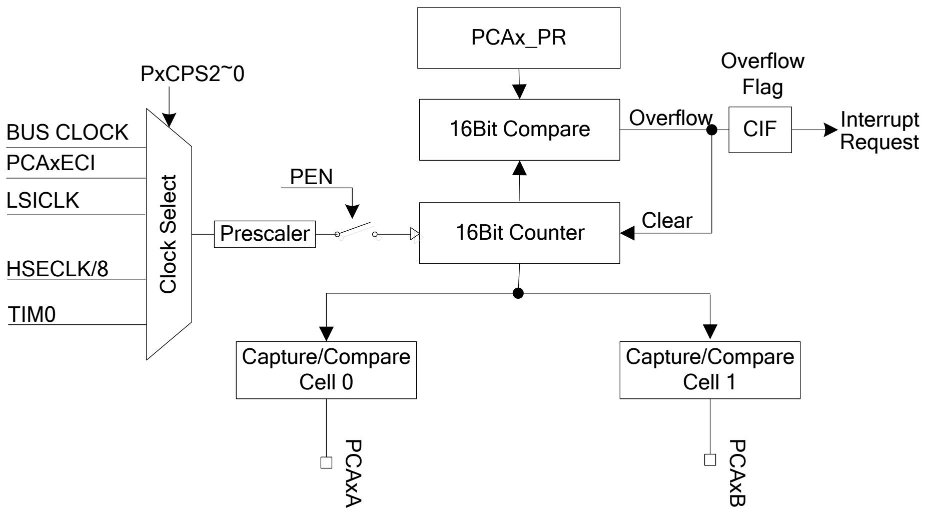
The programmable counter array PCA provides enhanced timer functions, and can provide four enhanced functions of input capture, compare match (output), frequency adjustable square wave output, and PWM modulation output. SH30F9/SB0 series chip has 4 built-in PCA modules (PCA0/1/2/3), and each PCA module provides 2 input and output channels.

The PCA characteristics of SH30F9/SB0 series chip are:

- 16-bit timer
- 2 independent input and output channels
- Support input capture, compare match (output)
- Support square wave output with adjustable frequency
- 16/8 bit PWM, 4 output modes
- Two pairs of PCA0/1 and PCA2/3 can be cascaded into a 32-bit PCA module

The programmable counter array PCA consists of a basic counting unit and two 16-bit capture/compare modules. Each capture/compare module has independent pins (PCAxA/PCAxB), which can be used as capture signal input or waveform output.

Note: The small subscript x is the PCA serial number, such as PCAx (x=0, 1, 2, 3), PCAx will be used uniformly below, and the value of x will not be explained.



PCA block diagram

PCAx consists of a 16-bit period register PCAx_PR, a 16-bit count register PCAx_CNT and a 16-bit prescaler register PCAx_PSC to form a basic counting/timing unit.

The 16-bit counter has two counting modes, counting up and center-aligned counting. When the SDEN bit in the register PCAx_CFGR is 0, the counter works in the up-counting mode, and when SDEN is 1, the counter works in the center-aligned counting mode.

In the up-counting mode, the counter counts up from 0 to the value of the period register PCAx_PR, then starts counting from 0 again and generates a counter overflow event, and the CIF bit in the status register PCAx_SR will be set to 1 by hardware. An interrupt is generated if the CIE bit in register PCAx_CFGR is set to 1 by software.

In the center-aligned counting mode, the counter counts up from 0 to the value of the period register PCAx_PR, and then counts down to 1 to complete a cycle, and then starts counting from 0 again in the next cycle, and continues to cycle. When the counter value is the value of the period register PR, a period match event will be generated, and the PIF bit in the status register PCAx_SR will be set to 1 by hardware. If the PIE in the CFGR register is 1, an interrupt will be triggered. When the counter value counts to 0, a counter overflow event will be generated, and the CIF bit in the status register PCAx_SR will be set to 1 by hardware. If the CIE bit in register CFGR is set to 1 by software, an interrupt will be triggered.

Note: From the shape of the generated waveform, the up-counting mode can also be called the sawtooth wave mode, and the center-aligned counting mode can also be called the triangle wave mode.

10.2. Programming Setting

10.2.1. *IO setting*

When using the relevant input and output functions of PCA, the IO multiplexing (AF) function must be set first, that is, the multiplexing register (GPIOx->AFRH/L) corresponding to the IO must be set as the PCA-related AF function.

10.2.2. *Clock setting*

The clock source of the counter can be selected by setting CPS[2:0] in the register PCAx_CFGR, which are bus clock, PCAxECI pin clock input, built-in 128KHz RC (LSICLK), external high frequency Oscillator/8 (HSECLK/8) and TIM0 overflow, as shown in the table below.

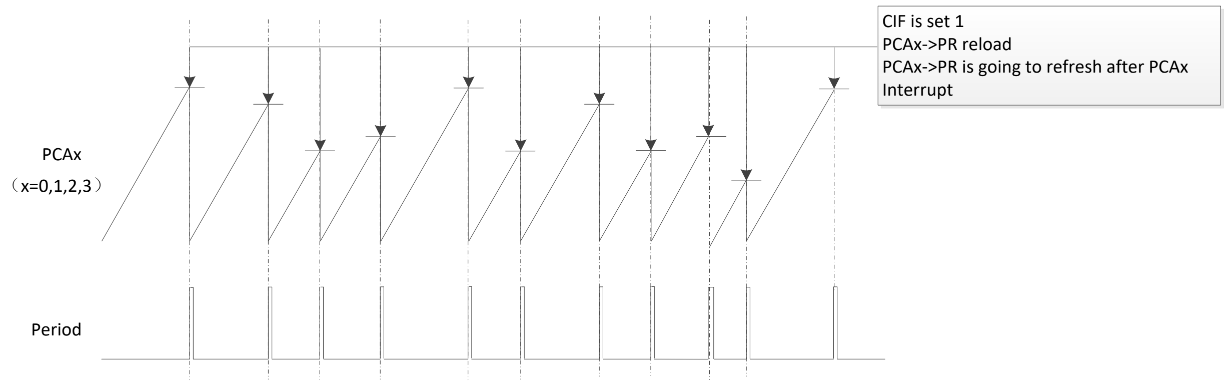
CPS[2]	CPS[1]	CPS[0]	
0	0	0	Bus clock
0	0	1	External pin input PCAxECI clock falling edge
0	1	0	LSICLK (RC128K)
0	1	1	HSECLK/8 (Crystal/Ceramic divide by 8)
1	0	0	TIM0 overflow
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

When CPS[2:0]=001, the external ECI pin input is selected as the clock source, and the filter parameters of the input clock source can be selected by setting the ECF[1:0] bit field in the CFGR register. The filter parameters are no filter, 8 times the bus clock, 16 times the bus clock, and 32 times the bus clock.

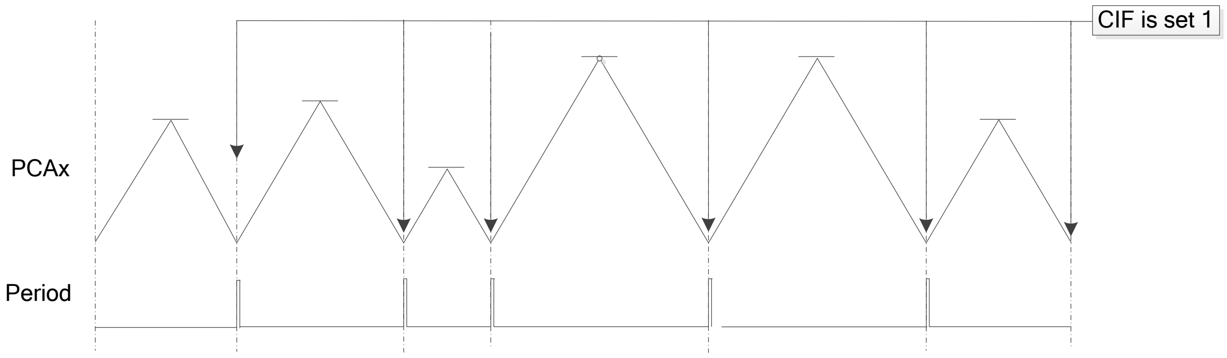
Note: The bus clock cycle must not be lower than 4 times the count clock cycle (except when the count clock is the bus clock), otherwise the PCAx counter will not count correctly.

10.2.3. *PCA count mode setting*

The PCAx counter/timer has a control bit that can select the counting mode. SDEN in the PCAx->CFGR register is 0 to indicate the sawtooth wave mode, and SDEN is 1 to indicate the triangle wave mode. The timing waveforms of sawtooth and triangle waves are as follows:



Waveform diagram of PCAx counter in up-counting (sawtooth) mode



Waveform diagram of PCAx counter in center-aligned counting (Triangle Wave) mode

10.2.4. *PCA mode setting*

PCAx counters can be mounted with 2-way capture/compare modules to achieve enhanced functions. Each capture/compare module can be configured to work independently (Note: 1 capture/compare module corresponds to 1 channel), each module has its own control registers in the system controller, these registers are used to configure the module work and exchange data with the module. You can enable the module to work in one of the following four working modes by configuring the two bits of SM[1:0] in the PCAx->CCMR register of each module: input capture, compare match output, frequency output, and PWM output mode. For details, see the description in the PCA chapter of the specification.

10.2.5. *Matters needing attention*

（**1**） Changing the PCAx->PR value must ensure that the new PCAx->PR value is not less than the value of all compare registers.

（2） When the comparison/capture module is used as a PWM output function, if PCAx->CCR is equal to 0x0000, the output will always be kept at low level; if PCAx->CCR is equal to PCAx->PR, the output will be kept at high level. Inverting the pin results in the opposite output.

（3） All compare/capture modules of PCAx can only work in the same counting mode (for example: compare/capture module 0 and compare/capture module 1 of PCA0 can only work in the same counting mode).

（4） After cascading, the PR value of PCA1 and CCR0/CCR1 are automatically mapped to the upper 16 bits of the corresponding register of PCA0, so reading only needs to read the 32-bit register of PCA0, and the rest of the PCA cascading is the same.

11. PWM

11.1. Overview

The SH30F9/SB0 series chip integrates four 16-bit PWM modules. The PWM module can generate pulse width modulation waveforms with adjustable period and duty cycle.

The PWM timer also provides 4 interrupt sources for PWMx (x = 0,1,2,3), which will generate an interrupt every PWM cycle. In this way, the user can change the period or duty cycle of the next cycle in each PWM cycle.

Has the following characteristics:

- 4×2 complementary outputs with dead zone control
- Provide every PWMx (x = 0,1,2,3) cycle overflow interrupt
- Two output polarities can be selected independently
- Provide error detection function to shut down PWM output in an emergency
- Provide protection registers to protect important registers from interference errors

11.2. Programming Setting

11.2.1. *IO setting*

When using PWM related output functions, you must first set the IO multiplexing (AF) function, that is, set the corresponding IO multiplexing register (GPIOx->AFRH/L) to the corresponding AF. For specific multiplexing information, please refer to Refer to "Multiple Function Mapping Table".

11.2.2. *Protection setting*

The PWM module is equipped with protection registers. When you need to set PWM-related registers, you need to write 0x5AA5 to PWMx->PWMLOCK to unlock, and then operate other registers. When PWMx->PWMLOCK writes a value other than 0x5AA5, operate other registers. Invalid, this can effectively protect the PWM output from interference.

11.2.3. *Module setting*

PWM programming is divided into the following steps:

（**1**） Unlock the PWM module.

（**2**） Select the PWM module clock source.

（**3**） Set the PWM period/duty cycle by writing an appropriate value to the PWM Period Control Register (PWMPR) or PWM Duty Cycle Register (PWMDR).

（**4**） Select the PWMx output mode (active high or active low) by setting the PWMSx bit of the PWMx->CR register.

（**5**） Set the PWMEN enable module output of the PWMx->CR register

（**6**） If the PWM cycle or duty cycle needs to be changed, the operation process is as described in step 3 or step 4. The modified value of the reload counter is valid from the next cycle onwards.

（**7**） Lock the PWM module.

11.2.4. *Matters needing attention*

（**1**） When PWMx output is off (FLT occurs), the output of PWM can be cut off and enter an invalid driving state (which can be set to high level, low level, and high resistance state).

（**2**） Once the high/low level of the FLTx pin is detected, the internal state will be maintained and the PWMx output will be turned off.

（**3**） When the FLTx input signal is valid, the FLTS bit cannot be cleared. The FLTS status bit can only be cleared after the FLTx input signal disappears.

（**4**） When PP[15:0] = 0000H, if PWMSA=0, regardless of the PWMx duty cycle, PWMxA (x=0,1,2,3) outputs low level.
When PP[15:0] = 0000H, if PWMSA=1, regardless of the PWMx duty cycle, PWMxA (x=0,1,2,3) outputs high level.

（**5**） When PD[15:0] = 0000H, if PWMSA=0, regardless of the PWMx period, PWMxA (x=0,1,2,3) outputs low level.
When PD[15:0] = 0000H, if PWMSA=1, regardless of the PWMx period, PWMxA (x=0,1,2,3) outputs high level.
When PP[15:0] ≠ 0000H and PP[15:0] ⩽ PD[15:0], if PWMSA=0, PWMxA (x=0,1,2,3) output high level.
When PP[15:0] ≠ 0000H and PP[15:0] ⩽ PD[15:0], if PWMSA=1, PWMxA (x=0,1,2,3) output low level.

12. ADC

12.1. Overview

The SH30F9/SB0 series chip is equipped with a 12 bit A/D converter using successive approximation method. The conversion rate of AN4~AN5 channels can reach up to 1MSPS, while the conversion rate of other channels can reach up to 100KSPS. The reference voltage defaults to AVDD, with a total of 30 ADC analog input channels, supporting three conversion modes: sequence conversion, intermittent conversion, and continuous conversion, with comparison function. Not only does it support software triggered AD conversion, but it also supports 4 PWM modules (PWM0/1/2/3), 4 PCA timers (PCA0/1/2/3), a regular timer TIM2, and external interrupts (EXTI0/1/2/3) to trigger AD conversion.

12.2. Programming Setting

12.2.1. *ADC clock setting*

After the system clock is set, enable the ADC module clock control bit.

12.2.2. *ADC channel IO port function setting*

Enable the analog multiplexing function of the ADC channel corresponding to the IO port.

12.2.3. *ADC sampling rate configuration*

ADC clock and sampling time can be set through the register ADCON2. Set ADC clock by TADC[3:0]@ADCON2. Set the sampling time tSAMP of each channel through registers TS[3:0]@ADCON2, TGAP[2:0]@ADCON2 and GAPENx@ADGAPON, tSAMP = (TS[3:0]+ TGAP[2:0]* GAPEN +1) * tAD, where TGAP[2:0] GAPEN is the time interval setting bit field between adjacent channels in the sequence, and GAPEN is the GAP time enable control bit of ADC channel y.

For 12BIT mode AD conversion, the AD conversion time of each channel is fixed at 15 * tAD. So total sample conversion time per channel = tSAMP + 15 * tAD. For 10BIT mode AD conversion, the AD conversion time of each channel is fixed at 13 * tAD. So total sample conversion time per channel = tSAMP + 13 * tAD.

12.2.4. *ADC conversion mode configuration*

12.2.4.1. Single Conversion

ADCTU[1:0] = 00, the ADC is configured as a single sequence conversion mode, converting one sequence at a time. A sequence consists of a single channel or multiple channels. To convert a sequence means to convert the channels in the sequence one by one. In hardware, multiple signals can be converted at the same time point (the shortest sampling interval between 2 channels is 0.5us, which can be regarded as approximately at the same time).

The conversion result is stored in the corresponding result register. When the sequence conversion is completed, compare the value of a specified result register with the values of ADDGT and ADDLT, and indicate the result of the comparison with a flag.

In the single-sequence conversion mode, after ADSOC is set, it starts to convert from channel 0. After each single-channel conversion, this register is automatically incremented by 1, and then the next channel conversion is performed. When the number of single-sequence conversion channels reaches the set value value (ADMAXCH[2:0]), this sequence of conversion ends, and the ADSOC bit is cleared, indicating that this sequence of conversion is completed, and the ADPCH register is automatically reset to 0; at the same time, the ADIF bit in ADINTF will be set by hardware. At this time If the ADIE bit in ADCON1 is 1 (note: the interrupt of the ADC should not be masked), the sequence conversion completion interrupt will be triggered, and the ADIFC flag needs to be cleared by software.

12.2.4.2. Intermittent Conversion

ADCTU[1:0] = 01, then the ADC works in the discontinuous conversion mode, and the ADC starts converting directly from the channel specified by the ADPCH register. When the number of conversions reaches the set value (ADMAXCH[2:0]), this The sequence conversion ends. The maximum number of the ADC sampling channel is 7, so when the conversion channel pointer register ADPCH exceeds 7, it will automatically return to 0. After a sequence conversion ends, if the ADPCH software is rewritten to 0 at this time, the next conversion will resume from channel 0; if the ADPCH is not rewritten, the next conversion channel will start from the value of ADPCH, if the preset analog channel of the channel is not After setting, you may get unexpected results, so the program should be careful when rewriting this register (During ADC conversion (ADSOC = 1), ADPCH cannot be modified).

12.2.4.3. Continuous Conversion

ADCTU[1:0] = 1x selects the continuous conversion mode, after converting a sequence, the next conversion of this sequence will be performed. Some register settings of continuous conversion can refer to "single conversion mode". The difference from sequence conversion is that this mode will cycle conversion sequence, and the time from the completion of one sequence conversion to the start of the next sequence conversion is also controlled by TGAP[3:0] . When each sequence conversion is completed, compare the value of a specified result register with the values of ADDGT and ADDLT, and indicate the result of the comparison with a flag. When the software clears the ADSOC bit, the AD conversion stops immediately.

12.3. Matters needing attention

In order to ensure that the AD conversion can convert an accurate AD result under a certain conversion rate, it is necessary to calculate whether the equivalent input impedance matches according to the actual application situation, otherwise it will not be possible to obtain a conversion result with an accuracy error within 0.1LSB.

Before ADC conversion, you need to turn on ADON@ADCON1 for 10us and then set the ADSOC@ADCON1 bit to 1, because the ADC module needs a period of time to warm up to convert after it is turned on.

For the channel used for AD conversion, the IO port function must be multiplexed as an analog function, otherwise the correct conversion result cannot be obtained.

After the ADC conversion is enabled, the channel parameters cannot be changed, including core channel parameters such as TGAP, GAPENx, TS, ADMAXCH, TADC, ADPCH, SEQCHx, etc., and any modification will be considered invalid.

The same channel number can also be set in SEQCHx, for example, if all the values in SEQCHx are set to AN3, the result register will store the sampling values of AN3 in different time periods.

When ADIE, ADLIE, and ADGIE are all enabled, any one of ADIF, ADLIF, and ADGIF can cause an interrupt, and share an interrupt vector. By judging which of ADIF, ADLIF, and ADGIF is 1, determine the specific interrupt source and Customize specific actions.

The time interval between adjacent channels in the AD conversion sequence can be used to increase the sampling time of the ADC. The Gap Time of each channel can be set independently, so the Gap Time can also be used as the sampling time of each channel. The Gap Time of all channels can only be set to one value, but each channel can be individually set whether to enable Gap Time. The first channel can also set the Gap Time, which can also be used as the sampling time.

During any hardware trigger single-shot sequence, intermittent conversion and continuous conversion, if there is another hardware trigger, the last hardware trigger request will be ignored.

When the comparison function is applied, the written values of ADDGT and ADDLT take effect immediately, and the latest updated values of ADDGT and ADDLT will be used when comparing.

13. TIMER

13.1. Overview

There are three basic 16-bit timers and one basic 32-bit timer inside SH30F9/SB0 series chip.Each timer can run with the bus clock, but also with the internal low frequency RC (128KHz LSI/4).

Each timer can output a square wave with adjustable frequency through the actual GPIO pin, or drive the counting unit of the timer from an external input clock source through the GPIO pin.

13.2. Programming Setting

13.2.1. *Basic timing application*

The basic timer can be configured as a simple timing function, and is mostly used for system time base or software timing in practical applications.

13.2.2. *Square wave with adjustable output frequency*

By setting the GPIO pin to be used as a TIM function, and then enabling the TC bit of the CR register, the GPIO pin can be flipped according to the overflow time of the timer to generate a square wave. To change the frequency, you only need to modify the overflow time of the timer.

13.2.3. *Matters needing attention*

1) The overflow flag bit is generated when the counter (TCNT) changes from the value in the period register (TPR) to 0, so the actual number of clocks in one cycle is TPR+1, that is, when the application requires a cycle of 100 , TPR should be set to 99.

2) If the initial value of the counter TCNT is greater than the value of the period TPR, then after the timer is enabled, the counter will run up to 0xFFFF(TIM3 will run to 0xFFFFFFFF) and then overflow to 0, and will cycle between 0 and TPR in the next period.
3) The counter register, prescaler register and period register are all modified when the timer is stopped.
4) When selecting the external clock for Timer, it is necessary to ensure that the frequency of the bus clock is at least three times that of the external clock
5) The usage method of TIM3 is the same as that of TIM0-2, the difference is that TIM3 is a 32-bit count, while TIM0-2 is a 16-bit count.
6) When the Timer clock source is selected as Tx port input, do not write to TCNT. If there is a need to clear TCNT and then count, it can be replaced by the difference between the two TCNT values before and after reading.

14. UART

14.1. Overview

The SH30F9/SB0 series chip provides 6 sets of UART. Support flexible IO function mapping.

Each group of UART has four working modes, comes with a baud rate generator, supports hardware parity check, and UART0~2 can be configured with FIFO, supporting data transmission and reception based on FIFO.

In addition, it also supports 1/2 stop bit can be set, and supports hardware generation and detection of LIN bus synchronous interruption. Enhancements include framing error detection and automatic address recognition.

14.2. Programming Setting

14.2.1. *Sending*

There are TDR and TDR shift two-level registers inside the UART transmission logic, so customers can write 2byte transmission data continuously for the first time. When the UART is turned on, the transmit interrupt flag TI is valid by default, and will not be automatically cleared until TDR and shift are filled with data, and then the interrupt TI will be generated when the TDR data is loaded into the shift register and becomes empty. The TC flag is set when the data in the TDR and shift registers are all empty, and the flag will be automatically cleared to 0 with the data written.

14.2.2. *Receiving*

There are RDR and RDR shift two-level registers inside the UART receiving circuit. The user needs to remove the data in time after receiving the data. Otherwise, when there is data in the RDR memory, the data in the shift will be flushed after receiving. Therefore, the user needs to remove the RDR data before the shift data reception is completed at the latest. The RI flag is generated after receiving a complete byte of data. After the data in RDR is read empty, RI will be automatically cleared to 0.

【Note】 After the UART single-byte data is received, it must be removed before the next single-byte data is received, otherwise data loss or receiving overflow will occur.

14.2.3. *IO setting*

Since the TXDx/RXDx pins are multiplexed with I/O functions. If the AF function selects UART, the RXDx pin is automatically set to UART input, but the internal pull-up resistor needs to be manually turned on in advance; the TXDx pin is automatically set to UART output, but the open-drain output or push-pull output needs to be initialized in advance. Note that if it is multiplexed with the emulation port, you need to close the emulation port in SYSCFG first, otherwise the AF configuration will not take effect.

14.2.4. *LIN mode*

In LIN mode, detection of discontinuous frame and receiving data cannot be carried out at the same time. The correct use process is to detect synchronous discontinuity first, then switch to receiving data mode under LBD interrupt, and then switch back to synchronous discontinuity detection mode after receiving.

【Note】 When detecting the synchronization interval in UART LIN bus mode, it is normal for the frame error flag to be set, and there is no need for special treatment, just clear it.

14.2.5. *Interrupt*

When using interrupt mode, it is recommended to enable TIE before sending data after initializing UART, otherwise, enable TIE before enabling TEN, the interrupt service program will first transfer 2byte data to TDR and shift, and send it out when TEN is enabled.

14.2.6. *Parity check*

When using the parity check function, its priority is the highest, and the multi-device communication and custom ninth digit functions will not be available.

14.2.7. *Error flag*

Send conflict, receive overflow, parity error, frame error and other error flags will be set when they occur again, but will not trigger an interrupt. If customers want to use it, they need to query and process it in the UART interrupt.

14.2.8. *Other notes*

In order to avoid misjudgment at the receiving end, in the UART transmission initialization program, it is recommended to set the I/O state of the corresponding TXD to output high level.

15. SPI

15.1. Overview

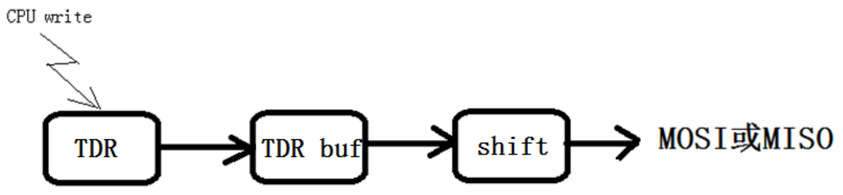
The SH30F9/SB0 series chip provides 2 sets of SPI. Support flexible IO function mapping.

Each SPI supports working in full-duplex mode, master/slave mode, programmable master clock frequency (up to bus frequency divided by 2), programmable polarity and phase of serial clock, and data word width per transmission 8, 16, and 32bit are optional, and the big and small endian modes are optional.SPI0 is configurable with FIFO and supports FIFO based data transmission and reception.

15.2. Programming Setting

15.2.1. *Sending*

The SPI transmission circuit contains TDR, TDR buffer and shift three-level registers (the latter two cannot be operated by the user); the user can write 2 data for the first time. When SPI is turned on, the transmit interrupt flag bit SPTI is set by default (indicating that TDR is empty), and it will not be cleared until TDR and TDR buffer are filled with data. The reset of SPTI occurs after the data in the TDR is loaded into the TDR buffer, indicating that one data can be written into the TDR again. The schematic diagram of the three-level register is as follows:



【Note】 The first data will be written through to TDR buf.

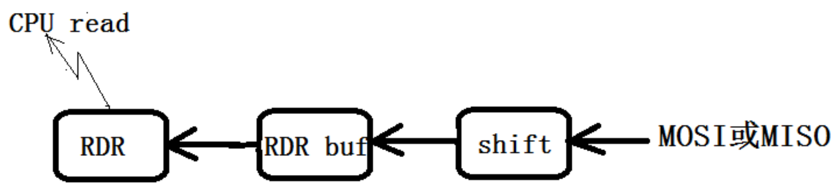
【Note】 SPI does not have an independent send enable switch. After the CPU writes a piece of data, the shift starts to send. There is no situation where the two-level registers are filled before sending. The process is always in the process of writing. in the process of sending.

【Note】 Writing to TDR will clear SPTI.

【Note】 It is not allowed to write TDR continuously, and only one data should be written each time.

15.2.2. *Reveiving*

There are RDR, RDR buffer and shift three-level registers inside the SPI receiving circuit. The user needs to remove the data in time after receiving the data. Otherwise, when there is data in the RDR and RDR buffer memory, the data in the shift is also full, and subsequent data will occur. Receive overflow event (RXOV), the subsequent data will be discarded. The user needs to remove the RDR data at the latest after the shift data is full and before new data arrives. The SPRI flag is set after the RDR stores 1 data (indicating that the RDR is full). After the data in the RDR is read empty, the SPRI will be automatically cleared to 0. If there is data in the lower buffer, the data will be automatically pushed to the RDR register, and Reset SPRI.



【**Note**】 shift is shared by sending and receiving, while buffer and data register (TDR/RDR) are two independent sets. The sharing of shift is clearly stated in the data sheet.

【**Note**】 Reading RDR means clearing SPRI.

15.2.3. *IO setting*

Since the SPI pins are multiplexed with I/O functions. If the AF function selects SPI, the signal input pin is automatically set to SPI input, but the internal pull-up resistor needs to be manually turned on in advance; the signal output pin is automatically set to SPI output, but the open-drain output or push-pull output needs to be initialized in advance .

15.2.4. *Fast slave mode*

When the SPI module selects the slave mode of CPHA=0, the $\overline{SS}$ pin must be used as a chip select signal, and it must be pulled low before sending 1 byte, and pulled high after sending 1 byte, and so on.

【**Note**】 The reason why the $\overline{SS}$ pin must be used is that when CPHA=0, the slave device has no clock edge to trigger the data loading operation, and can only use the $\overline{SS}$ signal line to complete the data loading. CPHA=1 does not have this problem, you can set SSDIS=1 to shield the $\overline{SS}$ signal.

In order to avoid the control of the $\overline{SS}$ pin from reducing communication efficiency, the slave fast mode (SPSFF) is introduced. You only need to pull the $\overline{SS}$ pin low when sending for the first time, and the subsequent data will be automatically sent by The SCK rear edge of the previous data is loaded, and the $\overline{SS}$ pin control is omitted.

【**Note**】 If it is full-duplex communication, for the slave, because it does not know when SCK will arrive, it is necessary to load a piece of data in TDR in advance, otherwise it will cause the delay of the first data from the slave to the master , it is often recommended to use CPHA=1 mode for such situations.

15.2.5. *Intterupt*

When using interrupt mode, SPTIE needs to be turned on after initializing SPI and before sending data. When a MODF error occurs, it is necessary to use software in the interrupt to switch to the slave and close the SPI. If the SPI needs to be switched to receive data from the slave, you need to re-enable the SPI. If the SPI still needs to send and receive data as a host, it needs to detect that the SS level is high before setting it as the host to restart the SPI communication.

15.2.6. *Error flag*

Sending conflicts and receiving overflow error flags will not trigger an interrupt. If necessary, the user needs to check and handle it in the program.

16. TWI

16.1. Overview

The TWI (Two-wire Serial Interface) interface is the inheritance and development of the I2C bus interface and is fully compatible with the I2C bus.

TWI is composed of a clock line and a data line, and is transmitted in units of bytes. It is compatible with the SMBus bus specification, automatically processes byte transmission, and tracks serial communication.

SCL/SDA is the signal line of TWI bus. SDA is a bidirectional data line and SCL is a clock line. To transmit data on the TWI bus, the highest bit (MSB) is first sent, the host sends a start signal, and then the host sends one or more bytes of data. After the data transmission is completed, the host sends a stop signal to complete the simplest TWI transmission.

16.2. TWI bus setting

16.2.1. *I/O setting*

The TWI bus is required to be configured as an open-drain output. Note that the external pull-up is limited to VDD or slightly higher than VDD when the open-drain output is used.

```
GPIOA -> AFRH.BIT.AFR14 = 4; // PA14 = AF4 ~ SDA0
GPIOA -> AFRH.BIT.AFR13 = 5; // PA13 = AF5 ~ SCL0
GPIOA ->OTYPER.BIT.OT14 = 1; // PA13/14: OD output
GPIOA->OTYPER.BIT.OT13 = 1;
GPIOA ->PUPDR.BIT.PHDR14 = 1; // PA13/14: pull high, use internal pull-up example here
GPIOA->PUPDR.BIT.PHDR13 = 1;
```

16.2.2. *Baud rate configuration*

```
// Prescaler, 0:64 frequency division, 1:16 frequency division, 2:4 frequency division, 3:1 frequency division

TWI->CR.BIT.CR = 3;

// Baud rate formula: Ftwi/(16+2*CR*BRT), when setting the baud rate to 100Kbps, Ftwi=24MHz, requires CR*BRT=112, take CR=1, BRT=112

TWI->BRT = 112;
```

16.2.3. *Communication address configuration*

// TWIAMR=0, do not enable shielding. ADDR[6:0]=0111011b, local machine address. GC=0b, disable responding to general address.

TWI->ADDR.V32 = 0x0076;

16.3. Four basic communication modes

The TWI communication of SH30F9/SB0 series chip is completed by the bottom drive circuit and application software based on interrupt. All bus events such as receiving a byte or sending a START condition generate an interrupt. So during the byte transfer, the application software can perform other operations.

Four basic operation modes of TWI:

- Host sending mode: the machine acts as the host, sending data
- Host receiving mode: the machine acts as the host to receive data
- Slave sending mode: This machine acts as a slave, sending data
- Slave receiving mode: the machine acts as a slave to receive data

All operations are combined from these four basic operating modes.

【Note】 The TWI bus does not perform sending and receiving operations during the TWINT setting period, and the status will not be updated. The status can only be updated after the flag is cleared.

【Note】 The setting of TWINT is only related to the state change of the local machine. It may be that the local machine has completed a certain transmission, or the local machine may have completed a certain reception, which will lead to a state change.

【Note】 In the initialization phase, AA=1 means setting as slave mode, and in the communication process, AA=1 means replying ACK signal.

【Note】 In the TWI bus, both the start condition and the stop condition are sent by the host.

Because TWI uses a bidirectional data line SDA for sending and receiving, it must strictly follow the timing when communicating. There are four basic communication modes in the data sheet. All TWI protocols are derived from these four basic modes.

16.4. TWI Interrupt and timeout

The TWI of SH30F9/SB0 series chip has 3 interrupt sources: communication interruption, bus timeout, and SCL high level timeout. The corresponding three interrupt flags are TWINT, TOUT, TFREE.

The bus timeout (TOUT) function and SCL high level timeout (TFREE) function of SH30F9/SB0 series chip are always on and cannot be turned off.

TOUT is generated after the SCL low level exceeds a certain period of time, indicating the upper limit of the time that a master or slave can pull the bus clock down. If the time exceeds this time, a bus timeout event will occur, and the hardware will automatically release the bus and set TOUT flag.

【Note】 The TOUT timeout function is permanently enabled. If the SCL is pulled down for a long time, a TOUT event will inevitably occur. The default timeout for TOUT is 25000 bus clocks.

TFREE occurs after the SCL high level exceeds a certain period of time, indicating that the bus is in an "idle" state. After the TFREE event occurs, the hardware automatically releases the bus and set TOUT flag.

【Note】 The TFREE timeout function is permanently enabled. If the SCL is pulled up for a long time (because SCL is pulled up by default, this state means that the SCL is idle), a TFREE event will inevitably occur. The default timeout time of TFREE is HOC×1024 bus clocks (the default value of HOC is 255).

【Note】 ETOT and EFREE are enable bits that control whether interrupts are generated when TOUT and TFREE events occur, not module function enable bits (emphasis again: the module function is permanently enabled).If ETOT and EFREE are turned off, there will be no interruption even if the timeout flag is set.

【Note】 TWINT is a communication interruption flag, and it will be set when the state of TWI changes, but it will not be set when entering the 0xF8 state from other states.

17. Aappendix

17.1. Aappendix 1：Function location and debugging method in KEIL MDK

Take SH30F9010 as an example to illustrate:

Code Area: 0x0000 0000~0x0003 FFFF

SRAM: 0x2000 0000~ 0x20003FFF

17.1.1. *Debugging the entire program in RAM*

Requirement: The debugged program must be smaller than the size of RAM.

Purpose: Divide the SRAM into two parts, put the entire program in the first half of the SRAM to run, and store variables in the second half.

step:

1. Add the Debug initialization file in the project directory and save it as "run_in_ram.ini". The file is in plain ASCII text format and reads

```
FUNC void Setup (void) {

     SP = _RDWORD(0x20000000);

     PC = _RDWORD(0x20000004);

  _WDWORD(0xE000ED08, 0x20000000);

}

FUNC void OnResetExec (void)  {

     Setup();

}

load %L incremental

Setup();

g, main
```

Statement description:

SP=_RDWORD(0x20000000): read 1 word from 0x20000000 and send it to SP

PC=_RDWORD(0x20000004): read 1 word from 0x20000004 and send it to PC

_WDWORD(0xE000ED08,0x20000000): Set the start address of the interrupt vector table

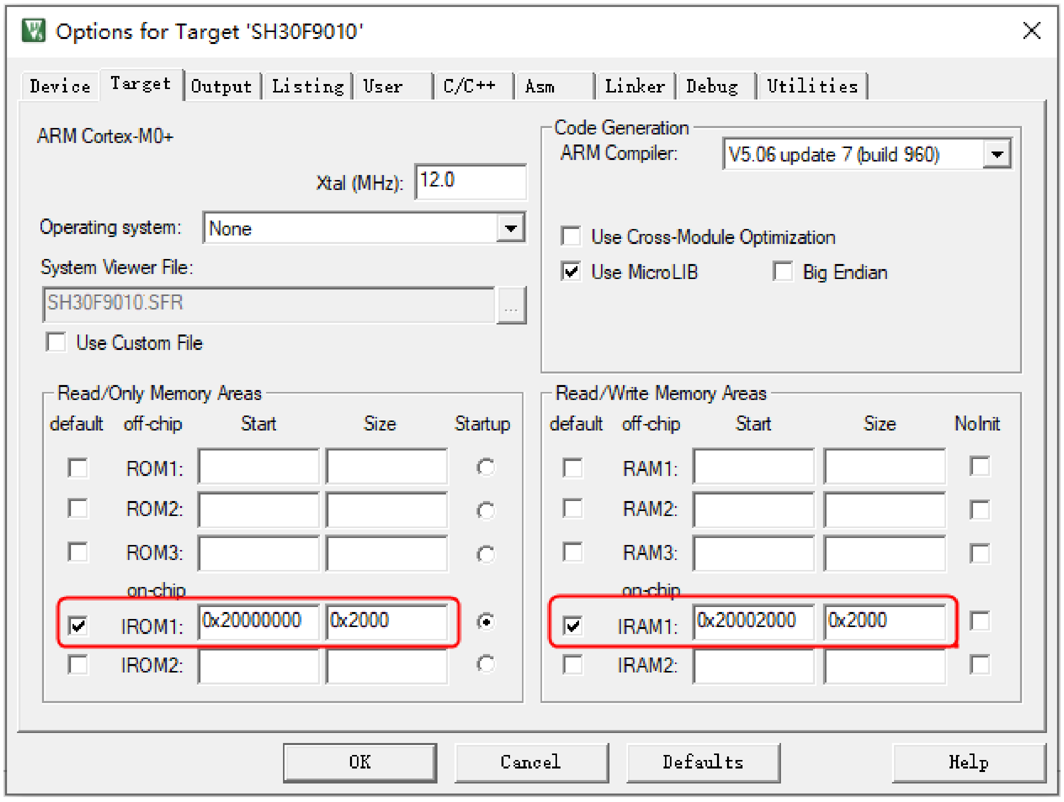
OnResetExec(); This function is called when the RESET button is pressed

Load %L incremental : Add the debugging information of the file, %L represents the Output file of the Link

g, main means to execute to the main function

**The statement outside the function is called when entering debug.

2. Modify the IROM settings and adjust the RAM settings as follows



Options for Target 'SH30F9010'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

ARM Cortex-M0+

Xtal (MHz): 12.0

Operating system: None

System Viewer File: SH30F9010.SFR

☐ Use Custom File

Code Generation

ARM Compiler: V5.06 update 7 (build 960)

☐ Use Cross-Module Optimization

☑ Use MicroLIB ☐ Big Endian

Read/Only Memory Areas

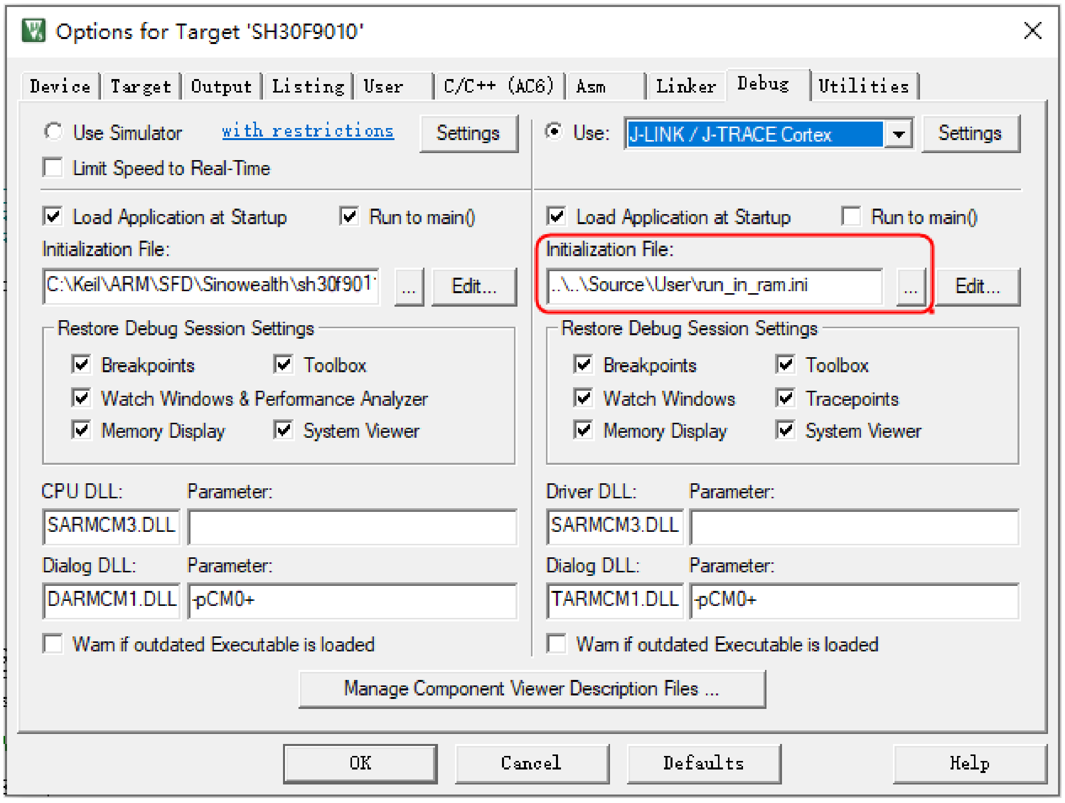
default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☑	IROM1:	0x20000000	0x2000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0x20002000	0x2000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help

3. Modify the DEBUG option as follows: Add the previously saved run_in_ram.ini to the Initialization File



Options for Target 'SH30F9010'

Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

○ Use Simulator with restrictions Settings

☐ Limit Speed to Real-Time

☑ Load Application at Startup ☑ Run to main()

Initialization File: C:\Keil\ARM\SFD\Sinowealth\sh30f901 ... Edit...

Restore Debug Session Settings

☑ Breakpoints ☑ Toolbox

☑ Watch Windows & Performance Analyzer

☑ Memory Display ☑ System Viewer

CPU DLL: SARMCM3.DLL Parameter:

Dialog DLL: DARMCM1.DLL Parameter: -pCM0+

☐ Warn if outdated Executable is loaded

● Use: J-LINK / J-TRACE Cortex Settings

☑ Load Application at Startup ☐ Run to main()

Initialization File: ..\..\Source\User\run_in_ram.ini ... Edit...

Restore Debug Session Settings

☑ Breakpoints ☑ Toolbox

☑ Watch Windows ☑ Tracepoints

☑ Memory Display ☑ System Viewer

Driver DLL: SARMCM3.DLL Parameter:

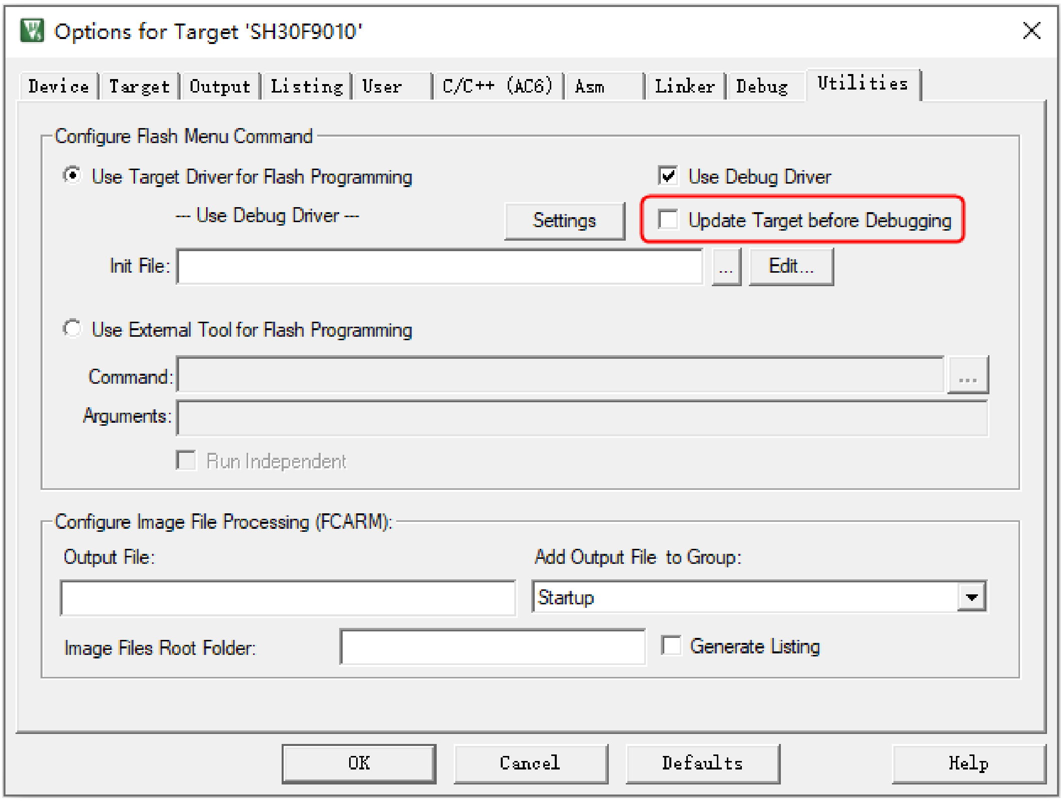
Dialog DLL: TARMCM1.DLL Parameter: -pCM0+

☐ Warn if outdated Executable is loaded

Manage Component Viewer Description Files ...

OK | Cancel | Defaults | Help

4. Modify the Utilities options as follows: Cancel Update Target before Debugging



Configure Flash Menu Command

(●) Use Target Driver for Flash Programming — [✓] Use Debug Driver

--- Use Debug Driver ---

[ ] Update Target before Debugging

Init File:

( ) Use External Tool for Flash Programming

Command:

Arguments:

[ ] Run Independent

Configure Image File Processing (FCARM):

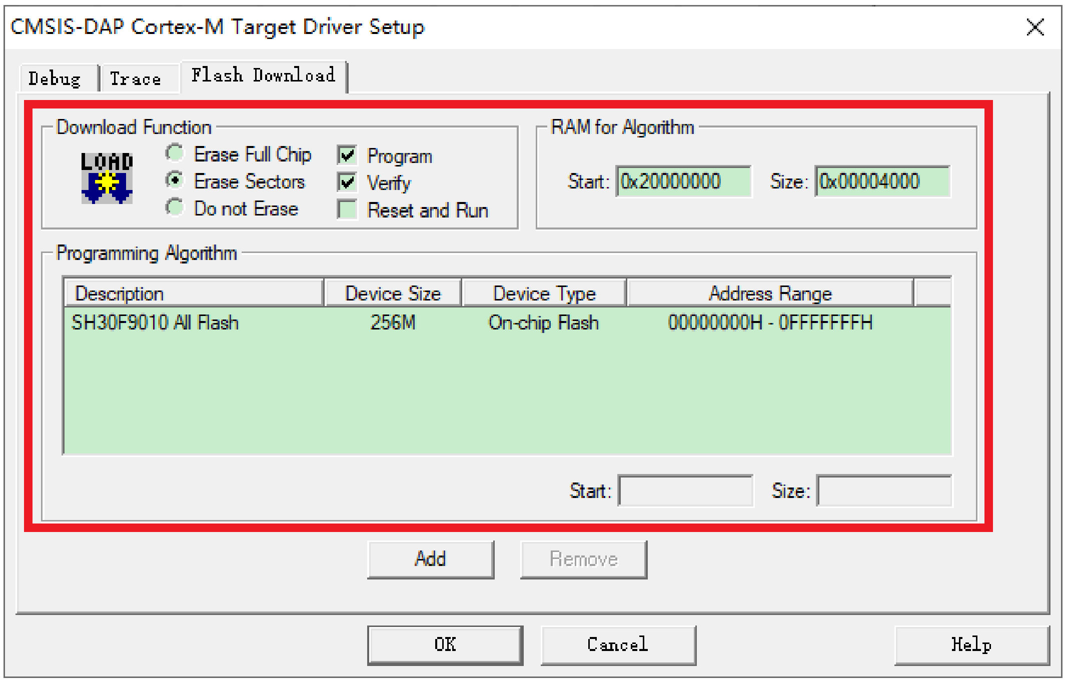
Output File:

Add Output File to Group: Startup

Image Files Root Folder:

[ ] Generate Listing

5. Modify the Flash Download settings as follows



Download Function

( ) Erase Full Chip — [✓] Program

(●) Erase Sectors — [✓] Verify

( ) Do not Erase — [ ] Reset and Run

RAM for Algorithm

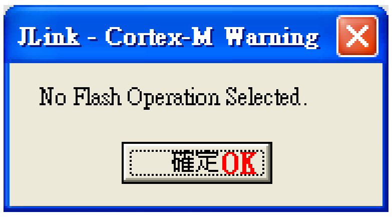
Start: 0x20000000 Size: 0x00004000

Programming Algorithm

Description	Device Size	Device Type	Address Range
SH30F9010 All Flash	256M	On-chip Flash	00000000H - 0FFFFFFFH

Start: Size:

6. Press the Debug button directly after rebuilding the project. There will be a warning "No Flash operation", click OK



JLink - Cortex-M Warning

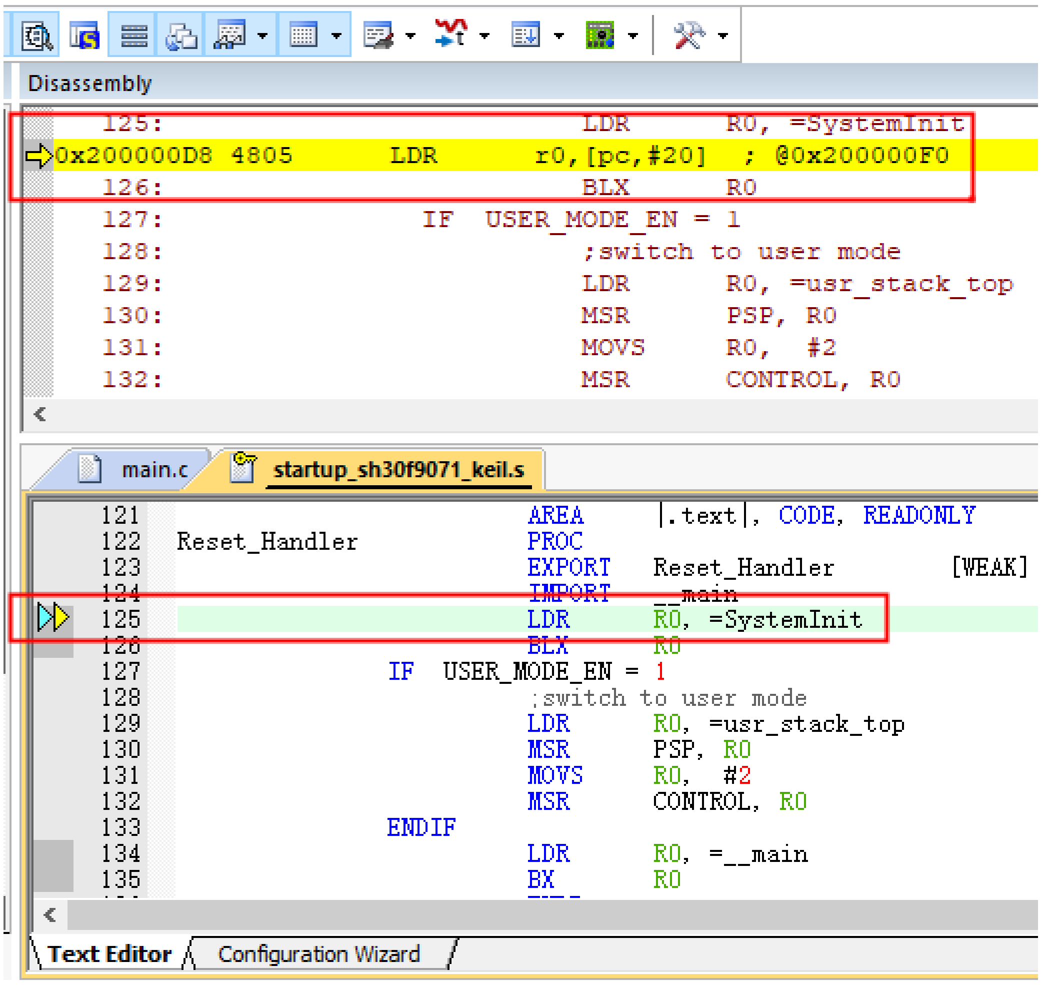
No Flash Operation Selected.

7. Already running normally in RAM. As shown below

Precautions:

You cannot use the RESET button to reset, you must exit Debug and enter again. Because after reset, the SP will be taken from the position of Code area 0, and the PC will be taken from the position of 4.

The program must relocate the location of the interrupt vector table to 0X20000000, otherwise the interrupt or exception cannot be correctly located.



```
   125:                     LDR      R0, =SystemInit
0x200000D8  4805      LDR       r0,[pc,#20]  ; @0x200000F0
   126:                     BLX      R0
   127:               IF  USER_MODE_EN = 1
   128:                     ;switch to user mode
   129:                     LDR      R0, =usr_stack_top
   130:                     MSR      PSP, R0
   131:                     MOVS     R0,  #2
   132:                     MSR      CONTROL, R0
```

```
Reset_Handler           AREA    |.text|, CODE, READONLY
                        PROC
                        EXPORT  Reset_Handler         [WEAK]
                        IMPORT  __main
                        LDR     R0, =SystemInit
                        BLX     R0
                IF  USER_MODE_EN = 1
                        ;switch to user mode
                        LDR     R0, =usr_stack_top
                        MSR     PSP, R0
                        MOVS    R0,  #2
                        MSR     CONTROL, R0
                ENDIF
                        LDR     R0, =__main
                        BX      R0
```

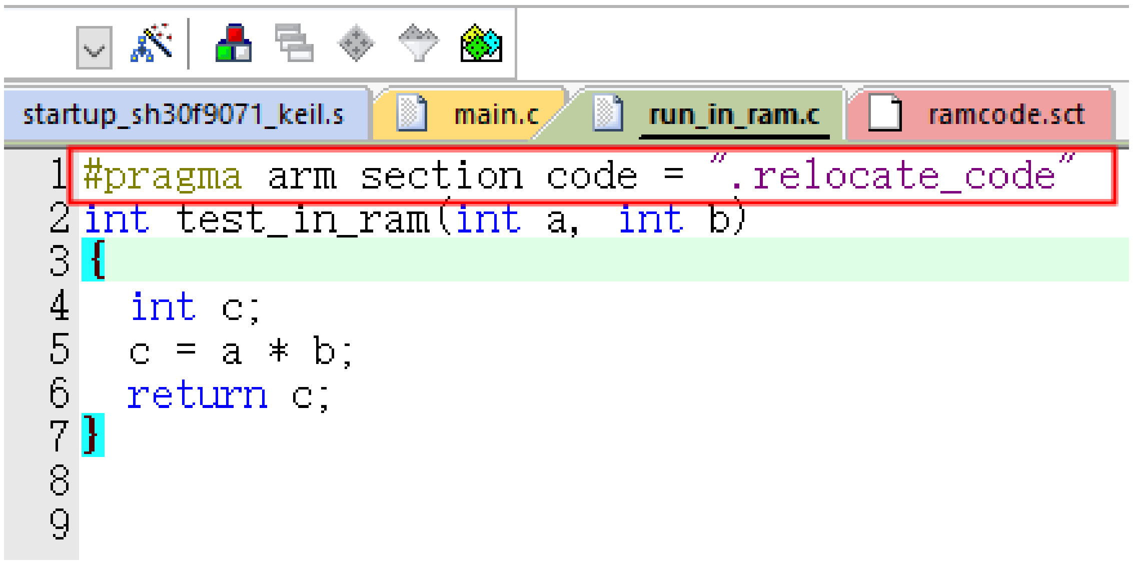
17.1.2. *Put the critical program into RAM to run*

Requirement: The program that needs to run in RAM must be smaller than the size of RAM.

Purpose: Improve the running speed of key code

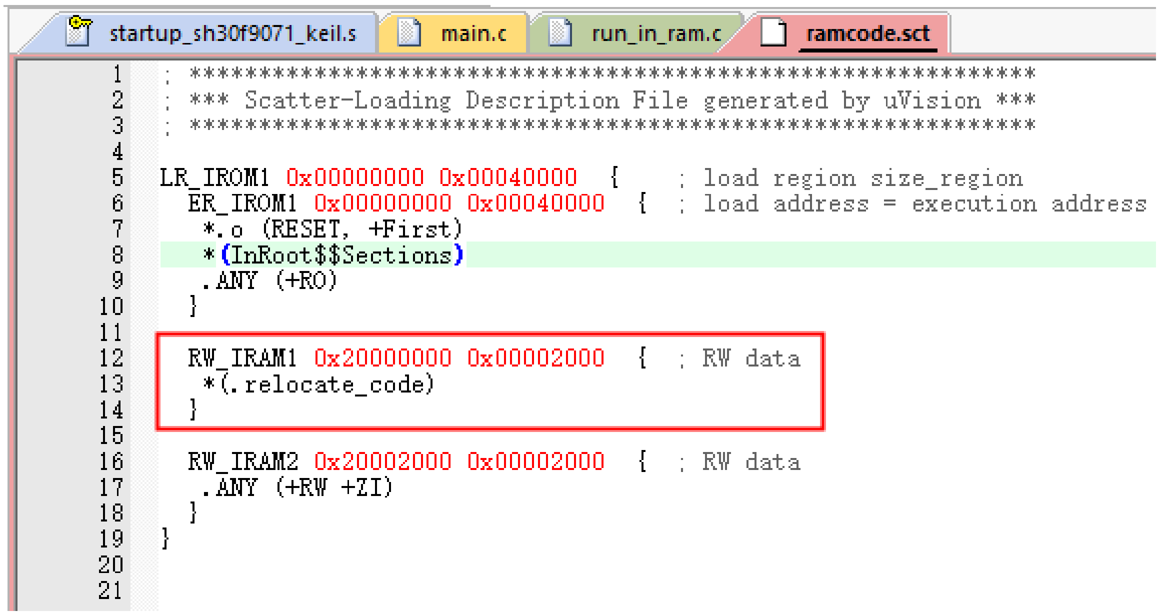
step:

1. Put the functions that need to be run in RAM into a file, and add the compiler instructions that define the section at the front. (The section name can be defined by yourself) as shown in the figure:



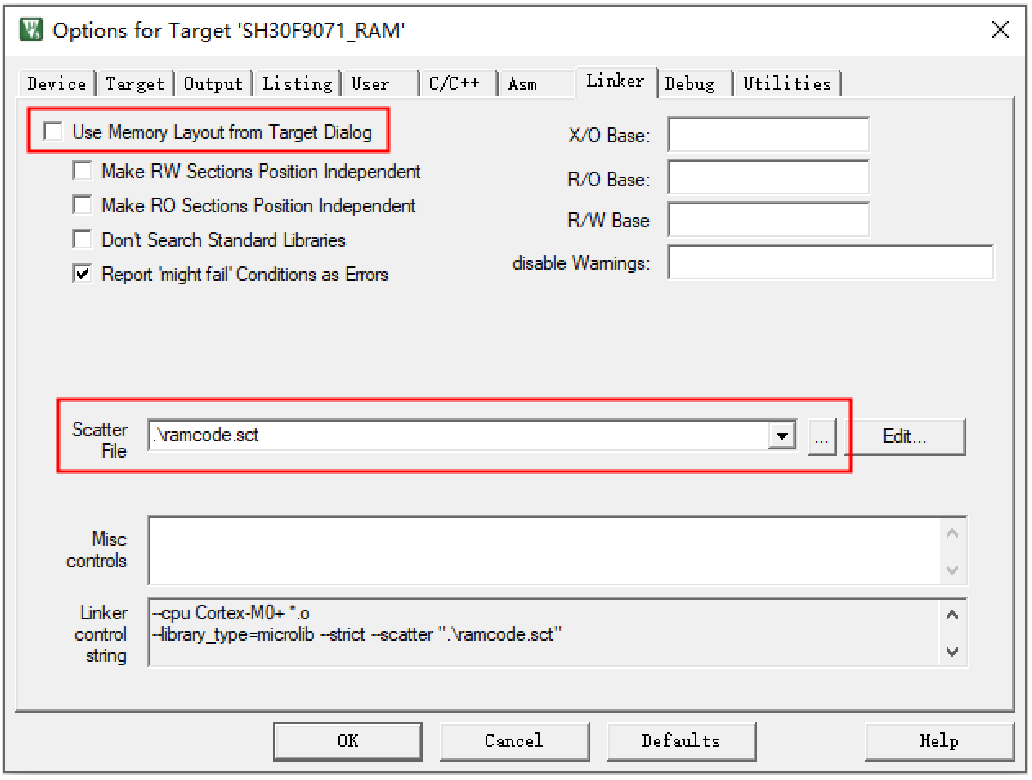
```
#pragma arm section code = ".relocate_code"
int test_in_ram(int a,  int b)
{
   int c;
   c = a * b;
   return c;
}
```

2. Add Link control file ramcode.sct. as shown in the picture



```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x00000000 0x00040000  {    ; load region size_region
  ER_IROM1 0x00000000 0x00040000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }

  RW_IRAM1 0x20000000 0x00002000  {  ; RW data
   *(.relocate_code)
  }

  RW_IRAM2 0x20002000 0x00002000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

3. Set the Link option, as shown in the figure



Options for Target 'SH30F9071_RAM'

- [ ] Use Memory Layout from Target Dialog
- [ ] Make RW Sections Position Independent
- [ ] Make RO Sections Position Independent
- [ ] Don't Search Standard Libraries
- [✓] Report 'might fail' Conditions as Errors

X/O Base:
R/O Base:
R/W Base
disable Warnings:

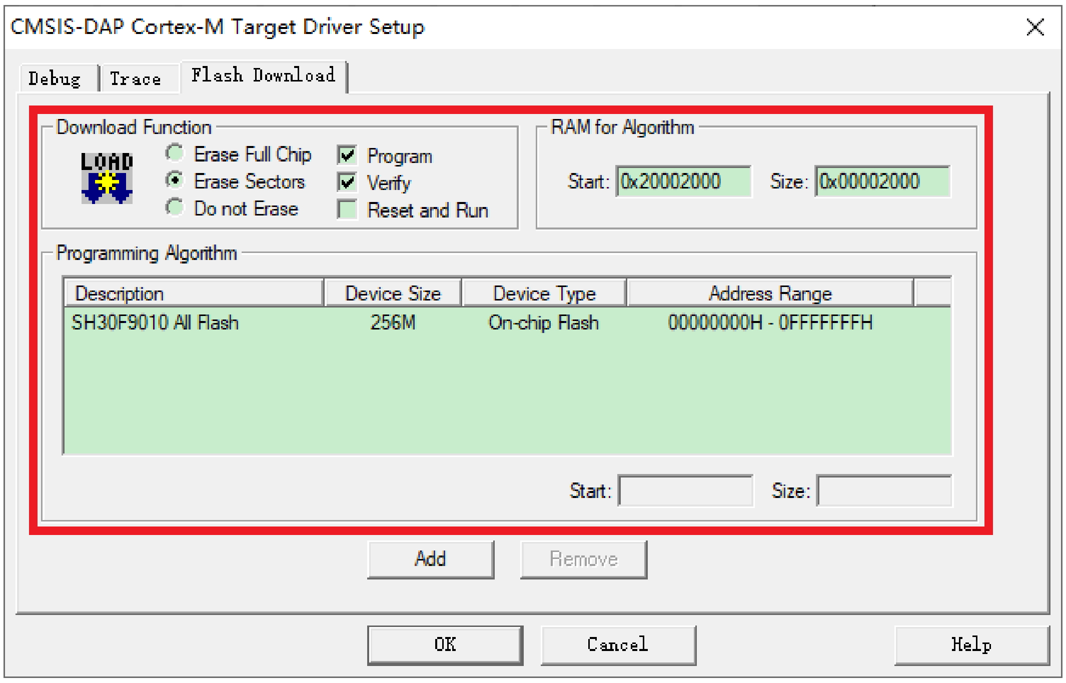
Scatter File: .\ramcode.sct

Misc controls:

Linker control string:

```
--cpu Cortex-M0+ *.o
--library_type=microlib --strict --scatter ".\ramcode.sct"
```

4. Modify the Flash Download settings as follows



CMSIS-DAP Cortex-M Target Driver Setup

Download Function

- Erase Full Chip
- [✓] Erase Sectors
- Do not Erase
- [✓] Program
- [✓] Verify
- [ ] Reset and Run

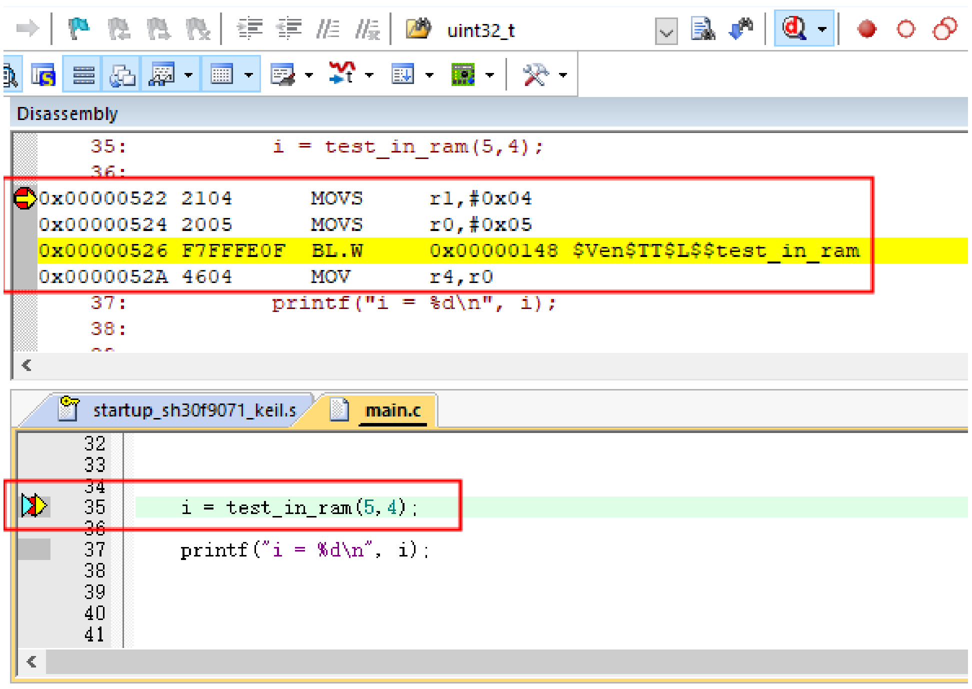
RAM for Algorithm: Start: 0x20002000 Size: 0x00002000

Programming Algorithm

Description	Device Size	Device Type	Address Range
SH30F9010 All Flash	256M	On-chip Flash	00000000H - 0FFFFFFFH

Start: Size:

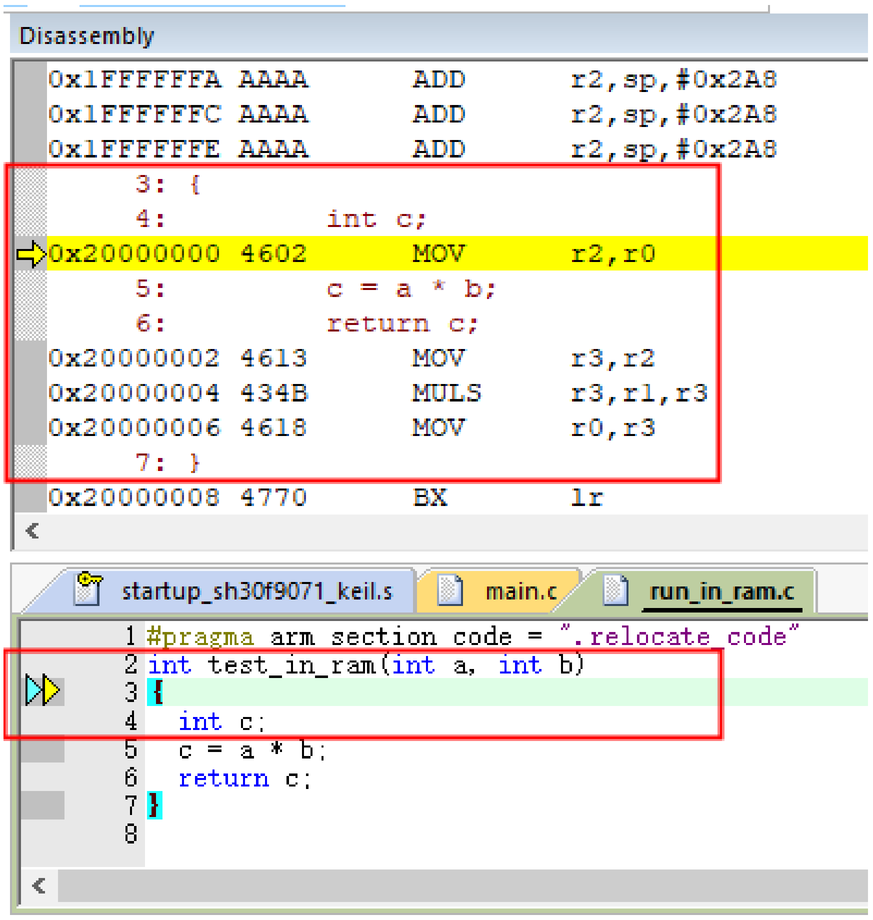
5. After rebuilding the project, press the Download button to download the program, and then press the Enter Debug button to enter the Debug interface. Run to the place where the function is called and you can see that it is running in Flash



```
    35:             i = test_in_ram(5,4);
    36:
0x00000522 2104      MOVS     r1,#0x04
0x00000524 2005      MOVS     r0,#0x05
0x00000526 F7FFFE0F  BL.W     0x00000148 $Ven$TT$L$$test_in_ram
0x0000052A 4604      MOV      r4,r0
    37:             printf("i = %d\n", i);
    38:
```

```
    i = test_in_ram(5,4);

    printf("i = %d\n", i);
```

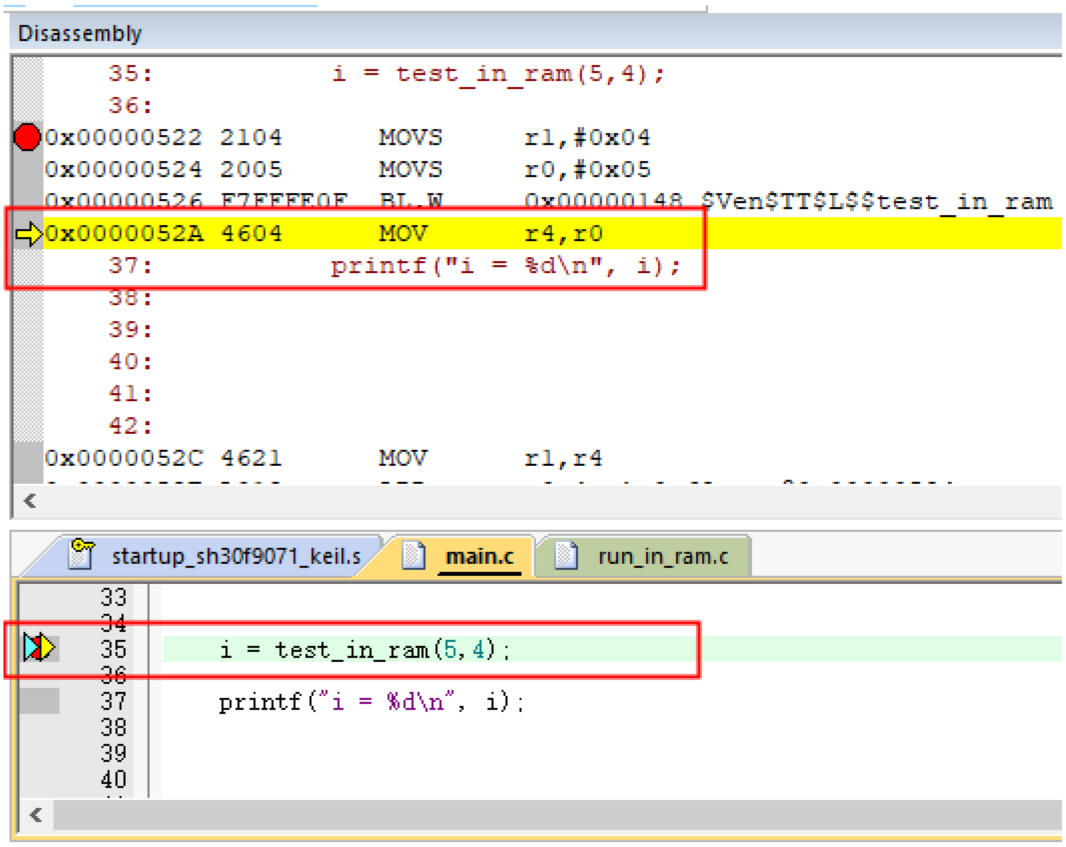
6. After running Step Into, you can see that it has jumped to RAM1.



```
0x1FFFFFFA AAAA      ADD      r2,sp,#0x2A8
0x1FFFFFFC AAAA      ADD      r2,sp,#0x2A8
0x1FFFFFFE AAAA      ADD      r2,sp,#0x2A8
     3: {
     4:         int c;
0x20000000 4602      MOV      r2,r0
     5:         c = a * b;
     6:         return c;
0x20000002 4613      MOV      r3,r2
0x20000004 434B      MULS     r3,r1,r3
0x20000006 4618      MOV      r0,r3
     7: }
0x20000008 4770      BX       lr
```

```
#pragma arm section code = ".relocate_code"
int test_in_ram(int a, int b)
{
  int c;
  c = a * b;
  return c;
}
```

7. Run Step Out again, you can see that it jumps back to the Flash area


Disassembly

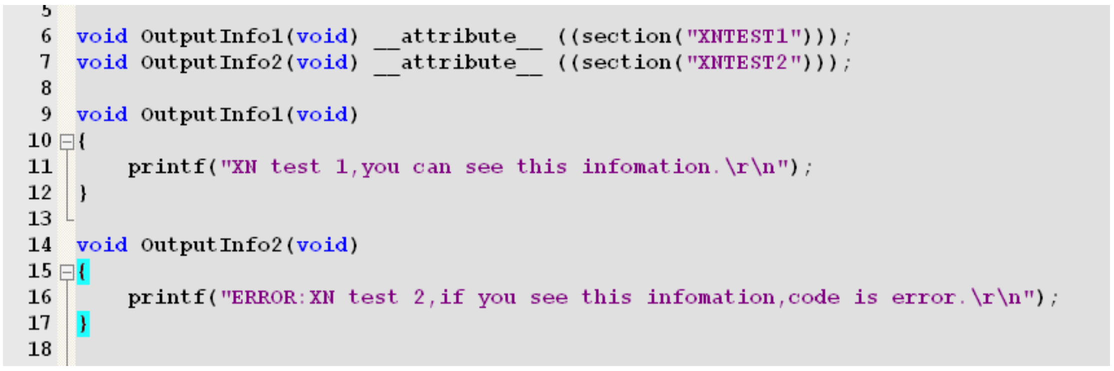
```
    35:             i = test_in_ram(5,4);
    36:
0x00000522 2104      MOVS     r1,#0x04
0x00000524 2005      MOVS     r0,#0x05
0x00000526 F7FFFE0F  BL.W     0x00000148 $Ven$TT$L$$test_in_ram
0x0000052A 4604      MOV      r4,r0
    37:             printf("i = %d\n", i);
    38:
    39:
    40:
    41:
    42:
0x0000052C 4621      MOV      r1,r4
```

startup_sh30f9071_keil.s | main.c | run_in_ram.c

```
33
34
35      i = test_in_ram(5,4);
36
37      printf("i = %d\n", i);
38
39
40
```

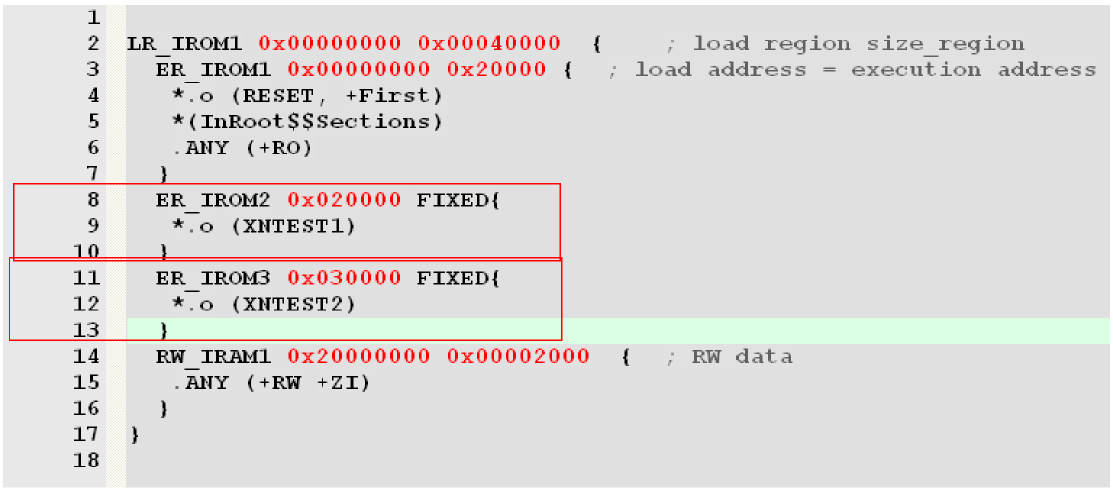
17.1.3. *Fix a function to a certain position*

1. Give the function a section name


```
void OutputInfo1(void) __attribute__ ((section("XNTEST1")));
void OutputInfo2(void) __attribute__ ((section("XNTEST2")));

void OutputInfo1(void)
{
    printf("XN test 1,you can see this infomation.\r\n");
}

void OutputInfo2(void)
{
    printf("ERROR:XN test 2,if you see this infomation,code is error.\r\n");
}
```

You can also use #pragma arm section code="XNTEST1" to define a section with the same name for all functions in the entire file.

2. Specify the address in the scatter file



```

LR_IROM1 0x00000000 0x00040000  {    ; load region size_region
  ER_IROM1 0x00000000 0x20000 {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  ER_IROM2 0x020000 FIXED{
   *.o (XNTEST1)
  }
  ER_IROM3 0x030000 FIXED{
   *.o (XNTEST2)
  }
  RW_IRAM1 0x20000000 0x00002000  {  ; RW data
   .ANY (+RW +ZI)
  }
}

```

As shown in the figure, the function OutputInfo1 is fixed at the position of 0x20000
As shown in the figure, the function OutputInfo2 is fixed at the position of 0x30000

17.2. Appendix 2: Automatically export similar data into a table

17.2.1. *Define an output macro*

E.g:

```
#define SECTION(x)                  __attribute__((section(x)))

#define RT_USED                     __attribute__((used))

typedef int (*drv_func) (void* param);

struct func_item{

    drv_func      func;

    const char*   desc;

};

#define EXPORT_FUNC_TABLE(name,desc)  const char _drv_##name##_desc[] = #desc; \

                              const struct func_item _drv_##name##_cmd SECTION("MySymTab") = \

                               {  \

                                   (drv_func)&name, \

                                   _drv_##name##_desc \

                               };
```

17.2.2. *Export call function*

Call this macro where you need to export

E.g:

```
static int func1 (void* param)

{

    return 3;

}
```

```
EXPORT_FUNC_TABLE(func1, func desp1);


static int func2 (void* param)

{

   return 1;

}

EXPORT_FUNC_TABLE(func2, func desp2);


static int func3 (void* param)

{

   return 0;

}

EXPORT_FUNC_TABLE(func3, func desp3);


static int func4(void* param)

{

   return 5;

}

EXPORT_FUNC_TABLE(func4, func desp3);
```

17.2.3. *Table lookup method*

**Note: In KEIL MDK, the header logo MySymTab$$Base and the tail logo MySymTab$$Limit will be automatically added

```
void my_func(void)

{

   struct func_item *ps;

   struct func_item *pe;

   extern const int MySymTab$$Base;

   extern const int MySymTab$$Limit;

   ps = (struct func_item*) MySymTab$$Base;

   pe = (struct func_item*) MySymTab$$Limit;

   while(ps != pe)

   {

      ps->func(0);

      ps++;

   }

}
```

17.3. Appendix 3: Fixing variables to a location in RAM

17.3.1. *Specify the address directly at the variable definition*

```
uint32_t g_var1 __attribute__((at(0x20000000))) = 0x33445566;

uint32_t g_var2 __attribute__((at(0x20000004)));
```

17.3.2. *Specify the section directly in the variable definition*

Specify the section at the variable definition, and then use the linker's scatter file to fix this section to a certain position:

```
uint32_t g_fvar1  __attribute__((section(".fixed1"))) = 0x12345678; /* RW */

uint32_t g_fvar2  __attribute__((section(".fixed1"))) = 0x87654321; /* RW */

uint32_t g_fvar3  __attribute__((section(".fixed1"))) = 0x87654321; /* RW */

uint32_t g_fixed1 __attribute__ ((section(".fixed_var")));
```

Define variables g_fvar1, g_fvar2, and g_fvar3 to belong to section ".fixed1", and variable g_fixed1 to belong to section ".fixed_var".

Then define it in the scatter file as follows:

```
RAM_IRAM_1 0X20000400 0x0000400{

   *(.fixed_var);

 }

 RAM_IRAM_2 0X20000800 0x0000400{

   *(.fixed1);

 }
```

The variable in Section ".fixed_var" is defined to the starting position of 0x20000400
The variables in Section ". fixed1" are defined to the starting position of 0x20000800

17.3.3. *Specify all variables in the file to a fixed location*

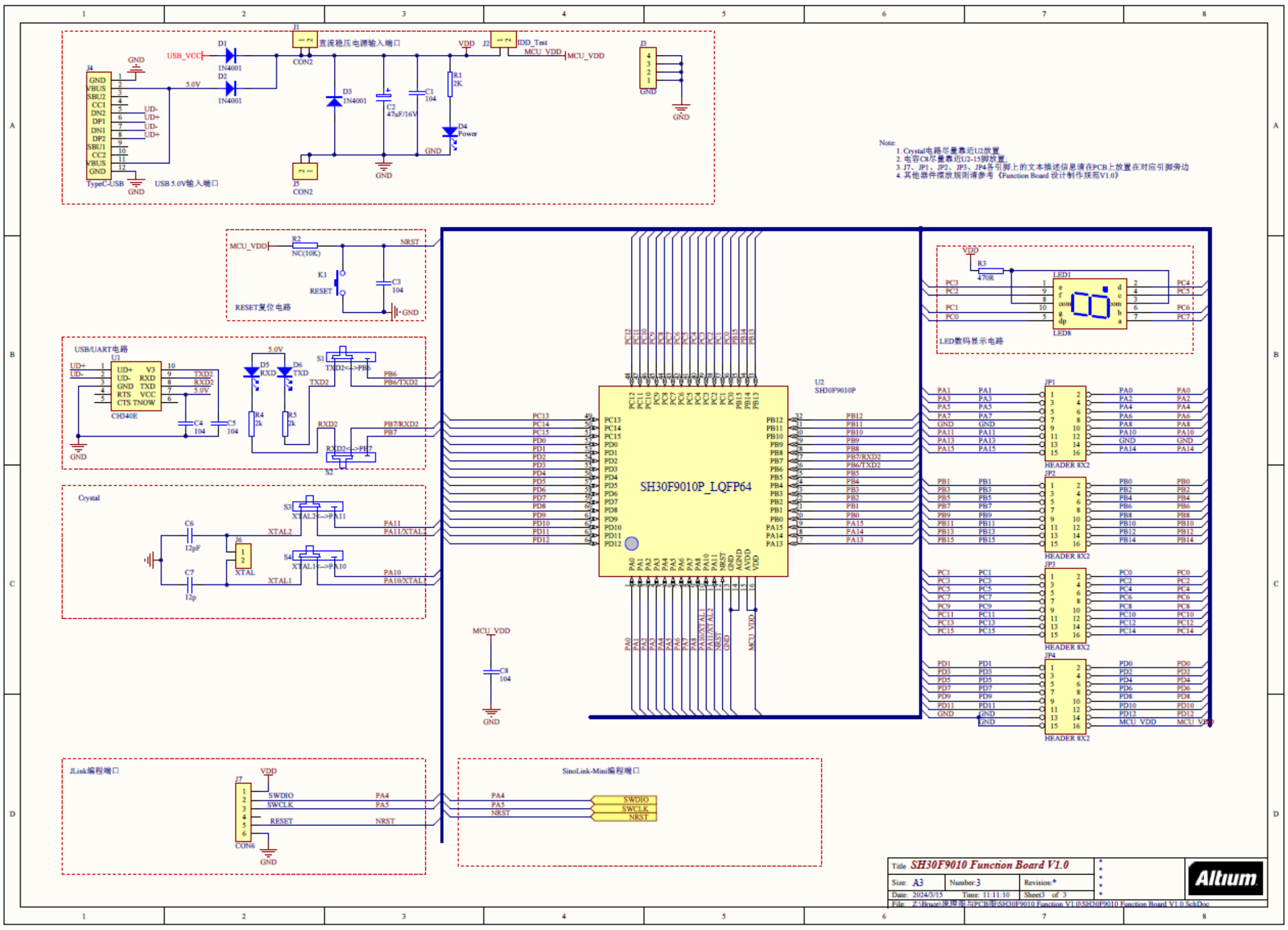
Define the section in the source file header, and the attribute is rwdata.

```
#pragma arm section rwdata=".fixed_var"

#pragma arm section zidata=".fixed_var"

uint32_t g_fixed1;

uint32_t g_fixed2;
```

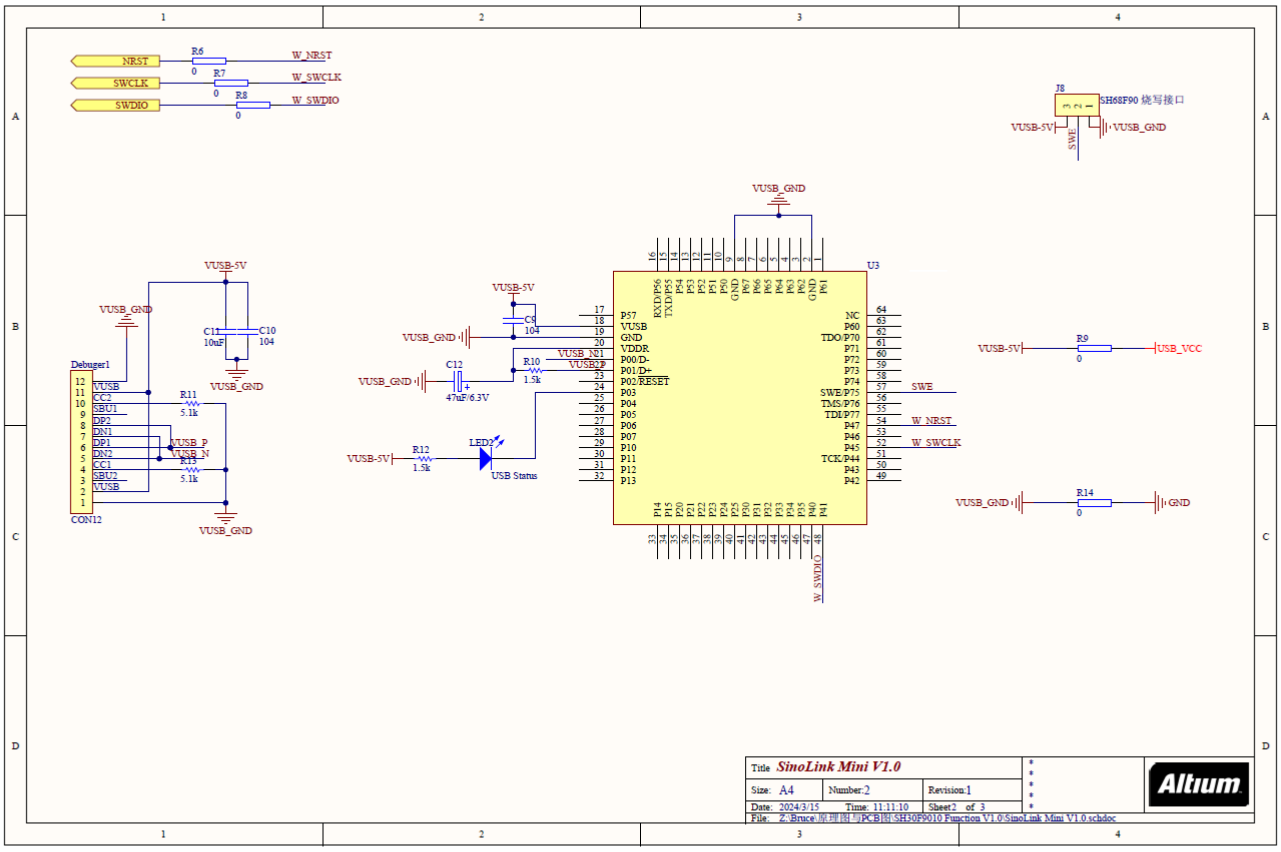
Then define it in the scatter file as follows

```
RAM_IRAM_1 0X20000400 0x0000400{

    *(.fixed_var);

}

The variable in Section ".fixed_var" is defined to the starting position of 0x20000400
```

17.4. Appendix 4: SH30F9010 Function Boardschematic diagram



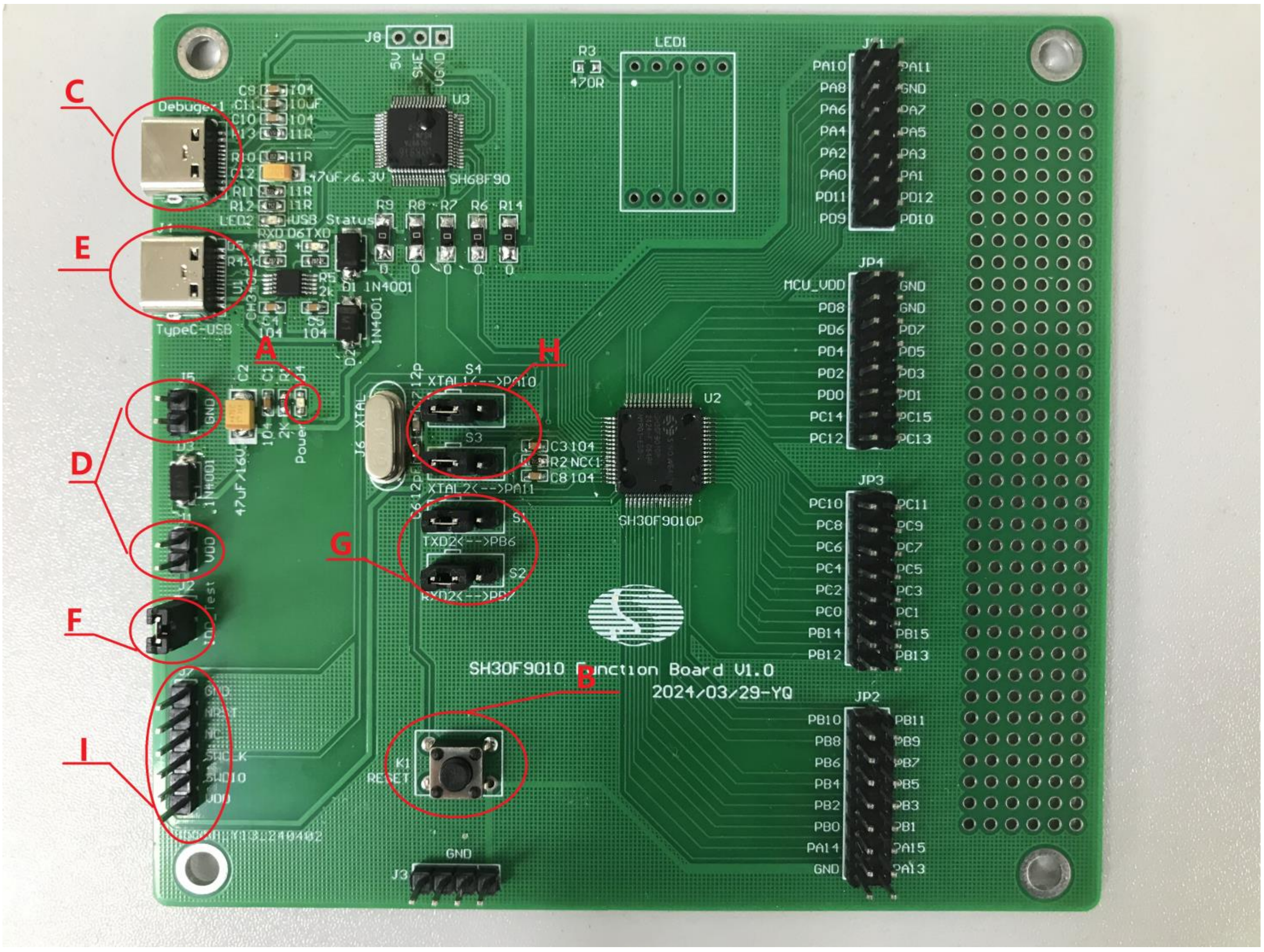
SH30F9010P_LQFP64
Title SH30F9010 Function Board V1.0
Size: A3
Number:3
Revision:*
Date: 2024/3/15
Time: 11:11:10
Sheet3 of 3
File: Z:\Bruce\原理图与PCB图\SH30F9010 Function V1.0\SH30F9010 Function Board V1.0.SchDoc
Note:
1. Crystal电路尽量靠近U2放置
2. 电容C8尽量靠近U2-15脚放置
3. J7、JP1、JP2、JP3、JP4各引脚上的文本描述信息请在PCB上放置在对应引脚旁边
4. 其他器件摆放规则请参考《Function Board 设计制作规范V1.0》
RESET复位电路
USB/UART电路
Crystal
LED数码显示电路
JLink编程端口
SinoLink-Mini编程端口
USB 5.0V输入端口
直流稳压电源输入端口
Altium



Title SinoLink Mini V1.0
Size: A4
Number:2
Revision:1
Date: 2024/3/15
Time: 11:11:10
Sheet2 of 3
File: Z:\Bruce\原理图与PCB图\SH30F9010 Function V1.0\SinoLink Mini V1.0.schdoc
SH68F90 烧写接口
Debugger1
CON12
USB Status
Altium

17.5. Appendix 5:SH30F9010 Function Board

SH30F9010 provides a set of demonstration boards for customers to familiarize themselves with chip functions. As shown in the following figure:



SH30F9010 Function Board

Function description of each part in the board:

A. Power light
B. Reset button
C. SinoLink-Mini Debugger interface

   The SinoLink Mini onboard debugger uses a USB Type-C interface, which enables onboard burning and simulation. Select CMSIS-DAP in the simulator configuration interface, and like jlink, add a burning algorithm file.

D. External power supply interface (VDD/GND)

   Function Board power supply interface, connected to the VDD and GND pins of the chip respectively. Choose external power supply, the chip power supply voltage can be adjusted.

E. Serial port

Using USB Type-C to connect with PC. Connect the UART to the MCU's UART2 (TXD2<->PB6, RXD2<->PB7) via USB to serial chip for UART debugging.

Note: The power supply of the demonstration board can be provided by an external power interface, USB interface, or simulation interface. Regardless of which power supply method is used, the voltage at the external power interface is the power supply voltage of the demonstration board.

F. Current testing interface. (Do not conduct current testing, please connect with a short-circuit jumper)

G. Jumper cap (UART PIN)

When PB6 and PB7 are used as UART2, the left two jumper caps are short circuited to form a USB to serial port circuit with the E part on the board; When PB6 and PB7 are used as IO, short-circuit the two on the right side of the jumper cap and disconnect it from the external USB to serial port circuit.

H. Jumper cap（CRY）

When PA10 and PA11 are used as Crystal PINs, short-circuit the left two jumper caps to connect to the external crystal oscillator. When PA10 and PA11 are used as IO, short-circuit the right two jumper caps to disconnect from the external crystal oscillator.

I. SWD Debugger interface

The SWD simulation interface can be connected to the VDD, SWDIO, SWCLK, RST, and GND of Jlink or SinoLink simulators using DuPont wires for downloading simulations. The power supply for the demonstration board can be obtained through the VDD pin of the simulation interface.