# Table of Contents

**Table of Contents** ..... 1

**1. Precautions for SinOne SC92F Series MCU Electrical Parameters** ..... 3

**2. Precautions for SinOne SC92F Series MCU Programming** ..... 3

**3. Precautions for MOVC Instructions of SinOne SC92F Series MCU** ..... 4

**3.1 Precautions for MOVC Instructions in C-language Programming** ..... 4
3.1.1 MOVC Instructions in C-language Development ..... 4
3.1.2 Specific Operation of C-language Development ..... 5

**3.2 Precautions for MOVC Instructions in Assemble Language Programming** ..... 10

**4. SinOne SC92F Series MCU EEPROM and Algorithm Explanation** ..... 11

**4.1 Precautions for Internal EEPROM Operations and Code IAP Operations** ..... 11

**4.2 EEPROM Operations and Code IAP Operations Codes** ..... 13

**4.3 EEPROM Use Algorithm** ..... 15

**5. Precautions for Circuit Design** ..... 21

**5.1 Circuit Design Examples** ..... 21
5.1.1 Circuit Design Instances ..... 21
5.1.2 LED Use and Connection ..... 21
5.1.3 Use of 1-digit Common Cathode Digital Tube ..... 22
5.1.4 RST Pin Circuit ..... 22
5.1.5 ADC Sampling Pin Circuit ..... 23
5.1.6 External Crystal Oscillation Circuit ..... 24
5.1.7 Touchkey Circuit ..... 24

**5.2 Precautions for Mode Setting of IO Port** ..... 25
5.2.1 Set I//O as High Resistance for Circuit Design ..... 25
5.2.2 Pull-up Input Mode ..... 25
5.2.3 Detection Keys of Pull-up Input Mode ..... 25
5.2.4 Implementation of I/O Open-drain Output Mode ..... 25
5.2.5 Precautions for I/O Read IO Functions ..... 26

**6. Precautions for Software Programming** ..... 26

**6.1 Precautions for PWM Setup and Use** ..... 26

**6.2 Precautions for PCON Register Setup** ..... 26

**6.3 Precautions for Checksum Setup** ..... 27

**6.4 Precautions for UART0 Setup and Use** ..... 27

6.5 Precautions for SPI/TWI/UART Universal Serial Port SSI Setup and Use ..... 28
6.6 Precautions for ADC Multi-channel Switch Acquisition ..... 28
6.7 Precautions for Writing External Interrupt 0/1 Service Functions When Using the Timer..... 29
6.8 Precautions for External Interrupt Setup ..... 30
6.9 Precautions for LCD/LED/PWM RAM Use ..... 30
6.10 Precautions for Code Option of Software Operations..... 31
6.11 Precautions for Touchkey Setup..... 31
7. Version Change History ..... 32
Statement ..... 33

# 1. Precautions for SinOne SC92F Series MCU Electrical Parameters

**Working Voltage:**
2.0V-5.5V (SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x)
2.4V-5.5V (Other models)
**Working Temperature:**
40 - 105℃ (SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x)
40 - 85℃ (Other models)
**Kernel:** High-speed 1T 8051
**Flash ROM：** (MOVC forbids addressing 0000H~00FFH) Repeatably write for 10,000 times
**LDROM:** 1-4 Kbyte variable, repeatably write for more than 100,000 times with more than 100 years of storage life at room temperature (SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x)
**EEPROM:** Independent 128Byte, repeatably write for 100,000 times with more than 10 years of storage life
**System Clock:**
Built-in high-frequency oscillator frequency error: No more than ±4% in the application environment spanning 4.0V - 5.5V and -20 - 85℃ (SC92F8x9x, SC92F7x9x)
Built-in high-frequency oscillator frequency error: No more than ±1% in the application environment spanning 4.0V - 5.5V and -20 - 85℃ (other models)
**Note**: LVR setting voltage must be lower than the power supply voltage to prevent constant reset of the chip!

# 2. Precautions for SinOne SC92F Series MCU Programming

1. The capacitance of CLK or DIO pins of SC92F series chips to GND shall not exceed 100pF and that of VDD to GND shall not exceed 1000uF.
2. Try not to allow series resistance before programming the lead-out point and the chip; if it is inevitable, guarantee that the resistance value of the series resistance does not exceed 100R and minimize the programming cable when programming.
3. Avoid connecting CLK and DIO of the chip to the same digital tube upon circuit design.
4. The length of SC-LINK programming cable shall not exceed 60cm.
5. SC_LINK and SOC Pro51 programming is not supported by SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x, please upgrade the models mentioned above to SC_LINK_PRO and programming upper computer SOC Programming Tool.

# 3. Precautions for MOVC Instructions of SinOne SC92F Series MCU

The start 256B ROM range of SinOne MCU Flash ROM, namely 0x0000-0x00FF, forbids MOVC addressing. Therefore, the user-defined data can not be stored in this region. For example, in C-language programming, initialized global variables and immutable-type data can not be stored in this address region.

The following are precautions for the application of MOVC instructions in programming specific for this feature.

## 3.1 Precautions for MOVC Instructions in C-language Programming

### 3.1.1 MOVC Instructions in C-language Development

In C-language development, MOVC instructions are usually used in three cases, that is to access Flash ROM.
1. Initialize the global variables
2. Immutable type data (code type data)
3. Look-up table operation of function call library files

After C-language compilation is completed, the user can open the .M51 file in the project to check the Code Memory section. By checking Code identifier, you can check if the operations in the preceding three conditions exist. See the table below:

| Identifier | Description | Remark |
|---|---|---|
| ?C_INITSEG | Initialize global variables | Call before entering MAIN |
| ?CO?Project_name | Constants or pointers in Code region | “Project_name” project name |
| ?C?LIB_CODE | Library file | Math function or floating-point operation may be used |

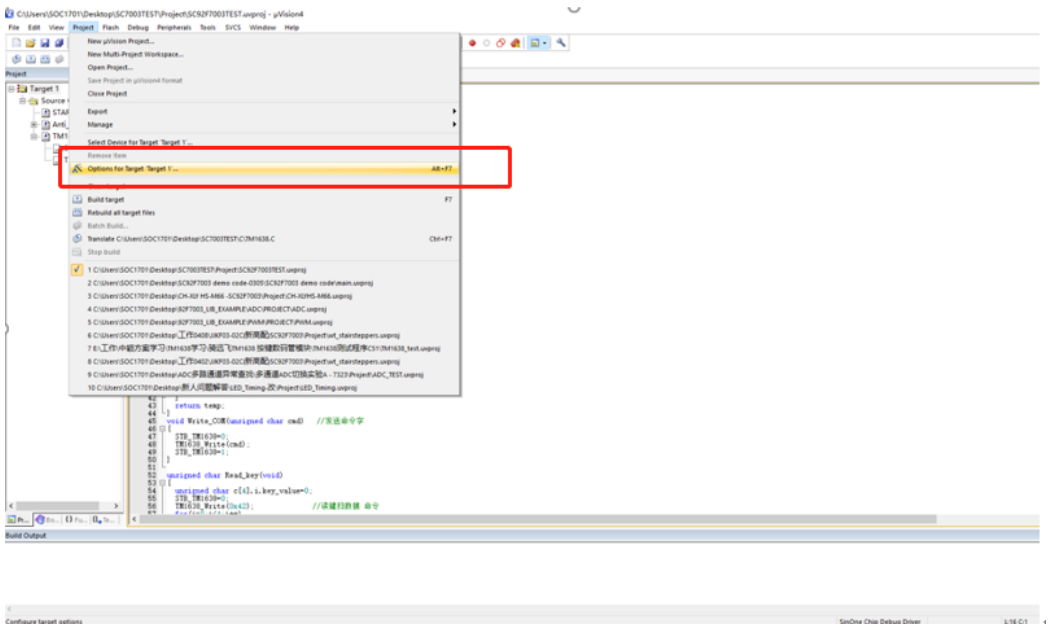
Note: ?C?LIB_CODE identifier is simply used to conduct look-up table operation to the library file called by a functions. In general, there is no need to call the Math function of the library file when the customers develop the products. (The library file may occupy large ROM space, such as sinx function).

The usage of each code segment in the above table is recorded in .M51 file in detail, including the start address and the length of Code. The user just needs to check if the start address of C_INITSEG, ?CO?Project_name and the function of call library file (if ?C?LIB_CODE available) is in the forbidden access area. If yes, refer to the following operations to change the start address.

### 3.1.2 Specific Operation of C-language Development

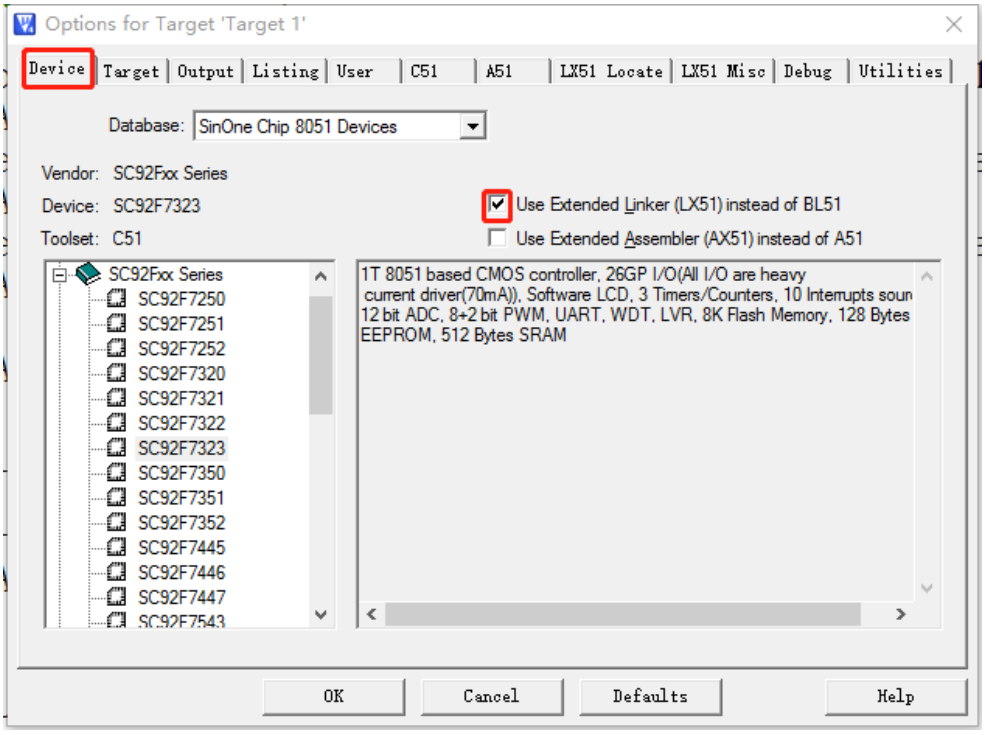
According to 3.1.1, the user needs to define the global variables and immutable type data (code type data) after the start 256B address of Flash ROM in the process of C-language development. Therefore, upon developing and debugging, it is able to shield the Flash Rom in this area, make adjustment after developing and debugging and generate the final program. The operation methods may vary depending on different linkers, the specific methods are as follows:

Make adjustment on LX51 Linker (recommended method):



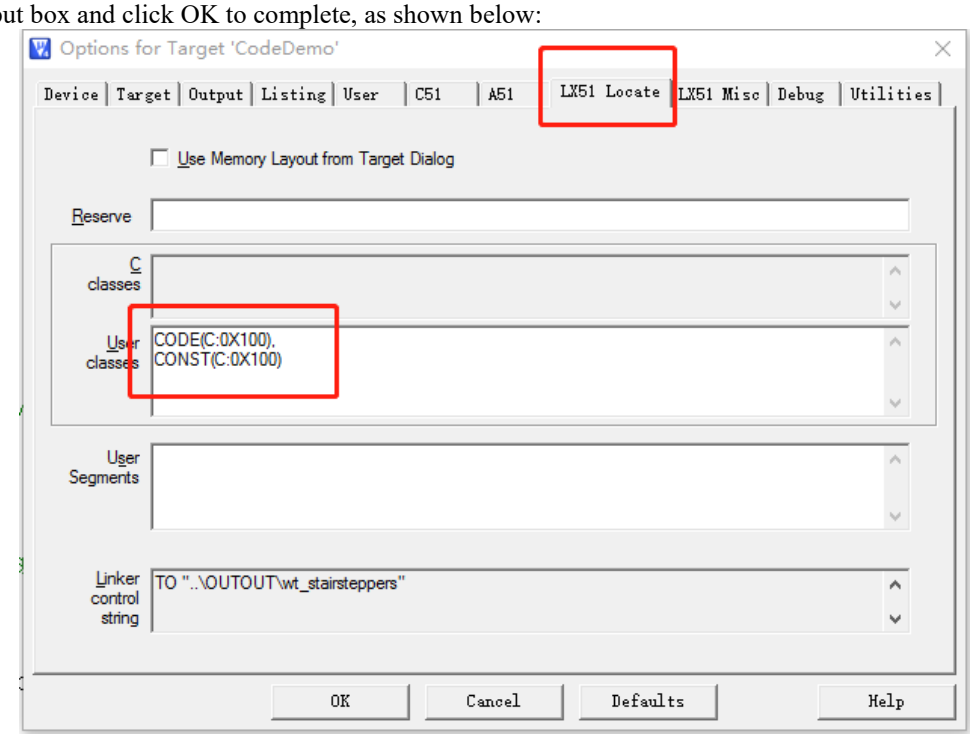
- **Select LX51 linkers.**

Open “Device” property page in the Project Option and check “Use Extended Linker(LX51) instead of BL51”, as shown below:



- **Add code-specified storage range in LX51 Setting Options.**

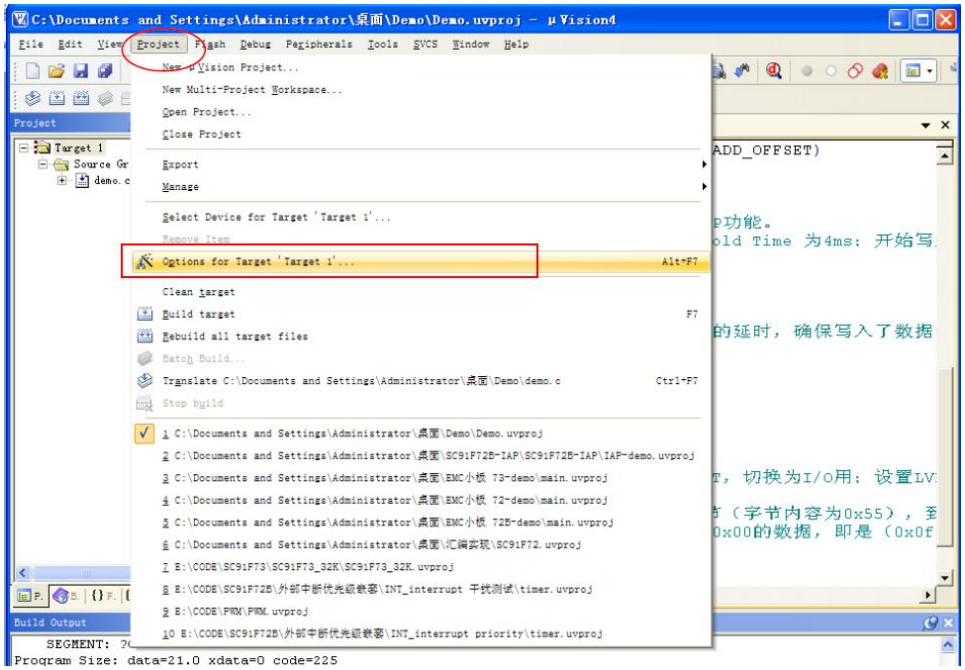
Open “LX51 Locate” property page in the Project Option”, fill in “CODE(C:0X100), CONST(C:0X100)” instruction in “User classes” input box and click OK to complete, as shown below:

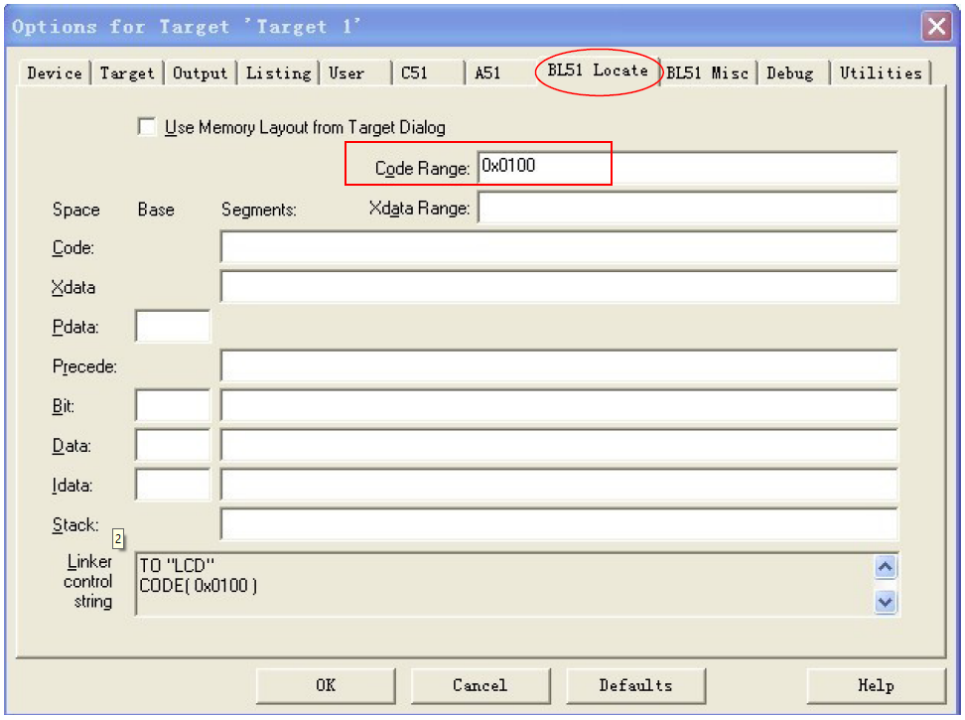



**Make adjustment on BL51 linker:**

- **Set code storage area for debugging, and place the code area after 0x0100.**

Open “BL51Locate” property page in the Project Option, input “0x0100” in Code Range for recompiling and debugging, as shown below:



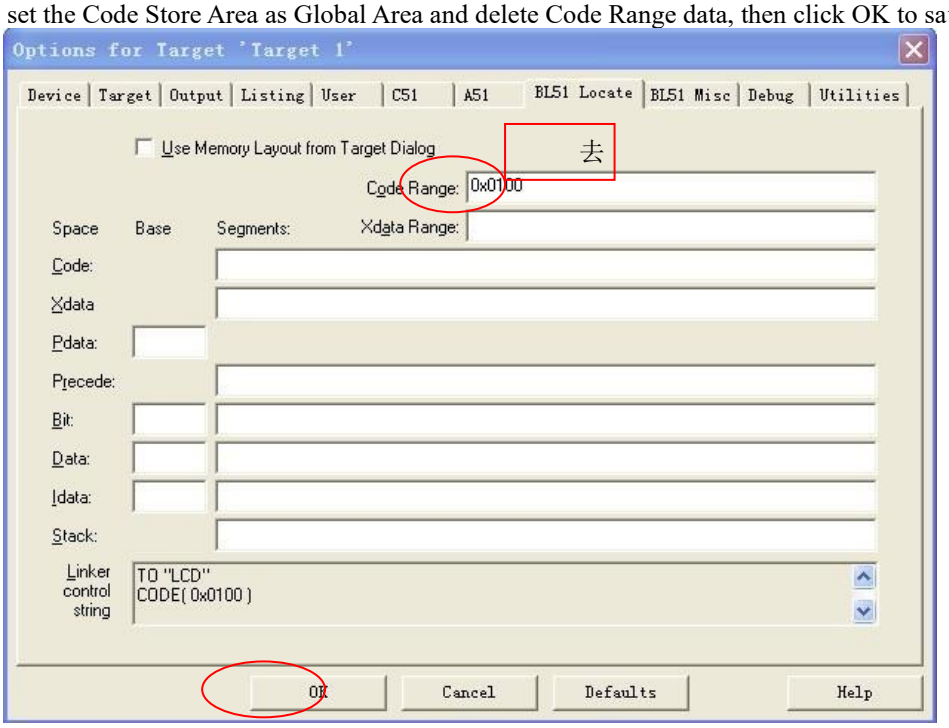



- **After debugging, generate the final program;**

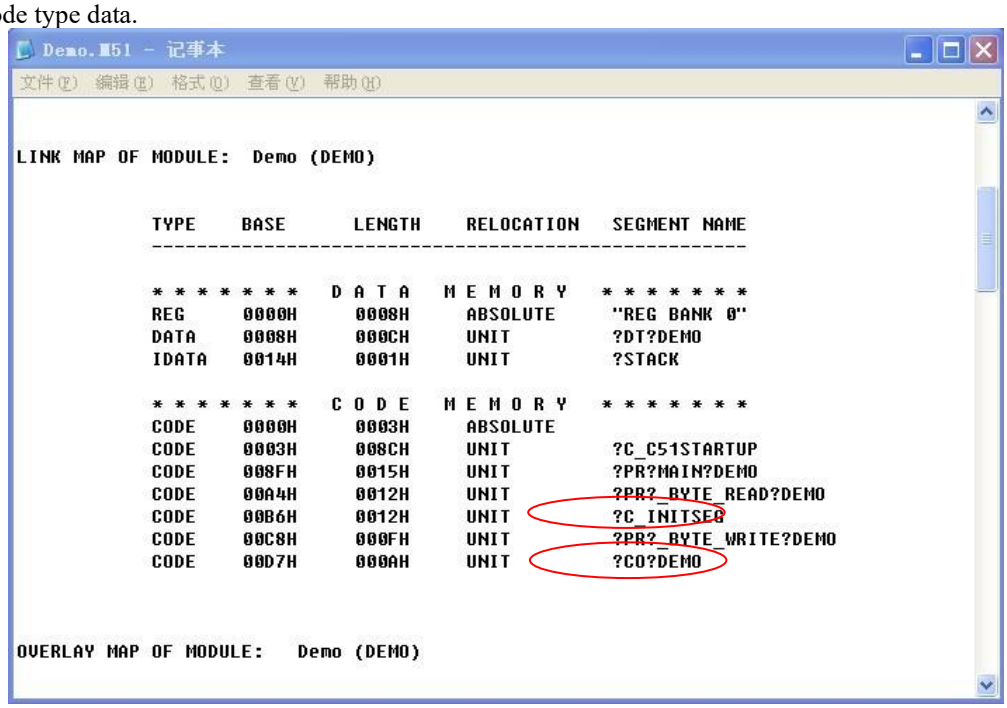
When the user needs to store codes before 0X100, global variables and immutable type data (code type data) can not be stored in this area because it is unable to read the data before 0X100; if there are global variables of code type in the programming codes, it is required to store these data types after 0x100 address.

The setting methods are as follows:

1) Cancel Step 1 to set the Code Store Area as Global Area and delete Code Range data, then click OK to save.



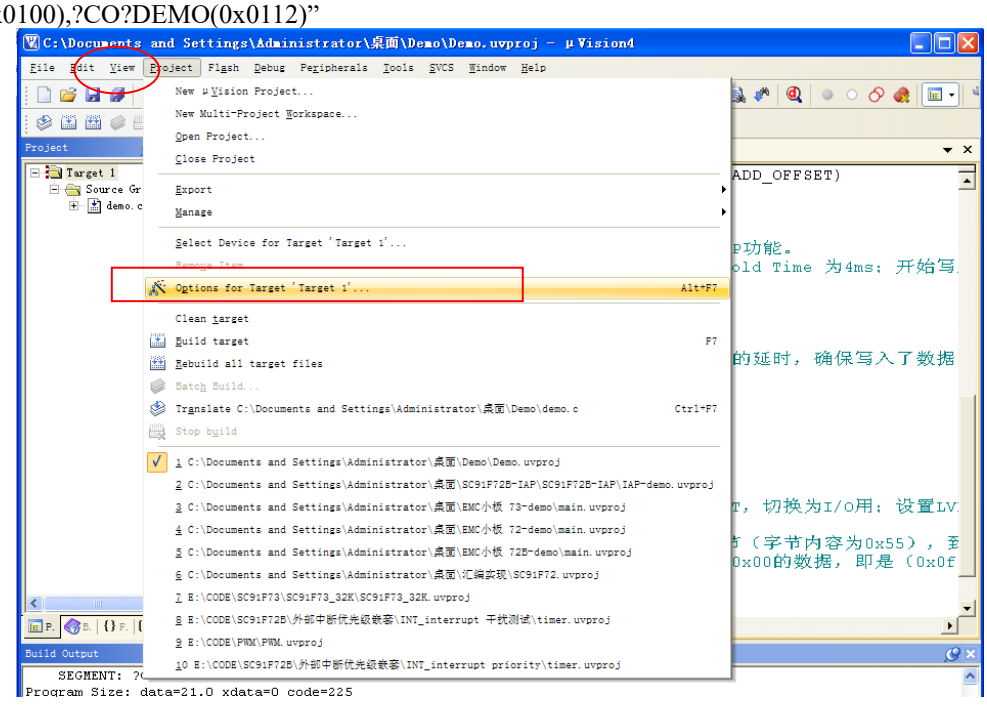
2) After recompiling, find and open .M51 file from the created project directory, then CODE MEMORY appears:
"?C_INITSEG": Initialized data of global variables.
"?CO?DEMO": code type data.

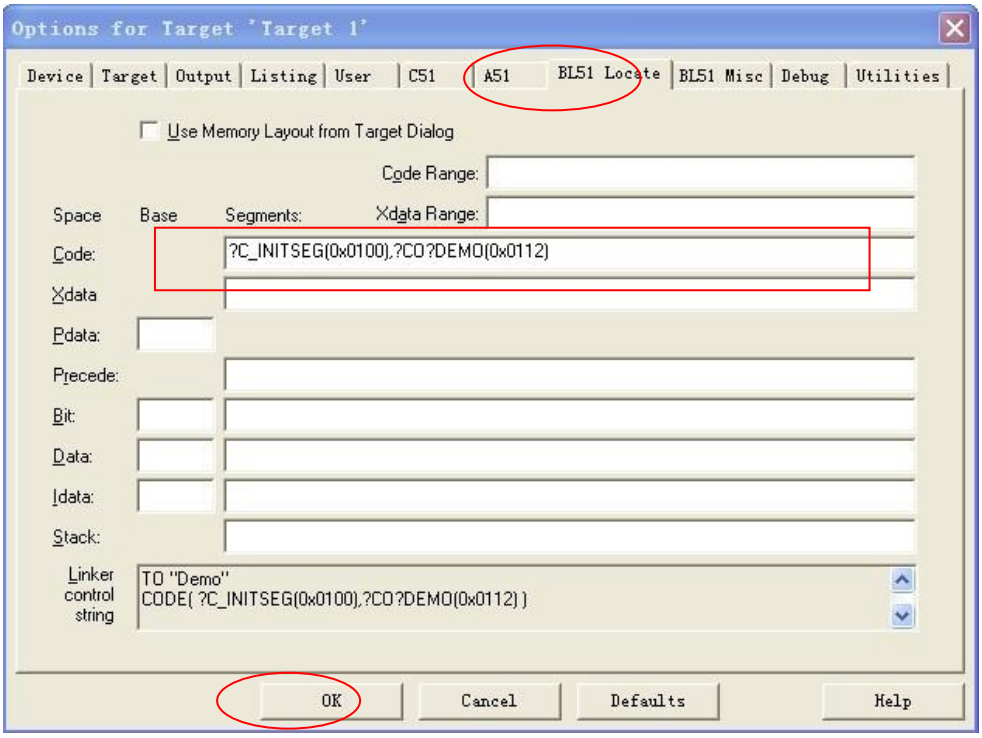



Note: From the "CODE MEMORY" information in the M51 file above, you can see "?C_INITSEG" with the link address of 00B6H and the length of 0012H bytes and "?CO?DEMO" with the link address of 00D7H and the length of 000AH bytes.

1) Calculate the relocation address respectively according to the length of "?C_INITSEG" and "?CO?DEMO":
The relocation address of "?C_INITSEG" is 0x0100
The relocation address of "?CO?DEMO" is 0x0112

2) Open the "BL51Locate" property page in the Project Option, and input the following statement in "Code" field:
"?C_INITSEG(0x0100),?CO?DEMO(0x0112)"






3) Click OK button and recompile to generate the final program.

- **Set up RAM clearing range.**

SinOne SC92F series MCU has 256B of internal RAM and 256B of external RAM (different models have different sizes of external RAM, refer to corresponding specifications in detail). After the microcontroller unit is reset, if it is required to clear all RAMs, modify the corresponding value in STARTUP.A51, open and set up STARTUP.A51 according to the area size of chip RAM, as shown in the figure, it is to clear 256Byte data in IDATA area and XDATA area of RAM.

```
17  ;
18  ;     Lx51 your object file list, STARTUP.OBJ  controls
19  ;
20  ;------------------------------------------------------------------------------
21  ;
22  ;  User-defined <h> Power-On Initialization of Memory
23  ;
24  ;  With the following EQU statements the initialization of memory
25  ;  at processor reset can be defined:
26  ;
27  ; <o> IDATALEN: IDATA memory size <0x0-0x100>
28  ;     <i> Note: The absolute start-address of IDATA memory is always 0
29  ;     <i>       The IDATA space overlaps physically the DATA and BIT areas.
30  IDATALEN        EQU     100H
31  ;
32  ; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
33  ;     <i> The absolute start address of XDATA memory
34  XDATASTART      EQU     0
35  ;
36  ; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
37  ;     <i> The length of XDATA memory in bytes.
38  XDATALEN        EQU     100H
39  ;
40  ; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
41  ;     <i> The absolute start address of PDATA memory
42  PDATASTART      EQU     0H
43  ;
44  ; <o> PDATALEN: PDATA memory size <0x0-0xFF>
45  ;     <i> The length of PDATA memory in bytes.
46  PDATALEN        EQU     0H
47  ;
48  ;</h>
49  ;------------------------------------------------------------------------------
```

## 3.2 Precautions for MOVC Instructions in Assemble Language Programming

Similarly, during the process of assemble programming, note that the custom ROM area data is defined after 0x0100. Operation method is relatively simple, which is to locate through ORG.

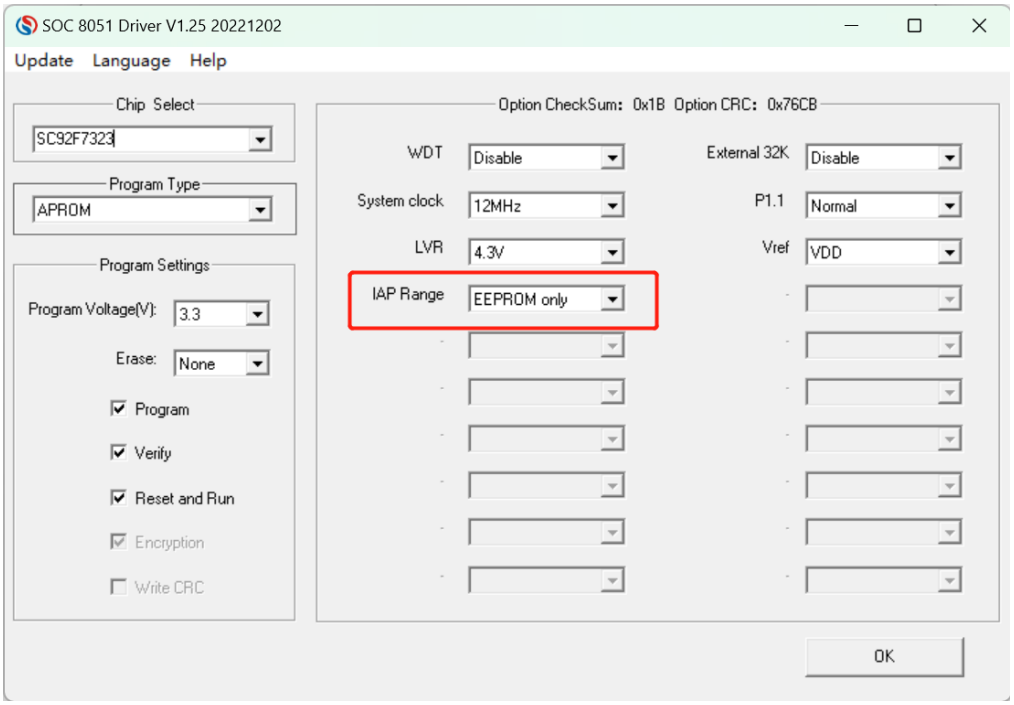
# 4. SinOne SC92F Series MCU EEPROM and Algorithm Explanation

## 4.1 Precautions for Internal EEPROM Operations and Code IAP Operations

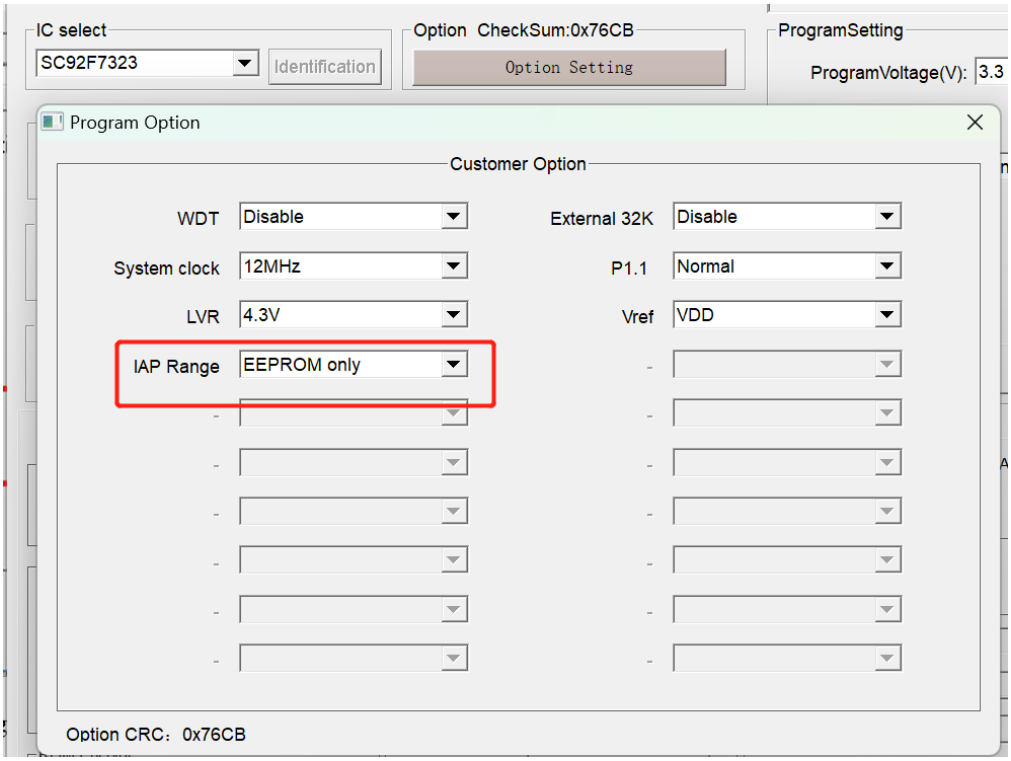
Take SC92F7323 as an example to explain SinOne MCU internal EEPROM use method and CODE IAP operation method.

SC92F7323 internal 8K Flash can be In Application Programming (IAP) operation, that is to allow the user program to dynamically write data into the internal Flash and have an independent 128Byte EEPROM.

When using IAP, it is required to set the range of IAP operations in the Option. The setting method in Keil is as follows: Select the range of IAP operations in IAP Range option.




Upon the upper computer programming, select the range of IAP operations in IAP Range of Option, as shown in the figure.




- **EEPROM read and write features:**

1. By Byte operation. Write and read one byte by one byte.
2. Similar RAM read and write: No need to erase before writing.
   Flash structure: Erase before writing.
   (Flash structure for SC92F8x8x, SC92F7x8x,SC92F8x9x and SC92F7x9x, similar RAM read and write for other models)

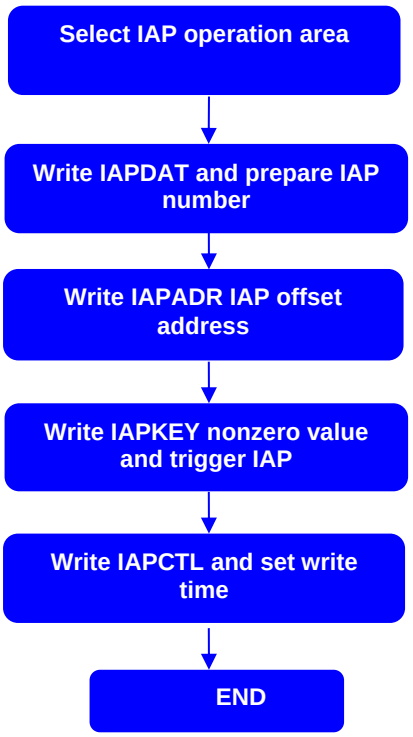
- **EEPROM Service Life: More than 100,000 times.**
- **You are advised not to erase more than the rated programming times of EEPROM, otherwise, exception will occur!**
- **If you perform IAP write operation after EEPROM is beyond its service life, the CPU Hold Time will become infinite, and you can not exit IAP mode even if the WDT is enabled. (Excluding SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x)**
- **FLASH read and write features:**

1. By Byte operation. Write and read one byte by one byte.
2. Similar RAM read and write: No need to erase before writing.
   Flash structure: Erase before writing.
   (Flash structure for SC92F8x8x, SC92F7x8x,SC92F8x9x and SC92F7x9x, similar RAM read and write for other models)

- **FLASH Service Life: More than 10,000 times.**
- **You are advised not to erase more than the rated programming times of EEPROM, otherwise, exception will occur!**
- **If you perform IAP write operation after EEPROM is beyond its service life, the CPU Hold Time will become infinite, and you can not exit IAP mode even if the WDT is enabled. (Excluding SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x)**

**IAP Write Process:**

Specify an address for writing each byte; specific IAP write process is as follows:




## 4.2 EEPROM Operations and Code IAP Operations Codes

SC92F series chips are not allowed to respond to external interrupt during IAP operations. Therefore, it is necessary to close the main repeater (EA=0) before performing related operations; restore the main repeater after the IAP operation is completed.

When using an **independent EEPROM** to perform IAP operations, make sure the IAPADE points back to CODE area after the operations are completed, so as to avoid program run-away.

1. SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x IAP operation routines:
   The head file shared by the following routine is as follows:

```
#include ”intrins.h”
unsigned int IAP_Add;
unsigned char IAP_Data;
unsigned char code * POINT =0x00;
```

**IAP operation: Sector erase**

```
EA = 0;                                  //Close main repeater
IAPADE = 0x00;                           //The extended address is 0x00, select Flash ROM
IAPADH = (unsigned char)((IAP_Add >> 8)); //Write high value of IAP target address
IAPADL = (unsigned char)IAP_Add;         //Write low value of IAP target address
IAPKEY = 0xF0;
IAPCTL = 0x20;                           //Set up sector erase bit
IAPCTL |= 0x02;                          //Perform block erase
_nop_(); //Wait (at least 8 _nop_()s)
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
```

```
_nop_();
EA = 1;                                  //Open main repeater
```

**IAP operation: Write data**

```
EA = 0;                                  //Close main repeater
IAPADE = 0X00;                           //The extended address is 0x00, select Flash ROM
IAPDAT = IAP_Data;                       //Send data to IAP data register
IAPADH = (unsigned char)((IAP_Add >> 8)); //Write high value of IAP target address
IAPADL = (unsigned char)IAP_Add;         //Write low value of IAP target address
IAPKEY = 0xF0;
//This value can be adjusted according to actual conditions; make sure that after executing this instruction and before assigning IAPCTL,
//the interval shall be less than 240 (0xF0) system clocks, otherwise, IAP function will be disabled;
//Take special care when opening the repeater
IAPCTL = 0X10;                           //Set up IAP write operation bit.
IAPCTL |= 0X02;                          //Perform write instruction
_nop_();                                 //Wait (at least 8 _nop_()s)
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
EA = 1;                                  //Open main repeater
```

**IAP operation: Read data**

```
EA = 0;                                  //Close main repeater
IAPADE = 0X00;                           //The extended address is 0x00, select APROM
IAP_Data = *( POINT+IAP_Add);            //Read IAP_Add value to IAP_Data
EA = 1;                                  //Open main repeater
```

2. IAP Operation Routine of Other Models

**Independent EEPROM Operation Routine**

```
#include ”intrins.h”
unsigned char EE_Add;
unsigned char EE_Data;
unsigned char code * POINT =0x0000;
```

**EEPROM write to operate C-language Demo program:**

```
EA = 0;                  //Close main repeater
IAPADE = 0X02;           //Select EEPROM area
IAPDAT = EE_Data;        //Send data to EEPROM data register
IAPADH = 0x00;           //High addres is 0x00 by default
IAPADL = EE_Add;         //Write low value of EEPROM target address
IAPKEY = 0XF0;           // This value can be adjusted according to actual conditions; make sure that after executing this instruction and before assigning IAPCTL,
// the interval shall be less than 240 (0xF0) system clocks, otherwise, IAP function will be disabled;
// Take special care when opening the repeater
IAPCTL = 0X0A;           //Perform EEPROM write operation, 1ms@24M/12M/6M/2M;
_nop_();                 // Wait (at least 8 _nop_()s)
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
```

```
_nop_();
_nop_();

IAPADE = 0X00;                  //Return ROM area
EA = 1;                 //Open main repeater
```

**EEPROM read to operate C-language Demo program:**

```
EA = 0;                         //Close main repeater
IAPADE =  0X02;                         //Select EEPROM area
EE_Data = *( POINT +EE_Add);        //Read IAP_Add value to IAP_Data
IAPADE =  0X00;                         //Return ROM area to avoid operating MOVC to EEPROM
EA = 1;                         //Open main repeater
```

**CODE area IAP Operation Routine**

```
#include ”intrins.h”
unsigned int IAP_Add;
unsigned char IAP_Data;
unsigned char code * POINT =0x0000;
```

**IAP write to operate C-language Demo program:**

```
EA = 0;                                 //Close main repeater
IAPADE = 0X00;                                  //Select Code area
IAPDAT = IAP_Data;                              //Send data to IAP data register
IAPADH = (unsigned char)((IAP_Add >> 8));       //Write high value of IAP target address
IAPADL = (unsigned char)IAP_Add;                //Write low value of IAP target address
IAPKEY = 0XF0;          // This value can be adjusted according to actual conditions; make sure that after executing this instruction and before assigning IAPCTL,
// the interval shall be less than 240 (0xF0) system clocks, otherwise, IAP function will be disabled;
// Take special care when opening the repeater
IAPCTL = 0X0A;          //Perform IAP write operation, 1ms@24M/12M/6M/2M;
_nop_();                // Wait (at least 8 _nop_()s)
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
EA = 1;                                 //Open main repeater
```

IAP read to operate C-language Demo program:

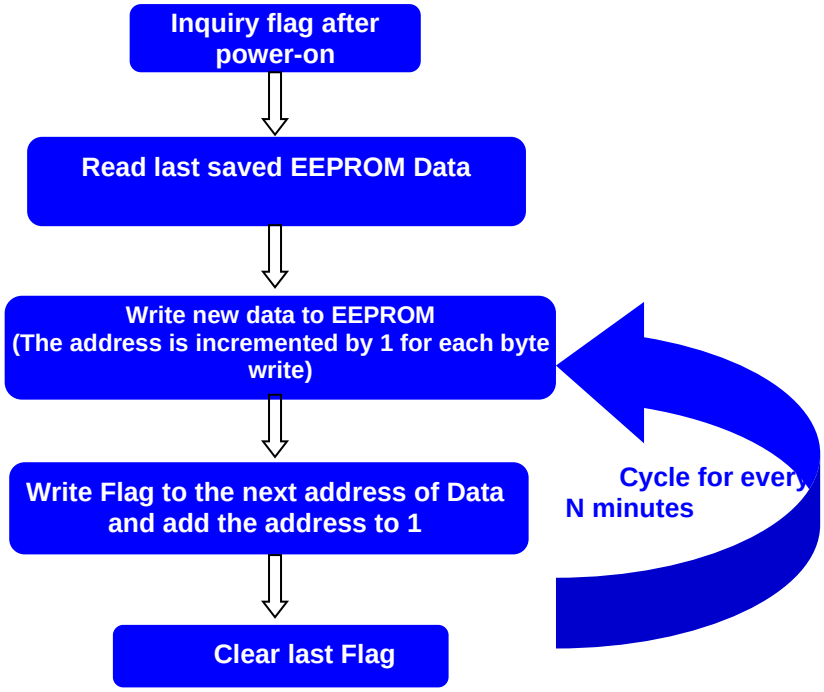
```
IAPADE = 0X00;          //Select Code area
IAP_Data = *( POINT+IAP_Add);//Read IAP_Add value to IAP_Data
```

Note: IAP operations in the Code area have certain risks, and the user needs to take corresponding security measures in the software. **Improper operations may cause user program to be rewritten!** This feature is not recommended unless it is required by the user (for remote program updates, etc.).

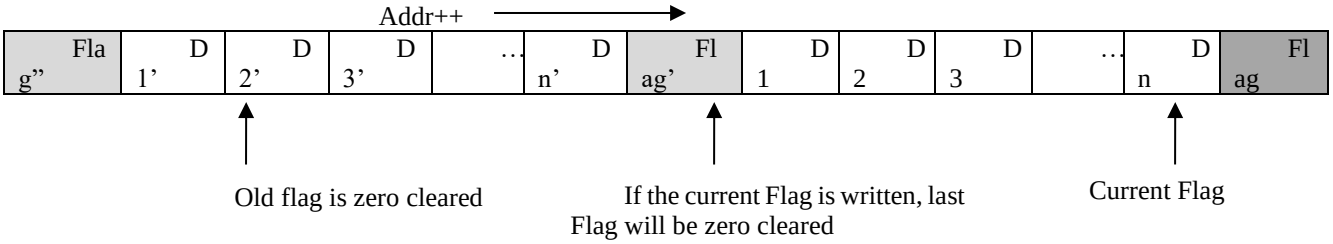
## 4.3 EEPROM Use Algorithm

As SC92F7323 has 8kB Flash and 128B EEPROM, it can be used for In Application Programming (IAP) operations; however, in actual product applications, such as electric pressure cooker, only a few Bytes of data need to be written to EEPROM. In order to make full use of all EEPROMs in MCU and prevent the service life of writing IAP from reaching too early, the following algorithm is provided for reference:

1. See the flow charge below:




2. Adopt the above algorithm and write EEPROM data, as shown below:




- **Features of above algorithms**

  ① Make full use of all EEPROMs in the MCU;

  ② **The algorithm is more robust, the data stored in EEPROM will not be destroyed by the change of power factors;**

  ③ The efficiency of the algorithm is high. 256B EEPROM can store 256/(N+1) times of data. N refers to the number of bytes to be written to the internal EEPROM;

  ④ If N+1 can not be divisible by 256 (N refers to the number of bytes to be written to the internal EEPROM), the service life of EEPROM can be performed to full potential; or else, the fixed address (Flag address) in the EEPROM will be written once more, and the service life of EEPROM will be reduced. Therefore, if N+1 can be divisible by 256, it is recommended to write an extra byte of empty data to EEPROM;

  ⑤ **Make sure that the Flag is unique**, and the selected Flag is different from each Data written to EEPROM.

- **Adopt the above-mentioned algorithm and implement the following demo program (For SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x, it is required to modify corresponding EEPROM write operation code) for reference:**

```
/*********************************************************************/
/*****This Demo uses the last 256B CODE area of SC92F7323 as power-down storage area (as EEPROM)*****/
//This Demo is a clock demo program with 4 Bytes data (Days, Hours
//Minutes, Seconds) Write internal EEPROM for every 3 minutes
/*********************************************************************/
#include "SC92F7323_C.H"
#include "Hex2Bin.h"
#include "intrins.h"
```

```
void display_shifen(void) ;                          //Digital display clock minutes
void Byte_Write(unsigned char DATA,unsigned char ADD_OFFSET); //IAP writes data functions
unsigned char Byte_Read(unsigned char AddOffset );//IAP reads data functions

#define uchar unsigned char                          //Simplify unsigned character
#define uint unsigned int                            //Simplify unsigned integer
#define   ADD_BASE   0x1f00                          //Define base address of IAP [Confirm the base address based on different types of IC of SOC]
/*****************************************/
/*****************************************/
/***Data to be stored in EEPROM, 4 Bytes*******/
uchar nSec;
uchar nMin;
uchar nHour;
uchar  nday;
/*****************************************/
/*****************************************/
uchar  ADD_OFFSET=0;                    //Offset address
uchar  code *POINT;                          //Define a pointer
uint offset,min3;

uint TusCounter;
uint nMinG;
uint nMinS;
uint nHourG;
uint nHourS;
uint nSecG;
uint nSecS;

uchar code chZimo [10]={0xc0,0xf9,0x64,0x70,0x59,0x52,0x42,0xf8,0x40,0x50};  //Store character die

/*****************************IAP writes data function*****************************/
void Byte_Write(unsigned char DATA,unsigned char Add_Offset)
{
EA = 0;                                 //Close main repeater
IAPDAT=DATA;                            //Send DATA to IAP data register
        if(Add_Offset>255)
        {
                ADD_OFFSET=0;
                Add_Offset=0;
        }
        IAPADH=0x1f;
        IAPADL =Add_Offset;                     //Write offset address;
        IAPKEY=0x09;                            //Write any non-zero value, enable IAP function.
        IAPCTL=0x0a;                            //Perform IAP write operation, and CPU Hold 1ms.
        _nop_();_nop_();_nop_();_nop_();
        _nop_();_nop_();_nop_();_nop_();   //Each time IAP data is written, there is a delay of 8 nops,
                                                To make sure the data is written.
EA = 1;                                 //Open main repeater
}
/************************IAP reads data functions************************/
unsigned char Byte_Read(unsigned char AddOffset )
{
        POINT=ADD_BASE+AddOffset;               //POINT points offset address;
        return (*POINT);                    //Return point contents, read is succeeded.
}
//Timer0 working mode 2-8-bit automatic reloading counter/timer: timing 50us
void timer0init()
{
        TMCON=_b00000001;               //fsys=fosc/4
        TMOD=_b00000010;                        //Mode 2
```

```
/*Load initial value********************Timing 50us
  200*(1/4us)=50us; initial value=（2^8-200)=56
  56=0x1060=_b 0011 1000
  High 8-bit  10000011=0x38
  ******************************************/
        TH0=0x38;
        TL0=0x38;
  /*Enable and start Timer*/
        TR0=0;
        ET0=1;
        TR0=1;
}
/********************Software Delay*********************/
void soft_delay(unsigned char n)
{
unsigned char k;
        for(k=0;k<n;k++)
                _nop_();
}
void display_shifen(void)
{
//Display one place of minutes
P1=chZimo[nMinG];
P21=1;
P20=0;
P07=0;
soft_delay(800);                    //Soft Delay
//Display ten place of minutes
P1=chZimo[nMinS];
P21=0;
P20=1;
P07=0;
soft_delay(800);                    // Soft Delay
//Display one place of hours
P1=chZimo[nHourG];
P21=0;
        P20=0;
        P07=1;
soft_delay(800);                    // Soft Delay
}

void PRA_Write(void)                        //Write data to EEPROM
{
        Byte_Write(nSec,ADD_OFFSET++);      //Write one Byte to EEPROM
        Byte_Write(nMin,ADD_OFFSET++);      // Write one Byte to EEPROM
        Byte_Write(nHour,ADD_OFFSET++);     // Write one Byte to EEPROM
        Byte_Write(nday,ADD_OFFSET++);      // Write one Byte to EEPROM
        Byte_Write(255,ADD_OFFSET++);       //Write mark 0xff;
        if(ADD_OFFSET==0)
                Byte_Write(0,250);          //Clear last mark as 0;
        if(ADD_OFFSET==1)
                Byte_Write(0,251);          // Clear last mark as 0;
        if(ADD_OFFSET==2)
                Byte_Write(0,252);          // Clear last mark as 0;
        if(ADD_OFFSET==3)
                Byte_Write(0,253);          // Clear last mark as 0;
        if(ADD_OFFSET==4)
                Byte_Write(0,254);          // Clear last mark as 0;
        if(ADD_OFFSET==5)
                Byte_Write(0,255);          // Clear last mark as 0;
        if(ADD_OFFSET>5)
```

```
                Byte_Write(0,(ADD_OFFSET-6));// Clear last mark as 0;
}

void PRA_Read(void)                         //Read data written to EEPROM before power-down
{
        if(ADD_OFFSET==0)
        {
                nSec=Byte_Read(256-4);
                nMin=Byte_Read(256-3);
                nHour=Byte_Read(256-2);
                 nday=Byte_Read(256-1);
        }
        if(ADD_OFFSET==1)
        {
                nSec=Byte_Read(256-4+1);
                nMin=Byte_Read(256-3+1);
                nHour=Byte_Read(256-2+1);
                nday=Byte_Read(0);
        }
        if(ADD_OFFSET==2)
        {
                nSec=Byte_Read(254);
                nMin=Byte_Read(255);
                nHour=Byte_Read(0);
                nday=Byte_Read(1);
        }
        if(ADD_OFFSET==3)
        {
                nSec=Byte_Read(255);
                nMin=Byte_Read(0);
                nHour=Byte_Read(1);
                nday=Byte_Read(2);
        }
        if(ADD_OFFSET>=4)
        {
                nSec=Byte_Read(ADD_OFFSET-4);
                nMin=Byte_Read(ADD_OFFSET-3);
                nHour=Byte_Read(ADD_OFFSET-2);
                nday=Byte_Read(ADD_OFFSET-1);
        }
}

void timer0()interrupt 1
{
        TH0=0x38;
        TusCounter++;
        if(TusCounter==20000)       //1s
                {
                        TusCounter=0;
                        nSec++;
                        P36=~P36;               //Flash once for every 1s
                        if(nSec>59)
                    {
                                nSec=0;
                                nMin++;                                 min3++;      //min3  increment  1  upon
running for every 1 minute
                                if(nMin>59)
                                {
                                        nMin=0;
                                        nHour++;
                                        if(nHour>23)
```

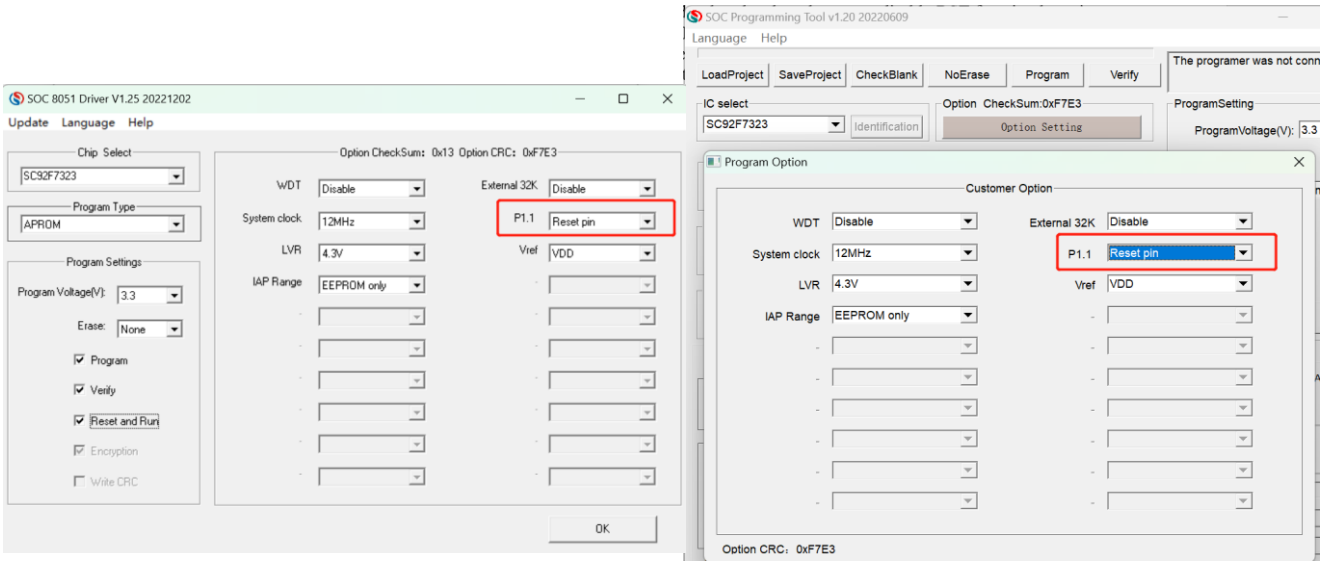
```
                                        {
                                                nHour=0;
                                                nday++;
                                        }
                                        if(nday>9)
                                        {
                                                nday=0;
                                        }
                                }
                        }
                }
        //Take seconds
        nSecS=nSec/10;
        nSecG=nSec%10;
        //Take minutes
        nMinS=nMin/10;
        nMinG=nMin%10;
        //Take hours
        nHourS=nHour/10;
        nHourG=nHour%10;
}

/**********************Main Program**********************/
void main()
{
        timer0init();
        EA=1;
        for(offset=0;offset<256;offset++)    //Inquiry mark 0xff
                if(Byte_Read(offset)==255)
                        ADD_OFFSET=offset;

        PRA_Read();                             //Read the data written to EEPROM before power-down
        do
        {
                display_shifen();                 //Display clock minute table
                if(min3>3)
                {
                        min3=0;
                        PRA_Write();          //Write data once for every 3 minutes.
                }
        }
        while(1);
}
```

# 5. Precautions for Circuit Design

## 5.1 Circuit Design Examples

The default GPIO power-on mode of SinOne SC92F series MCU is high resistance input mode.

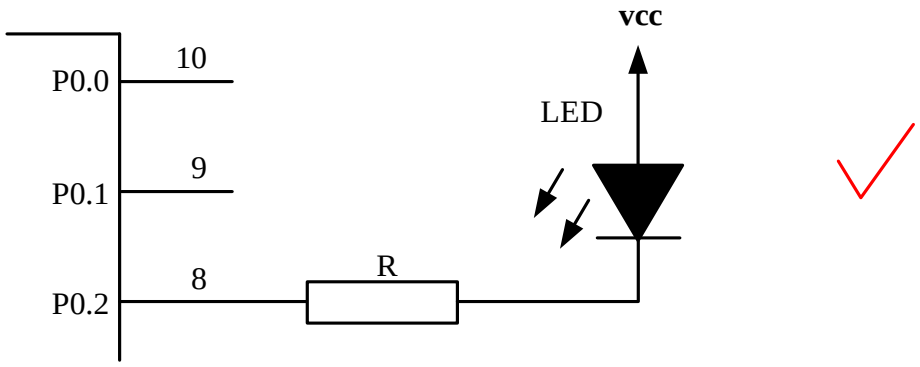
The RST pins of SinOne SC92F series MCU is enabled at low level, and you can disable RST function by using corresponding pin control option in Option to cancel RESET function and set Pin to GPIO. The pin will not perform reset operation in low level. Option facilities are shown in the figure (by taking SC92F7323 as an example). The setting situations of KEIL and programming upper computer are on the left and right, respectively.




### 5.1.1 Circuit Design Instances

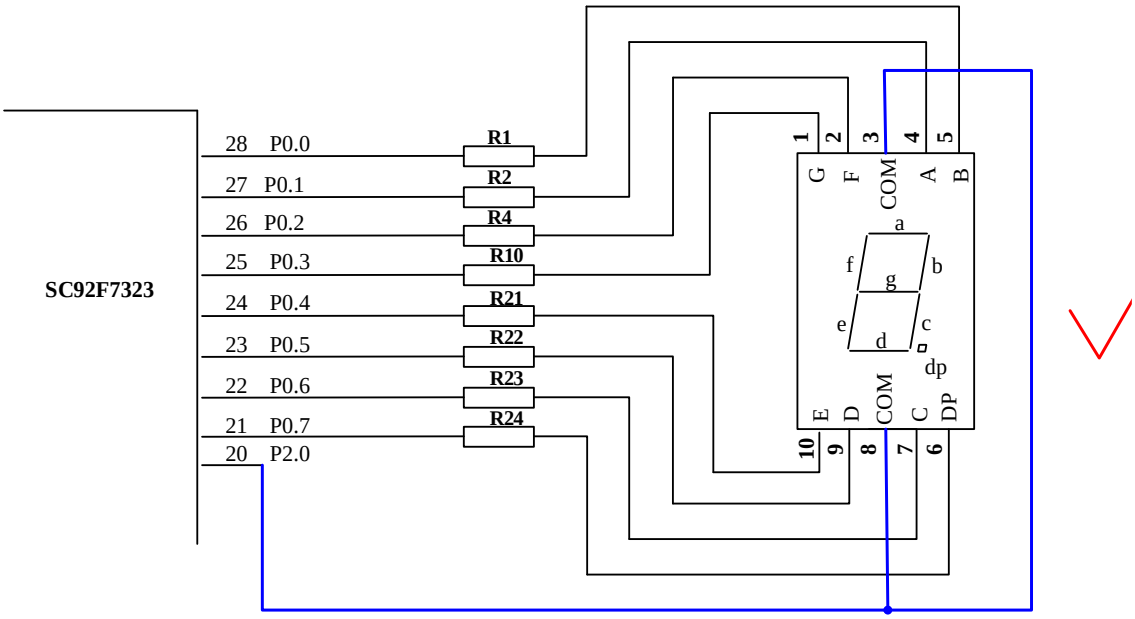
### 5.1.2 LED Use and Connection

- **Recommended connection: LED positive access to VCC and negative access to I/O, as shown in the figure below:**



### 5.1.3 Use of 1-digit Common Cathode Digital Tube

- **Recommended connection: as shown in the figure below.**

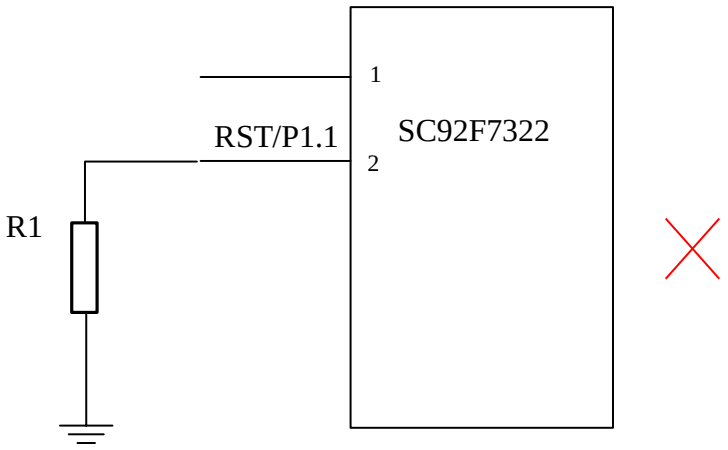



Note: As for other multi-digit digital tubes, there is no special requirements for the connection method, just follow the normal connection method, because the multi-digit digital tubes are independent of the default mode of MCU I/O power-on.

### 5.1.4 RST Pin Circuit

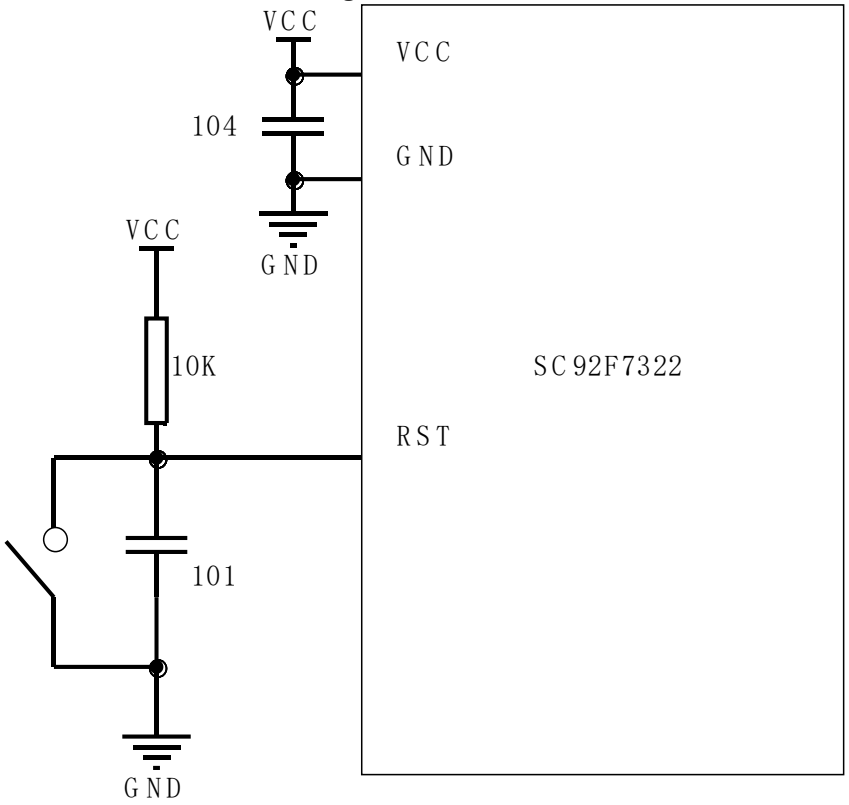
The RST pin of SC92F series MCU, which is multiplexed with I/O, is different from that of the traditional MCU (the latter can only be used for input), which can be used for both input and output. It should be noted that, for chips of SC92F7323, SC92F7322, SC92F7321, and SC92F7320, no matter whether the RST port in Option is set to RST mode or IO mode, the RST port will be set to reset mode by default after being powered on and then set to the corresponding mode. Therefore, when the RST port is used as the common IO port for these chips, make sure that the level of RST port is not low when the chips are powered on. After the IO port is set as the reset port, the RST port of the user circuit can not be always in low level after the device is powered on. Otherwise, the user circuit will always be in reset mode and can not work normally. Therefore, upon designing the circuit, the user needs to pay attention to:

- **Wrong connection**




Note: For the above circuit, if RST is connected with an external resistor R1, the system level will low when it is powered on, resulting in constant reset of the system and failure to work normally.

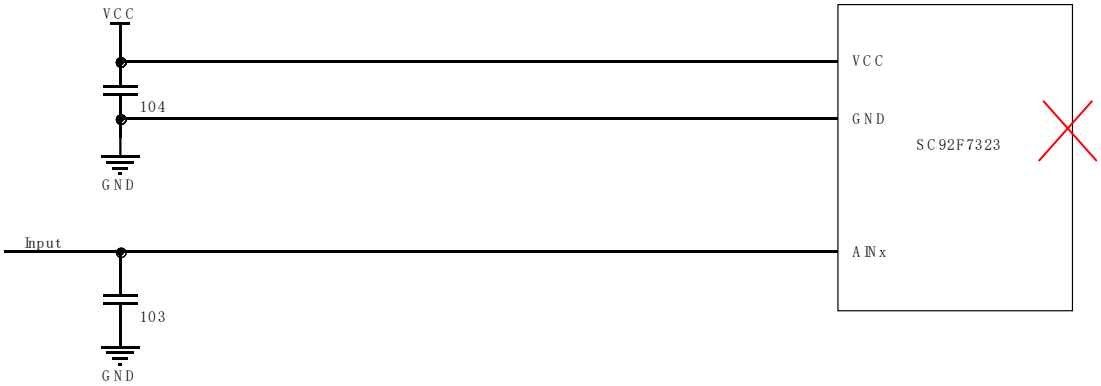
- **Recommended connection: as shown in the figure below.**



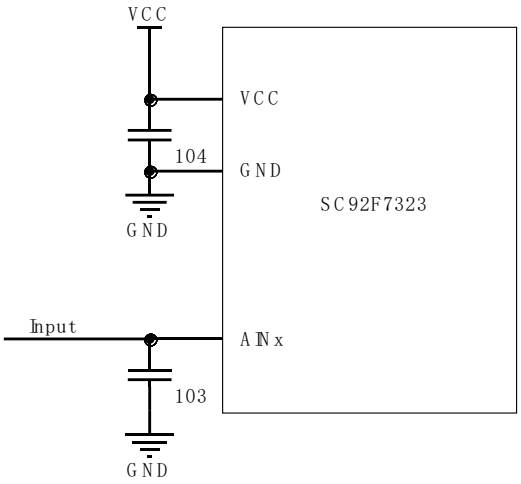

### 5.1.5 ADC Sampling Pin Circuit

For ADC sampling port of SinOne SC92F series MCU, 103 capacitor shall be added near the pin. The voltage of the power supply shall be stable for ADC conversion. Therefore, upon using ADC functions, 104 capacitor shall be added near the VCC and GND of the IC to guarantee accurate conversion results.

- **Wrong connection:** The 104 capacitor is too far away from the power pin, and the 103 capacitor is too far away from the ADC sampling port

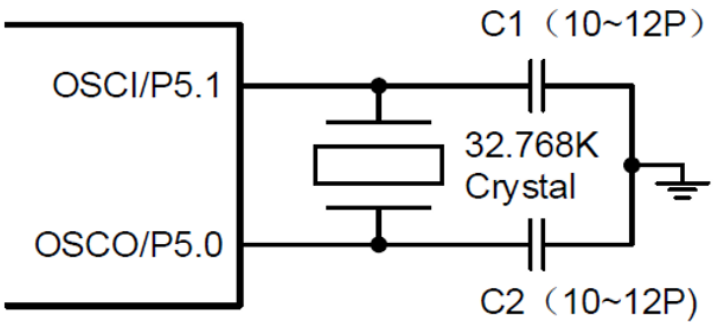



- **Recommended connection:** The 104 capacitor is close to the power pin and the 103 capacitor is close to the ADC sampling port, as shown in the figure below:




### 5.1.6 External Crystal Oscillation Circuit

Some models of SinOne SC92F series MCU provide high-frequency external crystal oscillation interface or low-frequency external interface. If the user needs to use the external crystal oscillation interface, the mating capacitor shall be selected according to the requirements of the selected crystal oscillation, which shall be close to the chip pin, as shown in the figure:




32.768K External Crystal Connection Diagram

### 5.1.7 Touchkey Circuit

The TouchKey architecture of SinOne MCU is divided into high-sensitivity TouchKey mode and high-reliability TouchKey mode.

The capacitance of external CMOD in high-sensitivity TouchKey mode ranges from 472 to 104. 103 capacitor is recommended, and there is no special requirements for the capacitor material.

The capacitance of external CMOD in high-reliability TouchKey mode ranges from 332 to 473. 473 capacitor is recommended. It is recommended to use the capacitor with small temperature coefficient and high accuracy to avoid inconsistency sensitivity or changes with the temperatures. For general plug-in capacitor, the polyester capacitor in 5% precision is recommended; for patch capacitor, NPO or X7R capacitors in 10% or higher precision are recommended.

CMOD capacitor shall be as close to the chip pin as possible.

## 5.2 Precautions for Mode Setting of IO Port

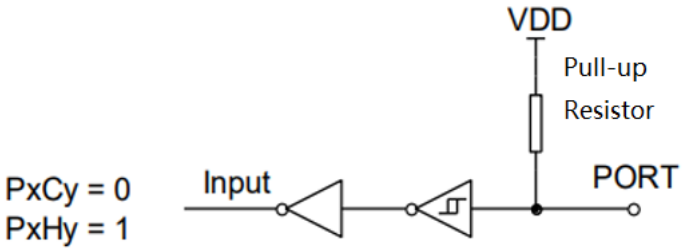
For SinOne SC92F series MCU GPIO, there are three working modes:
1. Pull-up input mode
2. High-resistance input mode
3. Strong push-pull output mode

### 5.2.1 Set I//O as High Resistance for Circuit Design

In general, some specific applications, such as voltage detection, zero-crossing detection, LCD application, etc., are adopting high-resistance mode. Therefore, the user can choose from MCU system on demand.

### 5.2.2 Pull-up Input Mode

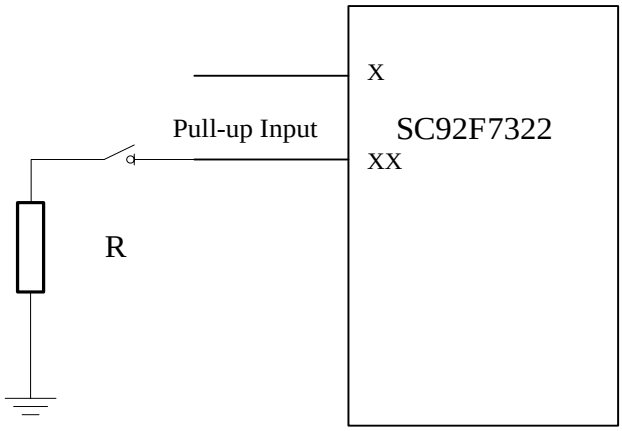
In pull-up input mode, the input port is constantly connected to a pull-up resistor, and the low-level signal will be detected only when the pull-up level of the input port is pulled low. The port structure of the pull-up input mode is shown below:




Pull-up Input Mode

### 5.2.3 Detection Keys of Pull-up Input Mode

When I/O port is used as key input, the pull-down resistance in series circuit R shall be less than 2K.




### 5.2.4 Implementation of I/O Open-drain Output Mode

There is no open-drain output setting options for SinOne SC92F series MCU. To enable the open-drain output function

of IO port, the user needs to switch the mode to achieve the open-drain output effect. When the pin output is low, the user needs to switch to the strong push-pull output mode; when the pin is suspended, the user needs to switch IO port to the high-resistance input mode.

The code examples are as follows:

```
P0PH &= 0XFE;  //Remove P00 pull-up resistance
P00 = 0;             //Output P00 as 0
P0CON &= 0xfe;//Set P0 as input mode, equivalent to open-drain mode of open-drain output
P0CON |= 0x01;//Set P0 as output mode, equivalent to low open-drain output
```

### 5.2.5 Precautions for I/O Read IO Functions

Most models of SinOne SC92F series MCU feature Read IO function, which is enabled by default. Even if GPIO is in output mode, the read port data register will return high and low values based on the level on the IO pin as well. If the Read IO function is turned off, the read port data register in output mode will not depend on the level on IO pin, but will read the value of the register. It is recommended to disable the Read IO function during the initialization. (Note! For earlier models including SC92F7X2X, SC92FXX5X, SC92FXX4X and SC92FXX7X, the Read IO function is in OFF state and this operation is not required). But the input mode will not be affected.

Code examples are as follows:

```
//Disable Read IO function
OPINX = 0X86;
OPREG&=0XF7 ;
```

# 6. Precautions for Software Programming

SinOne SC92F series MCU is equipped with abundant peripherals, which can be operated as long as corresponding register is configured. However, some operations need to be performed in accordance with the requirements. The following points shall be paid attention to during the process of user programming.

## 6.1 Precautions for PWM Setup and Use

To turn off the PWM in the application, the user needs to set the output register of IP port corresponding to the PWM as “1” or “0” according to the actual application requirements (when the PWM is turned off, the output of IP port changes from PWM to GPIO port, and the output of IO port will be in an uncertain state at this time).

The PWM of SinOne SC92F series MCU has three kinds of precision, namely 12-bit, 10-bit and 8-bit (please refer to the specification of corresponding model for specific precision descriptions). The PWM in both 12-bit precision and 10-bit precision features shared cycle and independently set duty ratio, but the PWM shares the duty ratio setting of last 2 bits.

When using 10-bit PWM, configure the last 2 bits and then the remaining 8 bits so as to guarantee the accuracy of PWM setting.

```
PWMDTYA = 0x00;          //First configure the last 2 bits of PWM
PWMDTY0 = 50;            //Then configure the remaining 8 bits of PWM
PWMDTY1 = 45;
PWMDTY2 = 40;
PWMCON0 = 0x38;          //First configure the last 2 bits of PWM cycle
PWMPRD  = 59;            //The configure the remaining 8 bits of PWM cycle
```

## 6.2 Precautions for PCON Register Setup

SinOne SC92F series MCU provides the power management function, which can make the chip in power saving mode and just operate the corresponding options of PCON register. However, after operating the PCON register, please connect at least 8 NOP instructions after its configuration instruction, otherwise, the program will go wrong.

Use examples are as follows:

```
#include "SC92F7323_C.H"
#include "intrins.H"

PCON |= 0X02;                    //Enter STOP mode, and connect 8 NOPs
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
```

Do not enable STOP and IDLE at the same time!

## 6.3 Precautions for Checksum Setup

Part of SinOne SC92F series MCUs features built-in Check Sum function, which can be used to generate the 16-bit Check Sum value for the program code in real time. The user can compare the Check Sum with the theoretical value to monitor whether the contents in the program are correct. Before operating, close EA, and then open EA after the operation, while connect at least 8 NOP instructions after its configuration instruction when operating OPERCON register, otherwise, the program may go wrong.

Use examples are as follows:

```
#include "SC92F846X_C.H"
#include "intrins.H"
EA = 0;                          //Close main repeater
OPERCON|= 0X01;                  //Perform check sum algorithm and connect 8 NOPs
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
EA =1;                           //Open main repeater
```

## 6.4 Precautions for UART0 Setup and Use

Part of SinOne SC92F series MCUs has similar baud rate configuration mode with traditional 51 microcontroller unit. When using UART0, it is necessary to set TR1 to 1 when using Timer 1 as the baud rate generator; however, if TIMER1 is selected as the baud rate generator when other models of the microcontroller unit of SinOne SC92F series adopts UART0, the Timer 1 must stop counting (TR1=0). When using UART0, it is required to set the corresponding TX port to the pull-up input mode so as to guarantee that the TX port is in high level in idle time.

Use examples are as follows:

```
P1CON &=0XDF;  //Set P15(TX0) as input mode.
P1PH |= 0x20;      //Add pull-up resistor to P15(TX0);
SCON   = 0X50;    //Set communication mode as Mode 1, allow to receive
T2CON &= 0XCF; //Select T1 as the baud rate generator
TR1 = 0;                  //Set Timer 1 as the baud rate generator, and Timer 1 must stop counting
TH1 = 0x06;         //In 16M, the baud rate is 9600; initial value of timer [TH1,TL1] = Fsys/baud rate
TL1 = 0x82;         //In 16M, the baud rate is 9600
EUART = 1;          //Enable Uart0 interrupt
```

The UART0 of SinOne SC92F series MCU (excluding SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x) can not directly send the value of SFR register. To send the value of SFR through UART0, please first assign the value of SFR to a temporary variable, then assign such temporary variable to SBUF.

Use examples are as follows:

```
unsigned char BufTemp;
BufTemp=ADCVH; //Store a temporary variable for the SFR value to be sent
SBUF= BufTemp;   //Assign the temporary variable to SBUF and then send it out.
Or the user can write the sending process as a function and take the data to be sent as an input parameter for sending.
void Uart0Send(unsigned char data0)
{
        SBUF= data0;
}
Uart0Send(ADCVH); //Call the function to send SFR data.
```

## 6.5 Precautions for SPI/TWI/UART Universal Serial Port SSI Setup and Use

When using SSI function of SinOne SC92F series MCU, set the SSI port to pull-up input mode.

SSI TWI function of SinOne SC92F series MCU can only be used as the slave. If it is required to be used as a host, please implement it through the software simulation.

When Mode 3 of SSI UART1 is used, RB8 can only be set as 1. Therefore, after receiving data with Mode 3, RB8 needs to be cleared, as follows:

```
uint16_t SSI_UART1_ReceiveData9(void)
{
        uint16_t Data9;
        Data9 =  SSDAT + ((uint16_t)(SSCON0&0X04)<<6); //Get the received data
        SSCON0 &= 0XFB; //Clear RB8.
        return Data9;
 }
```

SSI UART sends the interrupt flag TI and receives the interrupt flag RI on the same register, which can not be operated bit by bit. Therefore, when TI and RI are cleared, the whole registered will be operated. In such case, when UART performs full duplex communication, the sending and receiving interrupt may occur at the same time, or the interval between them will be very short. There is a risk that TI or RI will be cleared by mistake resulting in loss of interrupt; therefore, in the application scenarios of full-duplex communication, a fault-tolerant mechanism is required to ensure that the communication will not be collapsed due to the loss of interruption. After the data is sent, it is unable to determine if the transmission is completed by waiting for the sending flag. Timeout monitoring is required to guarantee that the waiting can be stopped after a certain period of time, as shown below:

```
SSDAT =0x55;//Push the sent data 0X55 to the sent cache
i=0x8000;   //For timeout processing; change this variable to change the timeout length, and the user can adjust it according to the baud rate.
while(!SSI0SendFlag)
{
i--;
if(i==0)
{
        break;              //Time out and exit
}
}
    SSI0SendFlag = 0;      //Clear the sending flag.
```

## 6.6 Precautions for ADC Multi-channel Switch Acquisition

Most models of SinOne SC92F series MCU features multiple ADC channels, but only one channel can be converted each time. To achieve the acquisition of multi-channel ADC signals, the conversion port of one ADC channel shall be switched to another channel of ADC, so as to achieve multi-channel ADC conversion after repeated operations. If AD conversion is performed immediately after switching ADC channel, the voltage on the channel port line may be unstable and the first value converted after switching the channel may be abnormal, we suggest the user conducting several times of acquisition and conversion to a channel and removing the first value or several values converted after switching the channel or removing the maximum and the minimum value and obtaining the average value of the remaining AD switch

values to get the acquisition results.

Use examples are as follows:

```
unsigned int ADC_Value0,ADC_Value1,ADC_Value2;
unsigned int ADC_Convert(void)
{
    unsigned  int Tad=0,MinAd=0x0fff,MaxAd=0x0000,TempAdd=0;
    unsigned char t=0;
    for(t=0;t<10;t++)
    {
        ADCCON |= 0X40;          //Start ADC conversion
        while(!(ADCCON&0x20));
        //Wait for ADC conversion to complete, different models may complete the conversion at different flat bit, some in Bit5 and others in Bit4, refer to the specification for details.
        ADCCON&=~(0X20);      //Clear interrupt flag bit
        Tad = ((unsigned int)ADCVH<<4)+(ADCVL>>4);     //Get the conversion value
        if (Tad>MaxAd)
        {
            MaxAd=Tad       ;//Get the current maximum value
        }
        if (Tad<MinAd)
        {
            MinAd=Tad;      //Get the current minimum value
        }
        TempAdd+=Tad; //Accumulate the conversion values
    }
    TempAdd-=MinAd;         //Remove the minimum value
    TempAdd-=MaxAd;         //Remove the maximum value
    TempAdd>>=3;            //Get the average value
    return(TempAdd);
}

void ADC_channel(unsigned char channel)
{
     ADCCON = ADCCON &0xE0| channel;              //Select ADCchannel port for ADC input
}

void ADC_Multichannel()
{
    ADCCFG0 = 0x07;         //Set AIN0, AIN1, AIN2 as ADC port, and remove pull-up resistance automatically.
    ADCCON  |= 0X80;        //Turn on ADC module power

    ADC_channel(0);         //Switch ADC entrance to AIN0 port
    ADC_Value1 = ADC_Convert();     //Enable ADC conversion, get the conversion value

    ADC_channel(1);         //Switch ADC entrance to AIN1 port
    ADC_Value1 = ADC_Convert();     //Enable ADC conversion, get the conversion value

    ADC_channel(2);         //Switch ADC entrance to AIN2 port
    ADC_Value2 = ADC_Convert();     //Enable ADC conversion, get the conversion value
}
```

## 6.7 Precautions for Writing External Interrupt 0/1 Service Functions When Using the Timer

When an external interrupt 0/1 occurs in the user program after initialization (excluding SC92F732X, SC92F725X, SC92F8x8x, SC92F7x8x, SC92F8x9x and SC92F7x9x), if any operation is required to TR1, TR0, TF1, TF0 bit of TCON in the following process, it is required to manually clear the external interrupt flag in the external interrupt 0/1 service program.

Otherwise, the external interrupt flag bit may not be cleared by the hardware.

Use examples are as follows:

```
void EX0() interrupt        0
{
   TCON &= 0xFD;
}

void EX1() interrupt        2
{
   TCON &= 0XF7;
}
```

Upon using SC92F732X and SC92F752X, when an external interrupt 0/1 occurs in the user program after initialization, if any operation is required to TR1, TR0, TF1, TF0 bit of TCON in the following process, **only one of the external interrupt 0/1 can be used, and two external interrupt flags need to be manually cleared in the external interrupt service function**. Otherwise, the external interrupt flag bit can not be cleared by the hardware. (If the subsequent program does not need to operate to TCON register, the external interrupt 0 and 1 can be used at the same time, and the software does not need to clear the flag and there is no need to do the following actions)

Use examples are as follows:

```
void EX0() interrupt        0
{
   TCON &= 0xF5;
}
void EX1() interrupt        2
{
   TCON &= 0XF5;
}
```

## 6.8 Precautions for External Interrupt Setup

When using the external interrupt function of SinOne SC92F series MCU, set the corresponding IO port to the input mode! You need to set IO port first, and then set the corresponding external interrupt configuration. The reverse operations may mistakenly result in an edge interrupt.

The external interrupts of the same group share the same interrupt vector. The user needs to read IO port level in the interrupt service function, judge the source of the interrupt, and then perform corresponding actions. It is not recommended to place multiple two-sided edge interrupts in the same set of external interrupt.

Use examples are as follows:

```
P1CON &= 0XFC;   //Set INT10(P10) port as input mode
P1PH1|=0X03;   //Open P10 and P11 pull-up resistance
INT1F = 0X03;  //Enable INT10, INT1 falling edge trigger
EINT1 = 1;                  //Enable external interrupt 1
EA = 1;            //Open main repeater
     void Interrupt_work() interrupt 2
     {
         if(P10==0)         //Judge if the external interrupt comes from INT10
         {
             //Executive code
}
         if(P11==0)         // Judge if the external interrupt comes from INT11
         {
             // Executive code
}
}
```

## 6.9 Precautions for LCD/LED/PWM RAM Use

SinOne SC92F series SC92F854X, SC92F754XS, C92F844X and SC92F744X MCU features LCD/LED and PWM-dedicated RAM area (700H-74FH), which can only write other than read. To obtain RAM data of this area, perform the following operations:

1. Create a cache regions on RAM for the user to read and rewrite.
2. Overwrite the contents of the cache area with LCD/LED/PWM RAM.

Use examples are as follows:

```
unsigned int xdata PWM_Duty[8] _at_ 0x740;
unsigned int PWM_Duty_Buff[8];   //Used to store the value written to PWM_Duty register
unsigned int temp;
PWM_Duty_Buff [0] = 0x0000;      //Store the value to be written to the cache
PWM_Duty[0]=PWM_Duty_Buff[0];//Write the value to the control register of corresponding PWM
Temp = PWM_Duty_Buff [0];        //Obtain the value in WM_Duty array via PWM_Duty_Buff
```

## 6.10 Precautions for Code Option of Software Operations

There is a separate Flash area inside SinOne SC92F Series MCU, which is used for storing the initial power-on value settings, it is called the Code Option area. When performing the IC programming, this part of the codes will be written into the IC. After the IC is reset and initialized, it will bring this setting into the SFR as the initial setting.

Option-related SFR read and write are controlled by OPINX and OPREG register respectively. The former determines the location of each Option SFR and the latter determines the write value of each Option SFR:

| Symbol | Address | Description | | Initial Power-on Value |
|---|---|---|---|---|
| OPINX | FEH | Option pointer | OPINX[7:0] | 00000000b |
| OPREG | FFH | Option register | OPREG[7:0] | nnnnnnnnb |

Upon operating Option-related SFR, the OPINX register will store the address of related OPTION register and the OPREG register will store the corresponding value.

For example: To configure OP_HRCR as 0X01, the specific operations are as follows:

C-language routine:

```
OPINX = 83H;          //Write OP_HRCR address into OPINX register
OPREG = 0x01;         //Write 0x01 into OPREG register (the value to be written into OP_HRCR register)
```

Assembly routine:

```
MOV OPINX,#83H;       // Write OP_HRCR address into OPINX register
MOV OPREG,#01H;       // Write 0x01 into OPREG register (the value to be written into OP_HRCR register)
```

**Notes:**

**1. Do not write values other than SFR address in the Customer Option area to the OPINX register! Otherwise, the system will run abnormally!**

2. SC92F8x8x, SC92F7x8x,SC92F8x9x and SC92F7x9x can not write software to DISRST bit

## 6.11 Precautions for Touchkey Setup

Upon initializing the application program, the IO port corresponding to TK needs to be set to strong push-pull output mode and high level output. IO corresponding to TK can not be operated during the process of TK scan. In addition, if there are more than two TouchKey chips on the same PCB, it is recommended to turn on “Anti-interference Setting” to prevent the same frequency interference.

For more TouchKey information, please refer to *SinOne SC92F_93F Series TouchKey MCU Application Guide*.

# 7. Version Change History

| Version | **Change History** | **Date** |
|---|---|---|
| V1.0 | Initial version | Dec. 2022 |

# Statement

Shenzhen SinOne Microelectronics Co., Ltd. (hereinafter referred to as SinOne) reserves the right to change, correct, enhance, modify and improve SinOne products, documents or services at any time without prior notice. SinOne believes that the information provided is both accurate and reliable. The information in this document becomes available since November 2021. In the actual production design, please refer to the latest data manual of each product and other relevant materials.