Table of Contents

1. Precautions for SinOne SC95F Series MCU Electrical parameters

Working Voltage: 2.0V - 5.5V
Working Temperature: -40 - 105℃
Kernel: Ultra High Speed 1T 8051; dual DPTR
Flash ROM: rewritable for over 100,000 times
LDROM:
1. Independent 1Kbyte, rewritable for 100,000 times with more than 10 years of storage life (SC95F8x1x, SC95F7x1x, SC95F8x2x and SC95F7x2x)
2. Independent 4Kbyte, rewritable for 100,000 times with more than 10 years of storage life (other models)
System Clock: Built-in high-frequency oscillator frequency error: No more than ±2% in the application environment spanning 2.0V - 5.5V and -40 - 105℃

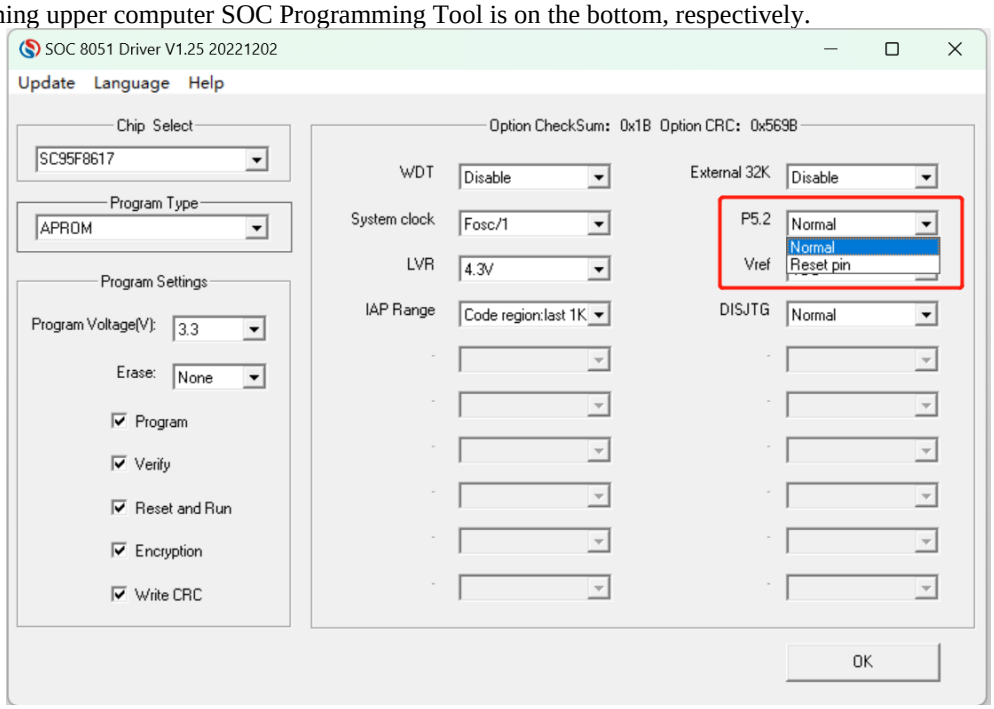
2. Precautions for SinOne SC95F Series MCU Programming

1. The capacitance of CLK or DIO pins of SC95F 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. SC_LINK and SOC Pro51 programming is only supported by SC95F8x1x, SC95F7x1x, SC95F8x2x and SC95F7x2x, for other models, please upgrade to SC_LINK_PRO and programming upper computer SOC Programming Tool.
5. Upon 95F series chip offline programming, if IAP write is required after the program is powered on, the check will be failed, because 95F series offline check will apply CRC check. Running the program may change ROM data resulting in failed check, it is required to add a period of time (100m is recommended) before initialization for successful programming.

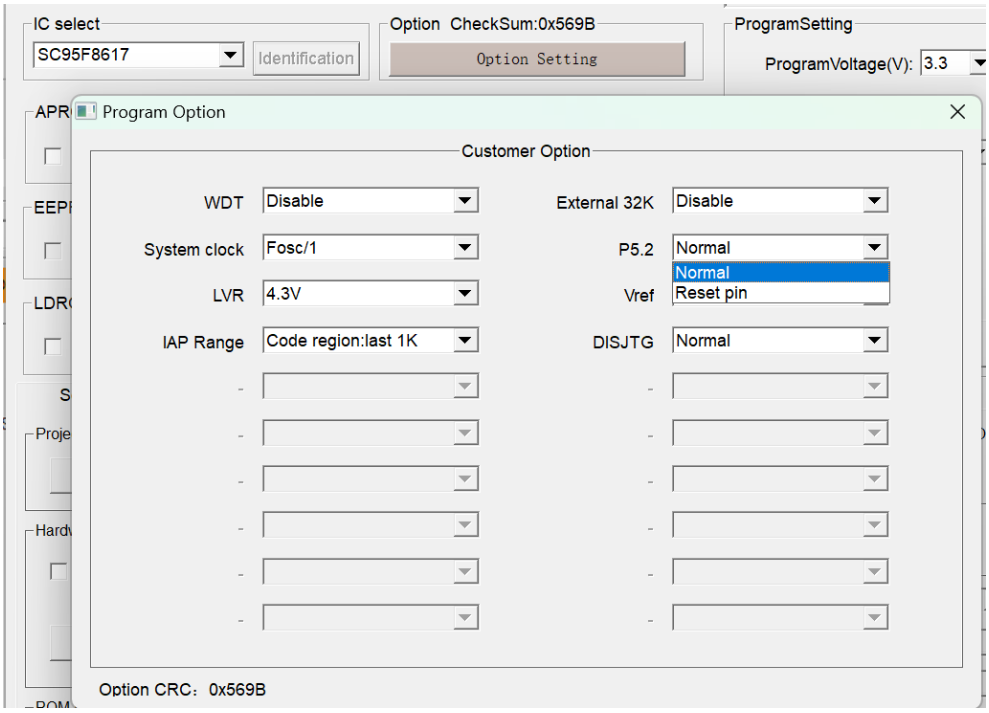
3. Precautions for Circuit Design

The default GPIO power-on mode of SinOne SC95F series MCU is Hi-Z input mode.

The RST pins of SinOne SC95F series MCU can be set as Reset pin or GPIO by using Option. When RST pin is used as Reset pin, it is enabled at low level; when RST pin is used as GPIO, the pin will not perform reset operation in low level. Option settings are shown in the figure (by taking SC95F8617 as an example). The setting page of KEIL plugin is on the top and the programming upper computer SOC Programming Tool is on the bottom, respectively.



KEIL Plugin Setting Page



Setting Page of Programming Upper Computer SOC Programming Tool

3.1 Circuit Design Instances

3.1.1 RST PIN Circuit

The RST pin of SC95F 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. 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:

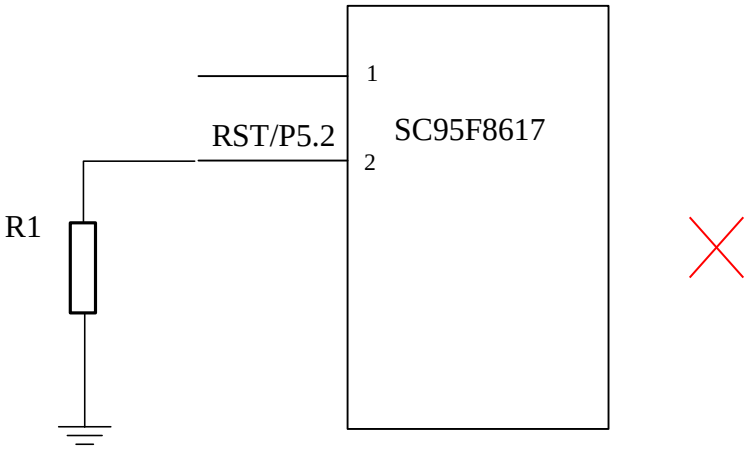


Diagram of Wrong Connection

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:

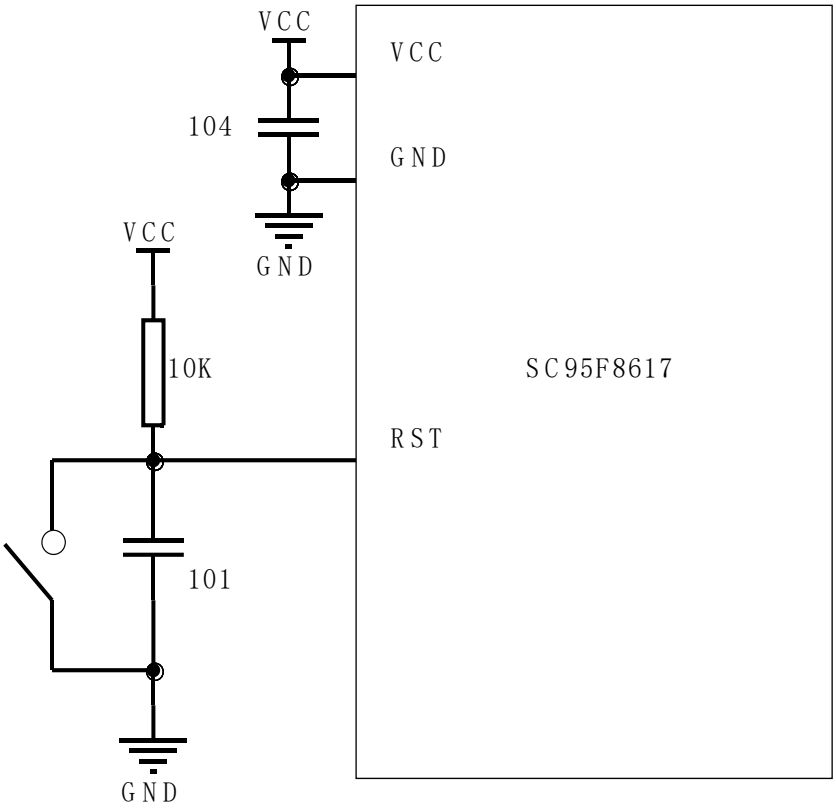


Diagram of Recommended Connection

3.1.2 ADC Sampling Pin Circuit

For ADC sampling port of SinOne SC95F 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.

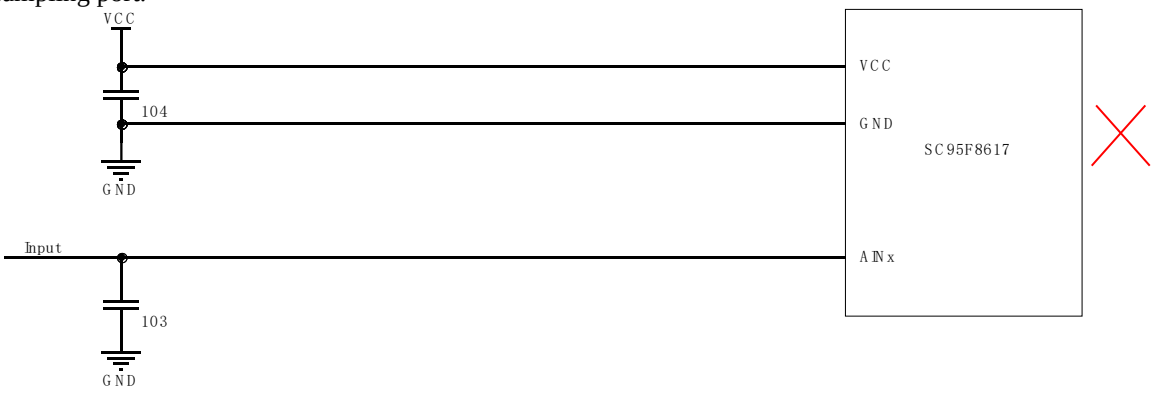


Diagram of Wrong Connection

Recommended connection: The 104 capacitor is close to the power pin and the 103 capacitor is close to the ADC sampling port.

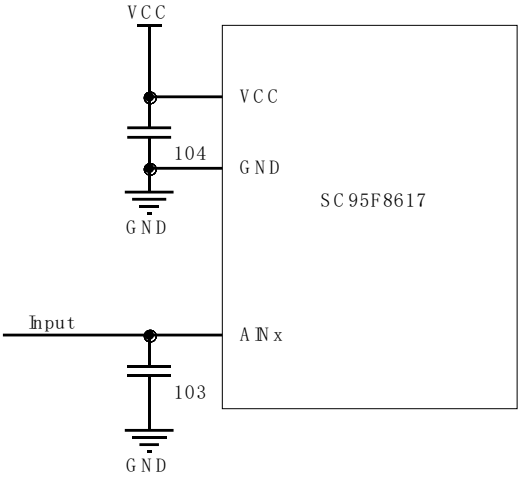
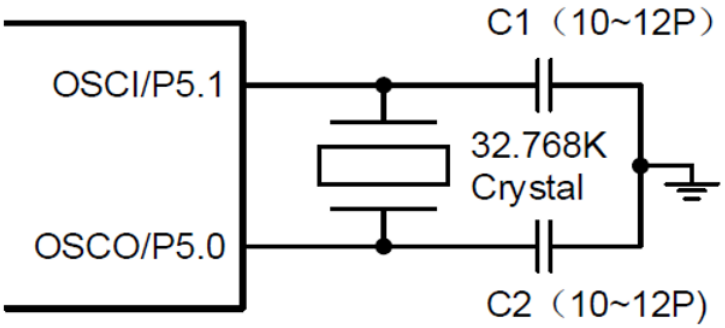


Diagram of Recommended Connection

3.1.3 External Crystal Oscillator Circuit

Some models of SinOne SC95F series MCU provide high-frequency external crystal interface or low-frequency external interface. If the user needs to use the external crystal interface, the mating capacitor shall be selected according to the requirements of the selected crystal oscillator, which shall be close to the chip pin, as shown in the figure:



32.768K External Crystal Connection Diagram

3.1.4 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.

3.2 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

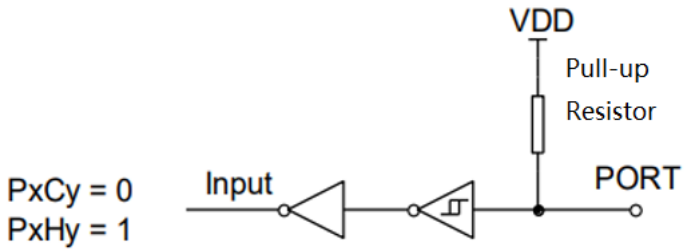
Note: Unused and encapsulated but unled IO port shall be set as Strong Push-pull Output Mode.

3.2.1 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.

3.2.2 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

3.2.3 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 10K.

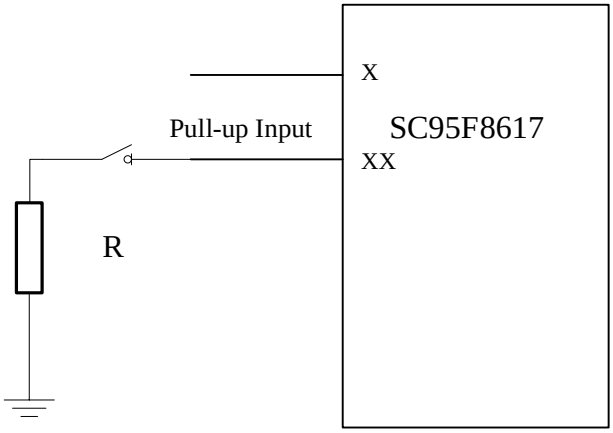


Diagram for Key Input Connection in Pull-up Mode

3.2.4 Implementation of I/O Open-drain Output Mode

There is no open-drain output setting options for SinOne SC95F 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
```

3.2.5 Precautions for I/O AND/OR Operations

The instruction execution by the CPU of the SC95F is fast, the level on GPIO may not be changed before executing the next instruction. Therefore, please do no change the output state of IO port by continuously conducting AND/OR operations on a GPIO register, as shown below:

```
P0|=0x04;
P0|=0x08;
```

Please merge the above operations into one for implementation, as follows:

```
P0|=0x0C;
```

Or change to bit operation, as follows:

```
P02=1;
P03=1;
```

3.2.6 Precautions for I/O READ IO Functions

SinOne SC95F series MCU features 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. But the input mode will not be affected.

Code examples are as follows:

```
OPINX = 0X86;       //Disable Read IO function
OPREG&=0XF7 ;
OPINX = 0X86;        //Enable Read IO function
OPREG|=0X08 ;
```

4. Precautions for Software Programming

SinOne SC95F 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.

4.1 Precautions for TIMER2/3/4 Use

SinOne SC95F series MCU contains three independent 16-bit Timers shared by register address, namely Timer2/3/4. All Timer2 work in four modes; and Timer3/4 of SC95F8x1x/7x1x series MCU only has one working mode = 16-bit automatic reloading timer, and only allows upward counting, while other TIM3/4 modules have four working modes.

The user configures the pointer register TXINX to make the TimerX register group point to Timer2/3/4, so that three independent Timers can be configured in one set of register.

Before operating TimerX register group, it is required to configure TXINX to point to Timer2/3/4 specified by the user; otherwise, operations on TimerX register group may not take effect on TimerX! This is especially important when Timer2/3/4 are used at the same time.

When using multiple Timer2/3/4, there are two important points to be noted when writing the interrupt service functions:

1. When TimerX register group is interrupted, add TXINX=0X02/ TXINX=0X03/ TXINX=0X04 pointing to Timer2/3/4 in the first line to configure TXINX register and guarantee the accuracy of the operation.

2. When the interrupt function is enabled, because Tim2/3/4 interrupt functions are required to be imported to TXINX register, a low-priority interrupt may be interrupted by a high-priority interrupt, and TXINX is modified in the high-priority interrupt function. TimerX register group may be changed upon re-entering low-priority interrupt functions. To avoid the above situations, before interrupting the service functions to configure TXINX register, it is required to save the value of TXINX in advance, and then assign the value of the variable to TXINX register after corresponding TXINX register group operation is completed.

The examples for using codes by Timer2/3/4 at the same time are as follows:

```
TXINX = 0X02;                           //Point TimerX register group to Timer2
                                        //Set Timer2
TXMOD = TXMOD & 0X7F | (1<<7);          //Set Timer2 clock frequency as FSYS/1
TXCON = 0X00;                           //Set Timer2 as 16-bit automatic reloading timer (counting up)
THX = (65535-32000) / 256;              //1ms@32M
TLX = (65535-32000) % 256;
RCAPXH = (65535-32000) / 256;
RCAPXL = (65535-32000) % 256;
TRX = 1;                                //Enable Timer2

TXINX = 0X03;                           //Point TimerX register to Timer3
                                        //Set Timer3
TXMOD = TXMOD & 0X7F | (1<<7);          //Set Timer3 clock frequency as FSYS/1
```

```
    THX = (65535-3200) / 256;                   //100us@32M
    TLX = (65535-3200) % 256;
    RCAPXH = (65535-3200) / 256;
    RCAPXL = (65535-3200) % 256;
    TRX = 1;                                    //Enable Timer3

    TXINX = 0X04;                               //Point TimerX register group to Timer4
                                                //Set Timer4
    TXMOD = TXMOD & 0X7F | (1<<7);              //Set Timer4 clock frequency as FSYS/1
    THX = (65535-32000) / 256;                  //1ms@32M
    TLX = (65535-32000) % 256;
    RCAPXH = (65535-32000) / 256;
    RCAPXL = (65535-32000) % 256;
    TRX = 1;                                    //Enable Timer4

    void Timer2_ISR() interrupt 5           //Timer2 interrupt service function
    {
        unsigned char TXINX_Stack = TXINX;
        TXINX = 0X02;                           //Enter Timer2 interrupt service function and first point TimerX
register group to Timer2
        ......
        TFX = 0;                                //Clear Timer2 interrupt flag TFX
        ......
         TXINX = TXINX_Stack;
    }
    void Timer3_ISR() interrupt 13              //Timer3 interrupt service function
    {
        unsigned char TXINX_Stack = TXINX;
        TXINX = 0X03;                           // Enter Timer3 interrupt service function and first point TimerX
register group to Timer3
        ......
        TFX = 0;                                //Clear Timer3 interrupt flag
        ......
        TXINX = TXINX_Stack;
    }
    void Timer4_ISR() interrupt 14              //Timer4 interrupt service function
    {
        unsigned char TXINX_Stack = TXINX;
        TXINX = 0X04;                           //Enter Timer4 interrupt service function and first point TimerX
register group to Timer4
        ......
        TFX = 0;                                //Clear Timer4 interrupt flag
        ......
        TXINX = TXINX_Stack;
}
```

4.2 Precautions for Conventional Pulse Width Modulation Counter PWM2/3/4 Use

SinOne SC95F8x3x/7x3x series MCU has 6 conventional PWMs, divided into 3 groups: PWM2, PWM3 and PWM4.

Note: The period register of these three groups of PWM are shared with PCAPXL and RCAPXH of Timer2, Timer3 and Timer4 respectively. Therefore, once any PWM2, PWM3 and PWM4 resources are used by the user, no timing/counting values for Timer2, Timer3 and Timer4 can be changed; otherwise, PWM period output may become abnormal!

For PWM2/3/4 output waveforms, the duty ratio can be changed by changing the value of the high-level setting register PDTxy (x=2~4, y=0~1). However, it should be noted that when the value PDTxy is changed, the duty ratio will not be changed immediately, and it will be changed in the next period.

Before using PWM2/3/4, consult Precautions for TIMER2/3/4 Use first.

4.3 Precautions for PWM Setting and Use

SinOne SC95F series MCU PWM can be grouped into PWM40/PWM41, PWM42/PWM43, PWM50/PWM51 and PWM52/PWM53 under complementary mode, with its duty ratio being adjusted by PDT40[11:0], PDT42[11:0], PDT50[11:0]

and PDT52[11:0]; at this time, the registers PDT41[11:0], PDT43[11:0], PDT51[11:0] and PDT53[11:0] are invalid. However, the output function enabling and polarity output are still independently controlled by ENPxy and INVxy.

SinOne SC95F MCU PWM provides fault detection function. After the fault detection function is enabled by the user, FTL pin can not be in high resistance state, otherwise, PWM output will become abnormal!

When any fault occurs, PWM stops outputting and PWM port is in high resistance state. The fault detection mode is divided into immediate mode and latch mode. In latch mode, when the fault signal meets the disabling condition, the hardware will not automatically clear the fault detection flag and the user can clear the PWMFLT =0x7f by software.

4.4 Precautions for PCON Register Setup

SinOne SC95F 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 "SC95F861X_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!

4.5 Precautions for UART0 Setup and Use

When SinOne SC95F Series MCU SC95F861X Microcontroller unit uses UART0, if TIMER1 is selected as the baud rate generator, the Timer 1 must stop counting (TR1=0). When Timer1/2 is selected as the baud rate , it is required to configure TXINX=0X02 first to make TimerX register set point to Timer2, then configure TXCON to select the baud rate generator.

Use examples are as follows:

```
SCON   = 0X50;        //Set communication mode as Mode 1, allow to receive
TXINX = 0X02;         //Point TimerX register set to Timer2
TXCON &= 0XCF;        //Select T2 as the baud rate generator
TRX = 1;              //Set Timer2 as the baud rate generator, and Timer 2 must enable counting
RCAPXH= 0x0d;         //In 32M, the baud rate is 9600; initial value of timer [RCAPXH,RCAPXL] = Fsys/baud rate
RCAPXL= 0x05;         //In 32M, the baud rate is 9600
EUART = 1;//Enable Uart0 interrupt
```

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

SinOne SC95F series MCU integrates three series interface circuit USCI, the user can configure USCIx for any communication mode in SPI/TWI/UART through USMDx[1:0].

4.6.1 Precautions for SPI Use:

USCI SPI has 8-bit and 16-bit transmission modes; when SPI is set to 16-bit mode, you must write the high byte SPDH[7:0] first, then write the low byte SPDL[7:0]. After that, start the transmission. Note: SPDH[7:0] is only for 16-bit mode.

To avoid CLK generated by mistake when SPI function is enabled (at this time, pin switches from IO to SPI), the IO port must be set to the corresponding level state according to the CLK idle level setting before enabling SPI. For example, if

CPOL=0 is set to idle , it is required to set corresponding IO of CLK as outputting 0 before enabling SPI (SPEN=1), vice versa, otherwise, it may lead to communication misalignment. At the same time, SPI needs to be reset frequently in the program upon SPI communication to make sure that CLK will not be mistakenly received due to noise and other reasons, resulting in continuous communication errors.

4.6.2 Precautions for TWI Use:

1. The user can set USCI TWI as host or slave as required. When TWI is used as the slave, the communication rate is up to 400kHz;
2. Upon using TWI function, it is recommended that the user queries the communication status through the status flag STATE[2:0] in the interrupt service function;
3. Since there is no interrupt flag to judge the Stop when TWI is used as the host, the user needs to add a short delay before and after sending STOP signal to prevent errors in communication.
4. To prevent abnormal communication caused by sending Start signal before the last data transmission is completed, please perform the following codes before sending Start:

```
US0CON0&=0xF7;          //First set AA as 0
Delay ();               //Delay for a period of time, longer than the transmission time of 1byte data
US0CON0 &=0x7f;         //Disable TWI
US0CON0 |= 0x80;        //Enable TWI
```

4.6.3 Precautions for UART Use:

Interrupt flag TI sent by USCI UART and interrupt flag RI received by USCI UART can be cleared in two ways:

1. Write 0 to clear the flag (SC95F8x1x, SC95F7x1x, SC95F8x2x and SC95F7x1x)

For USCI 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:

```
US0CON3=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(!USCI0SendFlag)
{
    i--;
    if(i==0)
    {
        break;          // Time out and exit
    }
}
USCI0SendFlag = 0;  //Clear the sending flag.
```

2. Write 1 to clear the flag (other models)

4.7 Precautions for USCI2/3/4 Configurations

SinOne SC9F8x3x and SC9F7x3x SC series MCU has USCI2/3/4 (USCI4 is only available for 44Pin chip), with the same functions as USCI1. For mode configuration, it is able to configure USCI0~4 interface as any communication mode in SPI, TWI and UART through TMCON USMD2[1:0], USMD3[1:0] and USMD4[1:0] bit.

It is worth noting that the control registers of USCI2/3/4 share the same set of addresses, the user can point the USCIX register group (USXCON0~3) to USCI2/3/4 through USINX[2:0], so as to realize effective operation of configuring three independent USCI interfaces in one set of registers. Only after USINX[2:0] is configured successfully, the USCIX register group will point to USCI2/3/4 specified by the user. In this case, the operation of USCIX register group is effective for corresponding USCI interface. Otherwise, it may be ineffective for USCIX! This is especially important when the user is

using USCI2/3/4 at the same time.

When using USCI2/3/4 interrupts, there are two important points to be noted when writing the interrupt service functions:
1. When USCI register is interrupted, first configure USINX register and guarantee the accuracy of the operation.
2. When the interrupt function is enabled, because USINX register needs to be interrupted, a low-priority interrupt may be interrupted by a high-priority interrupt, and USINX is modified in the high-priority interrupt function. USCIX register group may be changed upon re-entering low-priority interrupt functions. To avoid the above situations, before interrupting the service functions to configure USINX register, it is required to save the value of TXINX in advance, and then assign the value of the variable to USINX register after corresponding USINX register group operation is completed.

USCI2/3/4 interrupt service examples are as follows:

```
void USCI2_ISR() interrupt 16 //USCI2 interrupt service function
{
    unsigned char USINX_Stack = USINX;
    USINX = 0X02;                    //Enter USCI2 interrupt service function and first point USCIX register group to USCI2
    ......
    ......
    USINX = USINX_Stack;
}
void USCI3_ISR() interrupt 17 // USCI3 interrupt service function
{
    unsigned char USINX_Stack = USINX;
    USINX = 0X03;                    //Enter USCI3 interrupt service function and first point USCIX register group to USCI3
    ......
    ......
USINX = USINX_Stack;
}
void USCI4_ISR() interrupt 18          // USCI4 interrupt service function
{
    unsigned char USINX_Stack = USINX;
    USINX = 0X04;                    //Enter USCI4 interrupt service function and first point USCIX register group to USCI4
    ......
    ......
    USINX = USINX_Stack;
}
```

4.8 Precautions for ADC Multi-channel Switch Acquisition

Most models of SinOne SC95F 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
ADCCON&=~(0X20);                                //Clear interrupt flag
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.
    ADCCFG2 = 0x02;                 //Wrong ADCCFG configuration may influence ADC acquisition precision
  ADCCON  |= 0X80;              //Turn on ADC module power
    ADC_channel(0);             //Switch ADC entrance to AIN0 port
    ADC_Value0 = 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
}
```

4.9 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, 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.

```
void EX0() interrupt   0
{
   TCON &= 0xFD;
}
void EX1() interrupt   2
{
   TCON &= 0XF7;
}
```

4.10 Precautions for External Interrupt Setup

When using the external interrupt function of SinOne SC95F 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 interrupts.

Use examples are as follows:

```
P4CON &= 0XFC;   //Set INT10 (P40) port and INT11(P41) as input mode
P4PH |= 0X03;    //Open P40 and P41 pull-up resistance
INT1F = 0X03;    //Enable INT10, INT11 falling edge trigger
EINT1 = 1;       //Enable external interrupt 1
EA = 1;          //Enable global interrupts
void Interrupt_work() interrupt 2
{
if(P40==0)       //Judge if the external interrupt comes from INT10
    {
                 //Execute code
    }
    if(P41==0)   //Judge if the external interrupt comes from INT11
    {
                 //Execute code
    }
}
```

4.11 Precautions for Code Option of Software Operations

There is a separate Flash area inside SinOne SC95F 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:

```
EA = 0;                 //Disable global interrupts
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)
EA=1;                   //Enable global interrupts
```

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)
```

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

4.12 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.

Special Instruction: SC95F861X TK9/TK11 debugging communication port is multiplexed with that of TK; if TK debugging function is used, please avoid using TK9/TK11!

For more Touchkey information, please refer to *SinOne SC95F Series TouchKey MCU Application Guide*.

4.13 Precautions for CRC Use

1. CRCDRn write data and read data are different;
2. The CRC value calculated by the hardware is the 32-bit CRC checksum for the entire program area (Note that this does not include IAP area). If the address unit contains the residual value after last operation, the CRC value will be inconsistent with the theoretical value. Therefore, it is recommended that users erase the whole Flash ROM and then program the code to make sure that the CRC value is consistent with the theoretical value.
3. The hardware CRC calculation range does not include the IAP area. In addition, the 4 bytes address before IAP area is used to store the 32-bit CRC calculation results of the program code, so these 4 bytes are also not within the hardware CRC calculation range.
4. After the CRC starts the operation statement, be sure to add at least 8 NOP instructions to guarantee the CRC calculation is completed;
5. It is required to disable global interrupts EA upon performing CRC operation, and enable global interrupts after performing 8 NOPs.

4.14 Precautions for Software Security Encryption Function

SinOne 95 series chips allow users to enable the security encryption function, which needs to be used in conjunction with SC LINK and SOC PRO51 upper computer software. For specific operation instructions, see Section 4.7 of SOC LINK Series Programmer Simulator User Manual, which can be downloaded from SinOne website.

4.15 Precautions for Interrupt Disable

In practical application, it is generally unnecessary to disable the sub-interrupt again after enabling the sub-interrupt. If the user needs to disenable the sub-interrupt, it is required to disable the global interrupts before disabling the sub-interrupt, then disable the sub-interrupt; the following are the steps for disabling the sub-interrupts that you must follow

EA = 0;
Sub-interrupt switch=0; (for example ET0 = 0;)
EA = 1;

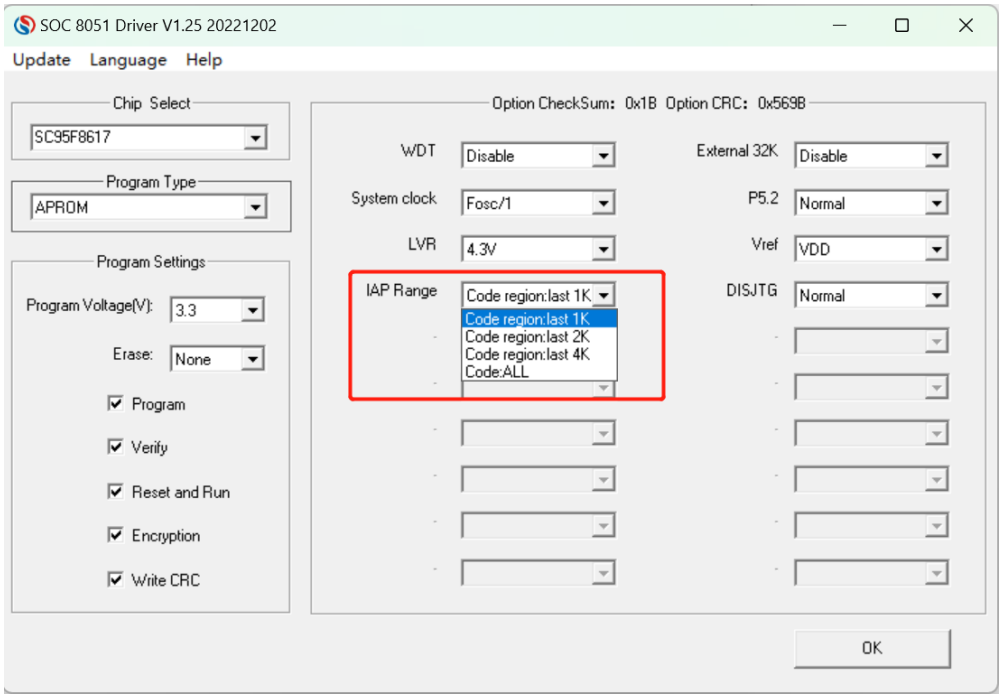
5. SinOne SC95F Series MCU IAP and Algorithm Explanation

5.1 IAP Operations

Take SC95F8617 as an example to explain SinOne MCU internal EEPROM use method and CODE IAP operation method.

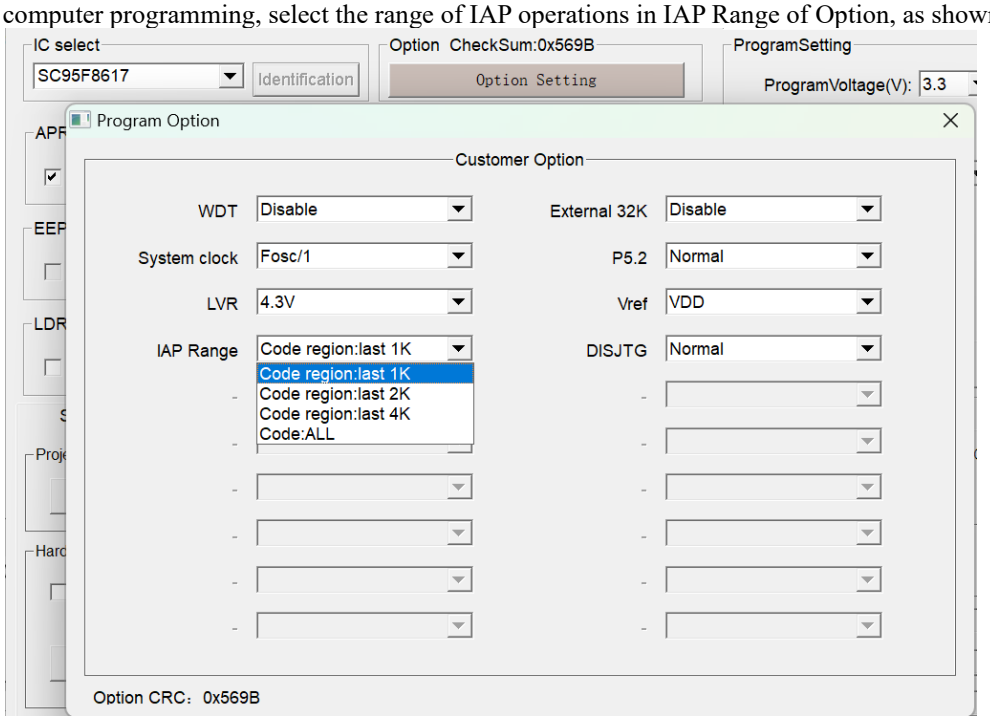
SC95F8617 internal 64K Flash can be In Application Programming (IAP) operation, that is to allow the user program to dynamically write data into the internal Flash.

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.



KEIL IAP Operating Range Configuration Interface

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



Programming Upper Computer IAP Operation Range Configuration Interface

FLASH read and write features:

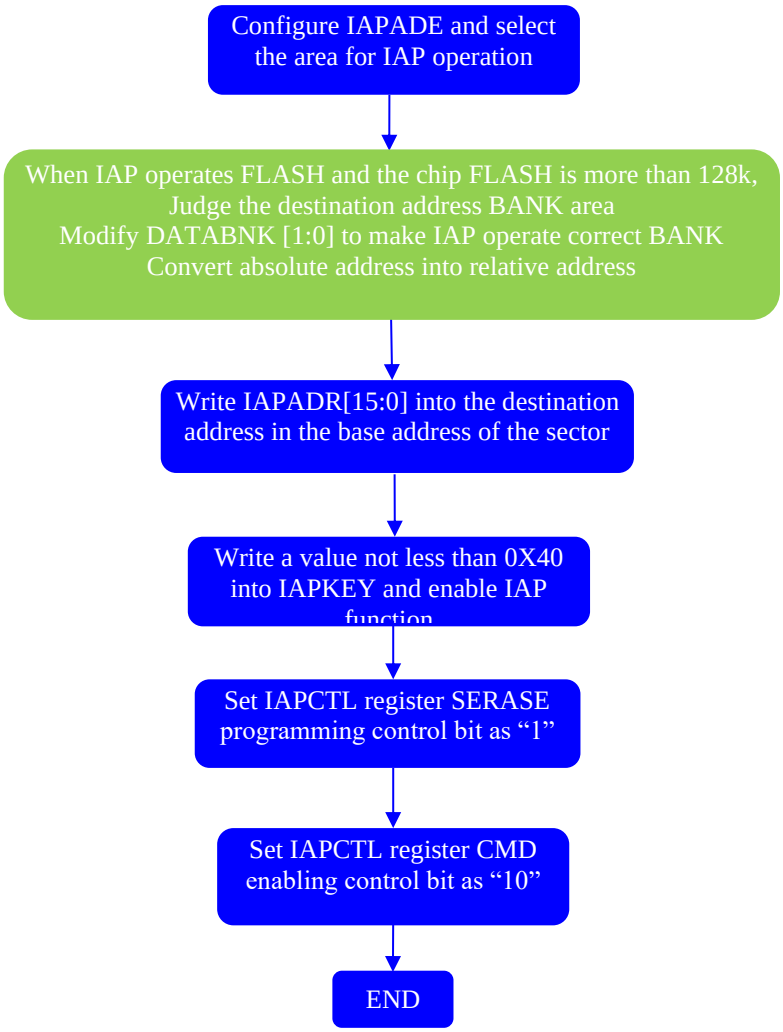
1. Single Byte read and write operation can be perform;
2. Flash ROM is divided into 128 sectors (0000h~FFFFh), it is required to erase the sector to which the operation address belongs before write (sector erase: 512 bytes), and just erase the base address written to the sector;

FLASH Service Life: More than 100,000 times

5.1.1 IAP Sector Erase Process

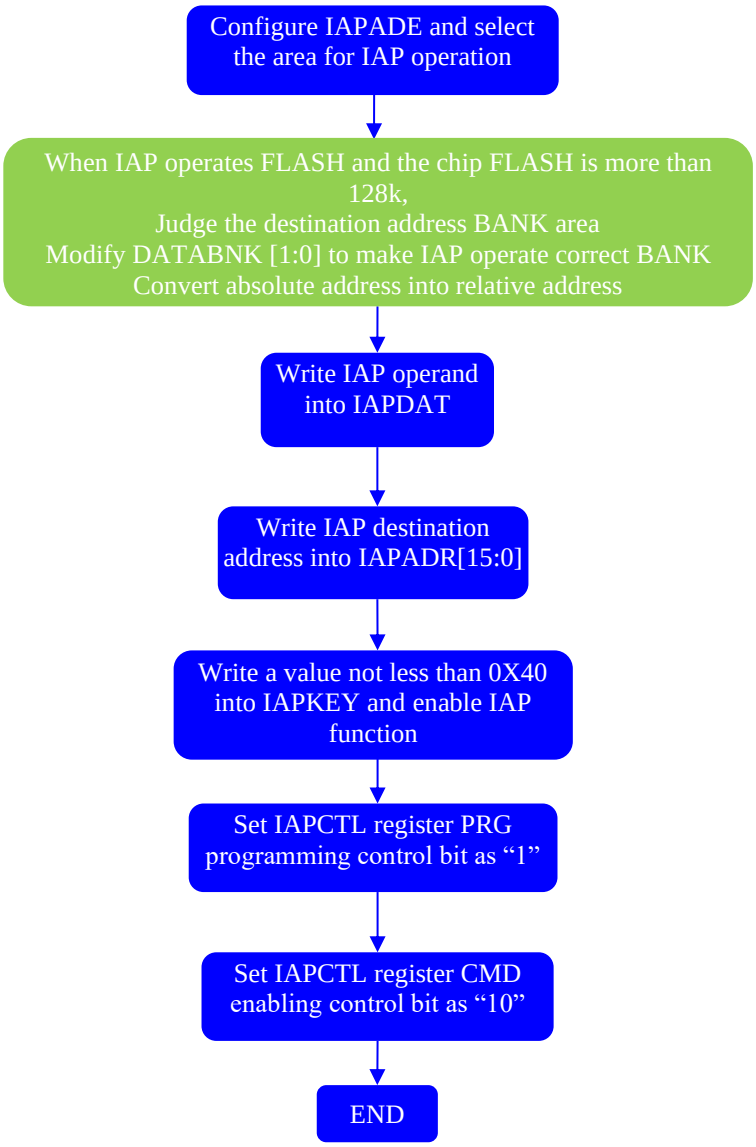
To guarantee successfully byte writing, make sure that the destination address has been erased to 0x00. SinOne SC95F series

MCU supports the sector erase. The specific IAP sector erase process is as follows:



5.1.2 IAP Write Process

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



5.2 IAP Operation Codes in CODE Area

SC95F 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.

5.2.1 When Chip FLASH Size No More than 64K

CODE Area IAP Operation Routines:

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

IAP sector erase operation C-language Demo program:

```
    EA_Buff = EA;
    EA = 0;                                     //Disable global interrupts
    IAPADE = 0X00;                              //Select ROM area
    IAPADH = (unsigned char)( IAP_Add>8);       //High value of address
    IAPADL = (unsigned char) IAP_Add ;          //Low value of address
```

```
    IAPKEY = 0XF0;              //This value can be adjusted according to actual conditions, the write value shall be
    more than 0x40; make sure that after executing this instruction
                                //and before executing sector erase command, the interval shall be less than 240
                                (0xf0) system clocks,
                                //otherwise, IAP function will be disabled;
    IAPCTL = 0X20;              //Set the sector erase control bit SERASE as 1
    IAPCTL |= 0X02;             //Perform EEPROM sector erase operation (do enable SERASE first)
    _nop_();                    //Wait (at least 8 _nop_()s)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
EA = EA_Buff;                   //Restore global interrupts
```

IAP write to operate C-language Demo program:

```
    EA_Buff = EA;
    EA = 0;                                         //Disable global interrupts
    IAPADE = 0X00;                                  //Select ROM area
    IAPDAT = IAP_Data;                              //Send data to IAP data register
    IAPADH = (unsigned char)((IAP_Add >> 8));       //High value of address
    IAPADL = (unsigned char)IAP_Add;                //Low value of address
    IAPKEY = 0XF0;              //This value can be adjusted according to actual conditions; make sure that after
    executing this instruction
                                //and before writing the command, the interval shall be less than 240 (0xF0) system
                              clocks,
    //otherwise, IAP function will be disabled;
    IAPCTL = 0X10;              //Set the program control bit PRG as 1
    IAPCTL |= 0X02;             //Perform EEPROM write operation (do enable PRG first)
    _nop_();                    //Wait (at least 8 _nop_()s)
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    EA = EA_Buff;               //Restore global interrupts
```

Note: IAP operations in the ROM 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.).

5.2.2 When Chip FLASH is 128K

CODE Area IAP Operation Routines

```
#include ”intrins.h”
unsigned int IAP_LogicAdd;
unsigned int IAP_MCUAdd;
unsigned char IAP_Data;
unsigned char code * POINT =0x0000;
bit EA_Buff;
```

Logical address to MCU address C-language Demo program:

```
if(IAP_ LogicAdd >=0x10000)         //If it exceeds 64K, Bank register is required to point to the target
address
```

```
{
        if(IAP_ILogicAdd<0x18000)
        {
                ROMBNK = (ROMBNK & 0xCF) |0x20;
                IAP_ MCUAdd = (IAP_ILogicAdd-0x8000);
        }
        else if(IAP_ILogicAdd<0x20000)
        {
                        ROMBNK = (ROMBNK & 0xCF) |0x30;
                IAP_ MCUAdd = (IAP_ILogicAdd-0x10000);
        }
}
else
{
        ROMBNK = (ROMBNK & 0xCF) |0x10;
        IAP_ MCUAdd = IAP_IlogicAdd;
}
```

IAP sector erase C-language Demo program:

```
EA_Buff = EA;
EA = 0;                                         //Disable global interrupts
IAPADE = 0X00;                                  //Select ROM area
IAPADH = (unsigned char)( EE_Add>>8);  //High value of address
IAPADL = (unsigned char)EE_Add;                 //Low value of address
IAPKEY = 0XF0;          //This value can be adjusted according to actual conditions, and the write value is more
than 0x40; make sure that after executing this instruction
                        //and before executing sector erase command, the interval shall be less than 240 (0xF0)
                    system clocks,
                        //otherwise, IAP function will be disabled;
IAPCTL = 0X20;          //Set the sector erase control bit SERASE as 1
IAPCTL |= 0X02;         //Perform EEPROM sector erase operation (do enable SERASE first)
_nop_();                //Wait (at least 8 _nop_()s)
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();

EA = EA_Buff;           //Restore global interrupts
```

IAP write operation C-language Demo program:

```
EA_Buff = EA;
EA = 0;                                                 //Disable global interrupts
IAPADE = 0X00;                                          //Select ROM area
IAPDAT = IAP_Data;                                      //Send data to IAP data register
IAPADH = (unsigned char)((IAP_MCUAdd >> 8));  //High value of address
IAPADL = (unsigned char)IAP_MCUAdd;             //Low value of address
IAPKEY = 0XF0;                  //This value can be adjusted according to actual conditions, and the write
value is more than 0x40; make sure that after executing this instruction
                                //and before executing sector erase command, the interval shall be less than
                            240 (0xF0) system clocks,
                                //otherwise, IAP function will be disabled;
IAPCTL = 0X10;                  //Set programming control bit PRG as 1
IAPCTL |= 0X02;                 //Perform EEPROM write operation (do enable PRG first)
_nop_();                        //Wait (at least 8 _nop_()s)
_nop_();
_nop_();
```

```
_nop_();
_nop_();
_nop_();
_nop_();
_nop_();
EA = EA_Buff;                    //Restore global interrupts
```

IAP Read Operation C-language Demo program:

```
EA_Buff = EA;
EA = 0;                                 //Disable global interrupts
IAPADE = 0X00;                          //Select ROM area
IAP_Data = *( POINT+IAP_MCUAdd);        //Read IAP_MCUAdd value to IAP_Data
EA = EA_Buff;                           //Restore global interrupts
```

5.2.3 Special Remind:

IAP operations in the ROM 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.).

5.3 IAP Use Suggestions and Precautions

1. It is required for SinOne SC95F series MCU to erase the sector where the destination address is located before writing data. It is recommended to conduct data backup before writing data to prevent old data from being erased but new data has not been written due to power failure during the erase process.
2. SC95F series chips can perform In Application Programming (IAP) operations, but in the application of actual products, only a few bytes of data need to be written to Flash, and using fixed address to write data will make some addresses reach the IAP lifetime too early. In addition, data backup and sector erase must be performed before writing data. Therefore, it is recommended to operate Flash by adopting sector-based and cyclic address write method.
3. SC95FXX1X chip out of-bounds IAP write or erase operations on LDROM area may result in chip damage. Do not perform out-of-bounds operations on the LDROM!
4. Pay attention to the value of ROMBANK[1:0] when using MOVC operations such as function pointer; otherwise, the value may be read incorrectly or run away.

6. Precautions for Simulation

6.1 Soft Reset Failure in Simulation State

SinOne SC95F series MCU will disable soft reset function in simulation state, so it will execute PCON|=0x08; the command will not perform the reset action.

6.2 No-reset of SFR in Simulation State

SinOne SC95F series MCU will not reset SFR in simulation mode.

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 XX 20XX. In the actual production design, please refer to the latest data manual of each product and other relevant materials.