SH79F6481A User Guide

Enhanced 8051 Microcontroller with 24 channels Touch-key input and PWM

1. SH79F6481A I/O User Guide

1.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's IO features:

- 46 bi-directional I/O ports
- Share with alternative functions
- Part of the second function can be configured by register
- 2 selectable open-drain I / O ports

The SH79F6481A has 46 bi-directional I/O ports. The PORT data is put in Px register. The PORT control register (PxCRy) controls the PORT as input or output. Each I/O port has an internal pull-high resistor, which is controlled by PxPCRy when the PORT is used as input (x = 0-5, y = 0-7).

1.2. Control Register

All control registers used by the IO module are shown in the following table:

Category	Abbreviation	Function Description
Module control	PxCRy（x=0-5，y=0-7）	IO port input and output control register
	PxPCRy（x=0-5，y=0-7）	IO pull-up resistor control register
Module output and input	Px.y（x=0-5，y=0-7）	IO port data register
Data pointer selection	INSCON	Special function register page selection

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

***Note**: Before setting the P5, P5CR, P5PCR registers in BANK1, please set the BSK0 bit of INSCON to 1 before operating it.*

1.3. IO Set

1.3.1 IO Output Set

IO as the output mode, you need to set its PxCRy (x = 0-5, y = 0-7) to 1, at this time, if you set Px.y (x = 0-5, y = 0-7) to 0, then IO outputs low level, and the value of low level is GND. If Px.y (x = 0-5, y = 0-7) is set to 1, IO outputs high level, and the value of high level is VDD.

1.3.2 IO Input Set

IO as the input mode, you need to set its PxCRy (x = 0-5, y = 0-7) to 0, if you set PxPCRy (x = 0-5, y = 0-7) to 0, the IO is in Enter the Floating state. If PxPCRy (x = 0-5, y = 0-7) is set to 1, the pull-up resistor is turned on and IO is in the input high state. The high level of the IO input is 0.8VDD and the low level of the IO input is 0.2VDD. For specific parameters, please refer to "SH79F6481A DATASHEET".

1.3.3 Open drain I / O settings

When the TWI function of SH79F6481A is enabled, the corresponding port will be automatically set to N-channel open-drain output. By setting the TWIPCR bit in TWITOUT, the internal pull-up resistor is turned on. The user can choose to connect or turn on the internal pull-up resistor according to the actual application.

1.4. Application Examples

1.4.1 Demand

Input a 0 ~ 5V triangle wave from P3.0, the value of P3.0 is output through P5.0.

1.4.2 Examples

```
bit a;
void main(void)
{
    P3CR &= 0xFE;            // P3.0 Input
    P3PCR |= 0x01;           // P3.0 Open the pull-up resister.
    INSCON |=0X40;           //Switch to BANK1
    P5CR |= 0x01;            // P5.0 Output
    INSCON & = 0XBF;         //Switch to BANK0
    while(1)
    {
        a = P3_0;
        INSCON |=0X40;           //Switch to BANK1
        P5_0 = a;                // P5.0 Output Value as the P3.0 Output Value
        INSCON & = 0XBF;         //Switch to BANK0
    }
}
```

2. SH79F6481A Flash&EEPROM User Guide

2.1 General Description

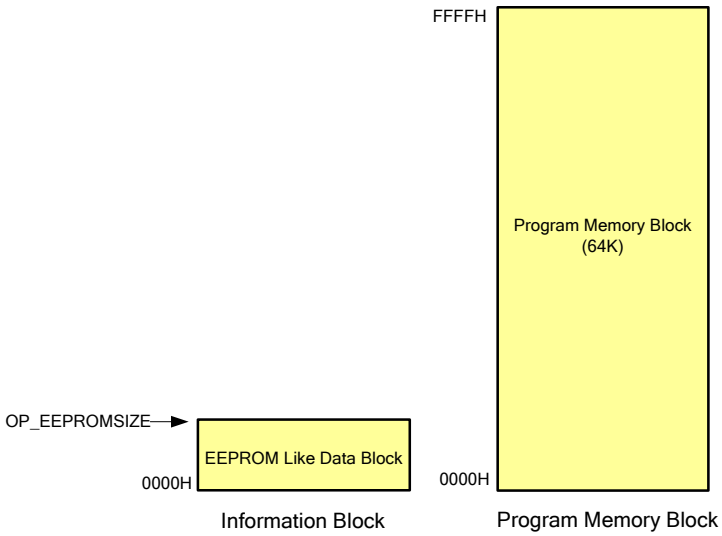
The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's Flash&EEPROM features:

- Flash memory includes 128 * 512Byte blocks，total 64KB
- EEPROM like memory 0 ~ 4KB (code option)
- Programming and erasing can be done over the full operation voltage range
- Write, read and erase operation are all supported by In-Circuit Programming (ICP)
- Fast mass/sector erase and programming
- Minimum program/erase cycles:

  Main program memory: 10,000

  EEPROM like memory: 100,000
- Minimum years data retention: 10
- Low power consumption



The SH79F6481A embeds 64K flash program memory for program code. The flash program memory provides electrical erasure and programming and supports In-Circuit Programming (ICP) mode and Self-Sector Programming (SSP) mode. Every sector is 512 bytes.

The SH79F6481A also embeds 4096 bytes EEPROM-like for program data with 512 bytes per sector, and maximum support 8 sectors. EEPROM Data block is located in Flash memory and sharing the space with program memory block. For example, when OP_EEPROMSIZE = 0000, selected 4KB EEPROM, the program memory size is 64KB-4KB=60KB, when OP_EEPROMSIZE = 0100,

selected 2KB EEPROM, the program memory size is 64KB-2KB=62KB. The specific content of selecting EEPROM size refers to code option chapter.

2.2 Control Register

All control registers used by the Flash&EEPROM module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	XPAGE	Address selection register for programing
	IB_OFFSET	Address offset register for programing
	IB_DATA	Data register for programing
	IB_CON1	SSP type selection register
	IB_CON2	SSP process control register 1
	IB_CON3	SSP process control register 2
	IB_CON4	SSP process control register 3
	IB_CON5	SSP process control register 4
	FLASHCON	Access control register

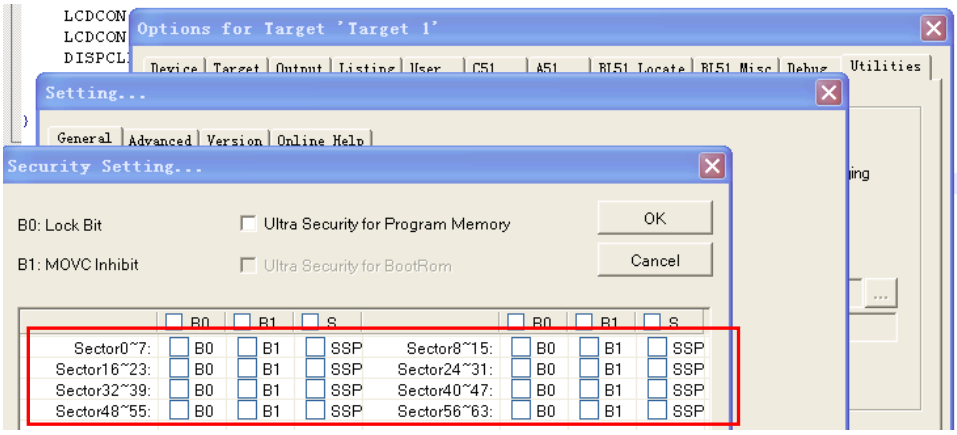
For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

2.3. SSP Programming

2.3.1. Flash SSP Programming

If you use the SSP function to erase or write the Flash, first, you need to set the FAC position of FLASHCON to 0, and then set the corresponding XPAGE, IB_OFFSET, IB_DATA (address and data to be operated), and then set IB_CON1, if IB_CON1 is 0XE6, the operation is sector erase. if IB_CON1 is 0x6E, the corresponding operation is program to memory cell. Then set IB_CON2 to 0x05, IB_CON3 to 0x0A, IB_CON4 to 0x06, and IB_CON5 to 0x09 to complete the entire Flash operation. It should be noted that the sector where the SSP program is located cannot be selected as the operation object in the SSP sector erase operation.

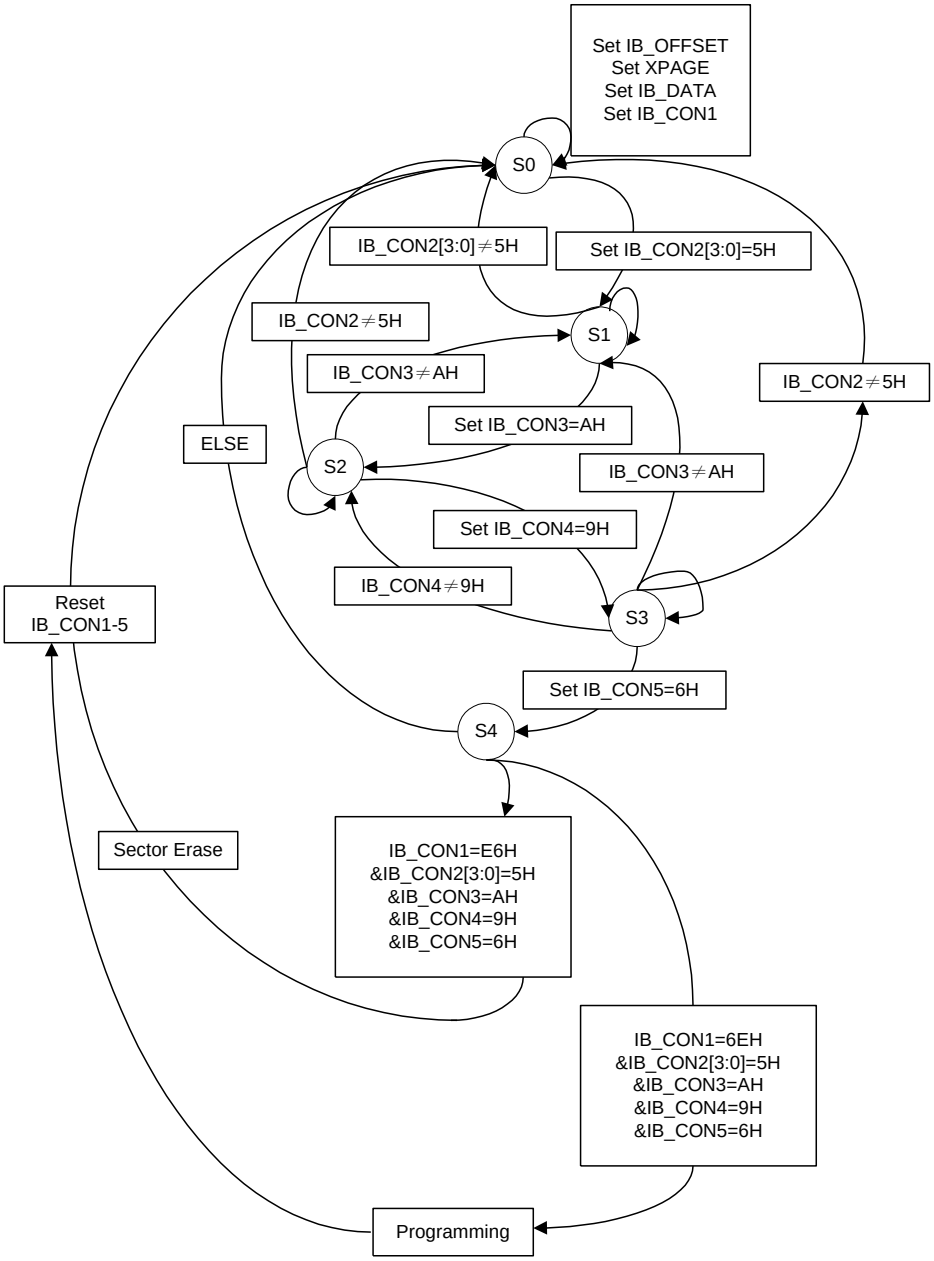
In particular, if you need to protect specific sectors, and prevent SSP disoperation, you can set Sector encryption in Keil through the "Options-> Utilities-> Settings-> Security" path. Among them, B0 encryption corresponds to Flash code protection mode 0; B1 encryption corresponds to Flash code protection mode 1. Please refer to "SH79F6481A DATASHEET" in detail about flash code protection mode introduce.



2.3.2. EEPROM SSP Programming

If you use the SSP function to erase or program the EEPROM, first, you need to set the FAC bit of FLASHCON, and then set the corresponding XPAGE, IB_OFFSET, IB_DATA (address and data to be operated), and then set IB_CON1, if IB_CON1 is 0xE6, then corresponding The operation is sector erase. If IB_CON1 is 0x6E, the corresponding operation is memory cell programming. Then set IB_CON2 to 0x05, IB_CON3 to 0x0A, IB_CON4 to 0x06, and IB_CON5 to 0x09 to complete the entire operation of the EEPROM. Flash control flow chart is shown in 2.3.3.

2.3.3. Flash & EEPROM SSP Control Flow Chart



2.3.4. Application Examples

2.3.4.1. Demand

Erase the Flash area first, then write 00 in sector0, 01 in sector1, and so on, and write 0F to sector15 in turn.。

2.3.4.2. Examples

```
#include <SH79F6481A.h>
#include <intrins.h>
```

```
#include <absacc.h>
#define uchar unsigned char
#define uint unsigned int
#define NOP _nop_();
uchar i,j,m;
uint n;
uint code *p;
uchar count = 0;
uchar Ssp_flag;

void main (void)
{
    uchar temp;
    CLKCON = 0;
    m = 0;
    n = 0;
    Ssp_flag = 0xA5;
/********************sector erase****************/
    EA = 0;
    for(i=0x00;i<0x40;i++)
    {
        FLASHCON = FLASHCON&0xFE;
        NOP
        EA = 0;
        IB_CON1 = 0xE6;
        IB_CON2 = 0x05;
        IB_CON3 = 0x0A;
        IB_CON4 = 0x09;
        if(!(Ssp_flag==0xA5)) goto Error;
        XPAGE = (i<<1)&0xFE;
        IB_CON5 = 0x06;
        NOP
        NOP
        NOP
        NOP
```

```
            NOP
        }
/********************sector write****************/
temp = 0x00;
for(m=0x00;m<0x10;m++)              //m sector
        {
        for(n=0x00;n<512;n++)
            {
                temp = m;
                FLASHCON = FLASHCON&0xFE;
                NOP
                EA = 0;
                IB_OFFSET = n;
                XPAGE |= n>>8;
                XPAGE |= m<<1;
                IB_DATA = temp;
                IB_CON1 = 0x6E;
                IB_CON2 = 0x05;
                IB_CON3 = 0x0A;
                IB_CON4 = 0x09;
                if(!(Ssp_flag==0xA5)) goto Error;
                IB_CON5 = 0x06;
                NOP
                NOP
                NOP
                NOP
                NOP
                XPAGE = 0;
            }
        }
while(1);
Error:
        Ssp_flag = 0;
        IB_CON1 = 0x00;
        IB_CON2 = 0x00;
```

```
    IB_CON3 = 0x00;
    IB_CON4 = 0x00;
    IB_CON5 = 0x00;
    XPAGE = 0x00;
    FLASHCON = 0x00;
    EA = 0;
    while(1);
}
```

2.4. Readable Identification Code

Each chip of SH79F6481A is cured with a 40-bit readable identification code after leaving the factory. Its value is a random value of 0-0xffffffffff. It cannot be erased (stored in the address information storage area 0x127b-127f), which can be programmed Or programming tool to read.

Example of program readout：

```
#include <SH79F6481A.H>
#include <intrins.h>
#include <absacc.h>
unsigned char Temp1,Temp2,Temp3,Temp4,Temp5;
void main()
{
    FLASHCON = 0x01;
    Temp1 = CBYTE[0x127b];
    Temp2 = CBYTE[0x127c];
    Temp3 = CBYTE[0x127d];
    Temp4 = CBYTE[0x127e];
    Temp5 = CBYTE[0x127f];
    FLASHCON = 0x00;
    while(1);
}
```

2.5. SSP Programming Note

To successfully complete SSP programming, the user's software must following the steps below:

(1) For Code/Data Programming：

1. Disable interrupt;
2. Fill in the XPAGE, IB_OFFSET for the corresponding address;

3. Fill in IB_DATA if programming is wanted;
4. Add 4 NOP for more stable operation;
5. Add 4 NOP for more stable operation;
6. Code/Data programming, CPU will be in IDLE mode;
7. Go to Step 3 if more data are to be programmed;
8. Clear XPAGE; enable interrupt if necessary.

(2) For Sector Erase：

1. Disable interrupt;
2. Fill in the XPAGE for the corresponding sector;
3. Fill in IB_CON1-5 sequentially;
4. Add 4 NOPs for more stable operation;
5. Sector Erase, CPU will be in IDLE mode;
6. Go to step 2 if more sectors are to be erased;
7. Clear XPAGE; enable interrupt if necessary.

(3) For Code Reading:

Just Use “MOVC A, @A+DPTR” or “MOVC A, @A+PC”.

(4) For EEPROM-Like

Steps is same as code programming, the differences are:

1. Set FAC bit in FLASHCON register before programming or erase EEPROM-Like.
2. One sector of EEPROM-Like is 256 bytes.

Note:

1. To ensure normal programming the system clock must above 200KHz.

2. It is needed to clear FAC after reading readable random code, otherwise it will influence on the instructions execution of reading program ROM.

2.6. Super anti-interference measures for FLASH / EEPROM-like programming / erasing

1）When downloading the program, select “Enable LVR function” in the code options.

2） When setting the sector, write immediate data to XPAGE.

3） Before calling the programming or erasing function, set a flag, such as 0A5H; Determine whether this flag is 0A5H in the programming or erasing function, otherwise clear this flag and exit the function.

The following EEPROM examples are for reference only:

C file：

```
uchar Ssp_flag;
```

```
/********************SSP Erase****************/
    Ssp_flag = 0xA5;
    EA = 0;
    FLASHCON = 0x01;
    IB_CON1 = 0xE6;
    IB_CON2 = 0x05;
    IB_CON3 = 0x0A;
    IB_CON4 = 0x09;
    if(!(Ssp_flag==0xA5)) goto Error;
    XPAGE = 0x01;          //XPAGE X write direct data
    IB_CON5 = 0x06;
    NOP
    NOP
    NOP
    NOP

/********************sector write****************/
    Ssp_flag = 0x5A;
    EA = 0;
    FLASHCON = 0x01;
    IB_DATA = data;        //data is programming data
    IB_OFFSET = addr;      //addr is the low address programming data
    IB_CON1 = 0x6E;
    IB_CON2 = 0x05;
    IB_CON3 = 0x0A;
    IB_CON4 = 0x09;
    if(!(Ssp_flag==0x5A)) goto Error;
    XPAGE = 0x01;          // Write high address address, write direct data
    IB_CON5 = 0x06;
    NOP
    NOP
    NOP
    NOP
    while(1);
    Error:
```

```
Ssp_flag = 0;
IB_CON1 = 0x00;
IB_CON2 = 0x00;
IB_CON3 = 0x00;
IB_CON4 = 0x00;
IB_CON5 = 0x00;
XPAGE = 0x00;
FLASHCON = 0x00;
EA = 0;
while(1);
```

3. SH79F6481A Timer3 User Guide

3.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed. Suitable for small appliances, white appliances, LED display.

SH79F6481A's Timer3 features:

- Timer3 is a 16-bit auto-reload timer that can work in power-down mode.

Timer3 is a 16-bit auto-reload timer, accessed through two data registers TH3 and TL3, controlled by T3CON register. The ET3 bit of the IEN1 register is set to 1 allow timer3 interrupt.(Refer to Interrupt Section for details)

Timer3 has only one operating mode: 16-bit Counter/Timer with auto-reload.Timer3 also supports selectable pre-scale setting and Operation during CPU Power-Down mode.

Timer3 consists of a 16-bit counter/reload register (TH3, TL3). When writing to TH3 and TL3, they are used as timer load register. When reading from TH3 and TL3, they are used as timer counter register. Setting the TR3 bit enables Timer 3 to count up. The Timer will overflow from 0xFFFF to 0x0000 and set the TF3 bit. This overflow also causes the 16-bit value written in timer load register to be reloaded into the timer counter register. Writing to TH3 also can cause the 16-bit value written in timer load register to be reloaded into the timer counter register.

Read or write operation to TH3 and TL3 should follow these steps:

Write operation: Low nibble first, High nibble to update the counter

Read operation: High nibble first, Low nibble followed.

Timer3 can operate even in CPU Power-Down mode.

If T3CLKS[1:0] is 00, Timer 3 can’t work in Power Down mode.

If T3CLKS[1:0] is 01, when T3 port input external clock, Timer3 can work in CPU normal operating or Power Down mode(entering Power Down mode when system clock is high frequency).

If T3CLKS[1:0] is 10, Timer3 can work in Power Down mode.Timer3 can’t work if the low frequency is closed.

It can be described in the following table.

T3CLKS[1:0]	Oscillator status	Can work in normal mode	Can work in Power Down mode
00	Unlimited	YES	NO
01	Unlimited	YES	YES
10	Low frequency is on, and low frequency is off in power-down mode	YES	NO
	Low frequency is on, and low frequency is on in power-down mode	YES	YES

3.2. Control Register

All control registers used by the Timer3 module are shown in the following table:

Category	Abbreviation	Function Description
Module control	T3CON	Control Register
	TL3	Low Bit Register
	TH3	High bit Register
Data pointer selection	INSCON	Special function register page selection

***Note*：**

1. *If timer3's clock is not the system clock, ensure TR = 0, when read or write theTH3 and TL3.*
2. *When Timer 3 uses the T3 port as the clock source and TR3 changes from 0 to 1, the falling edge of T3 is invalid within 1.5 system cycles.*
3. *The relevant register of timer3 are located in BANK1. For related operations of Timer3 register, please first set the BSK0 bit in INSCON to 1.*
4. *INSCON register needs to be pushed and un-stacked when entering and leaving an interrupt.*

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

3.3. Timer3 Set

Timer3 has only one working mode: 16-bit auto-reload counter / timer, and Timer3 can count in power-down mode, but counting in power-down mode is required for the working clock of Timer3, please refer to the table in the General Description.

To use Timer3, you need to set the working clock of Timer3, and then set the prescaler of the working clock, and then write TL3 and TH3 that meet your requirements. Then start Timer3.

Note: Timer3 interrupt flag is cleared by hardwareTimer3

3.4. Application Example

3.4.1. Demand

The Timer3 count clock selects the clock input from the T3 port. The clock frequency of the T3 port input is 2 kHz. Set Timer3 to wake up Power-Down once every second.

3.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.H>
void Timer3_init(void)
{
    CLKCON |= 0x00;     // Set the 24MHz as the system clock
    INSCON |= 0x40;     // Switch to BANK1
    T3CON = 0x01;       // T3 port is used as the clock input port of Timer3.
    TL3 = 0x30;         // Generate an interrupt at 1s
    TH3 = 0XF8;
    INSCON &= 0xBF;     // Switch to BANK0
}

void main(void)
{
    Timer3_init();
    EA = 1;
    IEN0 |= 0x20;               // Allow Timer3 interrupt
    INSCON |= 0x40;             // Switch to BANK1
    TR3 = 1;                    // Timer3 starts counting
    INSCON &= 0xBF;             // Switch to BANK0
    while(1)
    {
        SUSLO = 0x55;
        PCON |= 0x02;           // Enter power down mode
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
```

```
}

voidtimer3_int(void)    interrupt 5
{
    _push_(INSCON);    // Enter the interrupt and push INSCON to the stack
    INSCON = 0x40;      // Switch to BANK1
    TF3 = 0;
    INSCON = 0x00;     // Switch to BANK0
    _pop_(INSCON);     // INSCON out of stack
}
```

4. SH79F6481A Timer4 User Guide

4.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's Timer4 features:

- Timer 4 is a 16-bit auto-reload timer

Timer 4 is a 16-bit auto-reload timer. The two data registers TH4 and TL4 can be accessed as a 16-bit register, controlled by T4CON register. The ET4 bit in the IEN0 register is set to 1 to allow timer 4 interrupts (see the DATASHEET interrupt section for details).

When TH4 and TL4 are written, they are used as timer reload registers, and when they are read, they are used as count registers. The TR4 bit is set to cause timer 4 to start counting up. The timer overflows from 0xFFFF to 0x0000 and sets the TF4 bit to 1. At the same time, the 16-bit data of the timer reload register is reloaded into the count register. Writing to TH4 also causes the data of the reload register to be reloaded into the count register.

TH4 and TL4 read and write operations follow the following sequence:

Write operation: low bit first then high bit

Read operation: first high bit and then low bit

4.2. Control Register

All control registers used by the Timer4 module are shown in the following table:

Category	Abbreviation	Function Description
Module control	T4CON	Control register
	TL4	Low counter
	TH4	High counter

Note：

1、The relevant register of timer3 are located in BANK1. For related operations of Timer3 register, please first set the BSK0 bit in INSCON to 1.

2、INSCON register needs to be pushed and un-stacked when entering and leaving an interrupt.

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

4.3. Timer4 settings

Timer 4 has two working modes: 16-bit auto-reload timer and 16-bit auto-reload timer with T4 edge trigger. These modes are set by T4M [1: 0] in the T4CON register. Timer4 overflow can trigger ADC conversion.

4.3.1. Mode 0 setting

Timer 4 in mode 0 is a 16-bit auto-reload timer. If you want to use its function, you need to set T4PS [1: 0] to select a good frequency division ratio, choose the initial values of TL4 and TH4, and finally set TR4 to 1, so that Timer4 mode 0 starts, when the timer overflows from 0xFFFF to 0x0000 The TF4 bit is set to 1 in parallel, which can be used to query or generate an interrupt.

4.3.2. Mode 2 setting

Timer4 operates as 16-bit timer in Mode2.Timer4 can select system clock as clock source， other setting accords with mode 0.

The difference between mode 2 and mode 0 is that the automatic reload of mode 2 is triggered with an edge. In this mode 2, the T4 port cannot be used as an external clock input, and its function is to trigger Timer4 automatic reload. If you want to use its function, you need to set T4M [1: 0] to 10 (falling edge trigger) or 11 (rising edge trigger), then set T4PS [1: 0] to select a good frequency division ratio, and then set TC4 to 0. Timer4 can't be triggered anymore. If it is 1, Timer4 can be triggered again. Choose the initial value of TL4 and TH4. Finally, set TR4 to 1, so that the falling edge of TC4 can start Timer4. When the TF4 bit is 1, it will stop Timer4 timing and wait for the next falling or falling edge of T4 to start Timer4 (TC4 = 1).

Note: *When Timer 4 is used as a counter, the input signal frequency of the T4 pin should be less than half of the system clock.*

4.4. Application Examples

4.4.1. Mode 0 demand

The Timer4 clock is the system clock, and a 100ms timer interrupt is generated periodically.

4.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.H>
void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01;        //P1.0 as the output flag
```

```
}

void Timer5_init(void)
{
    CLKCON = 0x20;    //Set 12MHz as the system clock
    INSCON = 0x40;
    T4CON &= 0XF3;   // Working mode selection Mode0
    T4CON &= 0xFE; // Timer4 clock selects system clock.T4 port is used as ordinary IO port.
    T4CON & = 0xEF;
    T4CON | = 0x20;
    TL4 = 0x3E;          // Generate an interrupt at 100ms
    TH4 = 0x49;
}

void main(void)
{
    Timer4_init();
    Port_init();
    EA = 1;
    ET4 |= 0x40;                    //Enable the interrupt of timer4
    T4CON |= 0x40;
    TR4 = 1;                        //Enable Timer4
    while(1);
}

void timer4_int(void)    interrupt 9
{
    _push_(INSCON);
    INSCON = 0X40;     // Into the interrupt to push the INSCON stack
    TF4 = 0;
    INSCON = 0X00;    // INSCON out of stack
    _pop_(INSCON);
    P1_0 = ~P1_0;       // The reversal period of P1.0 is 200ms, that is, the frequency is 5Hz
}
```

4.4.3. Mode2 Demand

A P1.0 rollover occurs 100ms after each falling edge of T4.

4.4.4. Mode2 Example

```
#include <SH79F6481A.H>
#include <intrins.H>

void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01;          // P1.0 as output flag
}

void Timer4_init(void)
{
    CLKCON = 0x00;      //System clock =   12MHz
    INSCON = 0x40;
    T4CON |= 0x0C;    // Select Mode2，Trigger on rising edge
    T4CON &= 0xEF;
    T4CON |= 0x20;    // Prescaler ratio selection 1/64
    T4CON |= 0x40;    // TC4 = 1, Can be triggered again
    TL4 = 0x3E;          // Generate an interrupt at a time of 100ms
    TH4 = 0X49;
    INSCON = 0x00;
}

void main(void)
{
    Timer4_init();
    Port_init();
    EA = 1;
    IEN1 |= 0x40;      // Enable timer4 interrupt
    INSCON = 0x40;
    TR4 = 1;             // Enable timer4
    INSCON = 0x00;
```

```
    while(1);
}

voidtimer4_int(void)    interrupt 9
{
    _push_(INSCON);         // Into the interrupt to push the INSCON stack
    INSCON = 0x40;
    TF4 = 0;
    INSCON = 0x00;
    P1_0 = ~P1_0;           // The cycle of P1.0 is 200ms, that is, the frequency is 5Hz
    _pop_(INSCON);          // INSCON out of stack
}
```

5. SH79F6481A Timer5 User Guide

5.1 General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's Timer5 features:

- Timer 5 is a 16-bit auto-reload timer

Timer 5 is a 16-bit auto-reload timer. The two data registers TH5 and TL5 can be accessed as a 16-bit register, which controlled by T5CON register. The ET5 bit in the IEN0 register is set to 1 to allow timer 5 interrupts (see the DATASHEET interrupt section for details).

When TH5 and TL5 are written, they are used as timer-reload registers, and when they are read, they are used as count registers. The TR5 bit is set to cause timer 5 to start counting up. The timer overflows from 0xFFFF to 0x0000 and sets the TF5 bit to 1. At the same time, the 16-bit data of the timer reload register is reloaded into the count register. Writing to TH5 also causes the data of the reload register to be reloaded into the count register.

TH5 and TL5 read and write operations follow the following sequence:

Write operation: low bit first then high bit

Read operation: first high bit and then low bit

5.2. Control Register

All control registers used by the Timer5 module are shown in the following table:

Category	Abbreviation	Function Description
Module control	T5CON	**Control register**
	TL5	Low counter
	TH5	High counter

Note: *The registers related to Timer5 are located in Bank1. When performing related operations on the registers related to Timer5, please first set the BSK0 bit in the INSCON register to 1. INSCON register needs to be pushed and un-stacked when entering and leaving an interrupt.*

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

5.3. Timer5 settings

Timer 5 has a working model: 16-bit auto-reload timer.

Timer5 can only use the system clock as the module clock. When using it, first set the rescaled selection bits T5PS [1: 0] according to the required timing time base combined with the system

clock, then set TR5 to 1, enable Timer5, timing The device overflows from 0xFFFF to 0x0000 and sets the TF5 bit to 1 for query and interrupt. At the same time as overflow, the 16-bit data of the timer reload register is reloaded into the counter register.

5.4. Application Examples

5.4.1. Demand

Timer5 cycle timing generates a 5mS timer interrupt.

5.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.H>

void Port_init(void)
{
    P1 = 0x00;
    P1CR = 0x01;            // P1.0 as the output flag
}
void Timer5_init(void)
{
    CLKCON = 0x20;          //Put the12MHz as system clock
    INSCON = 0x40;
    T5CON = 0x00;           // Timer5 clock is selected as system clock = 12MHz
    TL5 = 0XA0;             // Generate an interrupt at 5ms
    TH5 = 0X15;
    INSCON = 0x00;
}

void main(void)
{
    Timer5_init();
    Port_init();
    EA = 1;
    ET5 |= 1;               // Enable the interrupt of timer5
    INSCON = 0x40;
    T5CON &= 0x02;          // timer5 start counting
    INSCON = 0x00;
```

```
    while(1);
}

voidtimer5_int(void)    interrupt 3
{
    _push_(INSCON);  // Enter the interrupt and push INSCON to the stack
    INSCON = 0x40;
    T5CON &= 0x7F;
    INSCON = 0x00;
    _pop_(INSCON);    // INSCON out of the stack
    P1_0 = ~P1_0;       // The reversal period of P1.0 is 10ms, that is, the frequency is 100Hz
}
```

6. SH79F6481A SPI User Guide

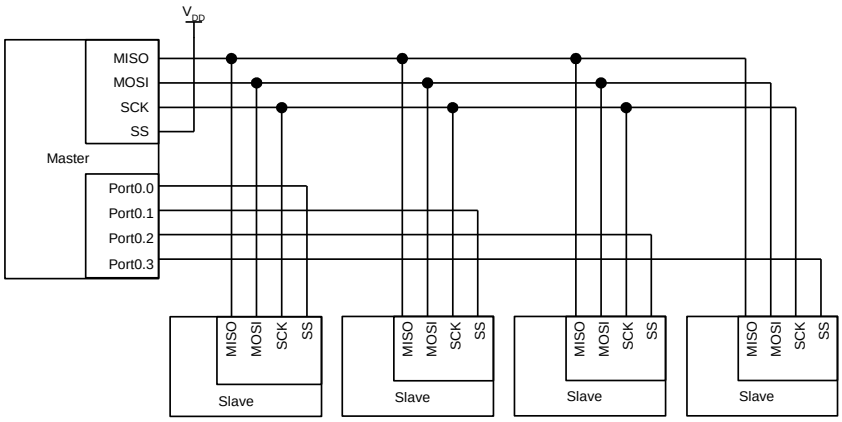
6.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's SPI features:

- Full-duplex, three-wire synchronous transfers
- Master or slave operation
- Seven programmable master clock rates
- Serial clock with programmable polarity and phase
- Master mode fault error flag with MCU interrupt capability
- Write collision flag protection
- Selectable LSB or MSB transfer

The Serial Peripheral Interface (SPI) Module allows full-duplex, synchronous, serial communication between the MCU and peripheral devices, including other MCUs.
The following diagram shows a typical SPI bus configuration using one master controller and many slave peripherals. The bus is made of three wires connecting all the devices. The master device selects the individual slave devices by using four pins of a parallel port to control the four $\overline{SS}$ pins of the Slave devices.



6.2. Control Register

All control registers used by the EUART module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	SPCON	Control Register
	SPSTA	Status Register
	SPDAT	Data Register

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note: *The INSCON register needs to be pushed and un-stacked when entering and leaving an interrupt.*

6.3. SPI Setting

6.3.1. Baud Rate Setting

In Master mode，the baud rate of SPI has 8 selectable frequencies, which are 4, 8, 16, 32, 64, 128, 256 or 512 of the internal clock, which can be set by SPR [2: in the SPCON register: 0] bit selection.

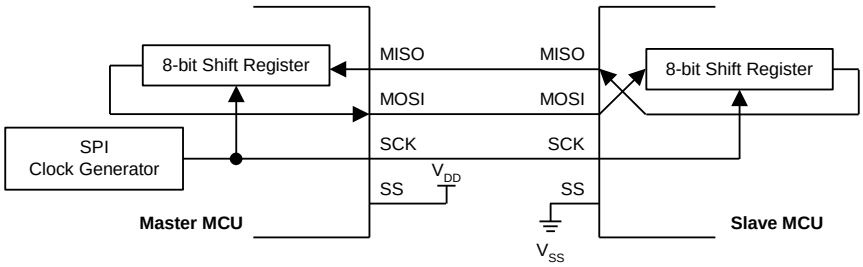
6.3.2. Work Mode Setting

SPI can be configured as one of master mode or slave mode. The configuration and initialization of the SPI module is accomplished by setting the SPCON register (serial peripheral device control register) and SPSTA (serial peripheral device status register). After the configuration is completed, the data transfer is completed by setting SPCON, SPSTA, SPDAT (Serial Peripheral Device Data Register).

During SPI communication, data is moved in and out serially synchronously. The serial clock line (SCK) synchronizes the movement and sampling of data on the two serial data lines (MOSI and MISO). The slave device selection line（$\overline{SS}$）can independently select the SPI slave device; if the slave device is not selected, it cannot participate in activities on the SPI bus.

When the SPI master device transmits data to the slave device through the MOSI line, the slave device sends data to the master device through the MISO line as a response, which realizes synchronous full-duplex transmission of data transmission and reception under the same clock.

The transmit shift register and the receive shift register use the same special function device address. Writing to the SPI data register SPDAT will write to the transmit shift register, and reading the SPDAT register will obtain the data of the receive shift register.



Full duplex master-slave interconnection diagram

- Master Mode

（1） Mode Start

The SPI master device controls the start of all data transfer on the SPI bus. When the MSTR bit in the SPCON register is set, the SPI is operating in master mode, and only one master device can initiate the transfer.

（2） Send

In SPI master mode, write a byte of data to the SPI data register SPDAT, the data will be written to the transmit shift buffer. If there is already data in the transmit shift register, then the main SPI generates a WCOL signal to indicate that writing is too fast. But the data in the transmission shift register will not be affected, and the transmission will not be interrupted. In addition, if the transmit shift register is empty, the master immediately shifts out the data in the transmit shift register to the MOSI line serially according to the SPI clock frequency on SCK. When the transfer is complete, the SPIF bit in the SPSTA register is set. If the SPI interrupt is enabled, an interrupt will also be generated when the SPIF bit is set.

（3） Receive

When the master device transmits data to the slave device through the MOSI line, the corresponding slave device also transmits the contents of its transmit shift register to the receive shift register of the master device through the MISO line to achieve full-duplex operation. Therefore, the SPIF flag bit 1 indicates that the transmission is completed and the data reception is completed. The data received from the device is stored in the receive shift register of the master device according to the MSB or LSB priority transmission direction. When a byte of data is completely moved into the receive register, the processor can obtain the data by reading the SPDAT register. If an overrun occurs (the SPIF flag is not cleared to 0, the next transmission is attempted), the RXOV bit is set to 1, indicating that a data overrun has occurred. At this time, the receive shift register retains the original data and the SPIF bit is set, so that until the SPIF bit Cleared to 0, the SPI master will not receive any data.

- Slave Mode

（1） Mode Start

When the MSTR bit in the SPCON register is cleared, the SPI operates in slave mode. Before the data is transferred, the $\overline{SS}$ pin of the slave device must be set low, and it must be kept low until one byte of data is transferred.

（2） Send and Receive

In the slave mode, according to the SCK signal controlled by the master device, data is shifted in through the MOSI pin, and the MISO pin is shifted out. A bit counter records the number of edges of SCK. When the receive shift register shifts in 8-bit data (one byte) and the shift register shifts out 8-bit data (one byte), the SPIF flag bit is set to 1. Data can be obtained by reading the SPDAT register. If the SPI interrupt is enabled, when SPIF is set to 1, an interrupt will also be generated.

To prevent overruns, the SPI slave device must also clear the SPIF flag bit before moving data into the receive shift register, otherwise the RXOV bit is set to 1, indicating that a data overrun has occurred. At this time, the receive shift register retains the original data and the SPIF bit is set, so that the SPI slave device will not receive any data until SPIF is cleared to 0.

The SPI slave device cannot start data transfer, so the SPI slave device must write the data to be transferred into the transmit shift register before the master device starts a new data transfer. If the slave device does not write data before the first transmission, the slave device will transfer the "0x00" byte to the master device. If the write SPDAT operation occurs during the transfer process, the WCOL flag of the SPI slave device is set to 1, that is, if the transfer shift register already contains data, the WCOL bit of the SPI slave device is set to 1, indicating a write SPDAT conflict. However, the data in the shift register is not affected, and the transfer is not interrupted.

6.3.3. Error detection

The flag bit in the SPSTA register indicates an error in SPI communication:

（1） Mode failure（MODF）

The mode fault error in SPI master mode indicates that the level state on the $\overline{SS}$ pin is inconsistent with the actual device mode. After the MODF bit in the SPSTA register is set to 1, it indicates that there is a multi-master conflict in the system control. In this case, the SPI system is affected as follows:

（2） Write conflict (WCOL)

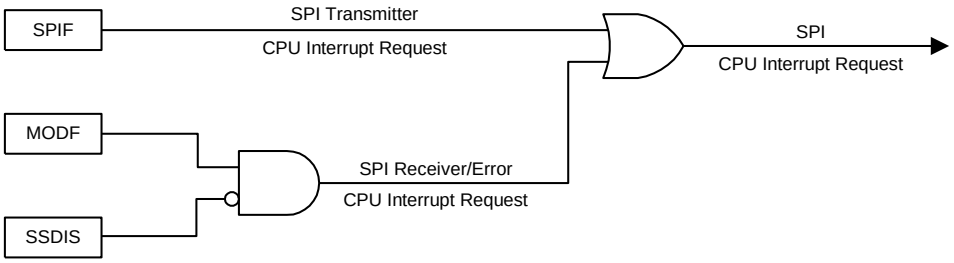
Writing to the SPDAT register during a data sequence will cause a write conflict, and the WCOL bit in the SPSTA register is set. Setting the WCOL bit will not cause an interrupt, nor will the transmission be aborted. WCOL bit needs to be cleared by software

（3） Overrun situation (RXOV)

When the master or slave device has not cleared the SPIF bit, and the master or slave device attempts to send several data bytes again, an overrun occurs. In this case, the receive shift register retains the original data, SPIF is set to 1, the same SPI device will not receive data until SPIF is cleared. Continue to call the interrupt before the SPIF bit is cleared, and the transmission will not be aborted. Setting the RXOV bit will not cause an interrupt. The RXOV bit must be cleared by software.

Two SPI status flags SPIF & MODF can generate a CPU interrupt request. Serial peripheral device data transmission flag, SPIF: set by hardware after sending a byte.

Mode failure flag, MODF: When this bit is set to 1, the level on the $\overline{SS}$ pin is inconsistent with the SPI mode. SSDIS bit is 0 and MODF is set to 1 will generate SPI receiver / error CPU interrupt request. When SSDIS is set to 1, no MODF interrupt request is generated.



SPI interrupt request generation

6.4. Application Examples

6.4.1. Demand

In main mode, SCK = SYS / 64, invalid SS pin, send 0x00 ~ 0xFF non-stop.

Examples

```
#include <SH79F6481A.H>
#include <intrins.h>
unsigned char out_flag = 0;
unsigned char r_data = 0;
void spi_init(void)
{
    CLKCON = 0x00;      // System clock = 24Mz
    SPCON |= 0x40;      // Master mode
    SPCON |= 0x08;      // Close SS Pin
    SPCON |= 0x05;      // Serial external device clock rate = system clock frequency / 64
    SPSTA = 0x00;
}
 void    main(void)
{
    spi_init();
    EA = 1;
    IEN1 |= 0x01;       //Enable SPI interrupt
    SPSTA |= 0x80;      // Enable SPIi module
    SPDAT = 0x00;
    while(1)
    {
        if(out_flag == 1)
        {
```

```
                SPDAT ++;
                out_flag = 0;
            }
        }
}
void spi_interrupt(void) interrupt   19
{
    _push_(INSCON);         // Into the interrupt to push the INSCON stack
    out_flag = 1;
    SPSTA &= 0xBF;
    _pop_(INSCON);          // INSCON out of the stack
}
```

7. SH79F6481A EUART User Guide

7.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's EUART features:

- SH79F6481A with 3 EUART, compatible with traditional 8051.
- EUART0/EUART1/EUART2 comes with a baud rate generator; The baud rate can choose the system clock frequency division or the baud rate generator overflow rate of 1/16.
- Enhanced functions include frame error detection and automatic address recognition
- Support TTL level conversion
- EUART has four working methods

7.2. Control Register

All control registers used by the EUART module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	SCON	EUART control and status register
	SBUF	EUART data register
	PCON	Power control register
	SADDR	EUART slave address register
	SADEN	EUART address mask register
	SFINE	Baud rate trimmer register
	SBRT	Baud Rate Generator Register
	UTOS	TTL level enable register

Note:

1. *The relevant registers of EUART1/ EUART2 are in BANK1. Before operating them, please set the BSK0 bit in INSCON. The INSCON register needs to be pushed and popped from the stack when the INSCON register is interrupted.*
2. *Suggest to set the I/O output height for Tx.*

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

7.3. EUART Settings

The EUARTx can be operated in 4 modes. Users must initialize the SCONx，involves selection of the Mode and the baud rate before communication.

In all four modes, any write operation that uses SBUF as the target register will initiate the transmission. In mode 0, the condition RI = 0 and REN = 1 initiate the reception. This will generate a clock signal on the TXD pin, and then shift the 8-bit data up on the RxD pin. In other modes, the reception is initiated by the input start bit. Start condition for external transmission by sending start bit.

EUARTx Mode Summary

SM0	SM1	Mode	Type	Baud Clock	Frame	Start Bit	Stop Bit	9th bit
0	0	0	Sych	f_{SYS}/(4 or 12)	8 bits	None	None	None
0	1	1	Ansych	own baud rate/16	10 bits	1	1	None
1	0	2	Ansych	f_{SYS}/(32 or 64)	11 bits	1	1	0,1
1	1	3	Ansych	own baud rate/16	11 bits	1	1	0,1

7.3.1. Mode0 setting

To use mode 0, first ,set the SM2 bit (SCON.5) to 0 or 1, and the baud rate is fixed to 1/12 or 1/4 of the system clock. When the SM2 bit is 0, the serial port runs at 1/12 of the system clock. When set to 1, the serial port runs at 1/4 of the system clock.

Then any write operation that uses SBUF as the target register will start the transmission. The next system clock TX control block starts to transmit. Data conversion occurs on the falling edge of the shift clock, the contents of the shift register are shifted from left to right one by one, and the empty position is 0. When all 8 bits in the shift register are transmitted, the TX control module stops the transmission operation and then sets TI to 1 (SCON.1) on the next rising edge of the system clock.

If data is received, REN (SCON.4) is set and RI (SCON.0) is cleared to initialize reception. The next system clock starts reception, latches data on the rising edge of the shift clock, and the contents of the receive conversion register are shifted to the left one by one. When all 8 bits have been received in the receive shift register, the RX control block stops receiving, and then RI is set to 1 on the rising edge of the next system clock until reception is cleared by software.

7.3.2. Mode1 settings

Method 1 provides 10-bit full-duplex asynchronous communication, the 10-bit data consists of a start bit (logic 0), 8 data bits (lower bit is the first bit), and a stop bit (logic 1). During reception, these 8 data bits are stored in SBUF and the stop bit is stored in RB8 (SCON.2). The baud rate in mode 1 is fixed at 1/16 of the overflow rate of the built-in baud rate generator.

Any write operation that uses SBUF as the target register will start the transmission, and also load TB8 into the 9th bit of the transmission shift register. The transmission actually starts from the system clock after the next transition in the divide-by-16 counter because this bit is synchronized with the divide-by-16 counter, but it is not synchronized with the write operation to SBUF. The start bit is first shifted out on the TXD pin, then the 9-bit data. After all 9 bits of data in the transmit

conversion register are sent, the stop bit is shifted out on the TXD pin, and the TI flag is set when the stop bit starts to transmit.

Reception is only allowed when the REN bit is set. When the RXD pin detects a falling edge, the serial port begins to receive serial data. For this reason, the CPU continuously samples RXD at a sampling rate of 16 times the baud rate. When a falling edge is detected, the divide-by-16 counter is immediately reset, which helps synchronize the divide-by-16 counter with the serial data bit on the RXD pin. The divide-by-16 counter divides the time of each bit into 16 states. In the 7th, 8th, and 9th states, the bit detector samples the level of the RXD terminal. In order to suppress noise, at least 2 times of sampling value consistent data in these 3 state samples are received. If the first bit received is not 0, it means that this bit is not the start bit of a frame of data. This bit is ignored and the receiving circuit is reset, waiting for the arrival of another falling edge on the RXD pin. If the start bit is valid, then shift into the shift register, and then shift into other bits to the shift register. After 8 data bits and 1 stop shift in, the contents of the shift register are loaded into SBUF and RB8 respectively, RI is set to 1, but the following conditions must be met:

1. RI = 0
2. SM2 = 0 or received stop bit = 1.

If these conditions are met, the stop bit is loaded into RB8, 8 data bits are loaded into SBUF, and RI is set. Otherwise the received frames will be lost. At this time, the receiver will re-detect whether the RXD terminal has another falling edge. The user must clear the RI with software before receiving it again.

7.3.3. Mode2 setting

Method 2 uses 11 bits in asynchronous full-duplex communication. A frame consists of a start bit (logic 0), 8 data bits (lower bit is the first bit), a programmable ninth data bit and a stop bit (logic 1). Method 2 supports multi-machine communication and hardware address recognition (see the multi-machine communication chapter for details). During data transfer, the ninth data bit (TB8 in SCON) can be written with 0 or 1, for example, the parity bit P in the PSW or the data / address flag bit in multi-machine communication. When data is received, the 9th data bit goes into RB8 and the stop bit is not saved. The SMOD bit in PCON selects the baud rate as 1/32 or 1/64 of the system operating frequency.

Any write operation that uses SBUF as the target register will start the transmission, and also load TB8 into the 9th bit of the transmission shift register. In fact, the transmission is started from the system clock after the next transition in the divide-by-16 counter, so the bit time is synchronized with the divide-by-16 counter and not synchronized with the write operation to SBUF. The start bit is first shifted out on the TXD pin, then the 9-bit data. After all 9 bits of data in the transmit conversion register are sent, the stop bit is shifted out on the TXD pin, and the TI flag is set when the stop bit starts to transmit.

Reception is only allowed when the REN bit is set. When the RXD pin detects a falling edge, the serial port begins to receive serial data to this end,

The CPU continuously samples RXD at a sampling rate of 16 times the baud rate. When a falling edge is detected, the divide-by-16 counter is immediately reset. This helps the divide-by-16 counter synchronize with the serial data bits on the RXD pin. The divide-by-16 counter divides the time of each bit into 16 states. In the 7th, 8th, and 9th states, the bit detector samples the level of the RXD terminal. In order to suppress noise, at least 2 times of sampling value consistent data in these 3 state samples are received. If the first bit received is not 0, it means that this bit is not the start bit of a frame of data. This bit is ignored and the receiving circuit is reset, waiting for the arrival of another falling edge on the RXD pin. If the start bit is valid, then shift into the shift register, and then shift into other bits to the shift register. After 9 data bits and 1 stop shift in, the contents of the shift register are loaded into SBUF and RB8 respectively, RI is set to 1, but the following conditions must be met:

1. RI = 0
2. SM2 = 0 or the 9th bit received = 1, and the received byte matches the actual slave address

If these conditions are met, the ninth bit is shifted into RB8, the 8-bit data is shifted into SBUF, and RI is set to 1. Otherwise, the received data frames will be lost.

Among the stop bits, the receiver returns to looking for another falling edge on the RXD pin. The user must clear the RI with software before receiving it again.

7.3.4. Mode 3 setting

Method 3 uses the transmission protocol of method 2 and the baud rate generation method of method 1.

7.3.5. EUART baud rate setting

In mode 0, the baud rate can be programmed to 1/12 or 1/4 of the system clock, which is determined by the SM2 bit. When SM2 is 0, the serial port runs at 1/12 of the system clock. When SM2 is 1, the serial port runs at 1/4 of the system clock.

In Mode 1 and Mode 3, the built-in baud rate generator is used, the baud rate can be fine-tuned, and the accuracy is a system clock. The formula is as follows:

$$BaudRate = \frac{Fsys}{16 \times (32768 - SBRT) + SFINE}$$

For example: Fsys = 8MHz, you need to get the baud rate of 115200Hz, SBRT and SFINE values are calculated as follows:

8000000/16/115200 = 4.34

SBRT = 32768 - 13 = 32764

115200 = 8000000/(16 X 4 + SFINE)

SFINE = 5.4 ≈ 5

The actual baud rate calculated by this fine-tuning method is 115942, and the error is 0.64%;

In Mode 2, the baud rate is fixed at 1/32 or 1/64 of the system clock, which is determined by the SMOD bit (PCON.7). When the SMOD bit is 0, EUART runs at 1/64 of the system clock. When the SMOD bit is 1, the EUART runs at 1/32 of the system clock.

$$BaudRate = 2^{SMOD} \times (\frac{f_{SYS}}{64})$$

7.3.6. EUART TTL level setting

For the application of 5V power supply, the communication level of UART and 3V device does not match. The EUART of SH79F6481A adds the TTL level conversion function of RXD pin, which is controlled by the UTOS register.

When the bit of UTOS is 0, the input high threshold of RXD pin is 0.8VDD, and the input low threshold is 0.2VDD. At this time, if the SH79F6481A's VDD = 5V, the input high threshold of the RXD pin is 4V, and the input low threshold is 1V, unable to communicate normally with the UART of the 3V device; when bit1 = 1 of UTOS, if VDD = 4.5V ~ 5.5V, the input high threshold of RXD pin is 2.0V, and the input low threshold is 0.8V. If VDD = 2.7V ~ 4.5V, the input high threshold of the RXD pin is 0.25VDD + 0.8, and the input low threshold is 0.15VDD, which can correctly receive the data sent by the 3V device.

7.4. Application Examples

7.4.1. Demand

Use EUART for serial communication, the baud rate of communication is 9600, 1 start bit, 8 data bits, 1 stop bit, the communication content is: send the received data out after receiving a data.

7.4.2. Examples

```
#include <SH79F6481A.H>
unsigned char     r_data;
unsigned char     out_flag;

void uart0_init(void)
{
    CLKCON = 0x40;              //Put the 6MHz as system clock
    SBRTH=0Xff;
    SBRTL=0xd9;
    SFINE=0x01;
```

```
    SCON = 0x50;                    //Work mode select
    }
void main(void)
{
    uart0_init();
    /* TX corresponding IO is set to output high */
    P0CR |= 0x02;
    P0 |= 0x02;         //TXD P0.1 OUTPUT HIGH
    r_data = 0x00;
    out_flag = 0;
    TI = 0;
    RI = 0;
    IEN0 = 0x90;
    REN = 1;
    TR2 = 1;
    while(1)
    {
        if(out_flag == 1)
        {
            SBUF = r_data;                  // Send the received data
            out_flag = 0;
        }
    }
}
void uart0_int(void) interrupt  4
{
    _push_(INSCON);         // Enter the interrupt and push INSCON to the stack
    if(TI == 1)
    {
        TI = 0;
    }
    else if(RI == 1)
    {
        r_data = SBUF;
        RI = 0;
```

```
            out_flag = 1;
        }
        _pop_(INSCON);          // INSCON out of the stack
    }
```

7.5. UART1/2

The usage of UART1 is the same as UART0, it should be noted that the relevant registers of UART1/UART2 are in BANK1.

8. SH79F6481A ADC User Guide

8.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's ADC features：

- 12-bit resolution
- The highest conversion rate can reach 1Msps
- The reference voltage VDD
- 9 external analog inputs, 1 internal analog channel
- 4 trigger sources can be selected to automatically trigger AD conversion
- Support sequence conversion, and each channel can be configured as any one of multiple analog inputs
- The time interval between adjacent channel conversions during sequence conversion can be set by software
- The 7-channel (AN0-AN2/AN4-7) ADC conversion rate is 100KSPS
- The 2-channel (AN3/AN8) ADC conversion rate is up to 1MSPS

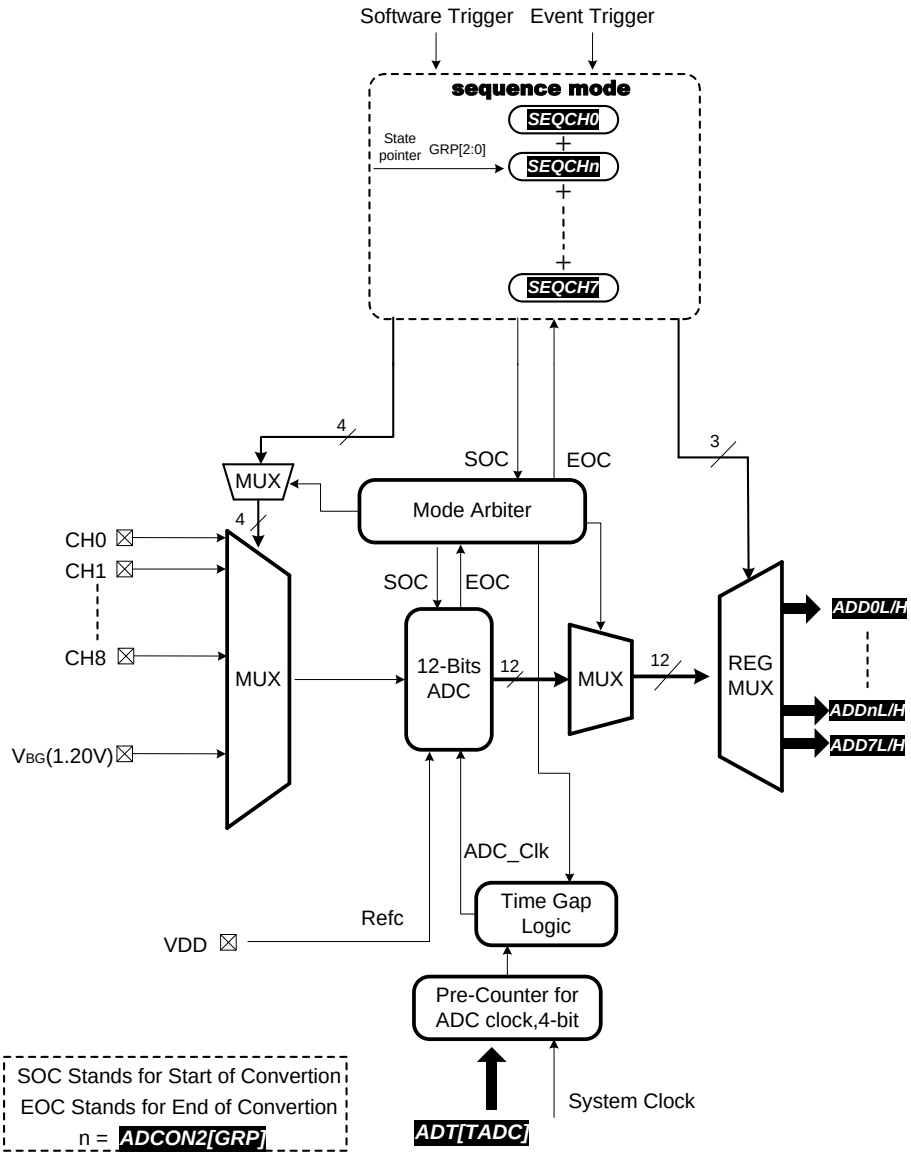
SH79F6481A includes a single-ended, 12-bit successive approximation analog-to-digital converter (ADC, Analog-to-Digit Converter), the module is shown in the figure. ADC is responsible for the conversion of analog signals to corresponding 12-bit digital signals. When the input is GND, the output is 0; when the input is greater than or equal to VDD – 1LSB, the maximum value is output. The reference voltage of the ADC can be VDD.

There are 9 analog inputs (CH0 – CH8, VBG) in this module, which can be programmed into the sequence for automatic conversion by configuring the channel register. The result is stored in the corresponding result registers ADDxH, ADDxL (x = 0-8), and the value of the result register is updated every time the sequence is converted. The mapping relationship between the result register and the analog input can be arbitrarily set to form a conversion sequence, and an analog input channel can be repeatedly set in the sequence, so that the result of multiple consecutive conversions of this analog input channel can be obtained in the result register.

For two of these channels, the conversion rate can be up to 1MSPS, and the ADC clock rate and sampling time can be set by registers. The time interval between adjacent channels in the sequence can also be set by register (TGAP [2: 0]).

The ADC module can work in Idle mode, and the ADC interrupt can wake up Idle mode. In Power-Down mode, the ADC module does not work.

The ADC module diagram is as follows:



8.2. Control Register

All control registers used by the Timer4 module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	ADT	Configuration of ADC clock, sample time
	ADCON1	Enable of ADC module, start a conversion, reference voltage selection, trigger mode and sources, interrupt flag
	ADCON2	The total number of a sequence and the time interval between neighboring channels selection
	SEQCON	Channels and the mapping control of the conversion result, alignment of conversion result selection
	ADCH1	Set the pin used as AD channel function or IO function
	ADCH2	Set the pin used as AD channel function or IO function
	SEQCHx	Configurate the channel and sequence , x = 0 - 7
	ADDxL	The low 8 bits conversion result of the channel which have been selected by SEQCHx (x = 0 – 7).
	ADDxH	The high 8 bits conversion result of the channel which have been selected by SEQCHx (x = 0 – 7).

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note:

It is necessary to push and pop the INSCON register when entering and exiting the interrupt.

8.3. ADC Setting

The Approach for AD Conversion by software:

(1) Enable ADC module

(2) Set sequence, including the number of channels and analog input channel

(3) Clear ADCIF zero

(4) Set GO/$\overline{\text{DONE}}$ to1,start ADC conversion

(5) Wait until GO/$\overline{\text{DONE}}$ = 0 or ADCIF = 1, if the ADC interrupt is enabled, the ADC interrupt will occur, user need clear ADCIF by software.

(6) By setting the mapping register to acquire the converted data of each channel from ADDxH/ADDxL in turn.

(7) Repeat step 2~6 if another conversion is required.

The Approach for AD Conversion by hardware:

(1) Enable ADC module

(2) Set sequence, including the number of channels and analog input channel

(3) Set the trigger source

(4) Clear ADCIF zero

(5) If the ADC interrupt is enabled, the ADC interrupt will occur, user need clear ADCIF by software.

(6) By setting the mapping register to acquire the converted data of each channel from ADDxH/ADDxL in turn.

8.4. Application Examples

8.4.1. Demand

Sequentially sample the input voltage of the AN7, AN6, AN5, AN4, AN3, AN2, AN1, and AN0 ports, Vref = Vdd, and the conversion result stort in the ADC_res array..

8.4.2. Examples

```
#include "SH79F6481A.h"
#include "intrins.h"
#include "cpu.h"
UINT16 ADC_res[8];
void init_adc()
{
    UCHAR i;
    ADCON1 = 0x80;    // ADC reference voltage select VDD, no trigger
    ADT=0x21;     // Put the 24M as sysclk clock, 500k sps
    ADCON2 = 0x72;    //The sampling interval time of 8 sampling channels is: 4Tad
    ADCH1 = 0xff;// P3.7 ~ P3.4, P4.3 ~ P4.0 as ADC sampling channels AN7 ~ AN0
                 // The sampling sequence of the configuration channel is: AN7, AN6, AN5, AN4,
                 //AN3, AN2, AN1, AN0
```

```
    SEQCON = 0x0;     //Configure SEQCH0
    SEQCHX = 0x7;
    SEQCON = 0x01;   // Configure SEQCH1
    SEQCHX = 0x06;
    SEQCON = 0x02;   // Configure SEQCH2
    SEQCHX = 0x05;
    SEQCON = 0x03;   // Configure SEQCH3
    SEQCHX = 0x04;
    SEQCON = 0x04;   // Configure SEQCH4
    SEQCHX = 0x03;
    SEQCON = 0x05;   // Configure SEQCH5
    SEQCHX = 0x02;
    SEQCON = 0x06;   // Configure SEQCH6
    SEQCHX = 0x01;
    SEQCON = 0x07;   // Configure SEQCH7
    SEQCHX = 0x0;
    SEQCON &= 0x7f;         // The conversion result is left aligned
    ADCON1 |= 0x01;     // Start conversion
    while(ADCON1 & 0x01);// Detecting whether the conversion is complete
    for(i = 0; i < 8; i++)    // Get conversion results
    {
        SEQCON = i;
        ADC_res[i] = ((ADDXH << 4) + (ADDXL >> 4));
    }
}
```

9. SH79F6481A PCA User Guide

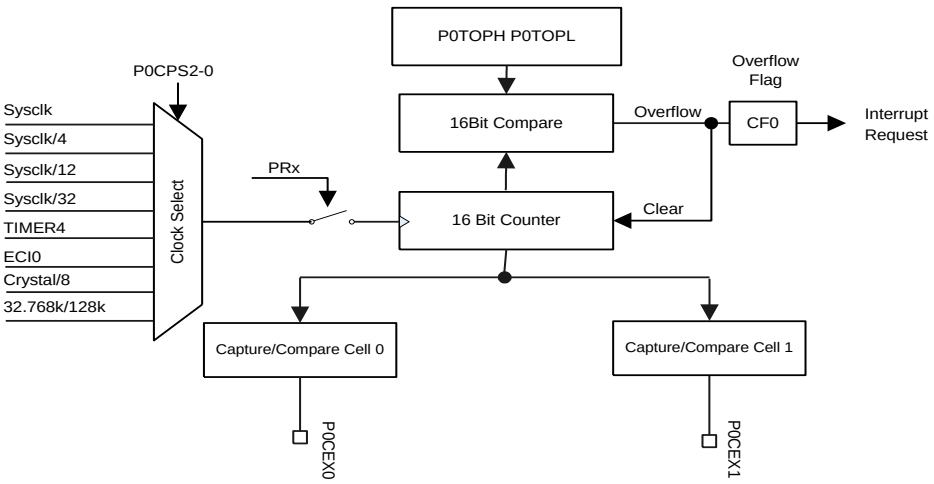
9.1 General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's PCA features：

- SH79F6481A have a PCA, which is a general purpose 16-bit Timer/Counter module, with two independent Output Compare Units.
- Phase Correct PWM Mode, Phase frequency Correct PWM Mode

The Programmable Counter Array (PCA0) provides enhanced timer functionality while requiring less CPU intervention than the standard 8051 counter/timers. The PCA0 consists of a dedicated 16-bit counter/timer and two 16-bit capture/compare modules. The PCA0 block diagram is shown blow, Each capture/compare module has its own associated I/O line (P0CEXn(n=0,1).



The 16-bit PCA0 counter/timer consists of a 16-bit SFRs, PxTOPH and PxTOPL.consist of the 16-bit counter/timer. PxTOPH and PxTOPL can be configured TOP value, initial value of PxTOPH and PxTOPL is 0XFFFF.

16 bit counter / timer is the most basic module of PCA0, Enable or disable bit PRX of PCACON register can Start or Stop counter, when PR0 is set to logic '0', 16 bit counter was also forced the clear '0'. When the counter/timer overflows , The counter counts from BOTTOM to TOP then restarts from BOTTOM when PCA work in single-slope operation or The counter counts repeatedly from BOTTOM to TOP and then from TOP to BOTTOM when PCA work in dual-slope operation. the Counter Overflow Flag (CF0) in P0CF is set to logic 1 and an interrupt request is generated (Setting the ECF0 bit in P0CMD to logic 1 enables the CF0 flag to generate an interrupt request).

Register P0TOPH and P0TOPL, P0CPHn and P0CPLn Write operation and Read operation rules: Write operation: first write MSB, then write LSB Read PxTOPH and PxTOPL、PxCPHn and PxCPLn had no effect on count.

9.2 Control Register

All control registers used by the PCA module are shown in the following table:

Category	Abbreviation	Function Description
Module control	P0CF	PCA0 Flag Register
	PCACON	PCA Enable Register
	P0CMD	PCA0 Mode Register
	P0CPM0	PCA Capture/Compare Register
	P0FORCE	Forced Output Control Register
	P0TOPL	Count Maximum Low Byte
	P0TOPH	Count Maximum High Byte
	P0CPLn	Capture/Compare Module Low Byte
	P0CPHn	Capture/Compare Module High Byte

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

9.3 PCA settings

9.3.1. PCA Clock Setting

The counter/timer is driven by a programmable time base that can select between eight sources: system clock, system clock divided by 4, system clock divided by 12, system clock divided by 32, timer 4 overflow , an external clock signal on the ECI input pin or built-in128kHzRC, The P0CPS2–P0CPS0 bits in the P0CMD register select the time base for the counter/timer as shown in following table.

P0CPS2	P0CPS1	P0CPS0	Clock source
0	0	0	System clock
0	0	1	system clock divided by 4
0	1	0	system clock divided by 12
0	1	1	system clock divided by 32
1	0	0	Timer 4 overflow
1	0	1	System clock
1	1	0	High-to-low transitions on ECI (max rate = system clock divided by 4)
1	1	1	built-in 128kHzRC

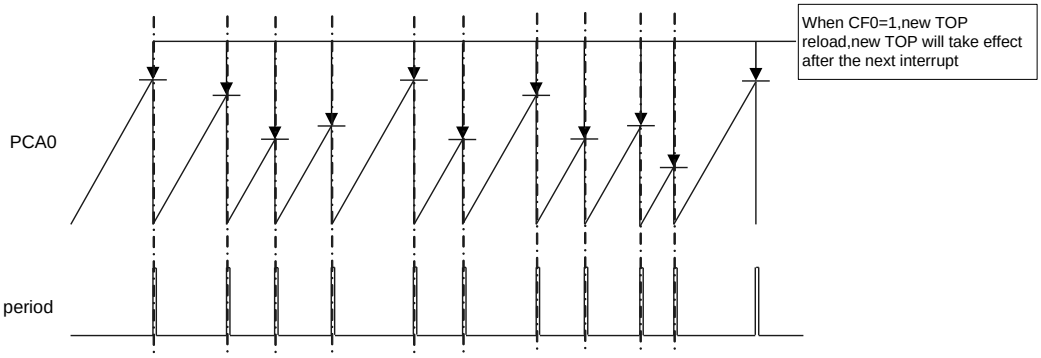
Note:

(1) Max rate value of time base is System clock divided by 4(except system clock as base time) ,otherwise PCA0 will not work well

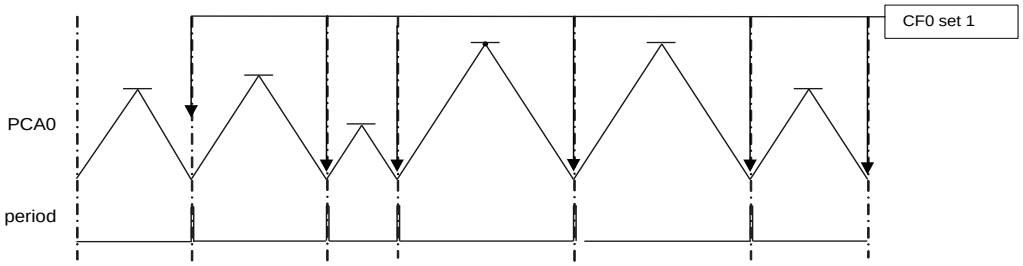
(2) If the clock source is 128kHzRC, the system clock must be dual-clock OP_OSC = 0011.

9.3.2. PCA Count mode setting

The counter/timer of PCA0 has a control bit that can be selected by the counting method. P0SDEN is 0 in the P0CMD register to indicate a single ramp, and 1 is double ramp, as shown in the following figure.



PCA0 counter / timer timing waveform of single slope



PCA0 counter / timer timing waveform of dual ramp

9.3.3. PCA Mode Settings

PCA0 has two independent Output Compare Units. Each module can be configured to operate independently. Each module has Special Function Registers (SFRs) associated with it in the CIP-51 system controller. These registers are used to exchange data with a module and configure the module's mode of operation. It could settings the bit(P0SMPn and P0SMNn) in the P0CPMn registers used to select the PCA capture/compare module’s operating four modes: Edge-triggered Capture, Software Timer, Frequency Output, PWM output.

Working mode selection is show in the following table:

Mode	P0SDEN	P0SMPn	P0SMNn	P0FSPn	P0FSNn	Function
Mode0	0	0	0	0	X	Capture triggered by positive edge (single slope)
				1	0	Capture triggered by negative edge on(single slope)
				1	1	Capture triggered by transition (single slope)
Mode1	0	0	1	0	X	Continuous software timer (single slope)
				1	X	Single software timer(single slope)
Mode2	0	1	0	X	X	frequency output (single slope)
Mode3	0	1	1	0	0	8 bit PWM(single slope)
				0	1	16 bit PWM(single slope)
	1			1	0	16 bit phase correction PWM (dual ramp)
	1			1	1	Phase frequency correction PWM (dual ramp)
Others						PCA0 Counter correctly, but the compare/ capture module does not

Note(used Output Compare Units):

(1) X: Don't Care;

(2) *When PCA0 Counter model work by its single-slope or dual-slope operation, Output Compare Units must be configured the same slope operation, otherwise Output Compare Units will not work.*

(3) *To change the value of P0TOP must ensure that the new P0TOP value not less than the values of all the comparison register.*

(4) *when compare / capture module work as 8-Bit Pulse Width Modulator, or 16-Bit Pulse Width Modulator, setting the P0CPHn equal to 0x00 or P0TOP will result in a constant high or low output(depending on the polarity of output set by the bit P0TCPn)*

(5) *ALL Output Compare Units of PCA0 must work in the same slope operation(example: PCA0 compare / capture module0 and compare / capture module1 can only work in the same slope mode).*

9.3.4. PCA Mode0 Settings

In this mode, a valid transition on the P0CEXn pin causes the PCA0 to capture the value of the PCA counter/timer and load it into the corresponding module's 16-bit capture/compare register (P0CPLn and P0CPHn).

Mode 0: Need to configure bit7:bit4 = 00 of P0CPMn (P0SMPn:P0SMNn)

Positive edge triggered mode: Need to configure bit5:bit4=00 or 01 of P0CPMn (P0SMPn:P0SMNn)

Negative edge triggered mode: Need to configure bit5:bit4=10 of P0CPMn

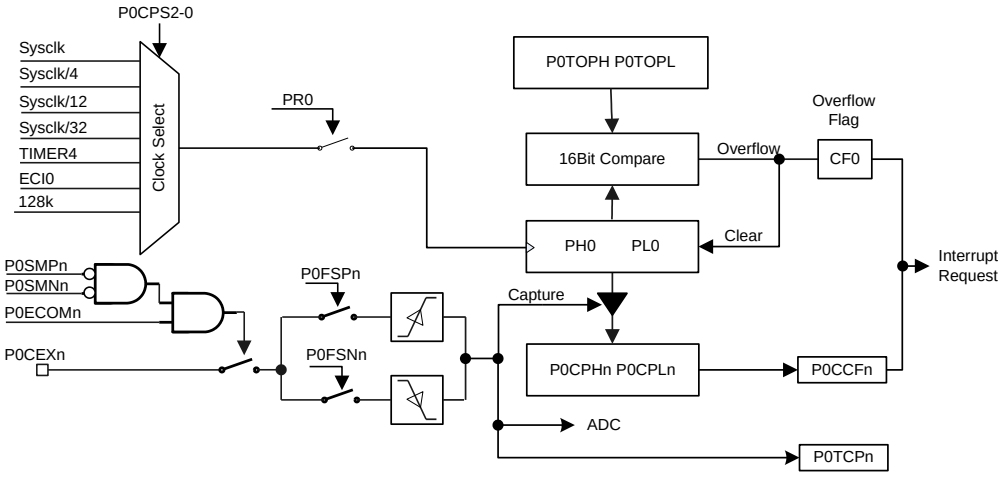
Any edge triggered mode: Need to configure bit5:bit4=11 of P0CPMn

Compare/capture module functions enable: Need to configure bit3=11 of P0CPMn

Interrupt of compare/capture module functions enable: Need to configure bit1=1 of P0CPMn

For a detailed explanation please refer to the description of register P0CPMn in SH79F6481A DATASHEET

The block diagram of the capture mode:



PCA0 Capture Mode Diagram

9.3.5. PCA Mode1 Settings

Mode 1 is called the software timing mode, and the software timer mode is also called the comparison output mode (by configuring P0SMPn: P0SMNn = 01 to enable this mode). In this mode, P0FSPn: P0FSNn = 0x, continuous software timing can be achieved. PCA0 compares the count value of the counter / timer with the module's 16-bit capture / compare registers (P0CPHn and P0CPLn). When a match occurs, the capture / compare flag (P0CCFn) in P0CF is set to logic '1' (when P0MATn = 1) and an interrupt request (such as if the P0CCFn interrupt is enabled), and the logic level on the P0CEXn pin of the module will change (set the P0TCPn bit to enable this function). When the CPU shifts to the interrupt service routine, the P0CCFn bit cannot be automatically cleared by hardware, and must be cleared by software.

Mode 1: P0CPMn bit7: bit6 = 01 (P0SMPn: P0SMNn)

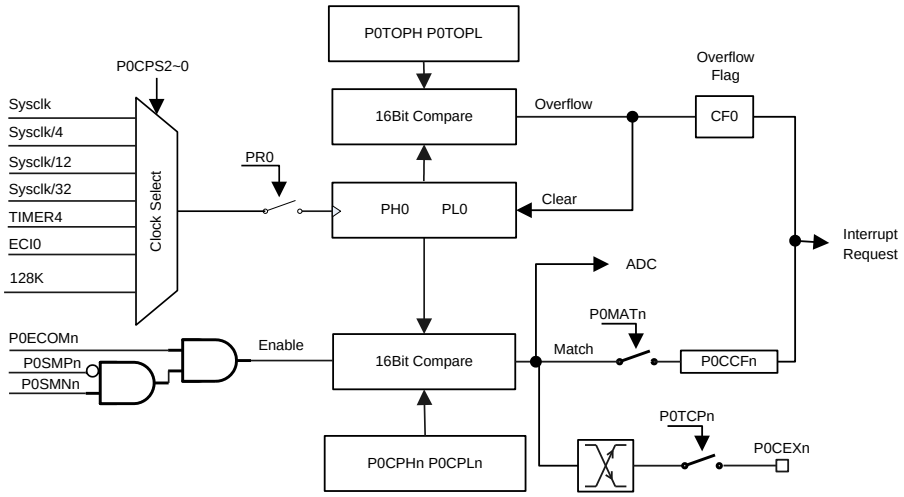
Continuous software timing: P0CPMn bit5: bit4 = 00 or 01 (P0FSPn: P0FSNn)

Single software timing: P0CPMn bit5: bit4 = 10 or 11 (P0FSPn: P0FSNn)

Enable comparison capture module: need to configure bit0 of P0CPMn = 1 (P0ECOMn)

P0CEXn pin output level: need to configure bit2 = 1 of P0CPMn (P0TCPn)

For detailed explanation, please refer to the SH79F6481A DATASHEET register P0CPMn description. The software timing principle block diagram is as follows:



Software Timer Mode Diagram

9.3.6. PCA Mode2 Settings

Mode 2 is called the frequency output mode. The frequency output mode can generate a square wave with a programmable frequency on the P0CEXn pin of the module (configuration P0SMPn: P0SMNn = 10 enables this mode. In this mode, the P0CPn register update does not use double buffering mechanism). The high byte P0CPH0 of the capture / compare module holds the number of PCA clocks to count before the output level changes. The frequency of the generated square wave FP0CEXn = FPCA0 / (2 × P0CPHn)

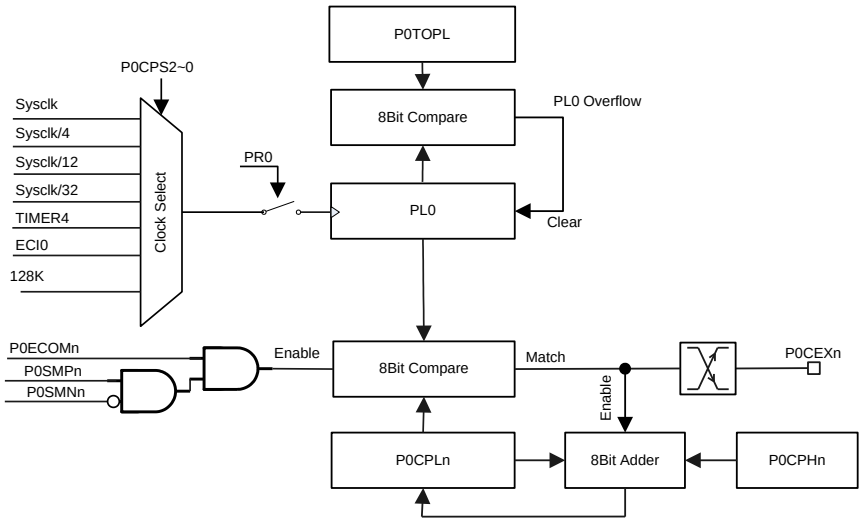
Note: For this equation, when the value in P0CPHn is 0x00, it is equivalent to 256.

The low byte P0CPLn of the capture / compare module is compared with the low byte PLx of the PCA0 counter; if the two match, the level of the P0CEXn pin changes, and the offset value in the high byte P0CPHn is added to P0CPLn, PLx continues to count until it matches again, the level of the P0CEXn pin changes, and the cycle is repeated. The output frequency of the P0CEXn pin is cont by P0CPH0. If a comparison / capture module of PCA0 enables this mode, the value of P0TOPL is fixed at 0XFF, and the user can configure the value of P0TOPH to change the maximum count value.

Mode 2: P0CPMn needs to be configured with bit7: bit6 = 10 (P0SMPn: P0SMNn)

Enable comparison capture module: P0CPMn needs to be configured with bit3 = 1 (P0ECOMn)

Frequency output principle block diagram is as follows:



PCA Frequency Output Mode

9.3.7. PCA Mode3 Settings

Mode3 is called PWM mode. There are four PWM modes, as shown in the following table:

POFSPn	P0FSNn	Function Description
0	0	8-bit PWM (single ramp)
0	1	16-bit PWM (single ramp)
1	0	16-bit phase correction PWM (double ramp)
1	1	16-bit phase frequency correction PWM (double ramp)

Mode 3: need to configure P0CPMn bit7: bit6 = 11

PWM mode needs to configure bit5: bit4 (P0FSPn: P0FSNn), as shown in the table above.

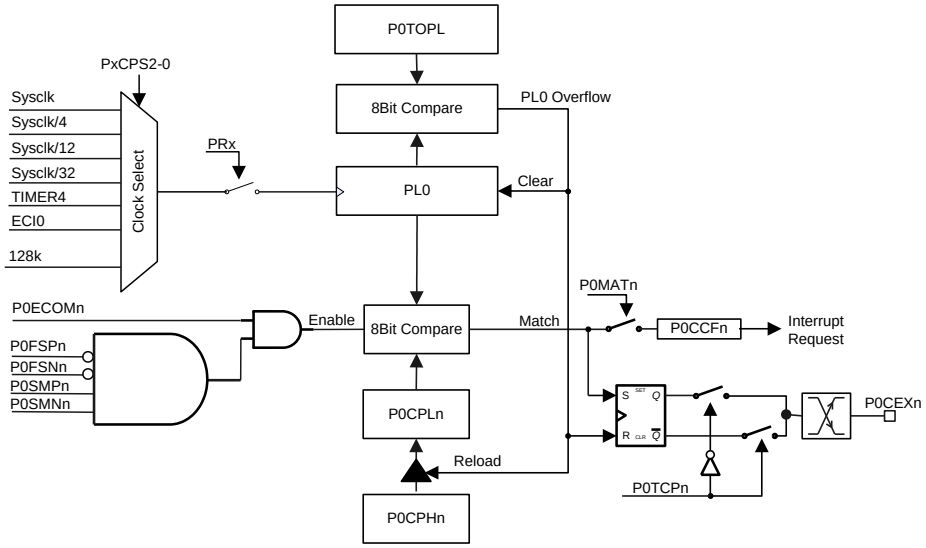
9.3.7.1. 8-bit PWM (single ramp)

When the compare / capture module works in 8-bit PWM function, the lower 8-bit PLx of PCA0 Counter counts up from 0x00 to P0TOPL (single ramp mode), and when PLx overflows (from 0xFF to 0x00), the value stored in P0CPHn is automatically Load into P0CPLn, this process does not require software intervention. When P0TCPn = 0, the low byte (PLx) of PCA counter / timer is equal to the value in P0CPLn, the output on P0CEXn pin is cleared to '0'; When the count value in PLx overflows, the P0CEXn output is set to '1'; when P0TCPn = 1, the P0CEXn pin outputs a waveform with the opposite polarity.

The duty ratio of 8-bit PWM is Duty = (256- (P0CPHn + 1)) / 256.

If a compare / capture module of PCA0 enables this mode, the value of P0TOPL is fixed at 0XFF. The user can configure the value of P0TOPH to change the maximum count value, but does not affect the 8-bit PWM output period.

The principle block diagram is shown below. See SH79F6481A DATASHEET for details.



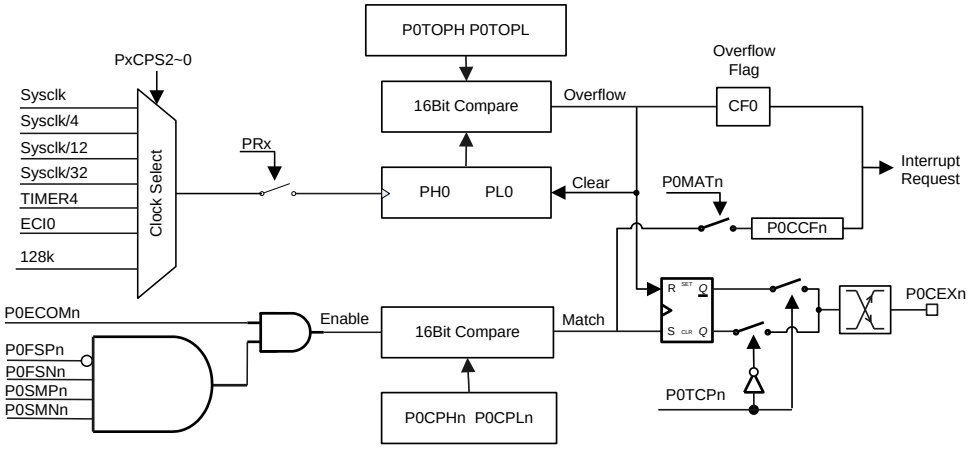
8-Bit PWM Mode Diagram

9.3.7.2. 16-bit PWM (single ramp)

16-Bit Pulse Width Modulator(single slope) is similar to 8-bit Pulse Width Modulator, In this mode, the 16-bit capture/compare module(P0CPn) defines the number of PCA clocks for the low time of the PWM signal. When P0TCPn =0 ,and the PCA counter matches the module contents (P0CPn), the output onP0CEXn is asserted low; when the counter overflows, P0CEXn is asserted High. When P0TCPn =1 , the P0CEXn pin outputs a waveform with the opposite polarity.

The duty ratio of 8-bit PWM is Duty = (65536- (P0CPHn + 1)) / 65536

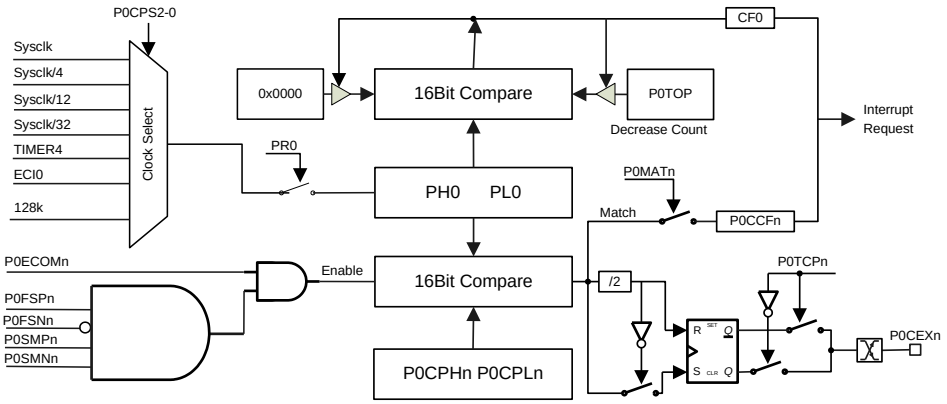
The principle block diagram is shown below. See SH79F6481A DATASHEET for details.



PCA 16-Bit PWM Mode (single slope)

9.3.7.3. 16-bit phase correction PWM (double ramp)

The phase correct Pulse Width Modulation or phase correct PWM mode provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM(0x0000) to TOP(P0TOP) and then from TOP(P0TOP) to BOTTOM(0x0000). When P0TCPNn=0, the Output of P0CEXn pin is cleared on the compare match between PCA0 counter and P0CPn while up counting, and set on the compare match while down counting. When P0TCPn=1, the output is inverted. The realization principle diagram is show shown below. When a match occurs, the Capture/Compare Flag (P0CCFn) in P0CF is set to logic 1 and an interrupt request is generated if interrupts are enabled. when the counter counts from P0TOP to 0x0000 overflow, the Flag CF0 in P0CF is set to logic 1 and an interrupt request is generated if interrupts are enabled.



16 bit phase and frequency correct

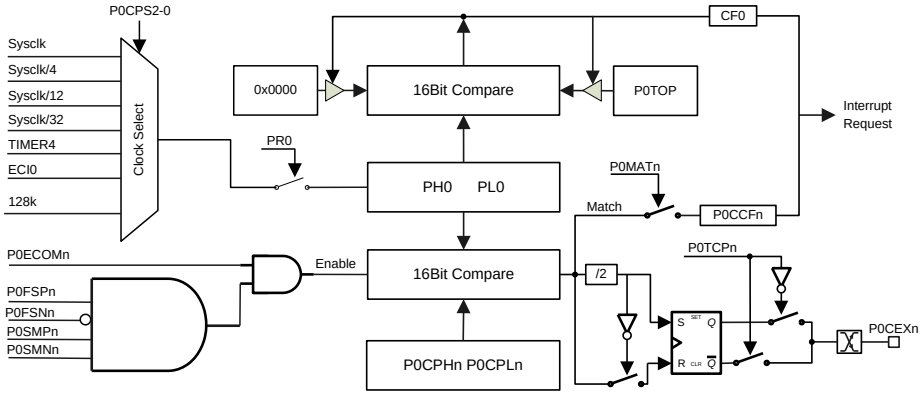
9.3.7.4. 16-bit phase-frequency correction PWM mode (double buffering mechanism, double ramp, BOTTOM terminal update)

Phase and frequency correction PWM (XPPWM) mode Hereinafter referred to as phase-frequency correction PWM mode. Similar to the phase correction mode, this function is also based on double ramp operation.

XPPWM can generate high-precision PWM waveforms with accurate phase and frequency. The block diagram of its implementation is shown in Figure XXX below. The timer repeatedly counts from 0x0000 to P0TOP, and then counts down from P0TOP to 0X0000. When P0TCPn = 0, when the timer counts to P0TOP, if PCA0 Counter matches PxCPn, P0CEXn will be cleared to low level;

and when the timer counts to 0x0000, if PCA0 Counter matches PxCPn, P0CEXn will be set to high power level. When P0TCPn = 1, the P0CEXn pin outputs a waveform with the opposite polarity. The PWM output by the phase frequency correction PWM is a symmetrical signal in all cycles. This is because the phase correction PWM updates the PxCPn and P0TOP register values at the P0TOP point, while the phase frequency correction PWM updates the PxCPn and P0TOP registers at the 0x0000 point Value.

The principle block diagram is shown below. Refer to SH79F6481A DATASHEET for details.



16 bit phase and frequency correct mode

9.4. Application Examples

9.4.1. Mode0 Examples

```
#include "SH79F6481A.h"
uchar Val_l;
uchar Val_h;
void main()
{
    IEN0 |=0x80; //Enable the total interrupt: EA
    IEN2 |= 0x20; // Allow PCA0 interrupt
    INSCON=0x40; //Switch to BANK1
    CEXCR = 0x60;     // P3.4 as P0CEX0,P5.2 as P0CEX1
    ECICR = 0x00;      // P3.6 as ECI0
    P0CMD=0x00;     //  Select system clock, single ramp mode, prohibit overflow interrupt
    P0CPM0=0x39;    //Mode 0, double edge trigger capture, allows capture interrupt
    P0TOPH=0x08;
    P0TOPL=0x00;
    PCACON=0x1; //Start counting
```

```
        While(1);
    }
      void INT_PCA0(void) interrupt 20
    {
            _push_(INSCON);
          INSCON=0x40;
         Val_l=P0CPL0;    // Assign a captured value to a variable after the rising or falling edge
                                 //is filled from P0CEX0
          Val_h=P0CPH0;
          P0CF =0x0;
          _pop_(INSCON);
     }
```

9.4.2. Mode1 Examples

Requirements: P0CEX0 output square wave

```
void main()
{
        IEN0 |=0x80;       // Enable total interrupt EA
        IEN2 |= 0x20;      // Allow PCA0 interrupt
        INSCON=0x40;
        CEXCR = 0x60;      // P3.4 as P0CEX0,P5.2 as P0CEX1
        ECICR = 0x00;       // P3.6 as ECI0
        P0CMD=0x00; // Select system clock, single ramp mode, prohibit overflow interrupt
        P0CPM0=0x47; //mode1，Allow P0CEX0 to output waveform, and allow compare
                                 //capture module interrupt
        P0TOPH=0x40;
        P0TOPL=0x20;
        P0CPH0=0x00;
        P0CPL0=0x20;
        P0CPM0|=0x08;    // Enable compare capture module
        PCACON=0x1;      // Start counting
        While(1);
 }
 void INT_PCA0(void) interrupt 20
 {
```

```
    _push_(INSCON);
    INSCON=0x40;
   P0CF =0x0;
   _pop_(INSCON);
}
```

9.4.3. Mode2 Examples

Requirement: P0CEX0 outputs a square wave with a certain frequency

```
#include "SH79F6481A.h"
void main()
{
IEN0 |= 0x80;       //Enable total interrupt:EA
IEN2 |= 0x20;       //Allow PCA0 interrupt
INSCON=0x40;    //Switch to BANK1
CEXCR = 0x60;      // P3.4 as P0CEX0,P5.2 as P0CEX1
ECICR = 0x00;       // P3.6 as ECI0
P0CMD=0x00;         // Select system clock, single ramp mode, prohibit overflow interrupt
P0CPM0=0x80;       //Mode2
P0CPH0=0x10;
P0CPL0=0x20;
PCACON=0x1;
P0CPM0|=0x08;    // Enable compare capture module
PCACON=0x1;      //Start counting
While(1);
}
void INT_PCA0(void) interrupt 20
{
_push_(INSCON);
INSCON=0x40;
P0CF =0x0;
_pop_(INSCON);
 }
```

9.4.4. Mode3 Examples

9.4.4.1. 8-bit PWM method

Requirement: P0CEX0 and P0CEX1 output a pair of square waves without dead zone

```
#include "SH79F6481A.h"
void main()
{
        IEN0 |= 0x80;    // Enable total interrupt:EA
        IEN2 |= 0x20;          // Allow PCA0 interrupt
        INSCON=0x40;
        CEXCR = 0x60;     // P3.4 as P0CEX0,P5.2 as P0CEX1
        ECICR = 0x00;       // P3.6 as ECI0
        P0CMD=0x80;    // Select system clock, single ramp mode, prohibit overflow interrupt
        P0CPM0=0xC3; // Compare capture module 0 mode3 (8-bit PWM), allows compare
                              //capture module interrupt
         P0CPH0=0x30;
        P0CPL0=0x30;
        P0CPM0|=0x08;// Enable compare capture module 0
        P0CPM1=0xC3;// Compare capture module 1 mode 3 (8-bit PWM), allow compare
                            //capture module interrupt
        P0CPH1=0x30;
        P0CPL1=0x30;
        P0CPM1|=0x0c; // Enable compare capture module 1
        PCACON=0x1;    //Start counting
        while(1);
}
void INT_PCA0(void) interrupt 20
{
        _push_(INSCON);
        INSCON=0x40;
        P0CF =0x0;
        _pop_(INSCON);
}
```

9.4.4.2. 16-bit PWM mode

```
#include "SH79F6481A.h"
```

Requirement：P0CEX0 and P0CEX1 output a pair of square waves without dead zone

```
void main()
{
        IEN0 |= 0x80;     // Enable total interrupt EA
        IEN2 |= 0x20;     //Allow PCA0 interrupt
        INSCON=0x40;
        CEXCR = 0x60;    // P3.4 as P0CEX0,P5.2 as P0CEX1
        ECICR = 0x00;     // P3.6 as ECI0
        P0CMD=0x80;     // Select system clock, single ramp mode, prohibit overflow interrupt
        P0CPM0=0xD3;    //Comparison capture module 0 mode 3 (16-bit PWM), allows
                          //comparison capture module interrupt
        P0CPH0=0x30;
        P0CPL0=0x30;
        P0CPM0|=0x08;    // Enable compare capture module 0
        P0CPM1=0xD3;    // Compare capture module 1 mode 3 (16-bit PWM), allow compare
                         //capture module interrupt
        P0CPH1=0x30;
        P0CPL1=0x30;
        P0CPM1|=0x0c; // Enable compare capture module 1
        PCACON=0x1;    //Start counting
        While(1);
}
void INT_PCA0(void) interrupt 20
{
        _push_(INSCON);
        INSCON=0x40;
        P0CF =0x0;
        _pop_(INSCON);
}
```

9.4.4.3. 16-bit phase adjustment PWM method

Requirement: P0CEX0 and P0CEX1 output a pair of square waves with no dead zone.

```
#include "SH79F6481A.h"
void main()
{
IEN0 |= 0x80;       //EA
```

```
IEN2 |= 0x20;    //Allow PCA0 interrupt
INSCON=0x40;
CEXCR = 0x60;     // P3.4 used as P0CEX0,P5.2 used as P0CEX1
ECICR = 0x00;      // P3.6 used as ECI0
P0CMD=0xC0;       // Select system clock, single ramp mode, prohibit overflow interrupt
P0TOPH=0x01;
P0TOPL=0x00;
P0CPM0=0xE3;      // Comparison capture module 0 mode 3 (16-bit phase correction PWM),
                  //allowing comparison capture module interrupt
P0CPH0=0x0;
P0CPL0=0x90;
P0CPM0|=0x08; // Enable compare capture module 0
P0CPM1=0xE3; /// Compare capture module 1 mode 3 (16-bit PWM), allow compare
                  //capture module interrupt
P0CPH1=0x0;
P0CPL1=0x90;
P0CPM1|=0x0c; //Enable compare capture 1
PCACON=0x1;//Start Counting
while(1);
}
void INT_PCA0(void) interrupt 20
 {
_push_(INSCON);
INSCON=0x40;
P0CF =0x0;
_pop_(INSCON);
}
```

9.4.4.4. 16-bit PWM method

Demand: P0CEX0 and P0CEX1 output a pair of square waves with no dead zone.

```
#include "SH79F6481A.h"
void main()
{
        IEN0 |= 0x80;     // Enable total interrupt EA
        IEN2 |= 0x20;     //Allow PCA0 interrupt
```

```
        INSCON=0x40;
        CEXCR = 0x60; // P3.4 as P0CEX0,P5.2 as P0CEX1
        ECICR = 0x00;       // P3.6 as ECI0
        P0CMD=0x80;     // Select system clock, single ramp mode, prohibit overflow interrupt
        P0CPM0=0xD3; // Comparison capture module 0 mode 3 (16-bit PWM), allows
                        //comparison capture module interrupt
        P0CPH0=0x30;
        P0CPL0=0x30;
        P0CPM0|=0x08; //Enable compare capture mode0
        P0CPM1=0xD3;// Compare capture module 1 mode 3 (16-bit PWM), allow compare
                        //capture module interrupt
        P0CPH1=0x30;
        P0CPL1=0x30;
        P0CPM1|=0x0c; // Enable compare capture module 1
        PCACON=0x1;    //Start counting
        While(1);
}
void INT_PCA0(void) interrupt 20
 {
        _push_(INSCON);
        INSCON=0x40;
        P0CF =0x0;
        _pop_(INSCON);
 }
```

10. SH79F6481A LED User Guide

10.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

The LED driver contains a controller, 8 COM output pins and 12 SEG output pins. Support 1/1 ~ 1/8 duty ratio voltage drive mode.

The LED driver has two operating modes.

Mode 1: On and off LED mode

When the LED driver works in the on and off LED mode, each LEDRAM bit controls an LED light. When the LEDRAM bit is 0, the LED is off, when the LEDRAM bit is 1, the LED is on; in the LED After a frame or a COM scan is completed, the corresponding interrupt flag bit LEDIF or COMIF of the LED driver is set.

Mode 2: Dimming LED mode

When the LED driver works in dimming LED mode, each LEDRAM byte controls the duty cycle of the SEG during the COM cycle being scanned; the duty cycle can be selected in total 256 files; When the LEDRAM byte is When 0xff, SEG outputs the maximum duty cycle, when LEDRAM byte is 0x00, SEG outputs the minimum duty cycle; when LEDRAM byte is the intermediate value of 0x00 ~ 0xff, the corresponding duty cycle of SEG output; for the modification of LEDRAM byte, It will take effect in the next COM scan cycle;

After the completion of each COM cycle scan of the LED, the next COM cycle scan will be performed immediately. The corresponding interrupt flag bit COMIF of the LED driver will be set to 1, because the LED driver works in mode 2, so the LED After the driver is enabled, the COMIF interrupt flag bit will also be set to 1 to facilitate the user to modify the value of LEDRAM; after the scan is completed, the corresponding interrupt flag bit LEDIF of the LED driver will be set to 1;

no matter the LED driver works in mode 1 or mode 2 , The unselected levels of COM and SEG are floating levels, the effective level of COM output is low, and the effective level of SEG output is Output is high. The LEDCOM register controls the number of LED drivers COM; the SEG01 / SEG02 register controls the number of LED drivers SEG; the DISPCOM register can set the width of each COM cycle; in order to prevent the uncertain state between two COM display switching, the LED driver will Insert a dead time between two COMs. During the dead time, the COM outputs a

floating level. The length of the dead time is N system clock widths, which can be set through the LEDDZ register; Therefore, when using the LED interrupt mode detection, before the LED module is enabled, turn on the required LED interrupt source and the total interrupt source.

At power-on reset, pins reset, low voltage reset or watchdog reset, the LED is turned off. When the LED is turned off, the selected COM and SEG outputs are floating.

In IDLE mode, the LED driver remains active, and in Power-Down mode, the LED driver is turned off. In the dual clock mode, the LED works in the high-frequency clock, and the LED clock register DISCOM under the high-frequency clock needs to be set. When switching to the low-frequency clock, the LED clock register DISCOM needs to be modified accordingly.

10.2. Control Register

All control registers used by the LED module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	LEDCON	Control Register
	DISCOM	**Control Register**1
	LEDDZ	Clock Control Register
	SEG01/SEG02	SEG module select register
	LEDCOM	COM Module Select Register

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note:

It is necessary to push and pop the INSCON register when entering and exiting the interrupt.

10.3. LED Settings

Example of Mode 1:

TB is the LED single COM scan width, TSYS is the system clock width, TLED is the LED scan time width .

TB = TSYS * 256 * DISCOM

TLED = TB * S

S is the number of LED scan COM: scan 4COM is S = 4, scanning 5COM means S = 5, and so on.

Taking the LED frame frequency 200HZ (5ms) that needs to be displayed as an example, when the LED is 5COM and the system clock is RC 24MHz:

TB = TSYS * 256 * DISCOM

= 41.66ns * 256 * 94 (5EH)

= 1002506ns = 1002.506us = 1.002 ms

Set TLED = 5ms

TLED = TB * S = 1.002ms * 5 = 5.010ms

Set the interrupt function to be enabled (ELED = 1), select frame interrupt (LEDFY = 1) (required) or COM interrupt (LEDCY = 1) (Optional). Turn on the LED module and start the LED scan. When the frame / COM waveform scan ends, the corresponding frame interrupt (LEDIF = 1) / COM interrupt (COMIF = 1) interrupt flag will be set. (The unselected levels of SEG and COM are both floating levels, the effective level of COM is low, and the effective level of SEG is high.)

Example of Mode 2:

TB is the LED single COM scan width, and TSYS is the system clock Width, TLED is the LED scan time width

TB = TSYS * 256 * DISCOM

TLED = TB *S

S is the number of COMs scanned by the LED: scan 4COM means S = 4, scan 5COM means S = 5, and so on. Taking the LED frame frequency 200HZ (5ms) that needs to be displayed as an example, when the LED is 5COM and the system clock is RC 24MHz:

TB = TSYS * 256 * DISCOM

= 41.66ns * 256 * 94 (5EH)

= 1002506ns = 1002.506us = 1.002 ms

Set TLED = 5ms

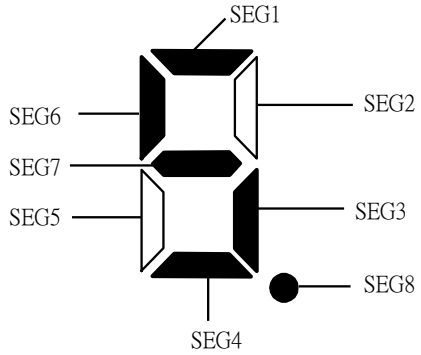
TLED = TB * S = 1.002ms * 5 = 5.010ms

1. SEGX duty (X = 0 ~ 16) of the SEG waveform of the first COM (if the output full scale is set to 0XFF, that is, the first COM cycle displays 100% duty cycle high level, if it is not displayed, set 0X00, that is Display the floating level, if you set 0X7F, SEG output 50% duty cycle high level), set the interrupt function that needs to be turned on (ELED = 1), you need to select the frame COM interrupt (LEDCY = 1) (required), or add Frame interrupt (LEDFY = 1) (optional). Turn on the LED interrupt and LED module.

2. Before turning on the LED function, set the first COM waveform and start the LED function. When the LED interrupt (COMIF = 1) comes, rewrite the SEG waveform SEGXduty (X = 0 ~ 16) of the second COM cycle. If not modified, the last cycle waveform is displayed. And so on, to the end of a frame (LEDIF = 1). Finally, when the second COM interrupt comes, fill in the SEG waveform of the first COM in the next frame.

10.4. Application Examples

10.4.1. Light off mode(mode0) routine




```
#include <SH79F6481A.H>
#include <intrins.h>
#include <absacc.h>
xdata unsigned char COM1 _at_ 0x530;
xdata unsigned char COM2 _at_ 0x531;
xdata unsigned char COM3 _at_ 0x532;
xdata unsigned char COM4 _at_ 0x533;

void LED_init(void)
{
   CLKCON = 0x00;       // System clock is set to 24MHz
   LEDCON = 0x0;        //module1
   DISCOM = 0x3f;       // Set single COM scan time
   LEDDZ = 0x10;        // Set dead time
   SEG01 = 0xFF;        // LED_S1 ~ LED_S8 multiplex on
   LEDCOM = 0x0f;       //LED_C1~LED_C4 multiplex on
}

void main(void)
{
   LED_init();
   COM1 = 0x06;         //show 1
   COM2 = 0x5B;    //show 2
```

```
    COM3 = 0x4F;          //show 3
    COM4 = 0x666;  //show 4
    LEDCON |= 0x80;       //Enable LED driver
    while(1);
}
```

10.4.2. Dimming mode (Mode2) routine

Demand: COM1-2 and SEG1-4 are selected to realize that the output duty cycle of each SEG port at COM1 is 1 in the first cycle, and the output duty cycle at COM2 is 1/2; in the second cycle, each The output duty ratio of each SEG port is 1/2 at COM1 and 1 at COM2; repeat the above process continuously;

```
#include <SH79F6481A.H>
#include <intrins.h>
#include <absacc.h>
xdata unsigned char LED_RAM[4] _at_ 0x530;
unsigned char Display_Data;
void LED_init(void)
{
    CLKCON = 0x00;       // System clock is set to 24MHz
    LEDCON = 0x20;       //Module 2
    DISCOM = 0x3f;       // Set single COM scan time
   LEDDZ = 0x10;         // Set dead time
    SEG01 = 0x0F;        //LED_S1~LED_S4 multiplex on
    LEDCOM = 0x03;       //LED_C1~LED_C2 multiplex on
    IEN1 = 0X02;         //Enable LED interrupt
    ELEDCON = 0X01;    //Enable LED_COM interrupt
}

void LED_Dispaly(unsigned char uc_temp)
{
    unsigned char i;
    for(i=0x00;i<4;i++)
{
        LED_RAM[i] = uc_temp;
}
}
```

```
void main(void)
{
    LED_init();
    Display_Data = 0xff;    // SEG duty cycle displayed by the first COM
    LED_Dispaly(Display_Data);
    EA = 1;                         //Enable total interrupt
    LEDCON |= 0x80;                 //Enable LED drive mode
    while(1);
}

void INT_LED(void) interrupt 8
{
    _push_(INSCON);
    INSCON = 0X00;
    LEDCON &= 0XF7;                 //Clear COM end flag
    Display_Data = Display_Data - 0x40;
    LED_Dispaly(Display_Data);
    _pop_(INSCON);
}
```

11. SH79F6481A TWI User Guide

11.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's TWI features:

- Address programmable
- Wake up system in idle mode
- Level bus timeout judgment (Timeout)
- Allow to send data (Transmitter) and receive data (Receiver)
- Support multi-host communication arbitration function
- With low-level bus timeout judgment (Timeout)
- Idle mode can wake the system
- Address programmable

11.2. Control Register

All control registers used by the TWI module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	TWICON	Control Register
	TWITOUT	Bus Timeout Count Register Status
	TWISTA	Status Register
	TWIFREE	High Timeout Count Register Bit Rate
	TWIBR	Bit Rate Register Address
	TWIADR	Address Register Data
	TWIDAT	Data Register Address Mask
	TWIAMR	Address Mask Register

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note:

Note: It is necessary to push and pop the INSCON register when entering and exiting the interrupt.

11.3. TWI Settings

When the TWI function is turned on, the SCL and SDA are configured as open-drain outputs. The open-drain output structure does not have the ability to output high levels. Therefore, a pull-up resistor (can be connected externally or TWIPCR = 1 to open the internal pull-up) is required to cooperate. At this time, make sure that the level on the IO port does not exceed VDD + 0.3V, otherwise the chip may be damaged.
You can configure TWI to different IO ports through the LCM function.
For the relevant agreement of TWI, please refer to "SH79F6481A DATASHEET"

11.4. Application Examples

11.4.1. Requirement

SH79F6481A is required as the master to transmit data to the slaves, which are "0x40,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0x4A .....".

11.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.h>
#define ERR                    0
#define ACK                    1
#define    NAK                 2
#define RCVCMDREAD             3
#define RCVCMDWRITE            4
#define RCVADDRESS0            5
#define LOSEARBITRATION        6
#define CMD_WRITE              7
#define    CMD_READ            8
#define FAIL                   0
#define OK                     1
void TwiInit()
{
   CLKCON |=0x08;
   Delay();
   CLKCON |=0x04;
   INSCON = 0x40;
   TWICR = 0x45;          // P4.2 as SCL , P4.3 as SDA
   INSCON = 0x00;
```

```
    TWITOUT = 0x02;   //Open the Pull-up resistor of SCL,SDA
    TWIBR=0x02;       // Configure send baud rate, disable bus timeout judgment
                      //f=fsys/(16+2*CR*TWIBR)
    TWISTA=0x0;       //Frequency divider 64
  TWICON = 0x40;      //ENTWI，Disable high level timeout
  TWTFREE = 0xff;     // Maximum timeout configuration
}
bit M_TwiSendStart()
{
  TWICON |=0x20;      // Send start condition when bus is idle
  //TWICON &= 0xdf;
  while(1)   //Wait TWI interrupt
  {
     if(TWICON&0x08)
     {
       if(((TWISTA&0xF8)==0x08)||((TWISTA&0xF8)==0x10))
       {
         return OK;          // Successful send start condition or repeated start condition
       }
       else
       {
         return FAIL;
       }
     }
     if(0x02 == (0x02 & TWICON))
     {
        TWICON &= 0xdf;
        return FAIL;         //Wait interrupt time out
     }
  }
}
UCHAR M_TwiSendCmd(UCHAR addr,UCHAR cmd)
{
    UCHAR SlaveAddr;
    UCHAR i=0;
```

```
if(cmd)
{
    SlaveAddr=((addr&0x7f)<<1)|0x01;          //Read
}
else
{
    SlaveAddr=((addr&0x7f)<<1);               //Write
}
TWICON &= 0xf7;       //Clear interrupt flag bit
TWIDAT=SlaveAddr;
while(1)   //Wait TWI interrupt
{
    i++;
    if(TWICON&0x08)
    {
        if(cmd)
        {
            if((TWISTA&0xF8)==0x40)
            {
                return ACK;          //Successful send SLA+R
            }
            else   if((TWISTA&0xF8)==0x48)
            {
                return NAK;
            }
        }
        else
        {
            if((TWISTA&0xF8)==0x18)
            {
                return ACK;          //Successful send SLA+W
            }
            else   if((TWISTA&0xF8)==0x20)
            {
                return NAK;
```

```
            }
        }
        if((TWISTA&0xF8)==0x38)
        {
            return LOSEARBITRATION;
        }
        else   if((TWISTA&0xF8)==0x68)
        {
            return RCVCMDWRITE;
        }
        else   if((TWISTA&0xF8)==0x78)
        {
            return RCVCMDREAD;
        }
        else   if((TWISTA&0xF8)==0xB0)
        {
            return RCVADDRESS0;
        }
        else
        {
            return ERR;
        }
    }
    if(i>=250)
    {
        return ERR;          //Wait interrupt time out
    }
}

UCHAR M_TwiSendData(UCHAR byte)
{
    UCHAR i=0;
    TWICON &= 0xf7;
    TWIDAT=byte;
    TWICON=0x40;
```

```
        while(1)
        {
            i++;
            if(TWICON&0x08)
            {
                if((TWISTA&0xF8)==0x28)
                {
                   return ACK;
                }
                else if((TWISTA&0xF8)==0x30)
                {
                   return NAK;
                }
                else if((TWISTA&0xF8)==0x38)
                {
                   return LOSEARBITRATION;
                }
                else
                {
                     return ERR;
                }
            }
            if(i>=250)
            {
               return ERR;            // Wait interrupt time out
            }
        }

}
bit M_SendDataToSlave()
{
        UCHAR ret,i=0;
        TwiInit();
        H_Timeout_check; //Enable bus time out, SCL bus high time out judgment
        ret=M_TwiSendStart();
```

```
    if(ret==OK)     //send-STA OK
    {
      if(M_TwiSendCmd(0x2D,0x0)==ACK)
      {
          while(i < 0xff)
          {
              ret=M_TwiSendData(0x40+i);
              i++;
              if(ret==NAK)
              {
                  return 1;           //Send data over
              }
              else if((ret==LOSEARBITRATION)||(ret==ERR))
              {
                  return 0;
              }
          }
      }
    }
    else      //err process
    {
       while(1);
    }
}
void main()
{
    M_SendDataToSlave();
    while(1);
}
```

12. SH79F6481A LPD User Guide

12.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's LPD features:

- Low voltage detection and interrupt generation
- Optional LPD detection voltage
- LPD includes bilateral debounce function, when the detection voltage is higher / lower than the detection point for about 10ms, LPD occurs and generates an interrupt.

12.2. Control Registers

All control registers used by the LED module are shown in the following table:

Category	Abbreviation	Function Description
Module Control	LPDCON	Control Register
	LPDSEL	Voltage detection gear control register

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note: *It is necessary to push and pop the INSCON register when entering and exiting the interrupt.*

12.3. LPD Settings

Low voltage detection is mainly used when the system voltage is lower than a certain value, the system believes that there is a risk of power outage, and needs to do some protective measures to prevent short-term power loss, if the system is powered on again within a certain time, You can continue to run in the state before power off. When using LPD, you need to set the detection mode and an appropriate voltage value first. SH79F6481A provides 2 detection modes and 16 detection voltages, which can be selected by LPDCON and LPDSEL. After setting, the LPD module can be enabled. You can query LPDF or use LPD can be interrupted to quickly detect power failure.

12.4. Application Examples

12.4.1. Demand

The operating voltage of the system is required to be 5V. When the voltage is less than 3.6V, the system considers that there is a risk of power failure. If the Vdd voltage is less than 3.6V, the key parameters are stored in C0H ~ CFH, and then the system enters Power-Down mode.

12.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.h>
unsigned char outflag = 0;
idata unsigned char temp0 _at_ 0xC0;
idata unsigned char temp1 _at_ 0xC1;
idata unsigned char temp2 _at_ 0xC2;
idata unsigned char temp3 _at_ 0xC3;
idata unsigned char temp4 _at_ 0xC4;
idata unsigned char temp5 _at_ 0xC5;
idata unsigned char temp6 _at_ 0xC6;
idata unsigned char temp7 _at_ 0xC7;
idata unsigned char temp8 _at_ 0xC8;
idata unsigned char temp9 _at_ 0xC9;
idata unsigned char temp10 _at_ 0xCA;
idata unsigned char temp11 _at_ 0xCB;
idata unsigned char temp12 _at_ 0xCC;
idata unsigned char temp13 _at_ 0xCD;
idata unsigned char temp14 _at_ 0xCE;
idata unsigned char temp15 _at_ 0xCF;
unsigned char keydata0;
unsigned char keydata1;
unsigned char keydata2;
unsigned char keydata3;
unsigned char keydata4;
unsigned char keydata5;
unsigned char keydata6;
unsigned char keydata7;
unsigned char keydata8;
unsigned char keydata9;
unsigned char keydata10;
unsigned char keydata11;
unsigned char keydata12;
unsigned char keydata13;
```

```
unsigned char keydata14;
unsigned char keydata15;
void main(void)
{
    CLKCON = 0x00;                  //System clock = 24MHz
    LPDCON = 0x80;                  //Enable LPD module
    LPDSEL = 0x08;                  //Set Vlpd =3.5V
    Delay();                //Delay 20us
    LPDCON &= 0xef;         //Clear flag bit
    EA = 1;                 //Enable total interrupt
    IEN1 |= 0x40;           //Enable LPD interrupt
    while(1)
    {
        if(outflag == 1)        //LPD interrupt happened
        {
            temp0 = keydata0;
            temp1 = keydata1;
            temp2 = keydata2;
            temp3 = keydata3;
            temp4 = keydata4;
            temp5 = keydata5;
            temp6 = keydata6;
            temp7 = keydata7;
            temp8 = keydata8;
            temp9 = keydata9;
            temp10 = keydata10;
            temp11 = keydata11;
            temp12 = keydata12;
            temp13 = keydata13;
            temp14 = keydata14;
            temp15 = keydata15;
            LPDCON &= 0x7F;                     //Disable LPD module
            SUSLO = 0x55;
            PCON |= 0x02;
            _nop_();
```

```
                _nop_();
                _nop_();
            }
        }
    }
    void LPD_int(void)    interrupt    13
    {
        _push_(INSCON);
        LPDCON &= 0xEF;
        outflag = 1;
        _pop_(INSCON);
    }
```

13. SH79F6481A PWM User Guide

13.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

SH79F6481A's PWM features:

- 2 channels of 12bit precision PWM module
- Provide overflow interrupt for each PWM period
- Selectable output polarity

SH79F6481A integrates two channels of 12-bit PWM module. The PWM module can generate pulse width modulated waveforms with adjustable period and duty cycle. The PWMxCON (x = 0-1) register is used to control the clock, waveform output and period interrupt of the PWMx module, the PWMxPH / L register is used to control the period of the PWMx output waveform, and the PWMxDH / L (x = 0-1) register is used to control The duty cycle of the output waveform of the PWMx module. These three registers can be modified while the PWM output is enabled, but the modification will take effect in the next PWM cycle.

13.2. Control Register

All control registers used by the PWM module are shown in the following table:

Category	Abbreviation	Function Description
Control Mode	PWM0CON	Control Register
	PWM0PH	Period control register high 4 bits
	PWM0PL	Period control register low 8 bits
	PWM0DL	Duty control register low 8 bits
	PWM0DH	Duty control register high 4 bits

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

Note: *Before using the PWM module, the PWM output pin needs to be mapped to the required IO through the LCM module. Please refer to "SH79F6481A DATASHEET" LCM chapter.*

13.3. PWM0 Settings

PWM programming is divided into the following steps:

(1) Select the clock sources of the PWM module.

(2) Set the PWM period / duty cycle by writing the appropriate value to the PWM period control register (PWMxPH / L) or the PWM duty cycle register (PWMxDH / L). Set the low bit first, and then set the high bit. Note that even if the high-order value is unchanged, it should be rewritten once, otherwise, the low-order modification is invalid.

(3) Select the PWMx output mode (active high or active low) by setting the PWMxS bit in the PWM control register (PWMxCON).

(4) If the PWM period or duty cycle needs to be changed, the operation flow is as explained in step 2 or step 3. Modified reload count.The value of the device will become effective in the next cycle.

13.4. Application Examples

13.4.1. Demand

PWM0 is used to generate a 2KHz PWM wave with a 50% duty cycle to drive the buzzer.

13.4.2. Examples

```
#include <SH79F6481A.H>
#include <intrins.h>
void PWM0_init(void)
{
    CLKCON = 0x00;          //System clock = 24MHz
    INSCON = 0x40;
    PWMCR = 0x00;          // P3.0 as PWM waveform output
    INSCON = 0x00;
    PWM0CON = 0x29;      //The clock is divided by 32 from the system clock, enable
                         /output,high output during duty cycle
    PWM0PL = 0x77;
    PWM0PH = 0x01;          //The frequency of PWM0 is 2KHz
    PWM0DL = 0xBB;
    PWM0DH = 0x00;          //The duty cycle of PWM0 is 50%
    }
void main(void)
{
    PWM0_init();
    PWM0CON |= 0x80;              //Enable PWM0
```

```
        while(1);
}
```

The usage of PWM1 is the same as PWM0.

14. SH79F6481A LCM User Guide

14.1. General Description

The SH79F6481A is a high performance 8051 compatible micro-controller. The SH79F6481A can perform more fast operation speed and higher calculation performance, if compare SH79F6481A with standard 8051 at same clock speed.

- 12 kinds of digital logic function ports can be re-mapped to I / O through digital logic configurable modules, and each function can choose one of multiple IO ports.

SH79F6481A integrates two channels of 12-bit PWM module. The PWM module can generate pulse width modulated waveforms with adjustable period and duty cycle. The PWMxCON (x = 0-1) register is used to control the clock, waveform output and period interrupt of the PWMx module, the PWMxPH / L register is used to control the period of the PWMx output waveform, and the PWMxDH / L (x = 0-1) register is used to control The duty cycle of the output waveform of the PWMx module. These three registers can be modified while the PWM output is enabled, but the modification will take effect in the next PWM cycle.

14.2. Control Register

All control registers used by the LCM module are shown in the following table:

Category	Abbreviation	Function Description
Pin Configure Register	UART0CR	RXD0 and TXD0 configure register
	UART1CR	RXD1 and TXD1 configure register
	TWICR	SCK and SDA configure register
	PWMCR	PWM0 and PWM1 configure register
	CEXCR	P0CEX1 and P0CEX0 configure register
	ECICR	RXD1 and TXD1 configure register

For a detailed description of each register, please refer to "SH79F6481A DATASHEET".

14.3. Application Examples

14.3.1. Demand

Configure pin P0.0 as TXD0, P0.1 as RXD0, P2.6 as TXD1, and P2.7 as RXD1.

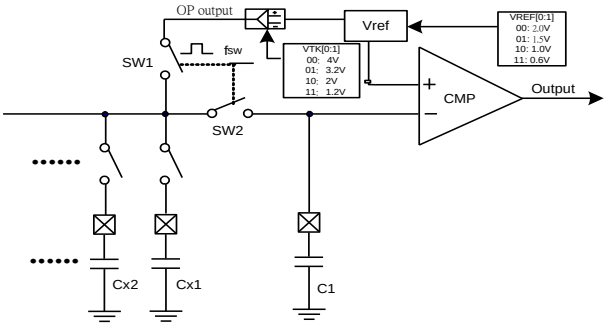
14.3.2. Examples

```
#include <SH79F6481A.H>
#include “intrins.h”
```

```
IO_config()
{
        _push_(INSCON);
        INSCON = 0x40;     //Select Bank1
        UART0CR = 0x01;  // P0.0 asTXD0, P0.1 as RXD0
        UART1CR = 0x01;  //P2.6 as TXD1, P2.7 asRXD1
        _pop_(INSCON);
}
```

15. SH79F6481A Touch Key User Guide

15.1. Touch Key Detection Basics

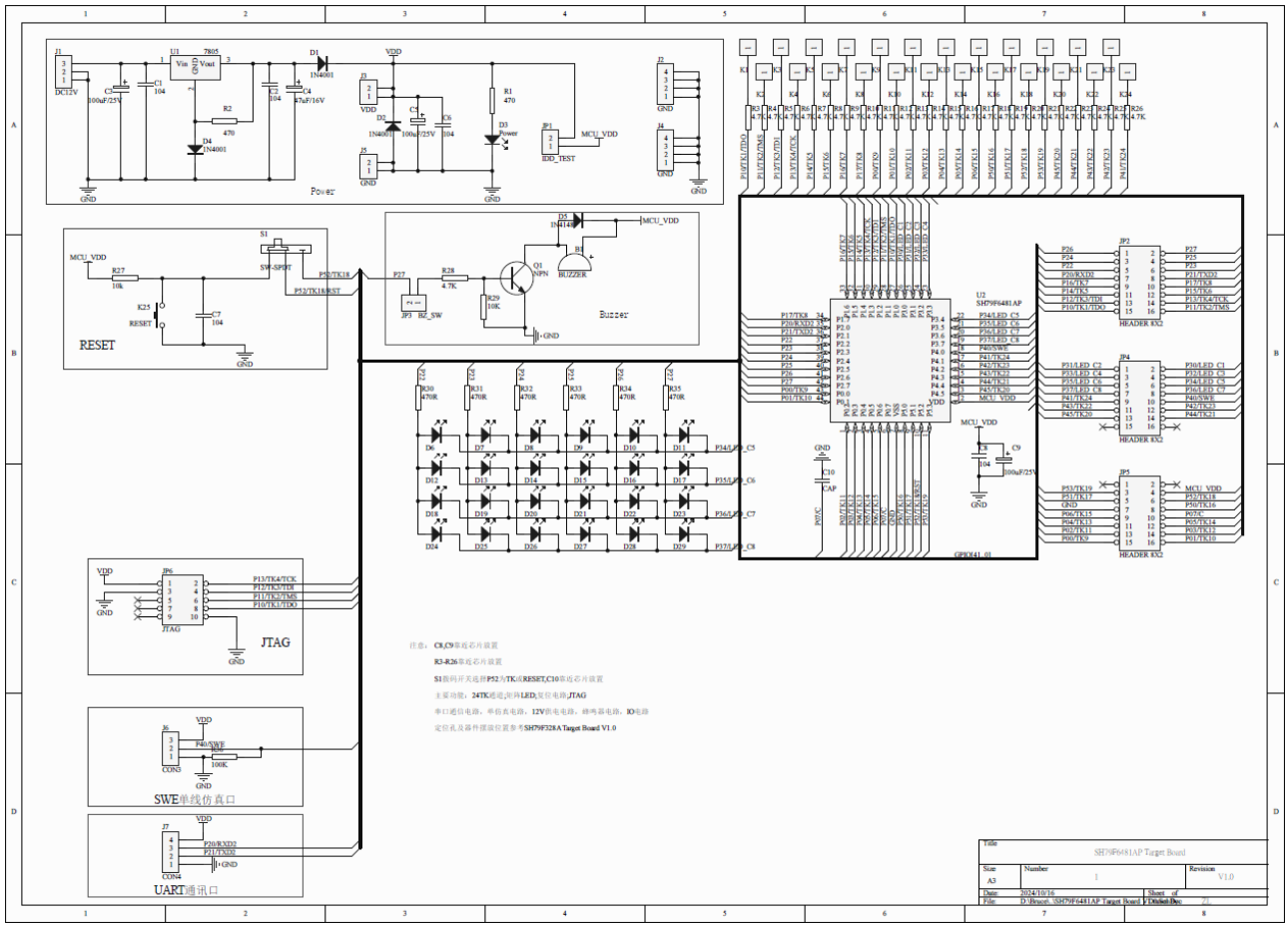


Touch block diagram

15.2. Touch Key startup process

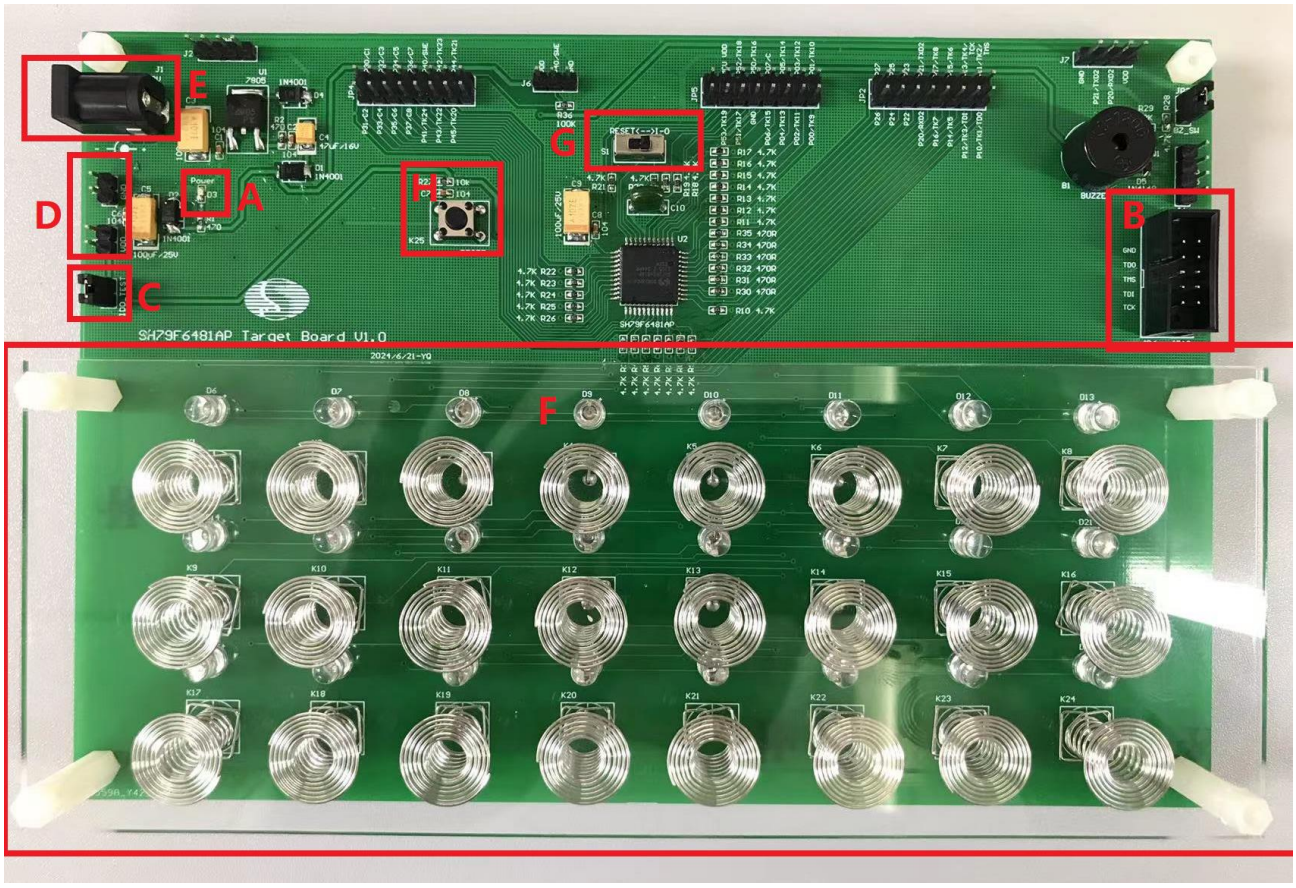
1、 Select key channel which is needed to scan;
2、 Set TKCON bit, enable Touch Key module works;
3、 Set switch frequency, reference voltage VREF, Touch Key sampling times and scanning sequence;
4、 Set the 28-bit amplification factor register
5、 Software delay 10us
6、 Register TKGO / DONE bit 1, start key scan
7、 When an interrupt is generated, the TKGO hardware is automatically cleare to 0;
8、 Judging interrupt flag bit: IFERR, IFGO, IFAVE, IFCOUNT;
   If IFAVE = 1, reading data register 500H – 52FH, program save data result, goto step 9 ;
   If IFERR = 1, data register arithmetic overflow error, clear IFERR flag bit, reset amplification coefficient register, reduce the value of amplification coefficient, goto step5 to restart scan;
   If IFGO = 1, key controller startup errors, clear IFGO flag bit, goto step 5 to restart scan;
   If IFCOUNT = 1, key scan count overflow error, clear IFCOUNT flag bit, reduce C1 capacitor. Goto step 5 to restart scan.
9、 Completing a group key scan.
   Note: *When the INSCON register is interrupted, it is necessary to push and pop the stack.*

Appendix 1：SH79F6481A Target Board V1.0 schematic diagram



Appendix 2：SH79F6481A Demo board（Target Board）

SH79F6481A provide for customers a demo board to familiarize the function of chip. As shown in the following table.



Description of each part functions:

A. Power LED

B. JTAG simulation interface

Communicate with the JET51A driver via a 10-pin cable.

C. JTAG VDD & MCU VDD interface

When downloading the program to select JTAG power supply, the two pins must be connected by jumper, at this time the entire power system is connected.

D. External power supply interface 1（VDD/GND）

The power supply interface of Target Board is connected to the VDD and GND pins of the chip respectively. The external power supply interface 1 is adopted for power supply, and the power supply voltage of the chip can be adjusted.

E. External power supply interface 2

Target Board power supply interface which can provide a stable 5V voltage through the U1 circuit on the board.

F. Touch Key

G. Dial switch（RST PIN）

When P5.2 as RST-PIN, the dial switch dials to the left and the part H on the board constitutes the RST circuit. The RESET will be happening a time when the button in part H is be pressed a time. When P5.2 as IO function, the dial switch dials to the right and disconnects from the external RST circuit.

H. Pin Reset Push Key

User Guide Revision History

Version	content	Data
1.0	Initial version	2024.10

Contents