SUNPLUS

SPCE040A/060A/061A PROGRAMMING GUIDE v1.2

05/05/2003

SUNPLUS TECHNOLOGY CO. reserves the right to change this documentation without prior notice. Information provided by SUNPLUS TECHNOLOGY CO. is believed to be accurate and reliable. However, SUNPLUS TECHNOLOGY CO. makes no warranty for any errors which may appear in this document. Contact SUNPLUS TECHNOLOGY CO. to obtain the latest version of device specifications before placing your order. No responsibility is assumed by SUNPLUS TECHNOLOGY CO. for any infringement of patent or other rights of third parties which may result from its use. In addition, SUNPLUS products are not authorized for use as critical components in life support devices/ systems or aviation devices/systems, where a malfunction or failure of the product may reasonably be expected to result in significant injury to the user, without the express written approval of Sunplus.

SUNPLUS TECHNOLOGY CO., LTD. 19, Innovation First Road, Science-Based Industrial Park, Hsin-Chu, Taiwan, R. O. C.
☎ 886-3-578-6005 ☏ 886-3-578-4418 www.sunplus.com.tw

0 TABLE OF CONTENT

1 Revision History

Revision	Date	By	Remark
V1.2	05/05/2003	Written by & Y. J. Liou Revised by Michael Lin	1. Modify the UART baud rate formula 2. Add SPCE040A 3. Add TimerA/B frequency limitation <200KHz 4. More descriptions on P_DAC_Ctrl($702A) bit1 5. Add watchdog will be turned off during sleep 6. Add 100KOhm on ICESCK pin when download 7. Modify the System Clock section 32KHz mode 8. More descriptions on DAC output characteristics 9. More descriptions on V_{MIC} behavior of ADC
V1.1	12/03/2002	Written by Arthur Shieh Revised by Michael Lin	Add 1. SPCE060A 2. DAC output characteristics (see DAC chapter for more information). 3. IOB13, IOB14 limitation
V1.0	11/01/2002	Written by Arthur Shieh & Y. J. Liou Revised by Michael Lin	First Edition

2 Important Notice Before Writing Your Program

- **Confirmation Sheet**

It is highly recommended programmers to obtain a copy of the appropriate confirmation sheet before writing program. The confirmation sheet, a requisite document before placing orders, contains useful information and checklist to help programmers avoiding mistakes during program development. Due to the possible changes may be made in the confirmation sheet occasionally, an up-to-date confirmation sheet can be downloaded from SUNPLUS web site at http://www.sunplus.com.tw. The following checklist is an abstract of the confirmation sheet.

SPCE040A/060A/061A

Customer		Date	
Project title		Code No.	
Project description			
Body type	□SPCE040A (24K-word) □SPCE060A (31K-word) □SPCE061A (31K-word)	Packaging Type	□Chip-form □Package, No. of pins: _______
Oscillator	□**Crystal (32768Hz)**	**CPU Freq (Hz):** 49.152M, 40.96M, 32.768M, 24.576M, 20.48M. □Check	
CPU working voltage (VDD)	□**2.4V ~ 3.6V**	I/O voltage (VDDIO)	□**2.4~3.6V** □**3.6~5.5V**
Release code file (fill “00H” for unused area)			
Binary filename (*.tsk):		Binary file check sum:	
SRAM: SPCE040A/060A/061A: 2K words ($0000 ~ $07FF) □Check ROM: 1. SPCE040A: Page0 ROM size: 24K words ($08000 ~ $0DFFF) 2. SPCE 060A/061A: □Check Page0 ROM size: 31K words ($08000 ~ $0FBFF) □Check			

Input / Output

IOA Port

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Input																
Output																
Wakeup Key Input																
A/D Line - In Input																

IOB Port

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Input																
Output																

Hardware /Software

A/D function

Mic_in: □Yes □No

Line_in: □Yes □No, if Line_in is used, check the following:

1.Line_In: input (shared with IOA0~6) voltage range (if used): _0 V ~ VDD □Check

2.Line_in: A/D top reference voltage(VEXTREF): □Internal(AVdd) □Built-in 2V regulator □External(_____V)

Serial Interface (SIO): □Yes (IOB0,IOB1) □Not Used

Using UART □Yes □No If yes, baud rate = ____________(minimum: 1500bps @ 24.576MHz, 3000bps @ 49.152MHz)	□RX (IOB7) □TX (IOB10)
LVR (Low Voltage Reset) and LVD (Low Voltage detector) always enable	□ Check
Watchdog enable (Bonding Option) If watchdog is enabled, please check the following: Watchdog port ($7012H) must be cleared by writing 0x0001 within every 0.75 seconds.	□Yes □No Check
Sleep mode (If applied) If sleep mode is applied, please check the following: The 32768Hz is enabled during sleep mode.	□Yes □No □Yes □No
IOB 13 & 14 can only operate in voltage range of 0V ~ VDD (<3.6V).	□Check

The general programming checklist intends to provide some general characteristics about SUNPLUS devices. It is the customer's responsibility to check all the information in the list. No responsibility is assumed by SUNPLUS for any non-checked box even this confirmation sheet has been approved by SUNPLUS. Make sure the following conditions are met and verified:

All used SRAM must be initialized after power on (Strongly recommended).	□Check
Make sure the used SRAM variables are not over stack reserved area.	□Check
Make sure the Interrupt section is located in the page0 or is declared as .TEXT section ($08000-$0FBFF).	□Check
(A)Make sure no current leakage in I/O or speaker amplifier during sleeping. (B) Make sure all I/Os are not floating during sleeping.	□Check □Check

Non-used I/O ports must be masked off (for input process). Example, if IOA[0:7] are input: R1 = [P_IOA_Data]; ; Read I/O port A Data R1 & = 0x00FF; ; Mask higher byte CMP R1,0x0011; ; Lower byte is available for compare operation	□Check

Document version

To make sure the correct version of document is used, please fill out the followings:

(A).SPCE040A/060A/061A Programming Guide Version____

(B). μ'nSP™ IDE User Manual Version____

(C). μ'nSP™ Assembly Tools User's Manual Version____

(D).Other documents (if any)____

Development tool / board version

To make sure the correct version of hardware is used, please fill out the followings:

(A).Emulation Board Version____

(B).Emulation Chip Version____

(C).Piggyback Version(if used) ____

(D).Others (if any)____

To make sure the correct version of software is used, please fill out the following:

(A). μ'nSP™ IDE Version____

(B).SACM Library Version(if used)____

□S200 □S240 □S480/530/720 □A1600 □A2000 □A2000 Encoder □A3200 □MS01 □MS02

(C).Others (if any)____

For Third Party Application

Cyberon voice recognition solution is applied in this code □Yes □No

If yes, please fill out the followings:

(A). Types of solutions: □SI only □SD only □SID(both SI & SD) □SV

(B). VStar Modeling Toolkit IDE Version____

(C). VStar Library Version: BSRV__________.lib

(D). VStar-SDK Programming Guide Version ____

(E). Others (if any)____

Customer note	SUNPLUS note
Signature: ______________________________	**Signature:** ______________________________

- **Other documentations**

To obtain the following documentation, please contact your SUNPLUS sales representatives.

u'nSP® IDE User's Manual: a manual for using u'nSP® IDE (Integrated Development Environment).

u'nSP® Assembly Tools User's Manual: a manual for using *μ'nSP®* assembly tools.

3 INTRODUCTION

3.1 General Description

The SPCE series is equipped with a 16-bit *μ'nSP®*, the newest 16-bit CPU developed by SUNPLUS, pronounced as *micro-n-SP*. Eight registers are involved in *μ'nSP®*: R1 ~ R4 (General-purpose registers), PC (Program Counter), SP (Stack Pointer), Base Pointer (BP) and, SR (Segment Register). The concatenation of R3 and R4 forms a 32-bit register, MR, which is used as the destination register for multiplication and inner-production. Moreover, Two types of interrupts are FIQ (Fast Interrupt Request) and eight IRQs (Interrupt Request) with one software interrupt, BREAK.

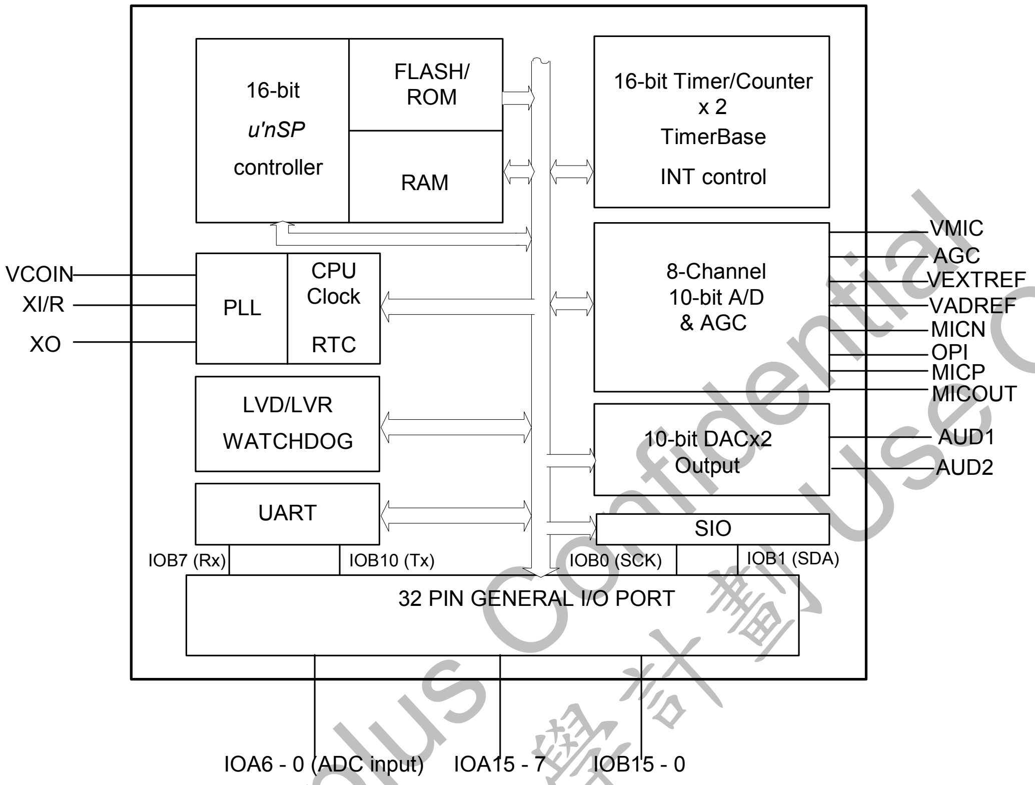
The *μ'nSP®* is a 16-bit microprocessor with 16-bit data and 22-bit address buses. Not only does the *μ'nSP®* perform general operations such as addition, subtraction and other logical operations, but it also supports multiplication and inner-product operations for digital signal processing. For more information about the *μ'nSP®*, please contact SUNPLUS to obtain the latest documents.

The SPCE040A/060A/061A, a 16-bit architecture product, carries the latest 16-bit microprocessor, *μ'nSP®* (pronounced as *micro-n-SP*), developed by SUNPLUS Technology. This high performance of *μ'nSP®* make itself capable of handling complex digital signal processes easily and rapidly. Therefore, the SPCE040/060/061A is applicable to the areas of digital sound process and voice recognition. The operating voltage of 2.4V through 3.6V and speed of 0.32MHz through 49.15MHz yield the SPCE040A/060A/061A to be easily applied in varieties of applications. The memory capacity includes 32K-word flash memory plus a 2K-word working SRAM. Other features include 32 programmable multi-functional I/Os, two 16-bit timers/counters, 32768Hz real time clock, Low Voltage Reset/Detection, eight channels of 10-bit ADC (one channel built-in MIC amplifier with Auto Gain Controller), two channels of 10-bit DAC output and plus many others.

3.2 FEATURES

- Working Voltage: 2.4V ~ 3.6V
- I/O Working Voltage: 2.4V ~ 5.5V
- CPU speed: 49.152MHz (max.)
- 2K-word SRAM with 12.288MHz
- Two 16-bit timers/counters (programmable and auto reload)
- Two 10-bit DACs(D/A output: 3mA or 2mA software option /channel)
- 32 general I/Os (bit programmable)
- 14 INT sources with two priority levels: Timer A / B, timebase, two external inputs, key wakeup
- PLL feature for system clock
- Real time clock (RTC): 32768Hz
- Eight channels of 10-bit ADC
- ADC external top reference voltage
- 2V voltage regulator, 5mA driving capability
- Built-in microphone amplifier and AGC function
- UART(Full duplex)
- Sunplus serial interface
- Low Voltage Reset: 2.2V
- Low Voltage Detection: Three programmable levels
- Built-in watchdog (bonding option)

3.3 Block Diagram



3.4 SPCE Series Feature Table

	SPCE040A	SPCE060A	SPCE061A
Working Voltage	2.4V~3.6V		3.0V~3.6V
I/O Working Voltage	2.4V~5.5V		3.0V~5.5V
Max. Speed	49.152MHz		49.152MHz
CPU	16-bit *μ'nSP*®		16-bit *μ'nSP*®
SRAM Size	2K Words		2K Words
Memory Size(Word)	24K Words ROM	32K Words ROM	32K Words Flash memory
PORTA	IOA15~0 (IOA6~0 shared with ADC input)		IOA15~0 (IOA6~0 shared with ADC input)
PORTB	IOB15~0		IOB15~0
Audio Type	DAC x 2		DAC x 2
Audio Output	Speaker x 2		Speaker x 2
Interrupt Source	TimerA/B TimeBase Ext INT Key-Wake-up		TimerA/B TimeBase Ext INT Key-Wake-up
Wakeup Source	IOA7~0 Interrupt		IOA7~0 Interrupt
Timer/Counter	Two 16-bits up count Timer/Counter		Two 16-bits up count Timer/Counter
UART	Yes		Yes
10-bit ADC	Yes		Yes
ADC Channel	One channel MIC and seven channels Line-in from IOA6~0		One channel MIC and seven channels Line-in from IOA6~0
ADC Top Reference Voltage	Yes		Yes
2V Voltage Regulator	Yes		Yes
IR (IRTx)	Yes		Yes
Serial interface	YES		YES
Crystal/ROSC Resonator	Crystal		Crystal
Low Voltage Reset	Yes		Yes
Low Voltage Detect	Yes		Yes
Watchdog	Yes (Bonding Option)		Yes (Bonding Option)

3.5 Difference between SPCE040A/060A/061A and SPCE Series

	SPCE040A/060A/061A	SPCE120A/250A/380A/500A
CPU type[1]	μ'nSP® 1.1	μ'nSP® 1.0
Working voltage	2.4V~3.6V	2.6V~5.5V
I/O Working voltage	2.4V~5.5V	2.6V~5.5V
CPU Max. speed	49.152MHz	24.576MHz
SRAM size (byte)	4K bytes	4K bytes
Memory Size (byte)	48K/64K bytes ROM/Flash (Bank0)	192K ~ 576K bytes ROM (Bank0~4)
RAM/ROM access cycle	RAM: 2 cycles Bank0 ROM: 2 cycles	RAM: 2/3 cycles Bank0 ROM: 3 cycles
Interrupt mask readable	Yes (P_INT_Mask(R/W)($702DH))	No (use RAM variable R_InterruptStatus)
10-bit ADC	Yes (8 channels)	Yes (1 channel)
ADC Channel	One channel of MIC and seven channels of Line-in from IOA0~6 with built-in MUX	One channel of MIC or Line-in
ADC input range	0~AV_{dd} (full range)	1/2 AV_{dd}
Serial interface	Yes	No
Crystal/ROSC resonator	Crystal	Crystal / Rosc
Low Voltage Reset	Yes	Yes (mask option)
Low Voltage Detect	Yes	Yes (mask option)
Watchdog	Yes (bonding option)	Yes (mask option)

[1] For the difference between μ'nSP® 1.1 and μ'nSP® 1.0, please refer to ***SUNPLUS μ'nSP® TOOL SET USER MANUAL.***

3.6 Guidance for SPCE 040A/060A/061A Programming Guide- a lookup Table

	Sections to refer to
SRAM/ROM	Memory Mapping
Max. CPU Speed	System Clock
PORTA, B	I/O
Timer/Counter	I/O System Clock Timer/Counter
Interrupt Source	I/O Timer/Counter Interrupt UART
Sleep / Wakeup control	System Clock I/O Interrupt Sleep and Wake-up
Real Time Clock	Timer/Counter
Analog to Digital Converter	I/O ADC Audio Output Timer/Counter
Audio Type	Audio Output
Audio Output	Audio Output Timer/Counter
UART	I/O UART Interrupt
IR (IRTx)	I/O IR UART
Serial interface	I/O Serial Interface I/O
Ext OSC	P_Feedback
Low Voltage Reset	LVR/LVD
Low Voltage Detect	LVR/LVD
Watchdog	Watchdog

4 MEMORY MAPPING

4.1 SRAM

The 2K-word SRAM (including Stack) area is located in $000000 ~ $0007FF with access time of two CPU clock cycles. Please refer to the ***System Clock*** for the CPU clock setting. The maximum CPU clock frequency of SPCE040/060/061 is 49.152MHz.

4.2 ROM

SPCE040A

SPCE040 equips 24K words ROM memory.

Note: The interrupt vector area is located at $0x00FFF5 ~ 0x00FFFF.

SPCE060A

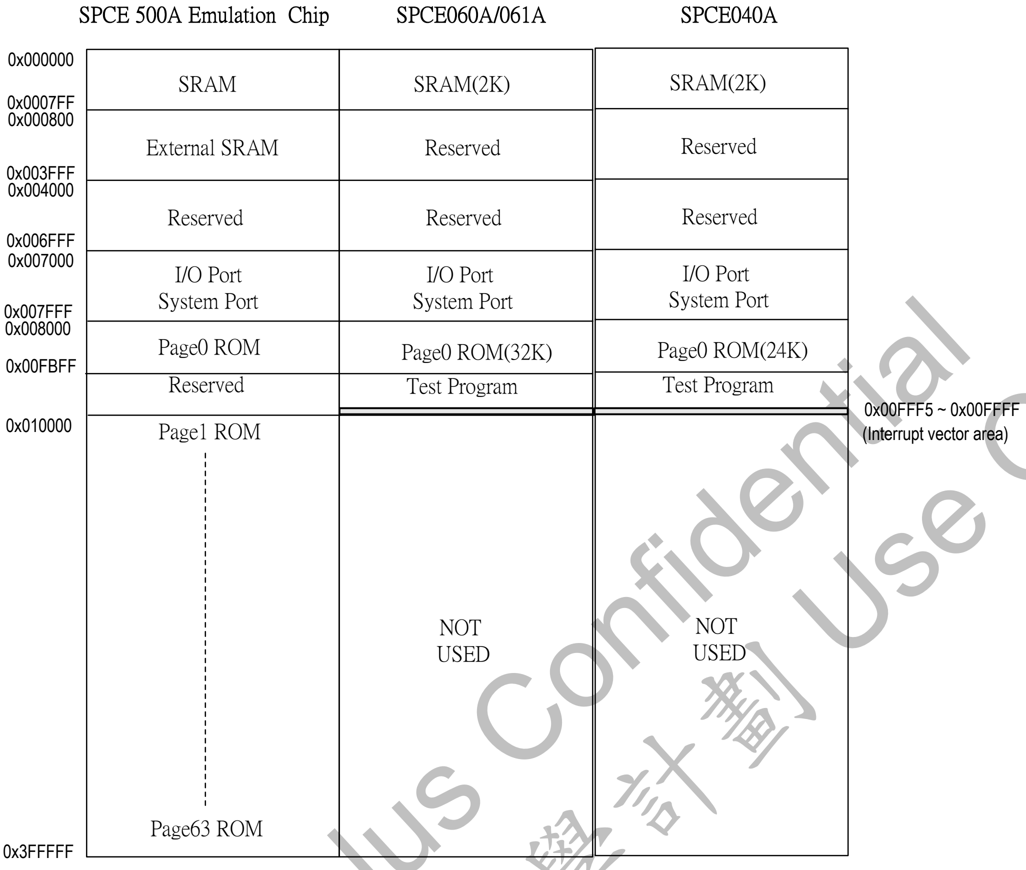
SPCE060 equips 32K words ROM memory.

Note: The interrupt vector area is located at $0x00FFF5 ~ 0x00FFFF.

SPCE061A

SPCE061 is a multi-time-programmable (MTP) microprocessor. It equips a 32K words programmable flash memory. The access time of the 32K-word flash memory ($008000 ~ $00FFFF) is two CPU clock cycles.

Note: The interrupt vector area is located at $0x00FFF5 ~ 0x00FFFF.



	SPCE 500A Emulation Chip	SPCE060A/061A	SPCE040A
0x000000 0x0007FF	SRAM	SRAM(2K)	SRAM(2K)
0x000800 0x003FFF	External SRAM	Reserved	Reserved
0x004000 0x006FFF	Reserved	Reserved	Reserved
0x007000 0x007FFF	I/O Port System Port	I/O Port System Port	I/O Port System Port
0x008000	Page0 ROM	Page0 ROM(32K)	Page0 ROM(24K)
0x00FBFF	Reserved	Test Program	Test Program
0x010000 0x3FFFFF	Page1 ROM Page63 ROM	NOT USED	NOT USED

0x00FFF5 ~ 0x00FFFF
(Interrupt vector area)

4.3 Memory Allocation

The memory allocation is determined at compile time. Sunplus *μ'nSP®* IDE will automatically allocate the RAM and ROM space when the project is compiled or built based on the predefined sections such as CODE, NB_DATA, DATA, TEXT, RAM, SRAM, IRAM, ISRAM, ORAM, and OSRAM. For details about the predefined sections, please refer to Sunplus *μ'nSP®* Assembly Tools Manual and *μ'nSP®* IDE on-line help.

4.3.1 RAM (0x000000~0x0007FF)

The RAM allocation follows the following priority to allocate the RAM (0x0000~0x07FF). It will allot the user-predefined RAM sections and next declare the RAM block, from the biggest to the smallest. That is, except the designated allocation, the rest of the space will be served by biggest-first rule. The first 64 words in RAM area (0x000000~0x00003F) can be used for RAM block allocation only when the RAM block is explicitly declared as SRAM. If not, the RAM block will be allocated in the range from 0x000040 to 0x0007FF. The stack will stack up from 0x0007FF and move up the stack pointer toward the address 0x000000. The stack size varies at run time as the program thread transits among foreground functions and background interrupts. Consequently, attention must be exercised to avoid the situation that, at run time, the stack size growth conflicts with the allocated RAM blocks.

4.3.2 ROM (0x008000~0x0FFFF)

The code allocation will start from the code segment, which is explicitly declared as TEXT and compiler will place those segments in page 0 (0x008000~0x00FBFF). The rest of the code declared other than TEXT will be automatically allotted to the unused space in page 0 and it also proceeds by the rule, the biggest first. Since there is only one page in SPCE040A/060A/061A, there is no the cross-banks issues.

4.3.3 Engineering issues

Users can opt to manually designate the allocation to the unoccupied location in *μ'nSP®* IDE. To manually allocate the RAM block, please refer to Sunplus *μ'nSP®* Assembly Tools Manual. The memory allocation examination by user is recommended after completing the program compilation. User can go to the debug or release folder under the project folder and click open the file, project_name.map, for memory allocation details or go to the *μ'nSP®* IDE menu->Tools-> Memory Map for graphic presentation.

5 I/O

The purpose of input and output ports is to communicate with other devices. Two programmable I/O ports are available in SPCE series, Port A and Port B. The Port A is an ordinary I/O with programmable wakeup capability and IOA0-6 can be shared as ADC input. In addition to regular I/O function, Port B also provides some special functions in certain pins. Please refer to ***Special Function in Port B*** for details.

5.1 I/O Configuration

The SPCE provides a bit-to-bit I/O configuration; every I/O bit can be defined individually. To setup a bit's configuration, three essential setups should be given: Data, Attribution, and Direction. The following table is a summary of setting up an I/O configuration. The Direction, Attribution and Data represent three ports. Each corresponding bit in these ports should be given a value to set one bit's configuration. The setting rules are as follows:

- The direction setting determines whether this pin is an input or output.
- The attribute setting gives a feature to the pin. Float / pull for input, not inverted/ inverted for output.
- The data setting affects the initial content of the pin. For inputs, it also decides the pull high or pull low setting.

Please refer to the table below for the functional configuration of I/O pins.

Please note that the wakeup function is only available when the IOA[0:7] is configured as input pull high, low or input with float. User can develop an low-power-consumed application by taking the advantage of wake-up function. Please refer to ***SLEEP AND WAKEUP*** for details.

If users intend to use IOA0~IOA6 as line-in input for ADC, be sure to configure the corresponding IOA bit as input with float. Also note that the ADC range is 0~AV$_{dd}$ and therefore, users must be careful with the signal range when inputting signal to the IOAn(n=0~6) and ADC line-in. If the input analog signal exceeds the valid operating range, the ADC performance may be affected. In case that user needs to use I/O pin for general control, if user has options, user is also encouraged to use different I/O pins other than IOA0~IOA6 for the purpose of general control. It eliminates the possibility of unexpected induction of power jittering from the invalid I/O signal, which may possibly deteriorate the ADC performance. Please refer to ***ADC*** for details.

To activate the special functions provided in IOB, correspondent setting also needs to be done by users besides the I/O setting described in this section. Please refer to the section ***Special Functions for port***

B for detail.

For example, suppose Port A.0 is used as input with pull low. The bit0 in Port A's Direction, Attribution and Data control ports should be given all 0s. If Port A.1 is used as input with float and wakeup function, the bit1 in Port A's Direction, Attribution and Data ports should be given "010". Note that the Port A and Port B own different Direction, Attribution and Data ports. Users should pay extra attention while configuring I/O.

Direction	Attribution	Data	Function	Wakeup	Description
0	0	0	Pull Low*	Yes**	Input with pull low
0	0	1	Pull High	Yes**	Input with pull high
0	1	0	Float	Yes**	Input with float
0	1	1	Float	No	Input with float***
1	0	0	Output High (inverted)	No	Output with data inverted (write "0" to the Data Port and will output "1" to the I/O pad)
1	0	1	Output Low (inverted)	No	Output with data inverted (write "1" to the Data Port and will output "0" to the I/O pad)
1	1	0	Output Low	No	Output with buffer (data not inverted)
1	1	1	Output High	No	Output with buffer (data not inverted)

*Default: Pull Low

**Only IOA [7:0] in the state of 000, 001, and 010 has wake up capability.

*** IOA[6:0] as ADC line-in input in input with float modes.

5.2 P_IOA_Data(R/W)($7000H)

Write data into the data register and read data from the I/O pad. Write data into $7000H will be the same as writing into $7001H.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

5.3 P_IOA_Buffer (R/W) ($7001H)

Reading means to read data from data buffer. Write data into $7001H will be the same as writing into $7000H.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Note: The reading of P_IOA_Data (R)($7000H) and P_IOA_Buffer (R)($7001H) is through different physical path.

Reading P_IOA_Data (R)($7000H) is reading data from I/O pad. Reading P_IOA_Buffer (R)($7001H) is reading data from I/O buffer. Please refer to the I/O block diagram.

5.4 P_IOA_Dir(R/W)($7002H)

Read/Write direction-vector from/into the Direction Register.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

5.5 P_IOA_Attrib(R/W)($7003H)

Read/Write attribute vector from/into the Attribute Register.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

5.6 P_IOA_Latch(R)($7004H)

Read this port to latch data on the I/O PortA for key change wakeup before getting into sleep mode (Please refer to ***Sleep/Wakeup*** section for details).

Example 1

Set IOA[3:0] as Pull Low, IOA[7:4] as Pull High, IOA[11:8] as Output Low, and IOA[15:12] as Output High.

```
R1 = 0xF0F0;
[P_IOA_Data] = R1;
R1 = 0xFF00;
[P_IOA_Attrib] = R1;
R1 = 0xFF00;
[P_IOA_Dir] = R1;
```

Although the data is written into the same register by writing P_IOA_Data(W) and P_IOA_Buffer(W), it is read from different locations, P_IOA_Buffer(R) and P_IOA_Data(R). In some I/O applications, it is able to save CPU RAM that is normally used to store the Port content. The I/O structure is able to change attribute easily. For example, the Float (011) can be changed to Output High (111) by only modifying the Direction bit from “0” to “1”. In addition, IOA[7:0] are key change wake-up sources. To activate the key change wake-up function, the P_IOA_Latch(R)($7004H) must be read to latch the data of PortA and key change wake-up function must be enabled before entering into standby mode. Wake-up is triggered when the I/O state of PortA is different from at the time latched.

Example 2

Two ways to Read data from IOA. Reading from P_IOA_Data is to read the signal existing on the I/O pin; however reading from P_IOA_Buffer will get the data on the buffer registers.

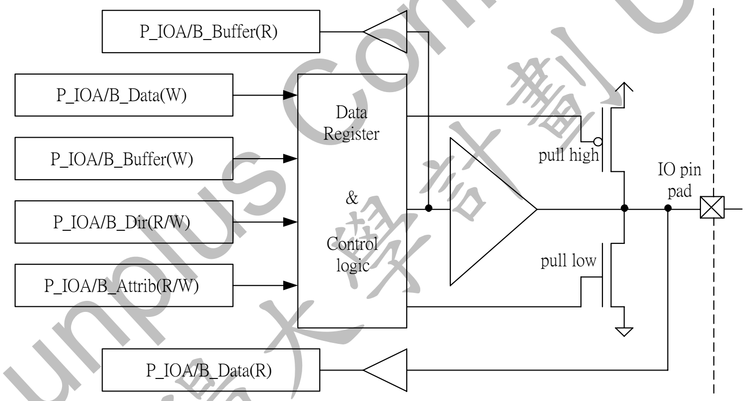
```
R1 = [P_IOA_Data] ;   // Read data from IOA pad
R1 = [P_IOA_Buffer];  // Read data from IOA buffer
```

Example 3

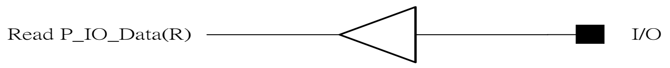
Two ways to write data to IOA. Writing on P_IOA_Data is the same as writing on P_IOA_Buffer.

```
R1 = 0x0000;
[P_IOA_Data] = R1 ;   // Write data to IOA through P_IOA_Data
[P_IOA_Buffer] = R1;  // Write data to IOA through P_IOA_Buffer
```

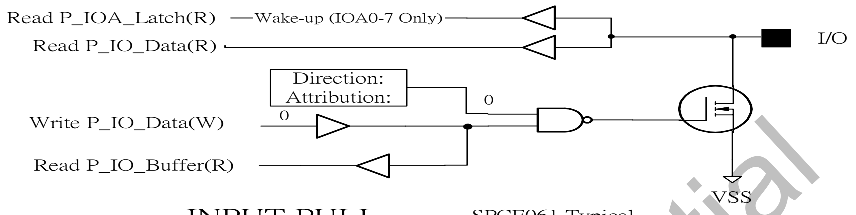
I/O block diagram



PortA, PortB

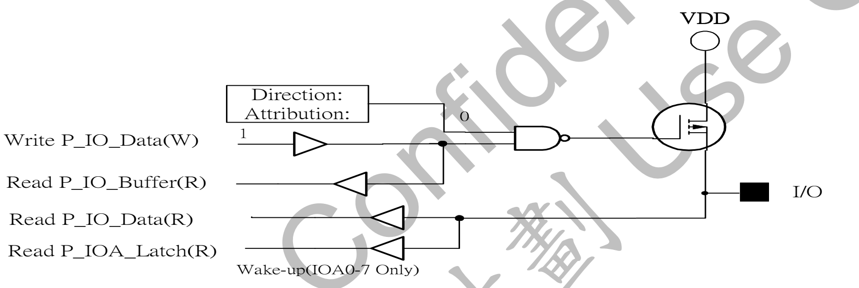


Float (Pure) INPUT



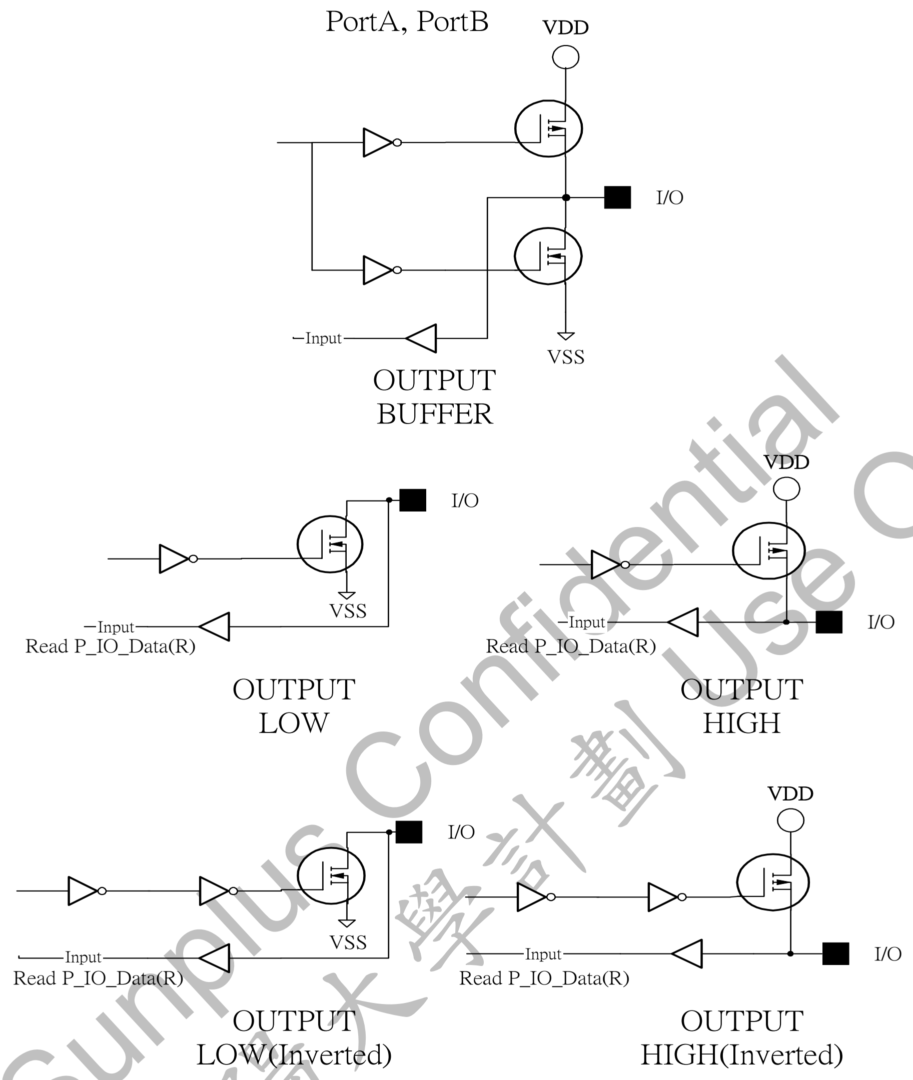
INPUT PULL LOW

SPCE061 Typical
233KOhms @ DC2.7V
180KOhms @ DC3.3V



INPUT PULL HIGH

SPCE061 Typical
325KOhms @ DC2.7V
242KOhms @ DC3.3V



5.7 P_IOB_Data(R/W)($7005H)

Write data into data register and read from I/O pad. Write data into $7005H will be the same as writing into $7006H.

Note:

1. **Due to physical architecture. IOB13 and IOB14 are advised to operate in the range from 0 to VDD. If user set VDDIO higher than VDD, IOB13 and IOB14 must operate below VDD.**

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

5.8 P_IOB_Buffer(R/W)($7006H)

Write data into the data register and read data from the I/O buffer . Writing data into $70006H will be the same as writing into $7005H.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Note: The reading of P_IOB_Data (R)($7005H) and P_IOB_Buffer (R)($7006H) is through different physical path. Reading P_IOB_Data (R)($7005H) is reading data from I/O pad. Reading P_IOB_Buffer (R)($7006H) is reading data from I/O buffer.

5.9 P_IOB_Dir(R/W)($7007H)

Read/Write direction vectors from/into the direction register.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

5.10 P_IOB_Attrib(R/W)($7008H)

Read/Write attribute vector from/into the Attribute Register.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Example

Set IOB[3:0] as Pull Low, IOB[7:4] as Pull High, IOB[11:8] as Output Low, and IOB[15:12] as Output High

```
R1 = 0xF0F0;
[P_IOB_Data] = R1;
R1 = 0xFF00;
[P_IOB_Attrib] = R1;
R1 = 0xFF00;
[P_IOB_Dir] = R1;
```

5.11 Special function for PortB

Besides serving as an ordinary I/O, PortB also provides some special functions in IOB0 ~ IOB10. In order to execute these functions properly, proper setup should be given. A description is depicted as follows:

PortB	Special Function	Function description	Note
IOB0	SCK	Serial interface clock	Refer to see SIO section
IOB1	SDA	Serial interface data	Refer to see SIO section
IOB2	EXT1	External interrupt source 1 (negative edge triggered)	Set IOB2 as input mode
	Feedback Output1	Work with IOB4 by adding a RC circuit between them to get an OSC to EXT1 interrupt	Set IOB2 as inverted output (See the block diagram below and refer to P_FeedBack(W)($7009H))
IOB3	EXT2	External interrupt source 2 (negative edge triggered)	Set IOB3 as input mode
	Feedback Output2	Work with IOB5 by adding a RC circuit between them to get an OSC to EXT2 interrupt.	Set IOB3 as inverted output (See the block diagram below and refer to P_FeedBack(W)($7009H))
IOB4	Feedback Input1		(See the block diagram below)
IOB5	Feedback Input2		(See the block diagram below)
IOB7	Rx	UART Receiver	Refer to UART section
*IOB8	APWMO	TimerA PWM output	Refer to Timer/Counter section, P_Feedback(W) ($7009H) section, and IR section
	IRTx	IR Transmitter	
IOB9	BPWMO	TimerB PWM output	Refer to Timer/Counter section
*IOB10	Tx	UART Transmitter	Refer to UART section

Default: Pull Low

PWM: Pulse Width Modulation

*For more information on the special function of IOB8 and IOB10, refer to the P_FeedBack(W)($7009H) for details.

5.12 P_FeedBack(W)($7009H)

This port controls the feedback function between IOB2 (IOB3) and IOB4 (IOB5).

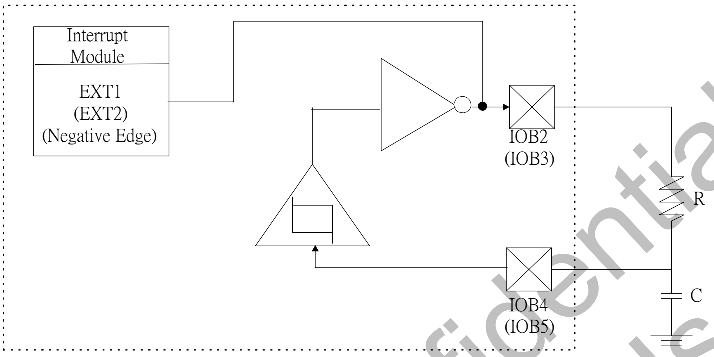
b15 – b4	b3	b2	b1	b0
---	FBKEN3	FBKEN2	---	IRTxEN

FBKEN3 b3 1 Enable the Feedback2 function of IOB3 and IOB5
0 IOB3 and IOB5 are normal I/Os (Default)

FBKEN2 b2 1 Enable the Feedback1 function of IOB2 and IOB4
0 IOB2 and IOB4 are normal I/Os (Default).

IRTxEN b0 0 IRTx disable (default)
1 Refer to Configuration of TAON, TxPinEN and IRTxEN in IOB8, IOB10 and the example.

The following circuit shows the configuration of IOB2, IOB3, IOB4, and IOB5 with feedback application. With the feedback function, an OSC frequency can be obtained to EXT1 (EXT2) by simply adding a RC circuit between IOB2 (IOB3) and IOB4 (IOB5). To make this feedback function work properly, the IOB2 (IOB3) must be configured as "output inverted" and IOB4 (IOB5) as float input mode.



Feedback function of IOB2, IOB3, IOB4 and IOB5
when FBKEN2(FBKEN3) of P_FeedBack(W)($7009H) is set to "1"

Example:

This program can obtain an OSC frequency (square wave) from IOB2 by connecting an external RC circuit between IOB2 and IOB4. This external clock is able to enable IRQ3_EXT1 with the external interrupt source EXT1 or can be used as an external clock input of a timer.

```
// Set IOB4 as floating input, IOB2 as inverted output
     R1=0x0004;
     [P_IOB_Dir]=R1;
     R1=0x0010;
     [P_IOB_Attrib]=R1;
// Write P_FeedBack Port
    R1=0x0004;
    [P_FeedBack]=R1;
```

Configuration of TAON, TXPinEn and IRTXEN in IOB8, IOB10

The combination of TAON, TXPinEn and IRTXEN determine the I/O mode for IOB8 and IOB10.

TAON	TXPinEn	IRTXEN	IOB8	IOB10
0	0	0	I/O	I/O
0	0	1	I/O	I/O
0	1	0	I/O	Tx
0	1	1	I/O	I/O

TAON	TXPinEn	IRTXEN	IOB8	IOB10
1	0	0	APWMO	I/O
1	0	1	APWMO	I/O
1	1	0	APWMO	Tx
1	1	1	IRTx	I/O

TAON: The enable flag of the PWM output of TimerA ***(Refer to the Timer/Counter section for details).***

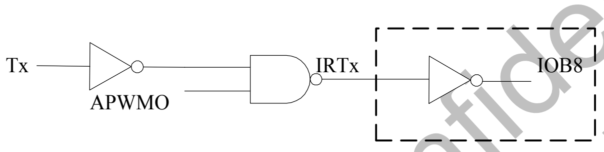
TXPinEn: UART Transmission mode enable ***(Refer to the UART section for details)***.

I/O: Normal I/O mode

APWMO: TimerA PWM output ***(Refer to the Timer/Counter section for details).***

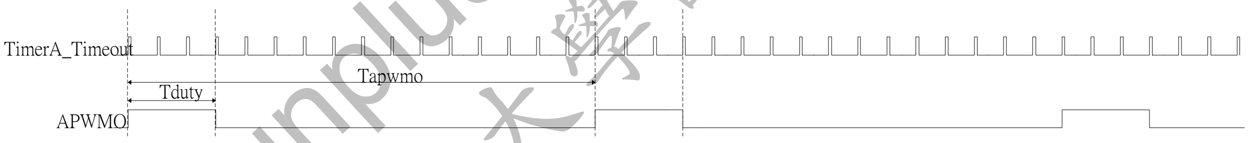
Tx: UART transmitter output ***(Refer to the UART section for details).***

IRTx: IR Transmitter output, which equals to $\overline{\mathrm{APWMO\,AND}\ \overline{\mathrm{Tx}}}$



Example:

To acquire the following waveform:



```
//Set IOB8 as output inverted

R1=0x0100;

[P_IOB_Dir]=R1;

R1=0x0000;

[P_IOB_Attrib] = R1;

[P_IOB_Data]=R1;

//Set TimerA, Tapwmo=(12.288MHz / 512) / 16 = 1.5KHz, Tduty= (3/16)*Tapwmo (Refer

// to the Timer/Counter Section)

R1=0xFDFF;

[P_TimerA_Data]=R1;

R1=0x00F0;

[P_TimerA_Ctrl]=R1;
```

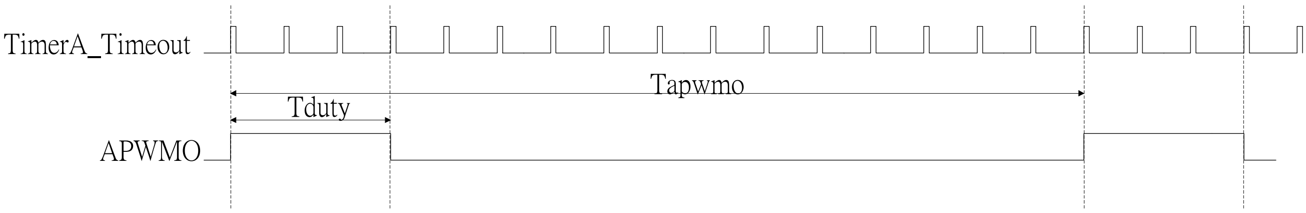
6 TIMER/COUNTER

SPCE provides two 16-bit timers (counters), Timer A and Timer B. The clock source of Timer A is from clock source A and clock source B which are ANDED to form varieties of combinations. In Timer B, the clock source is from clock source C. When timer overflows, a timeout signal (TAOUT/TBOUT) is sent to CPU interrupt module to generate a timer interrupt signal. In addition, Timer A and Timer B hardware interrupt events can be used to trigger the latch action of the ADC (auto mode) and the DAC audio output. (Please refer to ***AUDIO OUTPUT*** section).

Writing a N value into P_TimerA_Data(R/W)($700AH) (or P_TimerB_Data(R/W)($700CH)) and selecting an appropriated clock sources, timer will up-count from N, N+1, N+2,... 0xFFFE to 0xFFFF. An INT signal is generated at the moment of timer counting from “0xFFFF” to “0x0000” and the INT signal is processed by INT controller immediately. At the same time, N will be reloaded into timer and the timer starts counting again. (Please refer to ***INTERRUPT*** section). Note that the timeout signal(TAOUT/TBOUT) frequency can not exceed 200KHz.

In Timer A, the clock source A is a high frequency source and clock source B is a low frequency clock source. The combination of clock source A and B provides a variety of speeds to Timer A/Counter A. In Timer A block diagram, for the clock source selectors, the “1” represents pass signal, not gating and the “0” indicates deactivating the clock source. Clock source A selector has a “1” and a “0” inputs. Clock source B selector has a “1” input. For instance, if clock source A=”1” (that is, P_TimerA_Ctrl [b2~b0]=”101”), the clock output is depending on source B. If clock source A=”0” (that is, P_TimerA_Ctrl [b2~b0]=”110”), the Timer A is deactivated. The EXT1 and EXT2 are the external clock sources. In addition, counter can generate time-out signal for input clock source to a four-bit (16 levels) PWM pulse width counter. A variety of PWM clock duration can be generated and output from IOB8 (APWMO) and IOB9 (BPWMO).

The following example is a 3/16 duration cycle PWM output. The APWMO waveform is made by selecting a pulse width through P_TimerA_Ctrl(W)($700BH) [b9 ~ b6] (BPWMO by P_TimerB_Ctrl(W)($700DH)[b9 ~ b6]). As a result, each 16 cyclesof TimerA_Timerout will generate a pulse width, Tduty, as we defined in control port, P_TimerA_Ctrl(W)($700BH). These PWM signals are able to control the speed of motor or other devices.



The source A and source C are faster clock sources which come from Fosc (up to 49.152MHz) in PLL and source B comes from RTC system (32768Hz) which means the source B can be used as a precise counter for time tracking. For example, the 2Hz can be selected for real time counter.

6.1 P_TimerA_Data(R/W)($700AH)

Timer A Data Port. Write value into the 16-bit pre-load register or read data from the 16-bit counter value.

If users intend to program Timer A to a specific frequency, they can get the value to write from the following simple formula. Note that the desired timeout signal TAOUT frequency can not exceed 200KHz.

Data to Write = 0xFFFF – (Clock Source Frequency / Desired Frequency)

"Clock source frequency" is determined by the setting of P_TimerA_Ctrl.

"Desired Frequency" is the specific frequency as user's expectation.

User can then write the value coming from the formula to the P_TimerA_Data and turn on the specific interrupt for application.

For example, under the CPU clock rate 49.152MHz, if user needs a 8KHz TimerA interrupt , the data to be written will be 0xF3FF with the Fosc/2 clock source. Under the CPU clock rate 24.576MHz, if user needs a 8K Hz TimerA interrupt , the data will be 0xF9FF with the Fosc/2 clock source.

	b15	b14	b13	b12	B11	B10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	Timer A Data															

6.2 P_TimerA_Ctrl(W)($700BH)

Timer A Control Port. User can select the clock source of Timer A by setting b0 ~ b5. The b2~b0 determine the output of clock source selector A. The b5~b3 decide the output of clock source selector B. The outputs then will form various combination by a "AND" gate.

Also, various PWM pulses can be produced via Timer A counter by programming b6 ~ b9 of this port. User can program to output this PWM pulse to IOB8 (Please refer to ***P_FeedBack*** for details)

b15 - b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	Output_pulse_ctrl				Source B select bits			Source A select bits		

Output_pulse_ctrl:

b9	**b8**	**b7**	**b6**	**Output pulse duration (APWMO)**	**TAON**[a]
0	0	0	0	OFF	0
0	0	0	1	1/16	1
0	0	1	0	2/16	1
0	0	1	1	3/16	1
0	1	0	0	4/16	1
0	1	0	1	5/16	1
0	1	1	0	6/16	1
0	1	1	1	7/16	1
1	0	0	0	8/16	1
1	0	0	1	9/16	1
1	0	1	0	10/16	1
1	0	1	1	11/16	1
1	1	0	0	12/16	1
1	1	0	1	13/16	1
1	1	1	0	14/16	1
1	1	1	1	Toggle signal of TAOUT[b]	1

[a] TAON is the enable flag of the PWM output of TimerA(APWMO), and the default value is “0”. “1” will be given when b6 ~ b9 are not all “0”.

[b] TAOUT is the overflow signal of TimerA. When the counter in TimerA counts from “N” (user-defined by programming P_TimerA_Data (W)($700AH)) to “0xFFFF”, at the next count, overflow occurs and reload immediately N to pre-load register. At this moment, it will send an overflow signal, TAOUT, to the interrupt control module to trigger the TimerA interrupt source. The duty cycle of the toggle signal of TAOUT is 50%, and the frequency is $F_{TAOUT}/2$ while the frequency of other output pulse is $F_{TAOUT}/16$.

Source A select bits:

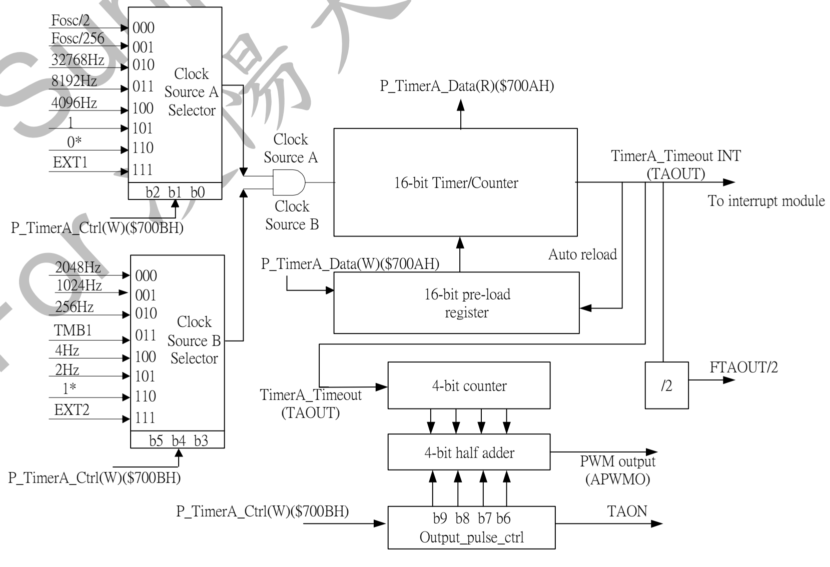
b2	b1	b0	Clock Source A
0	0	0	Fosc/2
0	0	1	Fosc/256
0	1	0	32768Hz
0	1	1	8192Hz
1	0	0	4096Hz
1	0	1	1
1	1	0	0*
1	1	1	EXT1

*Default: Clock Source A = 0

Source B select bits:

b5	b4	b3	Clock Source B
0	0	0	2048Hz
0	0	1	1024Hz
0	1	0	256Hz
0	1	1	TMB1
1	0	0	4Hz
1	0	1	2Hz
1	1	0	1*
1	1	1	EXT2

*Default: Clock Source B=1



Timer A Block Diagram

6.3 P_TimerB_Data(R/W)($700CH)

Timer B Data Port. Write value into the 16-bit pre-load register or read data from the 16-bit counter value.

If user wants to program Timer B to a specific frequency, it will be much similar to the setting of Timer A. User can get the value to write from the following simple formula. Note that the desired timeout signal TBOUT frequency can not exceed 200KHz.

Data to Write = 0xFFFF – (Clock Source Frequency / Desired Frequency)

"Clock source Frequency" is determined by the setting of P_TimerB_Ctrl.

"Desired Frequency" is the specific frequency that user expects. User can then write the value coming from the formula to the P_TimerB_Data and turn on the specific interrupt for application.

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W								TimerB Data							

6.4 P_TimerB_Ctrl(W)($700DH)

Timer B Control Port. You can select the clock source of Timer B by setting b0 ~ b2. Also, various PWM pulses can be produced via Timer B counter by programming b6 ~ b9 of this port.

b15 – b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	Output_pulse_ctrl				---			Source C select bits		

Output_pulse_ctrl:

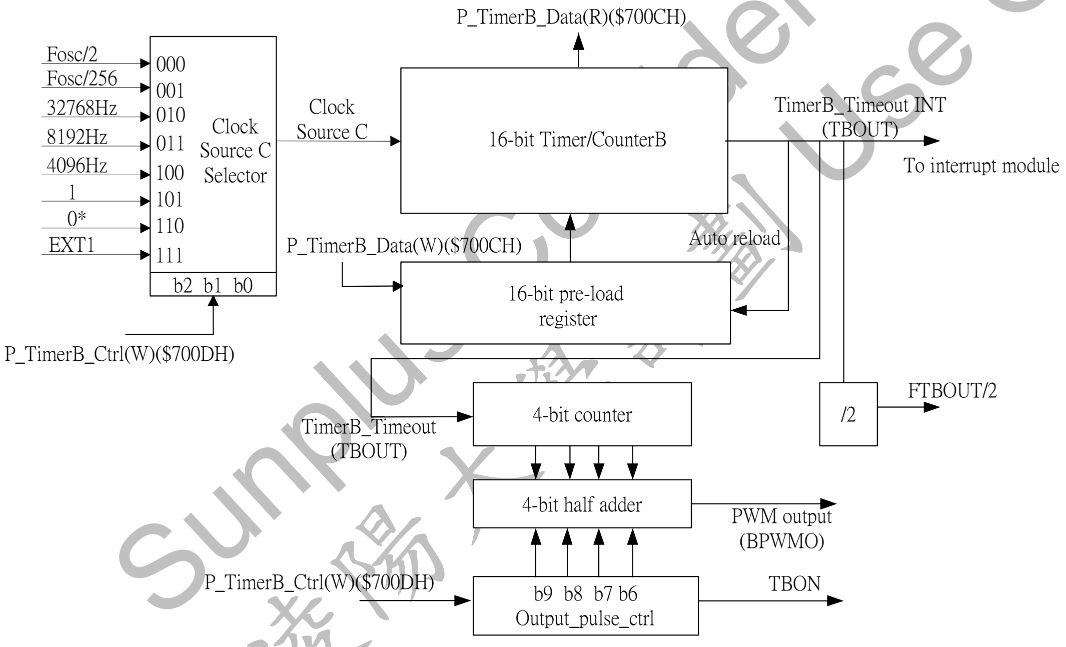
b9	b8	b7	b6	Output pulse duration (BPWMO)	TBON[a]
0	0	0	0	OFF	0
0	0	0	1	1/16	1
0	0	1	0	2/16	1
0	0	1	1	3/16	1
0	1	0	0	4/16	1
0	1	0	1	5/16	1
0	1	1	0	6/16	1
0	1	1	1	7/16	1
1	0	0	0	8/16	1
1	0	0	1	9/16	1
1	0	1	0	10/16	1
1	0	1	1	11/16	1
1	1	0	0	12/16	1
1	1	0	1	13/16	1
1	1	1	0	14/16	1
1	1	1	1	Toggle signal of TBOUT[b]	1

[a] TBON is the enable flag of the PWM output of TimerB(BPWMO), and the default value is "0".

Source C select bits:

b2	b1	b0	Clock Source C
0	0	0	Fosc/2
0	0	1	Fosc/256
0	1	0	32768Hz
0	1	1	8192Hz
1	0	0	4096Hz
1	0	1	1
1	1	0	0*
1	1	1	EXT1

*Default: Clock Source C = 0



Timer B Block Diagram

Example:

For the system clock Fosc = 49.152 MHz and if the clock source of Timer A = Fosc/2 = 24.576 MHz (That is, Clock Source A is set to Fosc/2, Clock Source B is set to 1). To generate a 8 KHz Timer A interrupt, user needs to write the data into P_TimerA_Data(W) ($700AH) by the formula

Data to Write = 0xFFFF – (Clock Source Freq / Desired Freq)

= 0xFFFF - (24.576 MHz / 8 KHz)

= 0xFFFF – 3072 (D)

= 0xFFFF – 0xC00

= 0xF3FF

Code:

```
// This code segment is setting a 8K IRQ1 interrupt at CPU speed= 49.152MHz.
// First, it sets the clock source for Timer A as Fosc/2
// Then it sets the Timer A value
// In that last 3 lines, it turns on the interrupt, IRQ1_TMA

  INT OFF;                    // Turn off IRQ,FIQ
  R1 = 0x0030;               // select Fosc/2 as timer A clock source
  [P_TimerA_Ctrl] = R1;      //
  R1 =  0xF3FF;              // Set the initial timer count for Fs = 8kHz
  [P_TimerA_Data] = R1;      //
  R1 = 0x1000;                // Enable IRQ1_TMA
  [P_INT_Ctrl] = R1;
  IRQ ON;                   // Turn on IRQ
```

Note:

Table: a quick reference for “Data to write” to set timer

Given Clock source for Timer A as Fosc/2, **Fosc = 49.152MHz**

	1K	**2K**	**4K**	**8K**	**10k**	**12K**	**16K**	**20k**	**24k**	**30k**
Timer A Data	0x9FFF	0xCFFF	0xE7FF	0xF3FF	0xF666	0xF7FF	0xF9FF	0xFB33	0xFBFF	0xFCCC

Here is another table for users’ reference

Given Clock source for Timer A as Fosc/2, **Fosc = 24.576MHz**

	1K	**2K**	**4K**	**8K**	**10k**	**12K**	**16K**	**20k**	**24k**	**30k**
Timer A Data	0xCFFF	0xE7FF	0xF3FF	0xF9FF	0xFB33	0xFBFF	0xFCFF	0xFD98	0xFDFF	0xFE66

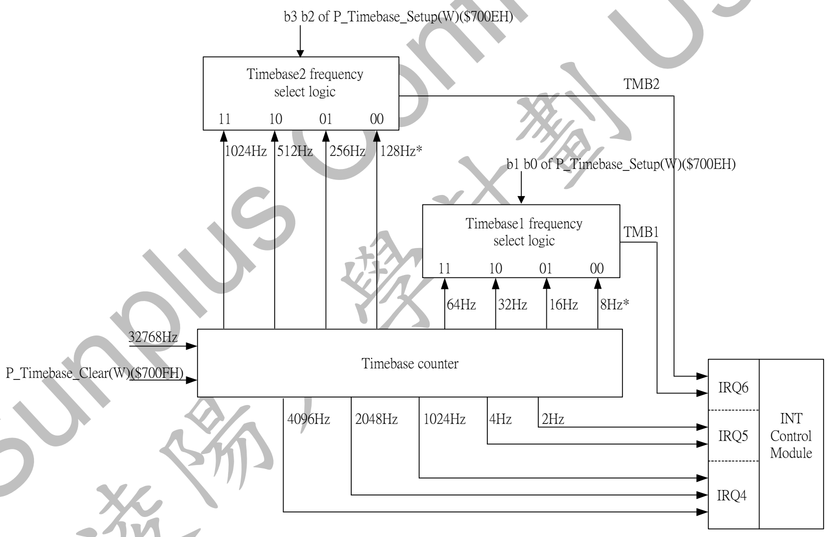
6.5 Timebase

Timebase, generated by 32768Hz, is a combination of frequency selections. The timebase blocks are TMB1 and TMB2. Both are for the frequency selections for Timer A (Clock source B) and Interrupt source (IRQ6). TMB1 provides a variety of frequency selections to the Clock Source B for Timer A. IRQ 6 can be triggered by TMB1 or TMB2 (Please refer to ***P_INT_Ctrl***). Also, timebase counter generates 2Hz, 4Hz, 1024Hz, 2048Hz, and 4096Hz interrupt sources (IRQ4 and IRQ5) for Real-Time-Clock.

6.6 P_Timebase_Setup(W)($700EH)

b15- b4	b3	b2	b1	b0
---	TMB2 Frequency select		TMB1 Frequency select	

b3	b2	TMB2	b1	b0	TMB1
0	0	128Hz*	0	0	8Hz**
0	1	256Hz	0	1	16Hz
1	0	512Hz	1	0	32Hz
1	1	1024Hz	1	1	64Hz
*Default: 128Hz			**Default: 8Hz		


b3 b2 of P_Timebase_Setup(W)($700EH)
Timebase2 frequency select logic
11
10
01
00
TMB2
1024Hz
512Hz
256Hz
128Hz*
b1 b0 of P_Timebase_Setup(W)($700EH)
Timebase1 frequency select logic
11
10
01
00
TMB1
64Hz
32Hz
16Hz
8Hz*
32768Hz
P_Timebase_Clear(W)($700FH)
Timebase counter
4096Hz
2048Hz
1024Hz
4Hz
2Hz
IRQ6
IRQ5
IRQ4
INT Control Module

Real Time Clock diagram

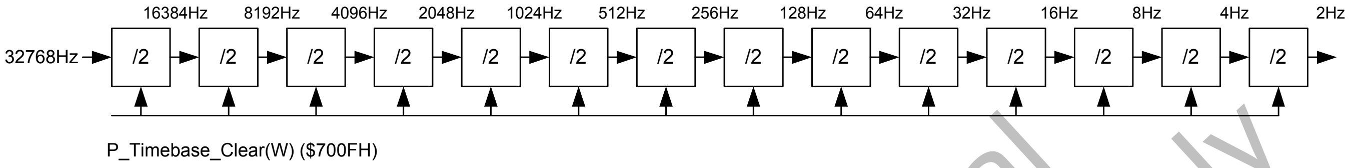
6.7 RTC (Real Time Clock)

Some applications require real time clock for time tracking. The SPCE provides several RTC interrupt sources (from 2Hz to 4096Hz) based on 32768Hz OSC. As an example of IRQ5_2Hz, the system wakes up every 0.5 second, which can be used for precise time counting.

6.8 P_Timebase_Clear(W)($700FH)

An additional control port, P_Timebase_Clear (W)($700FH), is for time reset and accuracy correction.

Write any data to this port resets all timebase counter stages to zero.



7 INTERRUPT

The *µ'nSP*® has two interrupt (INT) modes: FIQ (Fast Interrupt Request) and IRQ (Interrupt Request). The interrupt controller controls 14 INT sources; 3 of them correspond to FIQ and the other 11 INT sources correspond to IRQ. The FIQ is a high-priority interrupt and IRQs are low-priority interrupts. That is, an IRQ can be interrupted by a FIQ, which cannot be interrupted by any interrupt source. An IRQ cannot be interrupted by another IRQ unless two IRQs are generated simultaneously. If the interrupt requests of the same priority level are received simultaneously, an internal polling priority sequence (IRQ0 > IRQ1 > IRQ2 >…. > IRQ7) determines which request is served. The same criteria applies to the FIQs in which the priority is FIQ_PWM > FIQ_TMA > FIQ_TMB. Note that the polling priority in same level takes places only when two INTs occur simultaneously.

The P_INT_Ctrl(W)($7010H) or P_INT_Mask(W)($702DH) configures the interrupt sources. Writing data to P_INT_Ctrl(W)($7010H) will be equivalent to writing data to P_INT_Mask(W)($702DH). To enable interrupt(s), write an “1” to the corresponding bit(s) in P_INT_Ctrl(W)($7010H)(or P_INT_Mask(W)($702DH)). And by reading P_INT_Mask(R)($702DH), you can get the data you just write into it. That means you can keep track of interrupt enable/disable status by reading this port. Writing 0s to P_INT_Ctrl(W) ($7010H) or P_INT_Mask(W) ($702DH) will disable corresponding interrupt(s), but it will not clear the corresponding interrupt status flag existed in P_INT_Ctrl (R)($7010). When INT occurs, program should read P_INT_Ctrl(R)($7010H) to distinguish which interrupt source is activated. CPU will then process the INT according to the IRQ priority. After an interrupt (FIQ or IRQ) is executed completely, its INT vector should be cleared by writing an “1” to P_INT_Clear (W)($7011H) correspondingly and continue to process the rest of low priority IRQs until all INTs are executed. Next, CPU will leave from interrupt service routine and continue to execute other instructions. After each interrupt is completed, writing data to the correspondent bits in P_INT_Clear(W)($7011H) to clear the interrupt will allow next interrupt to come in.

For example, suppose the b5 of P_INT_Ctrl (W)($7010H) is enabled, the IRQ4_2KHz is activated. When the INT occurs, the b5 of P_INT_Ctrl(R)($7010H) turns HIGH. CPU should process the IRQ4_2KHz in INT service routine. After the INT is completed, write “0000 0000 0010 0000” to P_INT_Clear (W)($7011H) to clear the IRQ4_2KHz. This method will directly clear the processed IRQ without influencing other enabled interrupt sources.

In order to activate the interrupt service, the IRQ, FIQ or INT instructions need to be programmed correspondingly as well. User needs to explicitly activate the desired interrupt set after the P_INT_Ctrl (W)($7010H) is correctly set such as “FIQ ON” or “IRQ ON”. In case that user uses INT to activate the interrupt set and only one of the parameters, IRQ and FIQ is used, the other type of interrupt services will be turned off.

To access the interrupts, *μ'nSP®* reserves the following words for the interrupt entry points:

_FIQ, _IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _IRQ6, _IRQ7 where _IRQ7 is reserved for UART IRQ.

It also reserves _BREAK (INT Vector: 0xFFF5) for "Software Break" interrupt entry point.

Note: The "UART IRQ" is controlled by the UART setting of P_UART_Command1(W)(7021H) which is described only in the ***UART*** section.

7.1 P_INT_Ctrl(R/W)($7010H)

Writing data into P_INT_Ctrl(W)($7010H) will be the same as writing into P_INT_Mask(W)($702DH). Writing "1" enables the corresponding interrupt and writing "0" will disable the corresponding one. Reading P_INT_Ctrl(R)($7010H) will get the activated interrupt event flag.

b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	**b14**	**b13**	**b12**	**b11**	**b10**	**b9**	**b8**
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

b15	b14	
0	0	PWM(Fosc/1024) service OFF
0	1	IRQ0_PWM, set the PWM(Fosc/1024) service interrupt to IRQ0
1	-	FIQ_PWM, set the PWM(Fosc/1024) service interrupt to FIQ

b13	b12	
0	0	TimerA INT OFF
0	1	IRQ1_TMA, set TimerA interrupt as IRQ1
1	-	FIQ_TMA, set TimerA interrupt as FIQ

b11	b10	
0	0	TimerB INT OFF
0	1	IRQ2_TMB, set TimerB interrupt as IRQ2
1	-	FIQ_TMB, set TimerB interrupt as FIQ

IRQ3_EXT2	b9	EXT2 from IOB3, external interrupt with negative edge trigger
IRQ3_EXT1	b8	EXT1 from IOB2, external interrupt with negative edge trigger
IRQ3_KEY	b7	Key change Wake-up, refer to "Sleep and Wake-up" chapter for details
IRQ4_4KHz	b6	4096Hz
IRQ4_2KHz	b5	2048Hz
IRQ4_1KHz	b4	1024Hz

IRQ5_4Hz	b3	4Hz
IRQ5_2Hz	b2	2Hz
IRQ6_TMB1	b1	TMB1, TimeBase1 interrupt, refer to the “Timebase” section for details
IRQ6_TMB2	b0	TMB2, TimeBase2 interrupt, refer to the “Timebase” section for details

Interrupt Source	Interrupt	Priority	Vector	Reserved Word
PWM service(Fosc/1024)	FIQ_PWM/IRQ0_PWM	High (FIQ)	0xFFF6/ 0xFFF8	_FIQ/_IRQ0
Timer A	FIQ_TMA/IRQ1_TMA	High (FIQ)	0xFFF6/ 0xFFF9	_FIQ/_IRQ1
Timer B	FIQ_TMB/IRQ2_TMB	High (FIQ)	0xFFF6/ 0xFFFA	_FIQ/_IRQ2
EXT2	IRQ3_EXT2	Low	0xFFFB	_IRQ3
EXT1	IRQ3_EXT1	Low	0xFFFB	_IRQ3
Key-change wakeup	IRQ3_KEY	Low	0xFFFB	_IRQ3
4096Hz	IRQ4_4KHz	Low	0xFFFC	_IRQ4
2048Hz	IRQ4_2KHz	Low	0xFFFC	_IRQ4
1024Hz	IRQ4_1KHz	Low	0xFFFC	_IRQ4
4Hz	IRQ5_4Hz	Low	0xFFFD	_IRQ5
2Hz	IRQ5_2Hz	Low	0xFFFD	_IRQ5
TimeBase1	IRQ6_TMB1	Low	0xFFFE	_IRQ6
TimeBase2	IRQ6_TMB2	Low	0xFFFE	_IRQ6
UART (TxRDY or RxRDY)	UART_IRQ	Low	0xFFFF	_IRQ7

7.2 P_INT_Clear(W)($7011H)

This port clears the activated INT source for next interrupt run. Write “1” to clear the corresponding interrupt event.

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

7.3 P_INT_Mask(W/R)($702DH)

Writing data into P_INT_Mask(W)($702DH) will be the same as writing into P_INT_Ctrl($7010H). Writing “1” enables the corresponding interrupt and “0” for disabling the corresponding one. Reading data from P_INT_Mask(R)($702DH) can get the interrupt enable/disable status set by user.

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

7.4 Summary of Interrupt Control/Clear/Mask Ports

1. P_INT_Ctrl(W)(1): Enable INT/wake-up function, the same as P_INT_Mask(W)

2. P_INT_Ctrl(W)(0): Disable INT/wake-up function, , the same as P_INT_Mask(W), but it will not clear the existed interrupt event in P_INT_Ctrl(R).

3.P_INT_Ctrl (R)(1): INT event occurred

4. P_INT_Ctrl (R)(0): INT event not occurred

5. P_INT_Clear(W)(1): Clear the INT event

6. P_INT_Clear(W)(0): No-change the INT status

7. P_INT_Mask(W)(1): Enable INT/wake-up function, the same as P_INT_Ctrl(W)

8. P_INT_Mask(W)(0): Disable INT/wake-up function, the same as P_INT_Ctrl(W), , but it will not clear the existed interrupt event in P_INT_Ctrl (R) .

9. P_INT_Mask(R)(1): P_INT_Ctrl or P_INT_Mask Interrupt Enable Status

10. P_INT_Mask(R)(0): P_INT_Ctrl or P_INT_Mask Interrupt Disable Status

Example:

Demonstrate the difference of using FIQ, IRQ and INT to activate the interrupt service.

```
// Use IRQ to turn on/off IRQ
IRQ ON;        // IRQ is on
IRQ OFF;       // IRQ is off

// Use FIQ to turn on/off FIQ
FIQ ON;        // FIQ is on
FIQ OFF;       // FIQ is off

// Use INT to turn on/off FIQ,IRQ
INT FIQ IRQ;   // FIQ and IRQ are on
INT FIQ;       // FIQ is on and IRQ is off
INT IRQ;       // FIQ is off and IRQ is on
INT OFF;       // FIQ and IRQ are off
```

Example:

Use Timer A to enable FIQ_TMA(Fast interrupt source), and output a toggle signal from IOB.

```
//Set IOB as output
  R1=0x0FFFF;
  [P_IOB_Attrib]=R1;
  [P_IOB_Dir]=R1;
  R1=0x0000;
  [P_IOB_Data]=R1;
//Set TimerA=8KHz at CPU Speed = 49.152MHz
  R1=0xF3FF;
  [P_TimerA_Data]=R1;
  R1=0x0030;
  [P_TimerA_Ctrl]=R1;
//Set FIQ_TMA source
  R1=0x2000;
  [P_INT_Ctrl]=R1;
  FIQ ON;          // Turn on FIQ
// When TimerA (Interrupt source) is set, SPCE will enter FIQ sub-routine according
// to user-define frequency.
// Execution
  L_loop_fiq:
  JMP L_loop_fiq;
```

……………………………………………

//FIQ sub-routine

```
_FIQ:
PUSH R1,R5 to [SP];
R1 = 0x8000;
TEST R1,[P_INT_Ctrl];
JNE    L_FIQ_PWM;
R1 = 0x2000;
TEST R1,[P_INT_Ctrl];
JNE L_FIQ_TimerA;
R1=0x0100;
TEST  R1, [P_INT_Ctrl];
```

```
JNE    L_Quit_FIQ

L_FIQ_TMB:          // FIQ TMB entrance
   R1 = 0x0800;
   [P_INT_Clear] = R1;       //Clear FIQ TMB Event
   POP R1,R5 FROM [SP];
   RETI;

L_FIQ_PWM:               // FIQ PWM entrance
  [P_INT_Clear] = R1;        //Clear FIQ PWM Event
   POP R1,R5 FROM [SP];
   RETI;
L_FIQ_TMA:               // FIQ TMA entrance
[P_INT_Clear] = R1;      //Clear FIQ TMA Event
R1 = [P_IOB_Buffer];
R1 = R1 XOR0xFFFF;       // Toggle the IOB signal when FIQ_TMA occurs
[P_IOB_Buffer] = R1;
L_Quit_FIQ:
POP R1,R5 FROM [SP];
RETI;                    //Return to main program

_IRQ1:
  RETI;

_IRQ2:
  RETI;

_IRQ3:
  RETI;

_IRQ4:
  RETI;

_IRQ5:
  RETI;

_IRQ6:
  RETI;

_IRQ7:
  RETI;
```

8 WATCHDOG

The purpose of watchdog is to monitor if the system operates normally. Within a certain period, 0.75 seconds, watchdog counter must be cleared. If the watchdog is not cleared, CPU assumes the program has been running in an abnormal condition and therefore, CPU will reset the system to the initial state and start running the program all over again. It protects the system from incorrect code execution by launching a system reset when the watchdog timer overflows as a result of failure of software to clear the timer within 0.75 seconds. For SPCE040A/060A/061A, Watchdog function can be enabled or disabled by bonding option. That is, if pad 6(WDGOPT) of SPCE040A/060A/061A die is bonded with V_{DD}(pad 5), this chip will be watchdog-disable; if not, it will be watchdog-enable.

8.1 P_Watchdog_Clear(W)($7012H)

In SPCE, the watchdog should be cleared within every 0.75 seconds by writing “xxxx xxxx xxxx xx01b” to the P_Watchdog_Clear(W)($7012H). Thus, the system will not be reset in normal operation. If any inappropriate operation occurred which makes fail to write 0x0001 to the P_Watchdog_Clear (W)($7012H) within 0.75 seconds, the timer will generate a reset signal to reset system. If the data that are written into the P_Watchdog_Clear (W)($7012H) is not “0x0001”, SPCE040/060/061 will still reset the hardware. Therefore user is required to write the designated pattern (xxxx xxxx xxxx xx01b) to P_Watchdog_Clear(W)($7012H) to clear the watchdog; otherwise the system may function incorrectly. The watchdog function will be disabled by hardware during sleep mode.

Example:

```
R1 = 0x0001
[P_Watchdog_Clear] = R1;    // Reset the watchdog timer
```

9 SLEEP AND WAKE-UP

The purpose of sleep mode is to save powers while device is not in use. When sleep acts, the device runs from operating mode to standby mode. On the other hand, wake-up acts from sleep mode to operating mode. User is advised not to use sleep function to a frequency more than 5KHz. Every time the system wakes up from sleep mode, the PLL takes a wake-up time to oscillate stably. This wake-up time is about 200 μs. It implies that the sleep / wake-up frequency is not advised to exceed 5KHz. Otherwise, the system will never actually enter sleep mode at the frequency above 5KHz.

1. Sleep: After power on reset, IC starts working until a sleep command is given. When a sleep signal is accepted, IC will turn off system clock(PLL) and ADC and enter into sleep mode. You can change the system to sleep mode by programming P_SystemClock(W)($7013H). After entering sleep mode, program counter will stop at the next line and will start to execute after wake-up interrupt completed.

2. Wake-up: Waking up from sleep mode requires a wake-up signal to turn on the system clock (PLL). At the same time, a wake-up interrupt (FIQ or IRQ) is also generated. The interrupt signal leads CPU to complete the wake-up process as well as initialization. IRQ3_KEY key-change-wake-up (IOA7~0) and interrupt sources (FIQ, IRQ1~IRQ6 and UART IRQ) can be used as the wake-up IRQ sources. After wake-up interrupt completed, program counter will continue to execute the next command.

- For key change wake-up source, refer to ***I/O PortA configuration*** for details.

Note:

1. When entering the sleep mode , at the same moment, SPCE will disable two DACs and ADC. I.e. $V_{MIC} = V_{ADREF} = 0V$.
2. Sleep mode is also called standby mode in this documentation.

Example:

Enter Sleep mode and wake SPCE up with key change.

Before entering into sleep mode, set the IOA[7~0] as input mode and turn on IRQ3_KEY(Key change Wake-up) because Key change Wake-up is one of the interrupt source of IRQ3.

```
//IOA[7-0] configuration (for other I/O configurations, please refer to
//the I/O section).
R1=0x0000;
[P_IOA_Attrib]=R1;
R1=0x0000;
[P_IOA_Dir]=R1;
[P_IOA_Data]=R1;
```

```
         //Interrupt configuration (set IRQ3_KEY, Key change interrupt source, for other
         //interrupt configurations, please refer to the Interrupt section)
         INT OFF;
         R1=0x0080;
         [P_Int_Ctrl]=R1;
         IRQ ON;

    //Read this port to latch data on the I/O port for key change wake up
         R1=[P_IOA_Latch];

         // Write "1" to P_SystemClock(W)7013H($) B2-B0 to enter into sleep mode(please
         // refer to the System Clock section)
         R1=0x0007;
         [P_SystemClock]=R1;
```

…(After PortA key change wake-up source triggered, it will call IRQ3 subroutine)

Sub-routine of IRQ3:

```
 _IRQ3:
    R1 = 0x0100;
    TEST R1,[P_INT_Ctrl];
    JNZ L_IRQ3_Ext1;
    R1 = 0x0200;
    TEST R1,[P_INT_Ctrl];
    JNZ L_IRQ3_Ext2;
L_IRQ3_KeyChange_WakeUp:
      :
     (Tasks after wakeup)
      :
    R1 = 0x0080;          //Clear IRQ3(Key Change Wakeup) Event
    [P_INT_Clear]= R1;
    RETI

L_IRQ3_Ext2:
    [P_INT_Clear] = R1;        //Clear IRQ3 EXT2 Event
    RETI

L_IRQ3_Ext1:
    [P_INT_Clear] = R1;        //Clear IRQ3 EXT1 Event
    RETI
```

10 SYSTEM CLOCK

Basically, the system clock (Fosc) is provided by PLL and programmed by the P_SystemClock(W)($7013H) to determine the frequency of CPU clock for system. The default system clock(Fosc) is 24.576MHz and default CPU clock is Fosc/8 if not specified. Users are able to define the system clock and CPU clock by programming P_SystemClock(W)($7013H).

SPCE supports 32768Hz OSC running at normal mode and auto-power saving mode. The difference is the power supplied to the 32KHz crystal circuit. The output current of the normal mode is 10uA larger than the power saving mode. When the crystal begins to oscillate, the crystal circuit will need more power to make it starting properly. However, after the oscillation is stabilized, only small amount of current is needed. In normal mode the 32768Hz OSC always runs at the full power consumption. In auto-power saving mode, the first 7.5 seconds after power-on reset or wake-up is running at normal mode and it will then turn back to power-saving mode automatically to save powers. Since the 32KHz OSC is default as auto-power saving mode after power on reset and wake-up, please set the b3=1 of P_SystemClock (W)($7013H) after power on and wake-up to switch to normal mode manually.

Moreover, after power on reset and wake-up, the CPU clock is set to be Fosc/8 initially and then users can adjust to the desired CPU clock. This action avoids ROM running into read failure when system reset and wake-up. In SPCE040/060/061, user can maximal set the system clock (Fosc) and CPU clock up to 49.152MHz.

10.1 P_SystemClock(W)($7013H)

This port controls the system and CPU clocks. The system clock (Fosc) can be configured from 20.480MHz to 49.152MHz through programming b5 ~ b7 of P_SystemClock ($7013). By setting b0 ~ b2 to all 1s, it stops the CPU clock and enters sleep mode for power savings. Besides, you can turn 32KHz RTC on or off in sleep mode by programming bit4 and set the 32KHz to run at normal mode or auto-power saving mode by programming bit3.

b15-b8	b7	b6	b5	b4[a]	b3	b2	b1	b0
-----	Fosc2	Fosc1	Fosc0	32KHz Sleep Status	32KHz Mode	CPU Clock		

b7: Fosc2	b6: Fosc1	b5: Fosc0	System Clock Fosc
0	0	0	24.576MHz(default)
0	0	1	20.480MHz
0	1	0	32.768MHz
0	1	1	40.960MHz
1	X	X	49.152MHz

32KHz Sleep Status	b4	1	In sleep mode, 32.768KHz is still working (Default)
		0	In sleep mode, 32.768KHz is OFF
32KHz Mode	b3	1	Normal mode
		0	Auto-power saving mode (Default)

[a] b4 is effective only when b0, b1 and b2 are all set to “1” at the same time, i.e. to standby(sleep) mode

b2	b1	b0	CPU Clock
0	0	0	Fosc
0	0	1	Fosc/2
0	1	0	Fosc/4
0	1	1	Fosc/8[b]
1	0	0	Fosc/16
1	0	1	Fosc/32
1	1	0	Fosc/64
1	1	1	Stop(Sleep)

[b] Default CPU clock after power on reset and system wakeup = Fosc/8

Example:

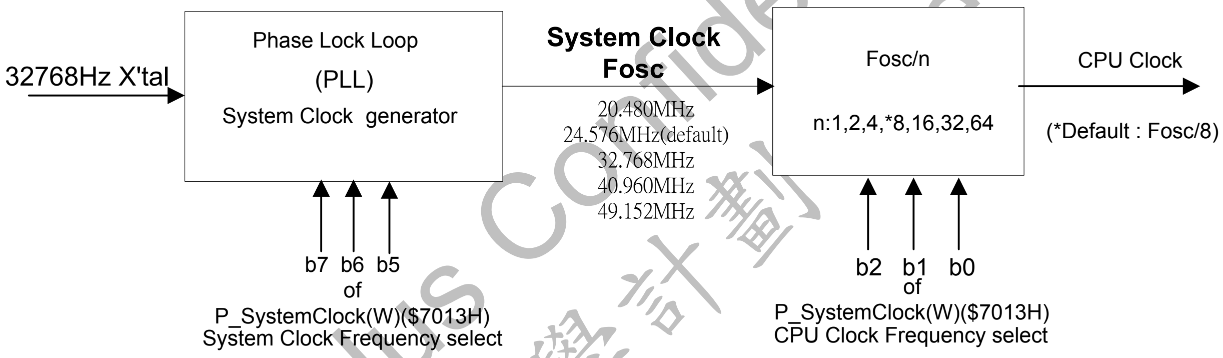
```
// User usually has to initialize the system clock, when starting the system.
R1 = 0x0080;           // System Clock Fosc = 49.152 MHz,
[P_SystemClock] = R1;  //  In sleep mode, 32.768KHz is OFF, Auto-power saving
mode, CPU Clock =Fosc
```

Example:

```
// User issues an instruction to force system into sleep mode to save power.
R1 = 0x0097;         //Sleep mode. System Clock Fosc = 49.152 MHz,
[P_SystemClock]= R1;
 // In sleep mode, 32.768KHz is ON , Auto-power saving mode, CPU Clock =Fosc
```

11 PLL (Phase Lock Loop)

The role of PLL is to provide a base frequency (32768Hz) and pumps the system clock to 20.480MHz ~ 49.152MHz. The default PLL free-run frequency after power on is 24.576MHz, which provides the default system clock Fosc. By setting the b0~b2 of P_SystemClock(W)($7013), the CPU clock is adjustable. The default CPU clock is Fosc/8 after power on and sleep wakeup. The PLL will go off in sleep mode. When program get back from sleep mode, the PLL requires a wake up time to bring PC back to work. It is about 200 μs' waiting. A diagram structure is shown as follows.



The oscillation from Crystal is fed to PLL as reference to generate system clock. System clock, Fosc, is decided by [b5 ~ b7] of P_SystemClock. CPU clock will then be determined by the combination of [b2~b0] of P_SystemClock. (Please refer to ***System Clock***). Once CPU clock is determined, user can have a good estimate on the MIPS of the system.

12 ADC (ANALOG TO DIGITAL CONVERTER)

SPCE provides 8 channels of 10-bit ADC with built-in multiplexer. One channel A/D is MIC_In with amplifier and AGC controller. The other seven channels are shared with IOA[6:0] as Line_In inputs. The A/D operating range of SPCE 040A/060A/061A is the full input range, i.e. the maximal A/D analog signal input can be from 0V to AV_{dd}. The presence of invalid analog signal (over VDD+0.3 V or below VSS–0.3 V) will degrade the ADC performance by affecting the operating region of the switch circuit. Due to the physical connection of ADC circuit and IOA0~6, it is advised that user can chose to use different I/O pins other than IOA0~6 to avoid the ADC performance deterioration from the power jittering caused by invalid I/O signal (over VDDIO+0.7V or below VSSIO-0.7V).

The ADC ceiling voltage can be set by the combination of b7 and b8 of P_ADC_Ctrl(W)($7015H). The VEXTREF, b7 of P_ADC_Ctrl(W)($7015H), can determine if ADC should refer to either AV_{dd} or external voltage reference. V2VREFB, b8 of P_ADC_Ctrl(W)($7015H), decides if the internal 2V voltage regulator should function. If the 2V regulator functions, user can feed this 2V reference to VEXTREF pin of SPCE040/060/061 and this connection, as a result, will set the ADC top reference as 2V. User can also provide user-specified voltage reference source to VEXTREF as ADC top reference voltage on the condition that this user-specified external voltage reference source does not exceed AV_{dd}.

In A/D converter, DAC0 and SAR (Successive Approximation Register) form the SAR ADC. To enable A/D converter, simply write “1” to the b0(ADE) of P_ADC_Ctrl(W)($7015H). The default value is “0” (A/D disabled). If A/D converter is enabled, users should further set the rest of control bits in P_ADC_Ctrl(W)($7015H) and P_ADC_MUX_Ctrl(W)($702BH) properly.

Users can select A/D input channel from the MIC_In input or one of the Line_In inputs by setting the b0~b2 of P_ADC_MUX_Ctrl(W)($702BH). In the runtime, if both MIC_In and Line_In oerate in direct mode, program would have to check the b15 of P_ADC_Ctrl(W)($7015H) or P_ADC_MUX_Ctrl (R) ($702BH). Only when the previous ADC session is done can the channel switching be successful. If MIC_In is in Timer latch mode, then MIC_In is prioritized to access AD converter. Then the FailB of P_ADC_MUX_Ctrl (R) ($702BH) can be used to indicate if the Line_In ADC was interrupted by MIC_In and was not successful.

To get the 10-bit ADC result from MIC_In, users must read P_ADC(R)($7014H).

To get the 10-bit ADC result of the selected Line_In input, users must read P_ADC_LINEIN_Data(R)($702CH).

In the MIC_In mode, you can also select the trigger event for A/D conversion. By setting the b3 and b4 of P_DAC_Ctrl(W)($702AH), the A/D MIC_In mode conversion is executed whenever P_ADC(R)($7014H) is read or TimerA/TimerB events are occurred. However, in the A/D Line_In mode, the A/D conversion is executed only when P_ADC_LINEIN_Data(R)($702CH) is read. The A/D conversion through Line_in can not use Timer to latch data automatically.

The ADC will be turned off(include AGC and V_{MIC}) when SPCE enters sleep mode. Please note that the V_{MIC}

signal(provide the power to the external connected MIC, V_{MIC} =AV_{DD}) is default as on after power on reset, even the ADC is default as disabled at this time. That means the V_{MIC} on/off is independent to the enable/disable status of the ADC which is controlled by the b0(ADE) of P_ADC_Ctrl(W)($7015H). Thus if you do not need to use the V_{MIC} signal, you have to set the b1(MIC_ENB)=1 of P_ADC_Ctrl(W)($7015H) to turn off it.

Note that the hardware A/D conversion speed limitation is **(Fosc/32/16) Hz.** Try to execute A/D conversion faster than this limitation may result in the acquisition of unexpected digital data while reading from P_ADC(R)($7014H) or P_ADC_LINEIN_Data(R) ($702CH).

The b5(DAC_OUT) of P_ADC_Ctrl(W)($7015H) configures the two channels audio DACs' maximal output current level to be 2mA or 3mA (default). This setting changes the driving power of the DAC output.

ADC 10-bit Data Maximum Conversion Rate: (Fosc/32/16)

System Clock	**20.48MHz**	**24.576MHz**	**32.768MHz**	**40.96MHz**	**49.152MHz**
Conversion rate	40KHz	48KHz	64KHz	80KHz	96KHz

12.1 P_ADC(R/W)($7014H)

The ADC includes a 10-bit DAC (DAC0) with a 10-bit buffer register (DAR0), a SAR, and a comparator.

b15 – b6	b5 – b0
DAR0(R/W)	---

P_ADC(R): Read A/D 10-bit digital output in MIC_In mode. Also, in the conversion by reading ADC mode (b4=b3=0 in P_DAC_Ctrl (W) ($702A)), reading this port will perform the MIC_In A/D conversion.

12.2 P_ADC_Ctrl(R/W)($7015H)

b15	b14	b13~b9	b8	b7	b6	b5~b3	b2	b1	b0
RDY (R)	---	---	V2VREFB (W)	VEXTREF (W)	DAC_OUT (W)	---	AGCE (W)	MIC_ENB (W)	ADE(W)

RDY b15 1 A/D digital data ready, 10-bit A/D conversion completed. A/D is ready for next conversion.
0 A/D digital data not ready, 10-bit A/D conversion not yet completed. A/D is not ready for next operation.

V2VREFB b8 1 V2VREF output disabled (Default). Turn off the built-in regulator 2V output.
0 V2VREF output enabled. Turn on the built-in regulator 2V output. Users can use this regulate 2V signal as the external A/D top reference voltage.

VEXTREF b7 1 VEXTREF pin enable. Users must input the external signal to VEXTREF as A/D top reference voltage.
0 VEXTREF pin disable. The A/D top reference voltage for Line_In is fix to AV_{dd} (Default).

DAC_OUT b6 1 DAC maximal output current = 2mA
0 DAC maximal output current = 3mA (Default)

AGCE b2 1 Enable the AGC function. This function is only effective for the MIC_In mode(b1(MIC_ENB)=0) and the MIC gain can be adjusted by external component.
0 Disable the AGC function in MIC_In mode(Default)

MIC_ENB b1 1 MIC_In disabled. V_{MIC} and AGC will be turned off. Note that V_{MIC} cannot be controlled by the b0(ADE) but only b1(MIC_ENB) can turn it off.
0 MIC_In mode enabled. Microphone power V_{MIC}=AV_{DD} is turned on when power on reset(Default)

ADE b0 1 Enable A/D converter
0 Disable A/D converter (Default)

12.3 P_ADC_MUX_Ctrl(R/W)($702BH)

b15	b14	b13-b3	b2	b1	b0
RDY(R)	FailB(R)	---	A/D MUX Channel select(R/W)		

RDY b15 1 A/D digital data ready, 10-bit A/D conversion completed. A/D is ready for next conversion.

0 A/D digital data not ready, 10-bit A/D conversion not yet completed. A/D is busy and not ready for next operation.

This bit equals to the b15(RDY) of P_ADC_Ctrl(R/W)($7015H)

FailB b14 0 10-bit A/D conversion (Mic Timer latch & Line_in) failed.(Default)

1 10-bit A/D conversion (Mic Timer latch & Line_in) completed successfully.

b2	b1	b0	A/D Channel
0	0	0	MIC_In
0	0	1	LINE_In1 from IOA0
0	1	0	LINE_In2 from IOA1
0	1	1	LINE_In3 from IOA2
1	0	0	LINE_In4 from IOA3
1	0	1	LINE_In5 from IOA4
1	1	0	LINE_In6 from IOA5
1	1	1	LINE_In7 from IOA6

IOA[6:0] shared with ADC Line_In input channels

IOA6	IOA5	IOA4	IOA3	IOA2	IOA1	IOA0
LINE_In7	LINE_In6	LINE_In5	LINE_In4	LINE_In3	LINE_In2	LINE_In1

12.4 P_ADC_LINEIN_Data(R)($702CH)

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

This port is read only. Read this port to get the 10-bit ADC result from the selected LINE_In input.

12.5 ADC operates in "MIC_IN" mode

12.5.1 ADC range

In MIC_In mode, the A/D top reference voltage is AV_{dd}, which means the A/D MIC_In input range is from 0V to AV_{dd} full range. The signal came from MIC_In will be passed through a buffer and amplified. And the amplifier gain can be adjusted by external circuit. Then, AGC(if enabled) will control the MIC_In signal in a range.

12.5.2 Setting

To operate ADC in MIC_In mode, users need to set the b0(ADE)=1 and b1(MIC_ENB)=0 of P_ADC_Ctrl(W)($7015H) to enable A/D and MIC_In mode(the MIC power V_{MIC} will be default turned on when power on reset happen). Then, users can further decide whether to enable the AGC function by setting the b2(AGCE). Also remember to set the b3 and b4 of P_DAC_Ctrl(W)($702AH) to determine the trigger mode for MIC_In A/D conversion (Timer latch or direct mode). Setting the b0~b2 to "0" of P_ADC_MUX_Ctrl(R/W)($702BH) will switch the multiplexer to the MIC_In channel input.

12.5.3 Operation

When the A/D MIC_In trigger event occurred, it generates a start signal with b15(RDY) = 0. Next, DAC0 output is compared with external signal using the binary search to find out the digital output of external signal. The successive approximation control first sets the most significant bit (MSB) and clears all other bits in SAR (10 0000 0000B). The output of the DAC0 (1/2 full scale) is compared to the input voltage V_{in}. If V_{in} is greater than the DAC0 voltage (V_{DAC}), the bit remains "1"; otherwise, it is cleared to "0". The successive approximation control now sets the next most significant bit (11 0000 0000B or 10 0000 0000B depends on the previous result), and V_{DAC} is compared to V_{in} again. If V_{in} is greater than V_{DAC0}, the bit being tested remains "1"; otherwise, it is cleared to "0". This procedure is repeated until all ten bits have been tested. During the conversion, the 10-bit A/D conversion result is held in SAR. When 10-bit A/D conversion is completed, the RDY (b15 of P_ADC_Ctrl (R) ($7015H)) will be set to "1".

12.5.3.1 Timer latch mode

At this moment, program should read the P_ADC (R) ($7014H) if in direct mode or ADC will latch data from MIC_In by Timer event if in Timer A/B latch mode (event sources set by P_DAC_Ctrl($702AH) to get the 10-bit A/D data. The timer event can be Timer A , Timer B or [(TimerA or Timer B) and PWM]. After reading data from P_ADC(R) ($7014H) either directly or auto event by Timer A, B, the b15(RDY) will be cleared to "0" and the ADC starts executing A/D conversion again. If the result of the ADC is not read from P_ADC (R) ($7014H) , b15(RDY) will remain being "1" and A/D will not proceed to execute the next A/D conversion. Note that the b15 (RDY) of P_ADC_Ctrl (R) ($7015H) is physically the same as b15(RDY) of P_ADC_MUX_Ctrl (R) ($702BH).

If in Timer latch mode, among the eight A/D channels, MIC_In is in the position of higher priority to other seven Line_In channels. It is to keep the real time signal capturing ability of the sound input coming from the MIC_In channel. That means if any Line_In channel is currently set as active channel, and during the 10-bit A/D conversion period for that Line_In channel, MIC_In is selected and triggered. The multiplexter will then be automatically switched to MIC_In channel coercively and A/D will start a new 10-bit A/D conversion for the MIC_In signal. In case that the Line_In ADC is interrupted by MIC_IV ADC request, the FailB, b14 of P_ADC_MUX_Ctrl(R/W)($702BH), will be low. In contrast, if a MIC_In channel is active and processing the A/D conversion, users will not be able to control the multiplexer (b0~b2 of P_ADC_MUX_Ctrl(R/W)($702BH)) to switch to any other Line_In channel until the b15(RDY) is set to "1". Please note the FailB, b14 of P_ADC_MUX_Ctrl(R/W)($702BH), is meaningful only when MIC_In in Timer latch mode and Line_In are working at the same time.

12.5.3.2 Direct mode

If b3 and b4 of P_DAC_Ctrl(W)($702AH) set the MIC ADC to operate in Direct mode, in order to launch a ADC session, user would have to read the P_ADC (R) ($7014H) first to start the ADC session, and then poll the status of ADC by reading b15 of P_DAC_Ctrl($702AH). When ADC is ready, program can read the P_ADC (R) ($7014H) again to get the data.

12.5.4 MIC_In Front End Amplifier

There are two stages of OP amplifiers in the MIC_In channel. Please refer to the following ADC Block Diagram for details. The gain of the first amplifier(OPAMP1) is ***15V/V*** when AGC is off. The gain of the second stage amplifier (OPAMP2) is ***60K/(1K+Rext)*** and thus the gain can be adjusted by the Rext value. The larger the Rext is, the smaller the OPAMP2 amplifier gain will be. For example if Rext is 5.1K ohm, the OPAMP2 gain is 60K/(1K+5.1K) = 9.8 (19.8dB). In this case, the overall MIC amplifier gain will be OPAMP1x OPAMP2 = 15 X 9.8 = 147(43.3dB).

If AGC is enabled(by setting b2 of P_ADC_Ctrl (W) ($7015H)), it will adjust the gain to prevent the signal saturation. When the output of OPAMP2 is larger than 0.9AVdd, AGC will lower the gain of OPAMP1 to avoid the saturation of the amplified MIC_In signal.

12.6 ADC operates in "LINE_IN" mode

SPCE provides seven Line_In channels input and these Line_In channels are shared with IOA[6:0] pins. Users need to configure the corresponding IOA pin(s) as " input "(s) to make this pin functioning as Line_In channel(s) properly. Please note that since the I/O input configuration of "Pull High" and "Pull Low" have internal physical pull resistor which might affect the external Line_In signal level. Thus, in most cases, it is advised to configure IOA[6:0] as " input with float " for being the Line_In channel input.

12.6.1 ADC range

The maximal valid input range of Line_In through IO[6:0] is from 0V to AV_{dd}.

In Line_In mode, the A/D top reference voltage can be selected by setting the b7(VEXTREF) of P_ADC_Ctrl(W)($7015H). When b7(VEXTREF) is set to "0", the A/D top reference voltage for Line_In is selected to be AV_{DD}, which means the Line-In signal input range can be from 0V to AV_{dd}, the full range. If b7(VEXTREF) is set to "1", the VEXTREF pin is enabled and user needs to input external signal to VEXTREF pin to be the A/D Line_In top reference voltage. The valid voltage range of VEXTREF is from 0V to AV_{dd}. In this case, the Line_In input range will be from 0V to VEXTREF. Note that the lower the VEXTREF(A/D top reference voltage) is, the smaller the A/D Line_In input range is. It also implies that the lower SNR(Signal-to-Noise-Ratio) for the A/D input signal to be converted will be. Since SPCE provides a built-in 2V regulator(obtained by setting b8(V2VREFB) of P_ADC_Ctrl(W)($7015H) to "0"), user can enable the V2VREF pin and turn on the built-in 2V regulator output. This regulated 2V signal can be externally connected to VEXTREF pin as the A/D Line_In top reference voltage.

12.6.2 Setting

SPCE provides seven Line_In channels input and these Line_In channels are shared with IOA[6:0] pins. Users need to configure the corresponding IOA pin(s) as " input "(s) to make this pin functioning as Line_In channel(s) properly. Please note that since the I/O input configuration of "Pull High" and "Pull Low" have internal physical pull resistor, which might affect the external Line_In signal level. Thus, in most cases, it is advised to configure IOA[6:0] as " input with float " for being the Line_In channel input.

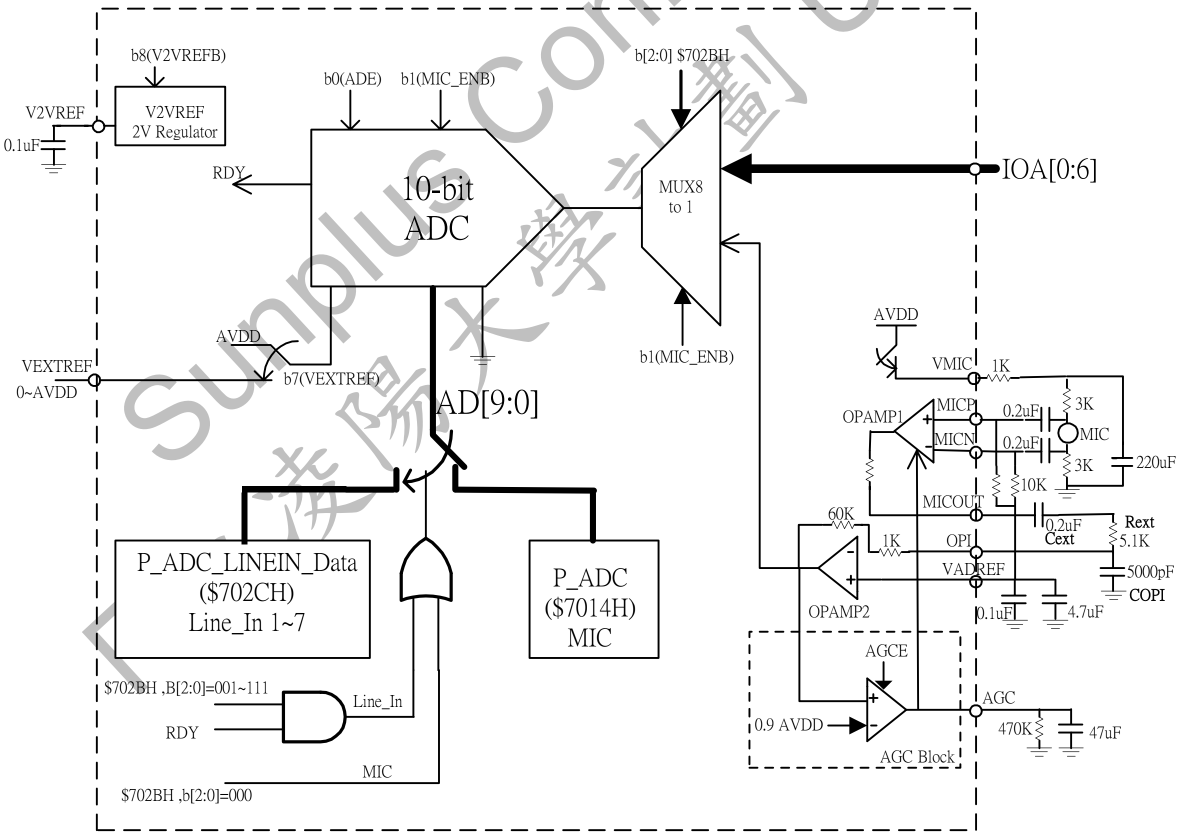
Since SPCE provides eight A/D multiplexing channels but only one physical A/D converter, before user can control the bit2~0 of P_ADC_MUX_Ctrl(R/W)($702BH) to switch to other Line_In channel, users must first check the b15(RDY) of P_ADC_MUX_Ctrl (R) ($702BH)(or b15(RDY) of P_ADC_Ctrl (R) ($7015H)) to make sure that the A/D is not busy and ready for next conversion. If b15(RDY) is not an "1", and thus ADC is busy, then any attempt to control the multiplexer (bit2~0 of P_ADC_MUX_Ctrl(R/W) ($702BH)) will be ignored.

12.6.3 Operation

Comparing with MIC_In mode, which can be triggered by different modes (set by b3 and b4 of P_DAC_Ctrl(W)($702AH)), the A/D conversion in Line_In mode can only be triggered by the event that user reads the P_ADC_LINEIN_Data(R)($702CH). Note that MIC_In channel is prioritized to Line_In channels if MIC_In is in Timer latch mode. Therefore in case that both MIC_In and Line_In are used simultaneously, the on-going 10-bit A/D conversion of Line_In might be interrupted by a MIC_In A/D conversion request. Thus, to make sure of getting a valid and complete 10-bit A/D result from the

P_ADC_LINEIN_Data(R)($702CH) , users should always check the b14(FailB) of P_ADC_MUX_Ctrl(R/W) ($702BH) to see whether the 10-bit A/D Line_In conversion was completed successfully or had been interrupted. It is meaningful to check FailB, b14 of P_ADC_MUX_Ctrl(R/W) ($702BH), only when MIC_In Timer latch mode and after first MIC ADC is executed.

When Line_In mode is selected successfully by setting bit2~bit0 of P_ADC_MUX_Ctrl(R/W)($702BH), read the P_ADC_LINEIN_Data(R)($702CH) will start executing A/D conversion and the b15(RDY) will be cleared to "0" at the same time. When b15(RDY) changes back to "1", it means the triggered A/D conversion is completed and user can read P_ADC_LINEIN_Data(R)($702CH) again to get the resulted 10-bit digital data if b14(FailB) of P_ADC_MUX_Ctrl(R/W)($702BH) is "1". Note that this reading of P_ADC_LINEIN_Data(R)($702CH) will again trigger the A/D to execute a new conversion. If the b14(FailB) of P_ADC_MUX_Ctrl(R/W)($702BH) is "0" at this time, it means a MIC_In A/D conversion has interrupted this Line_In A/D conversion and resulted an unexpected digital data in P_ADC_LINEIN_Data(R)($702CH).



ADC Block Diagram

Note:

The application circuit for microphone can be adjusted to enhance the noise immunity by user. Users can refer to the ADC block diagram above and the RC table below to tune the bandwidth of the microphone band pass filter for better the front-end A/D performance.

RC Table:

Cext	Rext	Copi	f upper –3db	f lower –3db	OPO gain
0.22uF	5.1K	0	476.6KHz	107Hz	43dB
0.22uF	5.1K	5000pF	31.9KHz	107Hz	43dB
0.22uF	5.1K	10000pF	15.0KHz	107Hz	43dB
0.22uF	5.1K	15000pF	9.8KHz	107Hz	43dB
0.22uF	5.1K	20000pF	7.3KHz	107Hz	43dB
0.22uF	5.1K	30000pF	4.8KHz	107Hz	43dB
0.22uF	5.1K	40000pF	3.5KHz	107Hz	43dB
0.22uF	5.1K	50000pF	2.8KHz	107Hz	43dB
0.10uF	5.1K	0	476.6KHz	216Hz	43dB

13 AUDIO OUTPUT

The audio of SPCE can be exported to two DACs. The DAC1 current signal is outputted through AUD1 pin, and the DAC2 is through AUD2 pin. The DAC data ranges from 0x0000 to 0xFFFF. If the DAC data will be processed as PCM data , user needs to know that the DC level for the DAC Data is 0x8000. Only ten higher bits (MSB) are required for processing. The data of DAC1 and DAC2 should be delivered to P_DAC1 (W) ($7017) and P_DAC2 (W) ($7016) respectively. After power on reset, the two DACs are default as turned on which will consume several mA current. So if users do not need to use the two DACs, you can turn off them to save normal operating current by setting b1=1 of the P_DAC_Ctrl(W)($702AH).

An application issue here is that the DC level of DAC needs to ascend or descend smoothly. Otherwise, it is possible to cause unexpected explosive noise in the speaker due to the abrupt DC level change. A ramp up/down technique is often used to cope with this problem by building/releasing the DC level gradually. It is encouraged for users to apply this technique to the first time use of DAC after power on or waking up from sleep mode and for the last time use of DAC before power off or entering sleep mode. It will significantly yield better audio output quality. Please refer to the example for the ramp/up/down technique.

13.1 P_DAC2(R/W)($7016H)

In DAC mode, this port is a 10-bit DAC (DAC2) with a 10-bit buffer register (DAR2).

b15 – b6	b5 – b0
DAR2_Data(R/W)	---

P_DAC2(W) : Write 10-bit data into DAR2 for DAC2 (unsigned)

P_DAC2(R) : Read 10-bit data from DAR2

13.2 P_DAC1(R/W)($7017H)

This port is a 10-bit DAC (DAC1) with a 10-bit buffer register (DAR1).

b15 – b6	b5 – b0
DAR1_Data(R/W)	---

P_DAC1(W) : Write 10-bit data into DAR1 for DAC1 (unsigned)

P_DAC1(R) : Read 10-bit data from DAR1

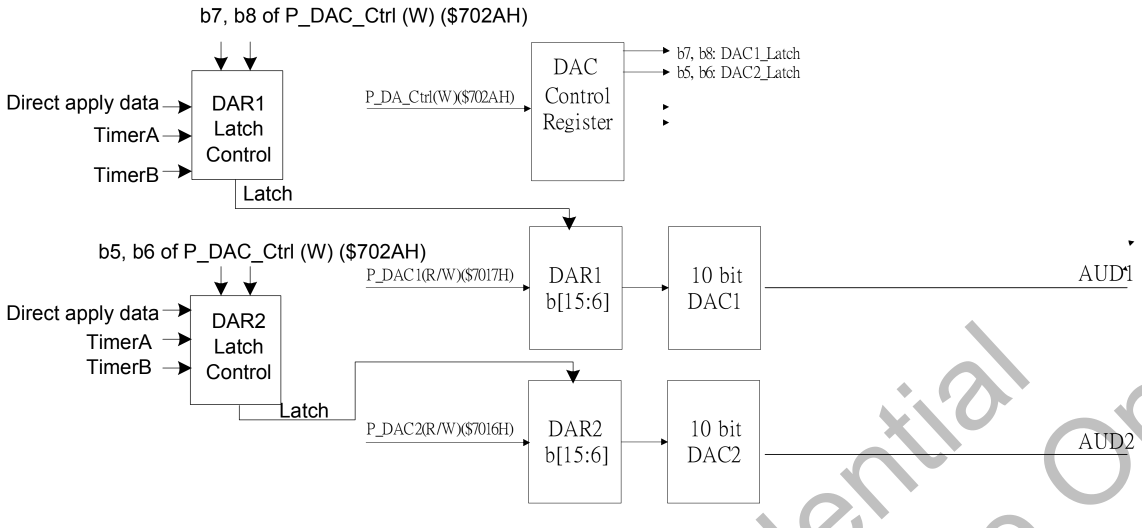
13.3 P_DAC_Ctrl (W)($702AH)

Configuration Port for DAC operation. The b1 controls the enable/disable of the two DACs. The two DACs can only be enable/disable at the same time but not individually. After power on the two DACs will be turned on(b1=0) which will consumes several mA current. If user do not need to use the two DACs, you can set b1=1 to turn off(disable) them to save normal operating current. The b5 ~ b8 provide different data latch modes for DAC output. Note that b3 and b4 configure the conversion mode of ADC. User can specify when the AD conversion should be performed It can be triggered by Timer A, Timer B, both Timer A/B or the action of reading P_ADC(R)($7014). (Please refer to ***ADC***)

b2	b1	b0
---	DAC Enable	---

b15 - b9	b8	b7	b6	b5	b4	b3
---	DAC1_Latch(W)		DAC2_Latch(W)		AD_Latch(W)	

DAC1_Latch (W)	b8	b7	
	0	0	Direct apply DAR1 data to DAC1 (Default)
	0	1	Latch DAR1 data to DAC1 by Timer A
	1	0	Latch DAR1 data to DAC1 by Timer B
	1	1	Latch DAR1 data to DAC1 by either TimerA or TimerB events occurred
DAC2_Latch(W)	b6	b5	
	0	0	Direct apply DAR2 data to DAC2 (Default)
	0	1	Latch DAR2 data to DAC2 by Timer A
	1	0	Latch DAR2 data to DAC2 by Timer B
	1	1	Latch DAR2 data to DAC2 by either TimerA or TimerB events occurred
AD_Latch(W)	b4	b3	
	0	0	Execute A/D MIC_In conversion by read P_ADC(R)($7014H) (Default)
	0	1	Execute A/D MIC_In conversion by Timer A
	1	0	Execute A/D MIC_In conversion by Timer B
	1	1	Execute A/D MIC_In conversion by the event , [(TimerA or TimerB) and PWM] .
DAC Enable	b1	1	Two DACs output disable.
		0	Two DACs output enable (Default)



Audio Output Block Diagram

Example

This example utilizes Timer A as a 8 KHz interrupt. The ADC samples voice data from microphone and sends the data to DAC1 and DAC2 in IRQ1.

In main.asm:

```
        INT OFF;
         R1=0x0080;                      // CPU Clock = 49 MHz
         [P_SystemClock]=R1;
         R1 = 0x0030;                   // select Fosc/2 as timer A clock source
         [P_TimerA_Ctrl] = R1;          // at CPU clock = 49.152 MHz
         R1 = 0xF3FF;                   // Fs = 8kHz
         [P_TimerA_Data] = R1;          //

         R1 = 0x0105 ;                  // AGC enable,
         [P_ADC_Ctrl] = R1;             // Mode=auto, INIB disabled, Source=MIC,

         R1 = 0x0000    // Mic
          [P_ADC_MUX_Ctrl] = R1

         R1 = 0x00A8;                   // Latch DAR1/DAR2 to DAC1/DAC2 by TimerA,
        [P_DAC_Ctrl] = R1;             // Latch A/D by Timer A , 2 DAC output
         R1 = 0x1000;                   //
         [P_INT_Ctrl] = R1;             // Enable IRQ1 for Timer A
         IRQ ON;
L_DeadLoop:
         R1 = 0x0001;                // Clear Watch Dog
         [P_Watchdog_Clear] = R1;
         jmp  L_DeadLoop;
```

In isr.asm

```
_IRQ1:
         push R1 to [SP]

         R1 = [P_ADC];
         [P_DAC1] = R1;
         [P_DAC2] = R1;

         R1 = 0x1000;
         [P_INT_Clear] = R1;
         pop R1 from [SP];
         reti;
```

Example

This example shows how to use ramp up/down to build/release DAC DC level smoothly.

You will see the function bodies showing how to ramp up/down DAC1/DAC2 below.

```
////////////////////////////////////////////////////////
// Function: Ramp Up/Down to avoid speaker "pow" noise
// Destroy: R1,R2
////////////////////////////////////////////////////////
_SP_RampUpDAC1:    .PROC
F_SP_RampUpDAC1:
      push r1,r2 to [sp]
       r1=[P_DAC1]
       r1 &= ~0x003f
       cmp    r1,0x8000
       jb     L_RU_NormalUp
       je     L_RU_End

L_RU_DownLoop:
       call   F_Delay
       r2 = 0x0001
       [P_Watchdog_Clear] = r2
       r1 -= 0x40
       [P_DAC1] = r1
       cmp    r1,0x8000
       jne    L_RU_DownLoop
L_RD_DownEnd:
       jmp    L_RU_End

L_RU_NormalUp:
L_RU_Loop:
       call   F_Delay
       r2 = 0x0001
       [P_Watchdog_Clear] = r2
       r1 += 0x40
       [P_DAC1] = r1
       cmp    r1, 0x8000
       jne    L_RU_Loop
L_RU_End:
         pop    r1,r2 from [sp]
         retf
     .ENDP

//..............................................................
_SP_RampDnDAC1:   .PROC
F_SP_RampDnDAC1:
     push r1,r2 to [sp]
     r1 = [P_DAC1]
     r1 &= ~0x003F
     jz     L_RD_End
L_RD_Loop:
     call   F_Delay
     r2 = 0x0001
     [P_Watchdog_Clear] = r2
     r1 -= 0x40
     [P_DAC1] = r1
     jnz    L_RD_Loop
```

```
L_RD_End:
       pop    r1,r2 from [sp]
       retf
.ENDP

//.............................................................
_SP_RampUpDAC2:   .PROC
F_SP_RampUpDAC2:
    push r1,r2 to [sp]
    r1=[P_DAC2]
    r1 &= ~0x003f
    cmp    r1,0x8000
    jb       L_RU_NormalUp_
    je     L_RU_End

L_RU_DownLoop_:
    call   F_Delay
    r2 = 0x0001
    [P_Watchdog_Clear] = r2
    r1 -= 0x40
    [P_DAC2] = r1
    cmp    r1,0x8000
    jne    L_RU_DownLoop_
L_RD_DownEnd_:
    jmp    L_RU_End_

L_RU_NormalUp_:
L_RU_Loop_:
    call   F_Delay
    r2 = 0x0001
    [P_Watchdog_Clear] = r2
    r1 += 0x40
    [P_DAC2] = r1
    cmp    r1, 0x8000
    jne    L_RU_Loop_
L_RU_End_:
    pop    r1,r2 from [sp]
    retf
  .ENDP
//.............................................................
_SP_RampDnDAC2:   .PROC
F_SP_RampDnDAC2:

    push r1,r2 to [sp]
    r1 = [P_DAC2]
    r1 &= ~0x003F
    jz      L_RD_End_
L_RD_Loop_:
    call    F_Delay
    r2 = 0x0001
    [P_Watchdog_Clear] = r2
    r1 -= 0x40
    [P_DAC2] = r1
    jnz     L_RD_Loop_
L_RD_End_:
    pop     r1,r2 from [sp]
    retf
.ENDP

//..................................................................
.define C_RampDelayTime 8    // This delay time can be adjusted based on the situation
F_Delay:
      push r1 to [sp]
      r1 = C_RampDelayTime          // Ramp Up/Dn delay per step
L_D_Loop:
      r1 -= 1
      jnz     L_D_Loop
      pop     r1 from [sp]
      RETF
```

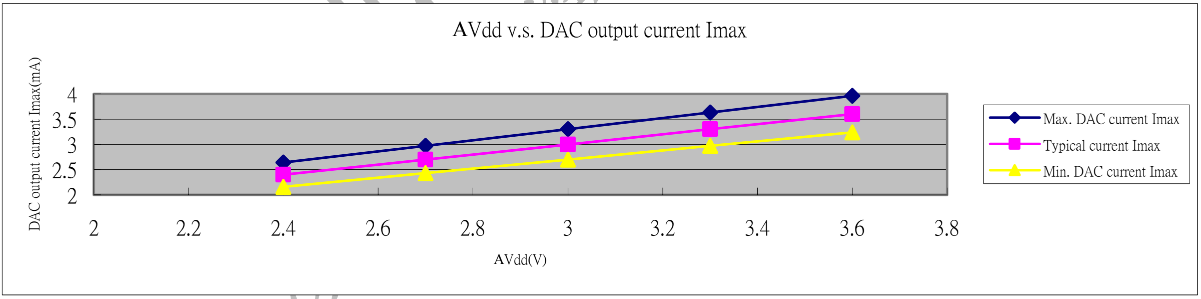
13.4 DAC output characteristics

The DACs are designed as the audio output device originally. The purpose of this section is for those applications using DAC as an analog output device. Basically the maximal DAC output current is proportional to the AVdd. Please refer to the following table "AVdd vs. DAC output current I_{MAX}", the maximal DAC output current range is "Typical current±10%". For example, when AVdd=3.0V and DAC@3mA mode, the maximal DAC output current will be in 2.7mA~3.3mA range.

Since we already know the maximal DAC output current, then the possible analog output voltage range depends on the effective loading of the DAC. But due to the DAC structure the maximal output voltage will be lower than AVdd by 0.3V~0.4V. For example, if the DAC output is in 3mA mode, when AVdd=3V, sending 0xFFC0 to P_DAC with effective loading 866Ohm then the maximal output voltage will be around 3mAX867Ohm=2.6V. If now you try to connect a loading larger than 867Ohm, the output voltage will not be able to increase accordingly due to the internal DAC structure. Please refer to the following table showing the example when AVdd=3V, the DAC maximal output voltage with various effective loading.

AVdd vs. DAC output current I_{MAX}

AVdd(V)	Min. DAC current(mA)	Typical current(mA)	Max. DAC current(mA)
2.4	2.16	2.4	2.64
2.7	2.43	2.7	2.97
3.0	2.7	3.0	3.3
3.3	2.97	3.3	3.63
3.6	3.24	3.6	3.96



DAC max. output voltage vs. Effective loading when AVdd=3V

AVdd=3V and DAC@3mA mode		AVdd=3V and DAC@2mA mode	
Effective Loading	Max output voltage (V)	Effective Loading	Max output voltage (V)
100 Ohm	0.3V	100 Ohm	0.2V
200 Ohm	0.6V	300 Ohm	0.6V
300 Ohm	0.9V	450 Ohm	0.9V
400 Ohm	1.2V	600 Ohm	1.2V
500 Ohm	1.5V	750 Ohm	1.5V
600 Ohm	1.8V	800 Ohm	1.8V
700 Ohm	2.1V	1.0K Ohm	2.1V
800 Ohm	2.4V	1.2K Ohm	2.4V
867 Ohm	(Max)2.6V	1.3K Ohm	(Max)2.6V

13.5 DAC Output RC circuit vs. frequency response

For DAC audio output, the external audio driver circuit is required. The bandwidth of the audio driver circuit can affect the perceived audio performance. User can adjust the bandwidth according to the reference table below. R is the effective resistance of the circuit and C is the effective capacitance. Please refer to SPCE 040A/060A/061A application circuit.

For example, in SPCE061 Emulation board V2.2,

R7,R11 and C22 (R12,R16 and C30) is C22(or C30) = 0.047uF for F3dB=4 KHz ; C22(or C30) = 0.01uF for F3dB=20 KHz

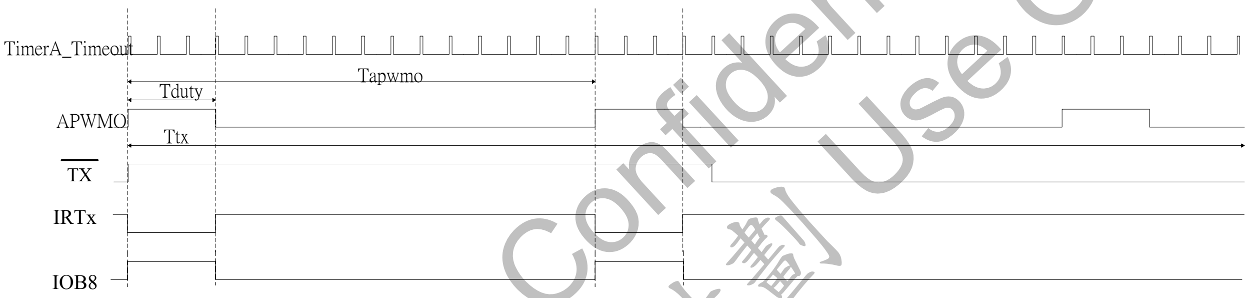
R8,R10 and C21 (R13,R15 and C29) is C21(or C29) = 0.1uF for F3dB=3 KHz ; C21(or C29) = 0.02uF for F3dB=15 KHz

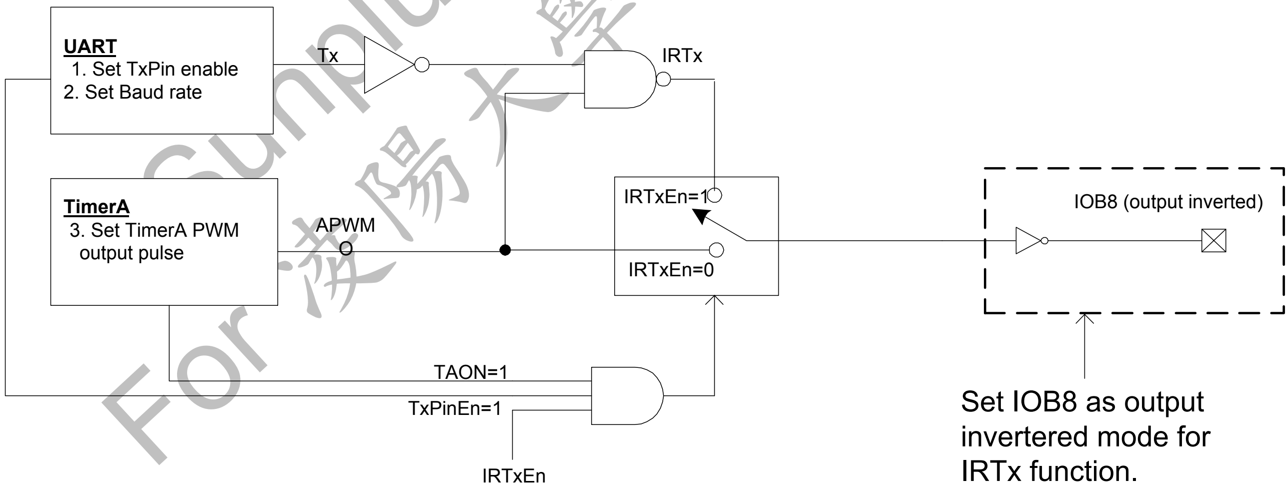
F_{3dB}(3dB frequency)		R (ohm)															
		100	120	180	220	330	390	470	510	560	680	750	820	910	1000	1100	1500
C(uF)	0.0047	338.628	282.19	188.127	153.922	102.614	86.8276	72.0485	66.3976	60.4693	49.7982	45.1504	41.2961	37.2118	33.8628	30.7843	22.5752
	0.01	159.155	132.629	88.4195	72.3432	48.2288	40.809	33.8628	31.2069	28.4205	23.4052	21.2207	19.4092	17.4896	15.9155	14.4686	10.6103
	0.02	79.5775	66.3146	36.1716	36.1716	24.1144	20.4045	15.6034	15.6034	14.2103	11.7026	9.70458	9.70458	8.74478	7.95775	5.30517	5.30517
	0.047	33.8628	28.219	18.8127	15.3922	10.2614	8.68276	7.20485	6.63976	6.04693	4.97982	4.51504	4.12961	3.72118	3.38628	3.07843	2.25752
	0.1	15.9155	13.2629	8.84195	7.23432	4.82288	4.0809	3.38628	3.12069	2.84205	2.34052	2.12207	1.94092	1.74896	1.59155	1.44686	1.06103
	0.22	7.23432	6.0286	4.01907	3.28833	2.19222	1.85495	1.53922	1.41849	1.29184	1.06387	0.96458	0.88223	0.79498	0.72343	0.65767	0.48229
	0.47	3.38628	2.8219	1.88127	1.53922	1.02614	0.86828	0.72048	0.66398	0.60469	0.49798	0.4515	0.41296	0.37212	0.33863	0.30784	0.22575
	1	1.59155	1.32629	0.88419	0.72343	0.48229	0.40809	0.33863	0.31207	0.28421	0.23405	0.21221	0.19409	0.1749	0.15916	0.14469	0.1061

🟨	The frequency range, 5K~4 KHz is for speech application and marked in yellow cells.
🟦	The frequency range, 15K~25K Hz is for Audio application and marked in blue cells..

14 IR (INFRARED) FUNCTION

The IR allows one device to transmit data to another and vice versa. The SPCE IR function is able to transmit IR data through IOB8 (IRTx) . To make the IR function working properly, IOB8 must be configured as output inverted (for IRTx). To transmit IR signal, you should set the b0 (IRTxEn) of P_FeedBack (W) ($7009H) and use the special function in IOB8 (IRTx). The IRTx signal is the combination of Tx (UART transmitter) and APWMO (TimerA PWM output). Therefore, to generate the IR transmitting output signal IRTx, the TimerA and UART transmission mode must be set properly. Then, IR transmitter signal can be sent through IOB8 where IOB8 = $\overline{\text{IRTx}}$ = $\overline{\text{Tx}}$ AND APWMO. Please refer to the ***Special Function of PortB***, ***P_FeedBack (W)($7009H)*** and ***UART*** for details.





Set IOB8 as output invertered mode for IRTx function.

15 IR (Tx) Block Diagram

LVR/LVD (LOW VOLTAGE RESET/LOW VOLTAGE DETECT)

The system may halt when power drops below 2.2V, normally caused by battery-bounce, heavy loading or low-battery condition. With the LVR functions, a reset signal is generated to reset the system when the power drops below 2.2V for 4 clock cycles.

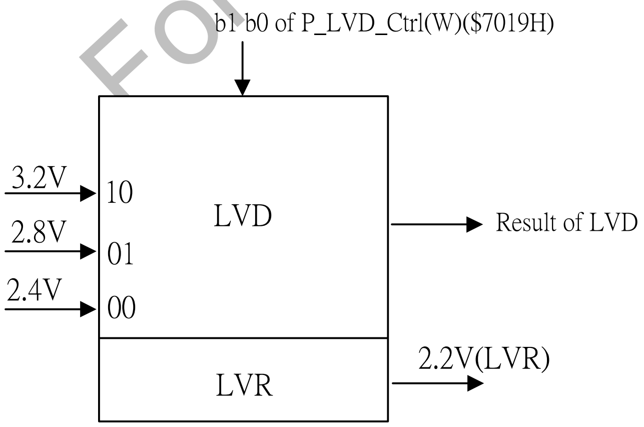
The LVD reports the circumstance of system voltage. When the system voltage (V_{cc}) drops below the selected LVD level (V_{LVD}), the LVD result-flag (b15 of P_LVD_Ctrl (R) ($7019) will be set and will be cleared to "0" when V_{cc} is greater than V_{LVD}. Three LVD levels can be programmed to specific levels by writing P_LVD_Ctrl (W)($7019H). Suppose, LVD is set to 2.8V, when the system power drops below 2.8V, the b15 of P_LVD_Ctrl(R)($7019H) returns HIGH. As a result, CPU is able to detect the low voltage condition via the programmed voltage level. The default value of LVD function is 2.4V and it can be disabled by mask option. The LVR and LVD functions will be turned off automatically in sleep mode.

15.1 P_LVD_Ctrl(R/W)($7019H)

b15	b14 - b2	b1	b0
Result of LVD	---	LVD level define(W)	

b1	b0	LVD level (V_{LVD})
0	0	2.4V* (Default)
0	1	2.8V
1	0	3.2V

b15: Result of LVD = 0 → $V_{DD} > V_{LVD}$
= 1 → $V_{DD} < V_{LVD}$



16 SERIAL INTERFACE I/O

Serial interface I/O is Sunplus serial interface, which provides a one-bit serial interface to communicate with other devices. You can use this serial interface to transmit or receive data via two I/O pins, IOB0 (SCK) and IOB1 (SDA). To make the serial I/O function work, in addition to the setting of P_SIO_Ctrl (R/W)($701EH), IOB0 (SCK) also needs to be configured as output pin by setting P_IOB_Dir ($7007H), P_IOB_Attrib ($7008H), and P_IOB_Data ($7005H).

The baud rate of serial I/O can be boosted up to (CPU CLK)/4. The default baud rate is (CPU CLK)/16. In SPCE 040A/060A/061A, the baud rate can be up to 12288 KHz. The address mode of the serial I/O can also be adjusted depending on the peripheral devices. The address mode can be 8 bits, 16 bits or 24 bits. The b5 of P_SIO_Ctrl (R/W)($701EH) can decide whether to add a control bit in front of each new read/write. The read/write control bit is ”0” for write operation and ”1” for read operation. Please refer to the read/write timing diagram below.

For users who intend to take advantage of this serial I/O function, the first step is to initialize the I/O and make sure that IOB0 is configured as an output. Next, P_SIO_Ctrl (R/W)($701EH) needs to be configured based on the user’s requirement on baud rate, address mode, read/write mode, and read/write control bit option. User then writes the necessary address to P_SID_Addr_Low (W) ($701BH), P_SID_Addr_Mid (W) ($701CH), and P_SID_Addr_High (W) ($701DH) by the setting of the addressing mode.

For write operation, users have to write P_SIO_Start (W)($701FH) to start writing session. Users write P_SIO_Data (W)($701AH) to send out the address followed by the data. Then P_SIO_Start (R)($701FH) should be read to poll the serial I/O status. When the status of serial I/O is ready, users have to write P_SIO_Stop (W)($7020H) to terminate this write session.

For read operation, users also have to write P_SIO_Start (W)($701FH) to start reading session. Users then need to read the P_SIO_Data (R)($701AH) once to send the address out and clear the read buffer. After that, users can read P_SIO_Start (R)($701FH) to poll the serial I/O status. Next, users should read the P_SIO_Data (R)($701AH) again to get the expected data from the serial I/O. When the status of serial I/O is ready, write P_SIO_Stop (W)($7020H) to terminate this read session.

Please note that reading P_SIO_Start (R)($701FH) to get the serial I/O status only reflects the status of serial I/O operation, not the status of the device connected. Users will need to take extra action to get the status of the device connected based on the interface provided by the device.

16.1 P_SIO_Ctrl(R/W)($701EH)

You have to set b7 of P_SIO_Ctrl to configure IOB0 as SCK and IOB1 as SDA. And you need to set IOB0 and IOB1 as output buffer mode . And by setting b6 to “1”, you can use SIO to transmit data out. Otherwise, if b6 set as “0”, SIO can read data from the address you specified. In addition, you can set the transmit speed with b4, b3 and the addressing bit numbers by setting b1 and b0.

b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**
SIO_Config	R/W	R/W_En	Clock select		-	Addr_select	

SIO_Config	b7	1	Set IOB0 = SCK(serial interface clock), IOB1 = SDA (serial interface data) for SIO function. IOB0 needs to be set as output. Please refer to ***I/O*** for the setting.
		0	Normal I/O(Default)
R/W	b6	1	Access SIO as write
		0	Access SIO as read
R/W_En	b5	1	No read/write control bit applied
		0	Read/write control bit applied (Default)
Clock select	b4	b3	
	0	0	(CPU CLK)/16(Default)
	0	1	(CPU CLK)/4
	1	0	(CPU CLK)/8
	1	1	(CPU CLK)/32
	b2:		N/A
Addr_select	b1	b0	Address mode select
	0	0	Address = 16 (A15 ~ A0) (Default)
	0	1	No address
	1	0	Address = 8 (A7 ~ A0)
	1	1	Address = 24 (A23 ~ A0)

Note: About the CPU CLK, refer to ***SYSTEM CLOCK*** section.

16.2 P_SIO_Data(R/W)($701AH)

This port receives or transmits one bit serial data. In SIO write mode, by writing this port, you can transmit the data byte in serial. In SIO read mode, you can get the received data byte by reading this port. Users should write to enable the P_SIO_Start ($701FH) before write/read data to/from P_SIO_Data ($701AH) from start address, P_SIO_Addr_Low, P_SIO_Addr_Mid and P_SIO_Addr_High. In SIO write mode, the first time write P_SIO_Data (W) ($701AH) right after write value to P_SIO_Start (W), the SIO will send the start address followed by the 8-bit data in P_SIO_Data (W) ($701AH). In SIO read mode, the first time read P_SIO_Data (R) ($701AH) right after write value to P_SIO_Start (W) ($701FH), the SIO will send the start address first.

b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	D2	D1	D0

16.3 P_SIO_Addr_Low(R/W)($701BH)

Serial interface lower byte (LSB) start address port (Default: 00H)

b7	b6	b5	b4	b3	b2	b1	b0
A7	A6	A5	A4	A3	A2	A1	A0

16.4 P_SIO_Addr_Mid(R/W)($701CH)

Serial interface middle byte start address port (Default: 00H)

b7	b6	b5	b4	b3	b2	b1	b0
A15	A14	A13	A12	A11	A10	A9	A8

16.5 P_SIO_Addr_High(R/W)($701DH)

Serial interface higher byte (MSB) start address port (Default: 00H)

b7	b6	b5	b4	b3	b2	b1	b0
A23	A22	A21	A20	A19	A18	A17	A16

16.6 P_SIO_Start(R/W)($701FH)

Writing any value to P_SIO_Start(W)($701FH) will enable(start) data transferring. After program writes value to P_SIO_Start (R/W) ($701FH), the next time reading/writing the P_SIO_Data (R/W) ($701AH) will lead SIO to send the read/write start address according to the P_SIO_Addr_Low, P_SIO_Addr_Mid and P_SIO_Addr_High. Then the address will not be sent again when reading/writing data from/to P_SIO_Data (R/W) ($701AH). To read/write from/to a new start address, program should stop SIO by writing value to P_SIO_Stop (W) ($7020H), set the new address to the three address ports and next write value to P_SIO_Start (W) ($701FH) to re-start SIO.

Program should read this port to acquire the transferring status of serial I/O. The bit7 of P_SIO_Start (R)($701FH) is the busy flag. When the data byte is completely transferred (read/write), the bit7 will be cleared to “0”. Program can read the bit7 to check the transferring status.

b7	b6	b5	b4	b3	b2	b1	b0
Busy	-	-	-	-	-	-	-

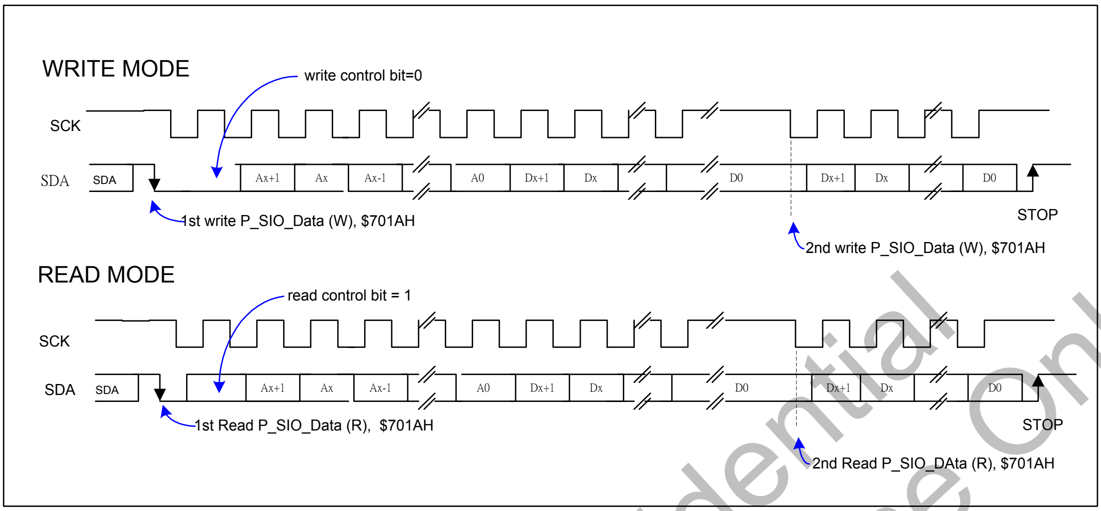
Busy(1) : Serial I/O is transferring(reading/writing)

Busy(0) : Serial I/O data byte transfer(read/write) completed. New data byte is acceptable.

16.7 P_SIO_Stop(W)($7020H)

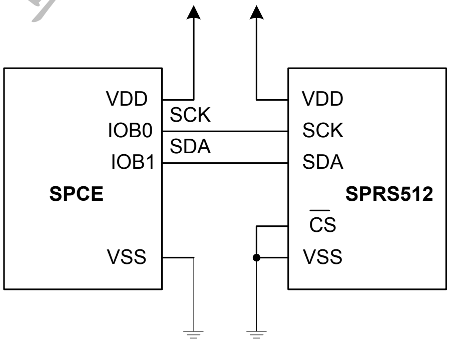
Write any value to P_SIO_Stop(W)($7020H) to stop data transferring. Usually, a stop command should be issued **before** a start command to enable data transferring. However, a stop command is not required before the first start command **after the power on reset**.

16.8 Write and Read Timing



16.9 Application Example

Ex. Communicate with SPRS512C SSRAM (Serial Static RAM) by SIO. In both write and read modes, the SSRAM will auto-increase its address counter whenever write/read to/from the start address.



Example:

This example writes 100 bytes of data to serial SRAM, reads them back from serial SRAM and compares the data sent and read. It has 16-bit addresses operating in the clock rate of CPUCLK/4 with write-bit-frame applied.

In Main.c

```
#include "sp_lib.h"

int main(){
  int Ret=0,addr,data,i,Error=0;


   // Initialize IOB0 as output for SCK
  SP_Init_IOB(0x0001,0x0001,0x0001);
  SP_Init_IOA(0xFFFF,0x0000,0xFFFF);      // Set PortA to be output buffer low

  addr=0;
  data=0x0000;
  // write 10 data
  for(i=0;i<100;i++){
       SP_SIOSendAByte(addr,data);
       addr++;
       data++;
   }
  // Read data back and compare
   addr=0;
   data=0x0000;
   // read data back and compare
   for(i=0;i<100;i++){

  Ret=SP_SIOReadAByte(addr);
  SP_Export(Port_IOA_Buffer,Ret);
  if (data!=Ret) Error++;
       data++;
       addr++;
   }

   SP_Export(Port_IOA_Buffer,Error);
  while(1){
  }
}
```

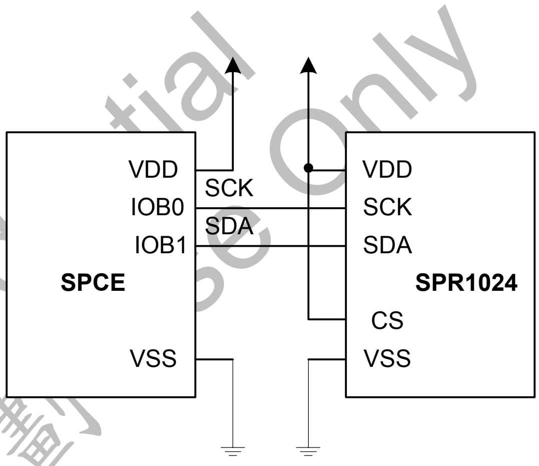
In SRAM.asm

```
.DEFINE C_SIOCLOCK         0x0008;       // CPUCLOCK/4

//////////////////////////////////////////////////////////////////
// Function: Send A Byte to Serial RAM
// Syntax: SP_SIOSendAByte(AddressLow,data)
// c level public
// Used register: r1,r2,r3
//////////////////////////////////////////////////////////////////
.public _SP_SIOSendAByte;
_SP_SIOSendAByte: .PROC
F_SIOSendAByte:
   PUSH BP,BP TO [SP];
   BP = SP + 1;
   R1 = [BP+3];
   [P_SIO_Addr_Low]=r1; // input SSRAM low address
   r1=r1 lsr 4;         // right shift 8
   r1=r1 lsr 4;
```

```
    [P_SIO_Addr_Mid]=r1; // input SSRAM mid address
    r1=0x00C0+C_SIOCLOCK;

    [P_SIO_Ctrl]=r1;       // clk=CPUclk/4, 16 bit address  ;write
    [P_SIO_Start]=r1;      // enable write mode
    R1 = [BP+4];
    [P_SIO_Data]=r1;       // state to transmit data
L_WaitSIOSendReady:
    r1=[P_SIO_Start];
    test   r1,0x0080
    jne    L_WaitSIOSendReady
    [P_SIO_Stop]=r1;       // disable write mode
    POP BP,BP FROM [SP];
    retf;
.ENDP;


//////////////////////////////////////////////////////////////////////
// Function: Read A Byte to Serial RAM
// Syntax: SP_SIOReadAByte(AddressLow)
// c level public
// Used register: r1,r2,r3
// Return register: r1
//////////////////////////////////////////////////////////////////////

.public _SP_SIOReadAByte;
_SP_SIOReadAByte: .PROC
F_SIOReadAByte:
       PUSH BP,BP TO [SP];
       BP = SP + 1;
       R1 = [BP+3];
       [P_SIO_Addr_Low]=r1;       // input SSRAM low address
       r1=r1 lsr 4;
       r1=r1 lsr 4;
       [P_SIO_Addr_Mid]=r1;       // input SSRAM mid address
       r1=0x0080+C_SIOCLOCK;
       [P_SIO_Ctrl]=r1;          // clk=CPUclk/4, 16 bit address  ;read
       [P_SIO_Start]=r1;         // enable read mode
       r2=[P_SIO_Data];          // Send address to read to SSRAM
L_WaitSIOReadReady1:             // Wait SSRAM send back the data byte
       r1=[P_SIO_Start];
       test   r1,0x0080
       jne    L_WaitSIOReadReady1
       r1=[P_SIO_Data];           //Read exact data byte
L_WaitSIOReadReady2:              //Wait next read data stop
       r2=[P_SIO_Start];
       test    r2,0x0080
       jne     L_WaitSIOReadReady2
       [P_SIO_Stop]=r2;          //disable read mode
       POP BP,BP FROM [SP];
       retf;
       .ENDP;
```

Example:

Communicate with SPR1024 (Serial Flash) by SIO. The SPR1024 is a low power 1M-bit FLASH memory device. Please aware that the SPR 1024 requires the supply voltage range from 2.7V ~ 3.6V. Therefore voltage adaptation circuit is required if the I/O voltage is other than this range.



Example:
It is the function body of F_SIOSendAByte. R1 is the address low, R2 is the address high and R3 is the data to send.

```
F_SIOSendAByte:
    PUSH BP,BP TO [SP];
    BP = SP + 1;
// Write address
R1 = [BP+3];
    [P_SIO_Addr_Low]=r1;         // input Serial flash low address
    r1=r1 lsr 4;                // right shift 8
    r1=r1 lsr 4;
    [P_SIO_Addr_Mid]=r1;        // input Serial flash mid address
    R1 = [BP+4];               // Port direction
    r1=r1&0x0007;              // input Serial flash hi address
    [P_SIO_Addr_High]=r1;
//Set SIO control Port
    r1=0x00D3;
    [P_SIO_Ctrl]=r1;          // clk=CPUclk/8, 24 bit address  ;write
    //Send SIO Data
    [P_SIO_Start]=r1;         // enable write mode
    R1 = [BP+5];
    [P_SIO_Data]=r1;          // state to transmit data
L_WaitSIOSendReady:
    r1=[P_SIO_Start];        // Polling the SIO status
    test   r1,0x0080
    jne    L_WaitSIOSendReady
    [P_SIO_Stop]=r1;         // disable write mode
    POP BP,BP FROM [SP];
    retf;
   .ENDP;
```

It is the function body of F_SIOReadAByte. R1 is the address low, R2 is the address high and the returned data will be store in R1.

```
F_SIOReadAByte:
       PUSH BP,BP TO [SP];
       BP = SP + 1;
       R1 = [BP+3];
       // Write Address
       [P_SIO_Addr_Low]=r1;   // input Serial flash low address
       r1=r1 lsr 4;
       r1=r1 lsr 4;
       [P_SIO_Addr_Mid]=r1;  // input Serial flash mid address
       R1 = [BP+4];           // Port direction
       r1=r1&0x0007;         // input Serial flash hi address
       [P_SIO_Addr_High]=r1;
       // Set SIO control Port
       r1=0x0093;
       [P_SIO_Ctrl]=r1;       // clk=CPUclk/8, 24 bit address  ;read
       // Read SIO data
       [P_SIO_Start]=r1;      // enable read mode and send address
       r2=[P_SIO_Data];       // Send address to read to serial flash
L_WaitSIOReadReady1:
       r1=[P_SIO_Start];      // Wait Serial flash send back the data byte
       test   r1,0x0080
       jne    L_WaitSIOReadReady1
       r1=[P_SIO_Data];        // Read exact data byte
L_WaitSIOReadReady2:           // Wait next read data stop
r2=[P_SIO_Start];
       test   r2,0x0080
       jne    L_WaitSIOReadReady2
       [P_SIO_Stop]=r2;         // disable read mode
     POP BP,BP FROM [SP];
       retf;
  .ENDP
```

17 UART

UART block provides a full duplex standard interface to communicate with other devices. With this interface, SPCE040A/060A/061A is able to transmit and receive data simultaneously. It is also capable of receiving buffered meaning that it can receive the next data before the previous received one being read from the register buffer. However, if the previous received data still has not been read by the time of the next data is received completely, the previous-data is lost. The P_UART_Data (W/R)($7023H) is the data access port for both transmitting and receiving. Writing data to P_UART_Data (W) will load the data transmitting a register. Reading data from P_UART_Data (R) accesses a separated received data register. The UART function can be accomplished by using I/O Port B and INT(UART IRQ). The Rx and Tx of UART are shared with IOB7 and IOB10. UART receives data through the UART receiver-pin, IOB7(Rx).

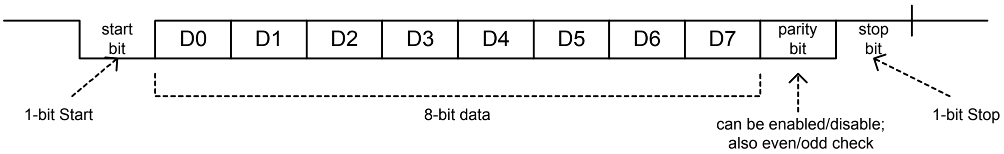
To make UART works properly, IOB7 must be configured as input (for Rx) and/or IOB10 as output (for Tx). Please refer to ***I/O*** for details. Then, users can set the baud rate through P_UART_BaudScalarLow(R/W)($7024H) and P_UART_BaudScalarHigh(R/W)($7025H). The default setting is 01H/00H. For SPCE 040A/060A/061A , the default baud rate is 48000 bps. Also, usershave to set the P_UART_Command1 (W)($7021H) and P_UART_Command2 (W)($7022H) properly to enable UART the receiving and/or transmitting function. UART will be activated when the setup is completed.

If users enable the UART IRQ by setting b7 and/or b6 of P_UART_Command1, the UART Rx and/or Tx event detection through IRQ will be activated. The UART IRQ is triggered by RxRDY and/or TxRDY. When *μ'nSP*® receives and/or transmits a byte of data, the b7 and/or b6 in P_UART_Command2 (R)($7022H) will be set to "1" and the program will enter UART IRQ at the same time. To process UART data, users can read b7 (RxRDY) and/or b6 (TxRDY) of P_UART_Command2 (R)($7022H) to distinguish the data receiving and transmitting event. Users can then process data according to the state of b7 (RxRDY) and/or b6 (TxRDY) in P_UART_Command2 (R) ($7022H) while UART IRQ is activating. And, the UART IRQ status should be cleared by reading the P_UART_Command2 (R)($7022H) when users are done with the processing.

Meanwhile, FIQ and IRQ1~6 use P_INT_Ctrl (R)($7010H) to distinguish the state of interrupts and use P_INT_Clear(W)($7011H) to clear interrupt status. Please be aware that even though the UART IRQ is not activated through P_INT_Ctrl (W)($7010H). The UART is still an IRQ type (IRQ 7) interrupt and therefore it will be turned on or off by the INT instruction.

Note: The UART IRQ interrupt vector is located in $FFFFH and is with the lowest priority among all IRQs.

17.1 UART Data Frame Format



17.2 P_UART_Command1(W)($7021H)

An UART control port. You can control the parity check function by setting b2 and b3. Setting either b6 to "1" or b7 to "1" will enable the UART IRQ. The difference is if you set b7 to "1", the UART IRQ will be triggered by the RxRDY, i.e. when data byte is received. If you set b6 to "1", the UART IRQ will be triggered by the TxRDY, i. e. when data byte is transferred completely. Setting b5 to "1" will reset the UART block and will make all the UART control/status registers to the default value.

b7	b6	b5	b4	b3	b2	b1	b0
RxIntEn	TxIntEn	I_Reset	-	Parity	P_Check	-	-

Default = $00

RxIntEn	b7	UART IRQ enable (Receiver Triggered)	1: enable 0: disable
TxIntEn	b6	UART IRQ enable (Transmitter Triggered)	1: enable 0: disable
I_Reset	b5	Internal Reset	1: reset 0: non-reset
Parity	b3	Parity check	1: even 0: odd
P_Check	b2	Parity check enable	1: enable 0: disable

17.3 P_UART_Command2(W)($7022H)

An UART Rx/Tx function enable control port. Set b7 and/or b6 to enable the Rx and/or Tx function.

b7	b6	b5	b4	b3	b2	b1	b0
RxPinEn	TxPinEn	-	-	-	-	-	-

Default = $00

RxPinEn	b7	Receiving Pin Enable	1: enable 0: disable
TxPinEn	b6	Transmitting Pin Enable	1: enable 0: disable

To receive and transmit data, these bits must be enabled correspondingly. Also, remember to set IOB7 as input for Rx and/or IOB10 as output for Tx. The IOB10 Tx output will be set to high automatically whenever the transmitting pin is enabled by setting b6 (TxPinEn=1).

17.4 P_UART_Command2(R)($7022H)

An UART function status port. The b7 is the RxRDY flag which will be set to “1” when data is received. The b6 is the TxRDY flag, which indicates data is transmitted completely. After enabling the transmitter function by setting b6 ($7022H) to “1”, the TxRDY will be set to “1” immediately. This means the transmitter’s data buffer is empty and it is ready to transmit data by writing data to P_UART_Data (W) ($7023H). When the UART RxRDY or TxRDY is set, it will be cleared whenever reading data from P_UART_Data (RxRDY) or writing data to P_UART_Data(TxRDY). The b3, b4 and b5 are error signals, which show the corresponding errors generated during the UART communication process. They will be cleared whenever reading data from P_UART_Data (R) ($7023H).

b7	b6	b5	b4	b3	b2	b1	b0
RxRDY	TxRDY	FE	OE	PE	-	-	-

RxRDY	b7	Data is received by UART	1: received 0: not received
TxRDY	b6	Transmitter is ready	1: ready 0: not ready
FE	b5	Frame Error	1: error 0: no error
OE	b4	Overrun Error	1: error 0: no error
PE	b3	Parity Check Error	1: error 0: no error

The error signals express the transmission errors. The reasons for why these errors occurred and how to improve the transmission are listed in the following table.

Error Type	Reason	Improve Solution
FE (Frame Error)	1. The frame format between Tx and Rx is inconsistent. 2. The baud rate of Tx and Rx is inconsistent.	1. Change the frame format. 2. Adjust the baud rate.
OE (Overrun Error)	The Rx data receiving is slower than the Tx data transmitting and causing the data overrun.	1. Improve the speed of Rx data receiving efficiency. 2. Slow the data transmission baud rate.
PE (Parity Check Error)	The transmission environment is not good and may present in the noise interference.	Improve the transmission environment.

17.5 P_UART_Data(R/W)($7023H)

Read / Write this port to get / send data from/to UART buffer. The reading of this port will clear the RxRDY and the writing of this port will clear the TxRDY. The data length is one byte long.

b7	b6	b5	b4	b3	b2	b1	b0
Data							

17.6 P_UART_BaudScalarLow(R/W)($7024H)

See $7025H for setup

17.7 P_UART_BaudScalarHigh (R/W) ($7025H)

In $7025H, the available bits are [4:0] only. To get a desired baud rate, write the corresponding value to the Port $7024H and $7025H according to the following Baud Rate equation.

For Fosc = 49.152MHz, 40.960 MHz or 32.768MHz.

Baud Rate = (Fosc / 4) / Scale

For Fosc = 24.576MHz, or 20.480MHz.

Baud Rate = (Fosc / 2) / Scale

where scale = (Fosc /4) / Desired Baud rate,

(Fosc /2) / Desired Baud rate

(scale is assigned to $7025H, $7024H)

Fosc = 20.480MHz~49.152MHz, depends on b5~b7 of P_SystemClock(W)($7013H)

For 49.152MHz and 24.576 MHz, the scales are the same.

For 40.960MHz and 20.480 MHz, the scales are the same.

The following table lists some usual values for the baud rates.

Baud Rate table

Fosc = 24.576 Mhz or 49.152Mhz

Baud Rate	High Byte (Hex)	Low Byte (Hex)	Scale (Hex)	Actual Baud Rate
1500**	1F	FF	1FFF	1500
2400	14	00	1400	2400
4800	0A	00	0A00	4800
9600	05	00	0500	9600
19200	02	80	0280	19200
38400	01	40	0140	38400
48000	01	00	0100*	48000
51200	00	F0	00F0	51200
57600	00	D5	00D5	57690
102400	00	78	0078	102400
115200**	00	6B	006B	114841

Fosc = 40.960 Mhz or 20.480 Mhz

Baud Rate	High Byte (Hex)	Low Byte (Hex)	Scale (Hex)	Actual Baud Rate
1500	1A	AB	1AAB	1500
2400	10	AB	10AB	2400
4800	08	55	0855	4801
9600	04	2B	042B	9597
19200	02	15	0215	19212
38400	01	0B	010B	38352
40000	01	00	0100*	40000
48000	00	D5	00D5	48075
51200	00	C8	00C8	51200
57600	00	B2	00B2	57528
102400	00	64	0064	102400
115200	00	59	0059	115056

Fosc = 32.768 Mhz

Baud Rate	High Byte (Hex)	Low Byte (Hex)	Scale (Hex)	Actual Baud Rate
1500	15	55	1555	1500
2400	0D	55	0D55	2400
4800	06	AB	06AB	4799
9600	03	55	0355	9604
19200	01	AB	01AB	19185
32000	01	00	0100*	32000
38400	00	D5	00D5	38460
48000	00	AB	00AB	47906
51200	00	A0	00A0	51200
57600	00	8E	008E	57690
102400	00	50	0050	102400
115200	00	47	0047	115380

*Default: 0100H

** The minimum baud rate is 1500bps and the maximum baud rate is 115200bps @ 49.152MHz.

Example:

Use UART to receive data from PC RS232 COM port. Because UART receives only eight bits of data each time, it saves data to 16-bit SRAM after two 8-bit data received.

```
//Set IOB10 as Output, IOB7 as Input floating
       R1=0x0480;
       [P_IOB_Attrib]=R1;
       R1=0x0400;
       [P_IOB_Dir]=R1;
//Set Baud Rate = 12.288MHz/107 = 114.84KHz (the frequency closest to 115.2KHz)
       R1=0x6B;
       [P_UART_BaudScalarLow]=R1;
       R1=0x00;
       [P_UART_BaudScalarHigh]=R1;
//Set UART_command and UART_command2
       R1=0xC0;
       [P_UART_Command1]=R1;
       [P_UART_Command2]=R1;
//Main Program
```

```
 L_begin_loop:
       R4=0;                 //The start address of the received data queue
 L_loop:
       R2=0;
       R2=R2 LSL 4;          // Clear shift buffer
       CALL F_UART_RECV;
       R1=R1 LSL 4;          // Shift left 8 bits
       R1=R1 LSL 4;
       R3=R1;
       R3=R3 and 0xFF00;     //R3 = MSB 8 bits
       R2=0;
       R2=R2 LSL 4;          // Clear shift buffer
       CALL F_UART_RECV;
       R1=R1 and 0x00FF; //R1 = LSB 8 bits
       R1=R1 or R3;          //R1=Final data word received from UART
       [R4++]=R1;            // Store the data word into SRAM data queue
       CMP R4,0x800;         // It has 2048 words in SRAM data queue
       JNE L_loop;
       JMP L_begin_loop;
// UART sub-routine
F_UART_RECV:
  PUSH R2,R3 to [SP];
 L_RxRDY:
           R2= 0X0080;            // Check if RxRDY=1
TEST R2, [P_UART_Command2];
       JZ  L_RxRDY
       R1 = [P_UART_Data];
POP R2,R3 from [SP];
  RETF;                  // Return to main program
```

18 Flash Access

SPCE061A is a MTP chip featuring its Flash instead of the mask ROM. The flash size is 32K word (32K*16 bits). User can take advantage of this internal flash to store user application data if the space allows. It may eliminate the need for extra external memory storage in some cases. For safety, the mass erase function is not open to users.

In order to access the internal flash, users have to write 0xAAAA to P_Flash_Ctrl (W) ($7555H) to enable the flash access first. Then user must write 0x5511 to P_Flash_Ctrl (W) ($7555H) for the page erase or write 0x5533 to P_Flash _Ctrl (W) ($7555H) for program Flash , Those instructions can't be intervened by any other operation, interrupts or ICE step-by-step trace function . It is because the flash controller must ensure that the state is correct to proceed with programming the internal flash. If some other procedure breaks in the sequence, the state will change. If the state is changed, then the page erase /program will not succeed.

Further, user also has to erase the page first before programming it. It is to ensure the success of programming. The page size is 0x100. The first page is [0x8000~0x80FF]. The last page is [0xFF00~0xFFFF]. However, user is advised to avoid using [0xFC00~0xFFFF] due to the system reservation.

For page erase, after user finish sending the page erase instruction, user needs to writes any value to any address in the specified page to erase, e.g. write 0x0001 to 0x85xx to erase page 5. For details please see the example.

For program, after sending the program instructions, user just needs to write the data to the address word by word e.g. write 0x0100 to 0x8500.

In SPCE061A the Flash address is from $008000 to $00FFFF. The interrupt vector area is located at $0x00FFF5 ~ 0x00FFFF. Users' discretion is required for handling the memory allocation. There is a potential hazard of damaging the code integrity when using this function since the code also resides in the same flash.

P_Flash_Ctrl(W)($7555H)

Write instructions into the flash controller register.

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W																

Instructions to write:

0xAAAA : Enable flash access

0x5511 : Page erase

0x5533 : Flash program

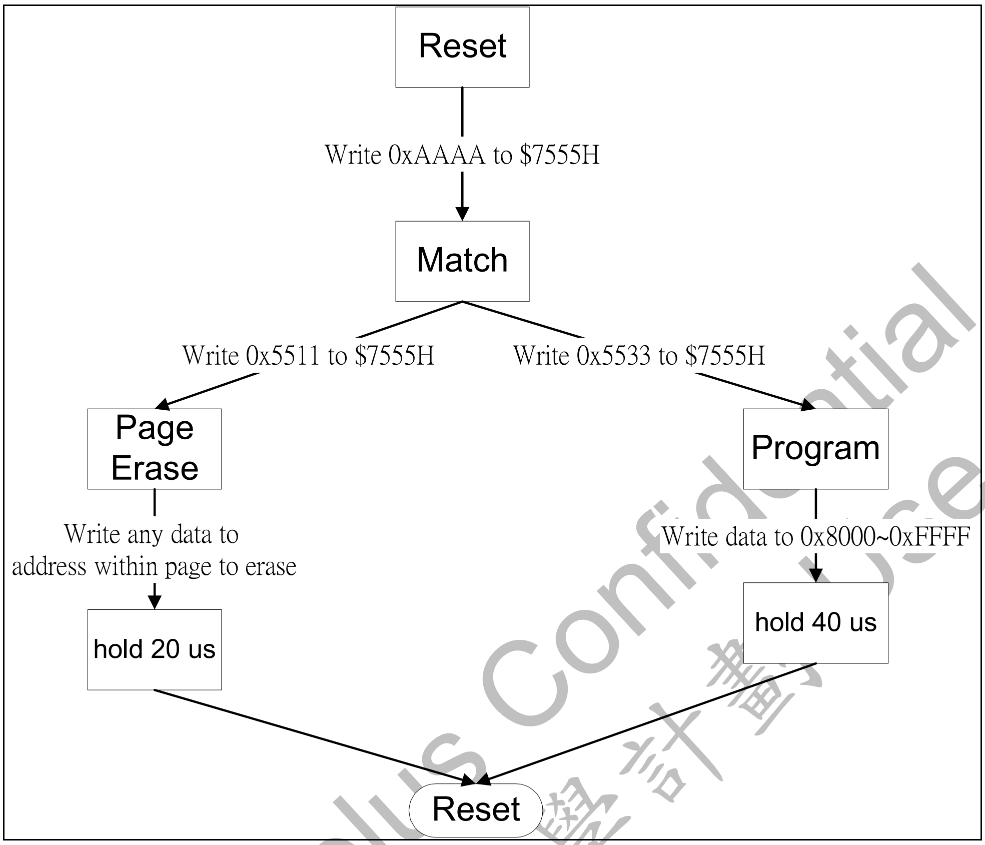


Fig: State diagram of page erase/program

Example 1

This example shows user how to page-erase, program and read the internal flash of SPCE061.

```
.DEFINE P_Flash_Ctrl 0x7555
_main:
INT OFF;
// Flash page erase
R1 = 0xAAAA;
[P_Flash_Ctrl] = R1;
R1 = 0x5511;
[P_Flash_Ctrl] = R1;
[0x8500] = R1;     // erase page 5, write any value to 0x85xx

// Flash program
```

```
R1 = 0xAAAA;
[P_Flash_Ctrl] = R1;
R1 = 0x5533;
[P_Flash_Ctrl] = R1;
R1=0x0100;
[0x8500] = R1;   // write 0x0100 to 0x8500

// Flash read
       R2 = [0x8500];
```

19 HARDWARE SUMMARY

P_IOA_Data(R/W)($7000H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOA_Buffer(R/W)($7001H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOA_Dir(R/W)($7002H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOA_Attrib(R/W)($7003H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

P_IOA_Latch(R)($7004H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOB_Data(R/W)($7005H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOB_Buffer(R/W)($7006H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOB_Dir(R/W)($7007H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_IOB_Attrib(R/W)($7008H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 0000 0000b)

P_FeedBack(W)($7009H)

b15 – b4	b3	b2	b1	b0
---	FBKEN3	FBKEN2	---	IRTxEN

(Default: xxxx xxxx xxxx 00x0b)

P_TimerA_Data(R/W)($700AH)

	b15	b14	b13	B12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	Timer A Initial Data															

(Default: 0000 0000 0000 0000b)

P_TimerA_Ctrl(W)($700BH)

b15 - b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	Output_pulse_ctrl				Source B select bits			Source A select bits		

(Default: xxxx xx00 0011 0110b)

P_TimerB_Data(R/W)($700CH)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R/W	Timer B Initial Data															

(Default: 0000 0000 0000 0000b)

P_TimerB_Ctrl(W)($700DH)

b15 - b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
---	Output_pulse_ctrl				---			Source C select bits		

(Default: xxxx xx00 00xx x110b)

P_Timebase_Setup(W)($700EH)

b15- b4	b3	b2	b1	b0
---	TMB2 Frequency select		TMB1 Frequency select	

(Default: xxxx xxxx xxxx 0000b)

P_Timebase_Clear(W)($700FH)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
W																

(Default: 0000 0000 0000 0000b)

P_INT_Ctrl(R/W)($7010H)

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

(Default: 0000 0000 0000 0000b)

P_INT_Clear(W)($7011H)

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

(Default: 0000 0000 0000 0000b)

P_INT_Mask(R/W)($702DH)

b7	b6	b5	b4	b3	b2	b1	b0
IRQ3_KEY	IRQ4_4KHz	IRQ4_2KHz	IRQ4_1KHz	IRQ5_4Hz	IRQ5_2Hz	IRQ6_TMB1	IRQ6_TMB2

b15	b14	b13	b12	b11	b10	b9	b8
FIQ_PWM	IRQ0_PWM	FIQ_TMA	IRQ1_TMA	FIQ_TMB	IRQ2_TMB	IRQ3_EXT2	IRQ3_EXT1

(Default: 0000 0000 0000 0000b)

P_Watchdog_Clear(W)($7012H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
W																

(Default: 0000 0000 0000 0001b)

P_SystemClock(W)($7013H)

b15-b8	b7	B6	b5	b4	b3	b2	b1	b0
-----	Fosc2	Fosc1	Fosc0	32KHz Sleep Status	32KHz Mode	CPU Clock		

(Default: xxxx xxxx 0001 0011b)

P_ADC(R/W)($7014H)

b15 – b6	b5 – b0
DAR0(R/W)	---

(Default: 0000 0000 000x xxxxb)

P_ADC_Ctrl(R/W)($7015H) b15	b14	b13~b9	b8	b7	b6	b5~b3	b2	b1	b0
RDY (R)	---	---	V2VREFB (W)	VEXTREF (W)	DAC_OUT (W)	---	AGCE (W)	MIC_ENB(W)	ADE(W)

(Default: 1xxx xxx1 00xx x000b)

P_ADC_MUX_Ctrl(R/W)($702BH)

b15	b14	b13-b3	b2	b1	b0
Ready(R)	Fail(R)	---	Channel select(R/W)		

(Default: 11xx xxxx xxxx x000b)

P_ADC_LINEIN_Data(R)($702CH)

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

(Default: 0000 0000 000x xxxxb)

P_DAC2(R/W)($7016H)

b15 – b6	b5 – b0
DA2_Data(R/W)	---

(Default: 0000 0000 00xx xxxxb)

P_DAC1(R/W)($7017H)

b15 – b6	b5 – b0
DA1_Data(R/W)	---

(Default: 0000 0000 00xx xxxxb)

P_DAC_Ctrl (W)($702AH)

b2	b1	b0
---	DAC Enable	---

b15 - b9	b8	b7	b6	b5	b4	b3
---	DAC1_Latch(W)		DAC2_Latch(W)		AD_Latch(W)	

(Default: xxxx xxx0 0000 0x0xb)

P_LVD_Ctrl(R/W)($7019H)

b15	b14 - b2	b1	b0
Result of LVD	---	LVD level define(W)	

(Default: 0xxxx xxxx xxxx xx00b)

P_SIO_Ctrl(R/W)($701EH)

b7	b6	b5	b4	b3	b2	b1	b0
SIO_Config	R/W	R/W_En	Speed select		-	Addr_select	

(Default: 0000 0000 0000 0x00b)

P_SIO_Data(R/W)($701AH)

b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	D2	D1	D0

(Default: xxxx xxxx 0000 0000b)

P_SIO_Addr_Low(R/W)($701BH)

b7	b6	b5	b4	b3	b2	b1	b0
A7	A6	A5	A4	A3	A2	A1	A0

(Default: xxxx xxxx 0000 0000b)

P_SIO_Addr_Mid(R/W)($701CH)

b7	b6	b5	b4	b3	b2	b1	b0
A15	A14	A13	A12	A11	A10	A9	A8

(Default: xxxx xxxx 0000 0000b)

P_SIO_Addr_High(R/W)($701DH)

b7	b6	b5	b4	b3	b2	b1	b0
A23	A22	A21	A20	A19	A18	A17	A16

(Default: xxxx xxxx 0000 0000b)

P_SIO_Start(R/W)($701FH)

b7	b6	b5	b4	b3	b2	b1	b0
Busy	-	-	-	-	-	-	-

(Default: xxxx xxxx 0xxx xxxxb)

P_SIO_Stop(W)($7020H)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
W																

(Default: 0000 0000 0000 0000b)

P_UART_Command1(W)($7021H)

b7	b6	b5	b4	b3	b2	b1	b0

RxIntEn	TxIntEn	I_Reset	-	Parity	P_Check	-	-

(Default: xxxx xxxx 000x 00xxb)

P_UART_Command2(W)($7022H)

b7	b6	b5	b4	b3	b2	b1	b0
RxPinEn	TxPinEn	-	-	-	-	-	-

(Default: 00xx xxxx xxxx xxxxb)

P_UART_Command2(R)($7022H)

b7	b6	b5	b4	b3	b2	b1	b0
RxRDY	TxRDY	FE	OE	PE	-	-	-

P_UART_Data(R/W)($7023H)

b7	b6	b5	b4	b3	b2	b1	b0
Data							

(Default: xxxx xxxx 0000 0000b)

P_UART_BaudScalarLow(R/W)($7024H)

b7	b6	b5	b4	b3	b2	b1	b0
Low Byte							

(Default: xxxx xxxx 0000 0000b)

P_UART_BaudScalarHigh(R/W)($7025H)

b7	b6	b5	b4	b3	b2	b1	b0
High Byte							

(Default: xxxx xxxx 0000 0000b)

P_Flash_Ctrl(W)($7555H) (SPCE061A only)

	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
W																

(Default:0000 0000 0000 0000b)

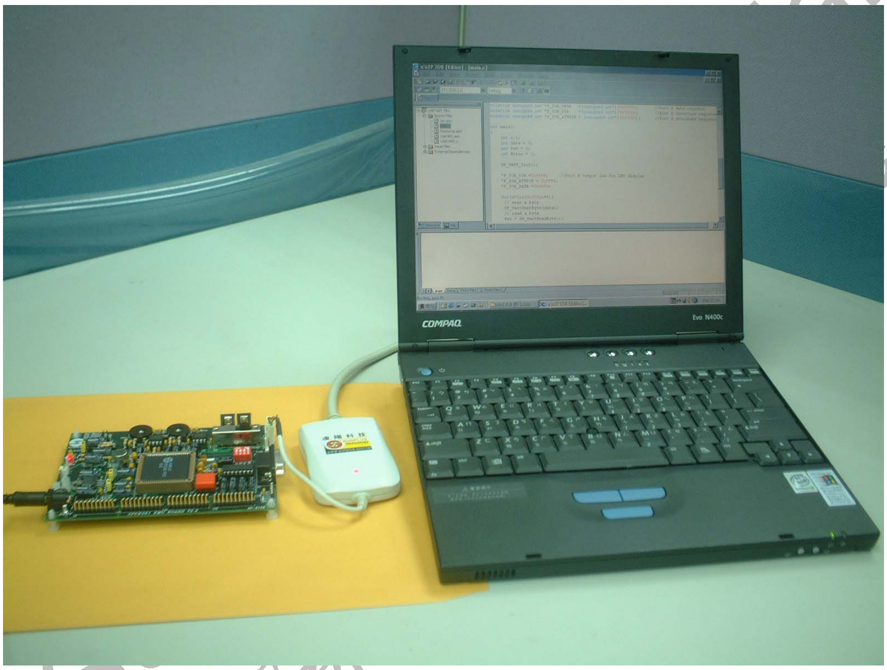
20 SPCE040A/060A/061A Development System

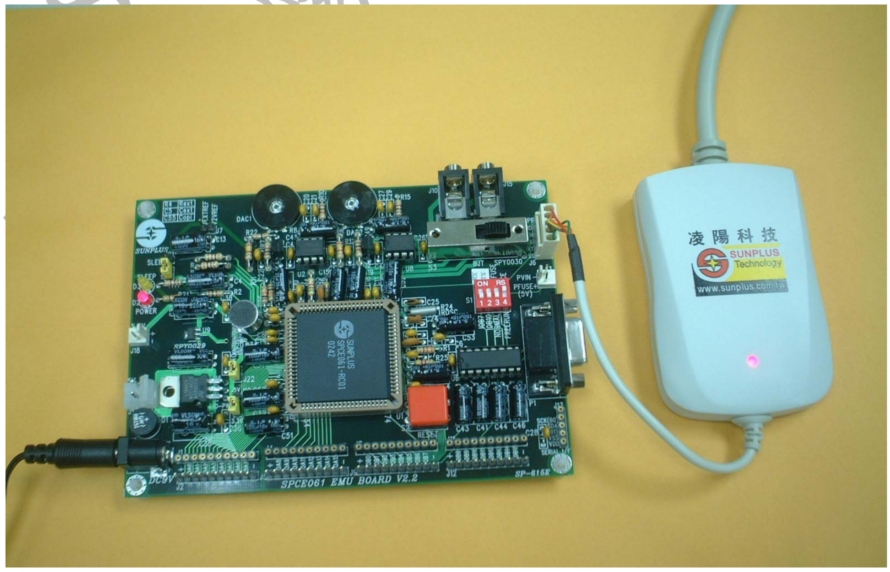
20.1 Development system Setup diagram

The SPCE040A/060A/061A development system contains:

1. Hardware: SPCE061 Emulation board and Sunplus probe
2. Software: *µ'nSP®* IDE(MS-Windows based Integrated Development Environment on PC)

Connection: PC parallel port → Sunplus Probe → SPCE061 EMU board

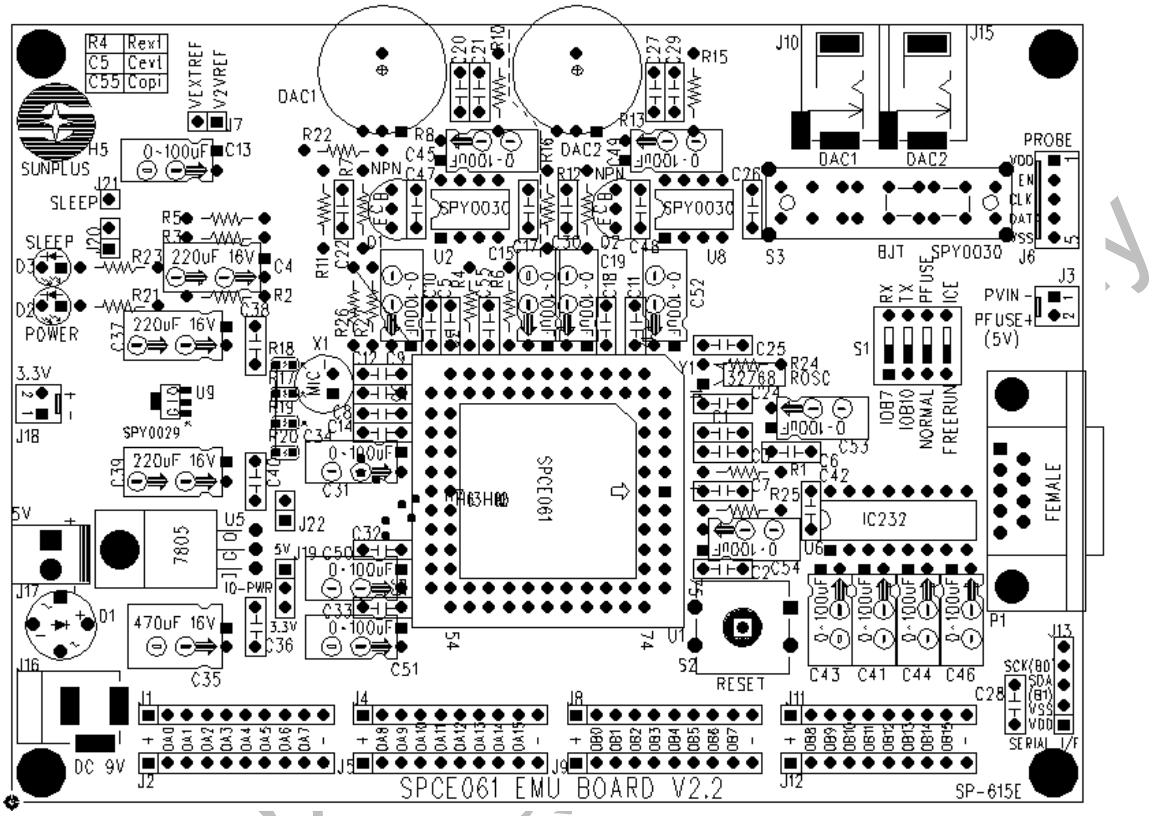




Watchdog option:

1. Since watchdog on/off is a bonding option for SPCE040A/060A/061A and is decided at bonding time. Thus we provide two kinds of SPCE061A package types for convenience. One is bonded as watchdog on and the other is watchdog off.
2. In the SPCE061 emulation board set, user will see 2 package chips marked as “WDOG ON” and “WDOG OFF”
   WDOG ON : Watchdog option is enabled
   WDOG OFF : Watchdog option is disabled
3. For project development needs, user has to change the SPCE061A chips on emulation board to change the specific watchdog condition before starting the project development.

20.2 SPCE040A/060A/061A EMU board v2.2



SPCE040A/060A/061A EMU Board Quick setup guide

1. Check the jumpers and switches on SPCE040A/060A/061A EMU board.

   J20: Short,

   J22: Short,

   J19: one of 5V and 3.3V must be short.

   S1: IOB7, IOB10, Normal and ICE are selected.

2. Power on the SPCE061 EMU board through one of the following jacks

   J16 (DC9V from adapter),

   J17 (5V from power supply),

   J18 (3.3V from power supply).

3. Connect SPCE040A/060A/061A EMU board via Sunplus probe to PC LPT1 port.
4. Launch *µ'nSP®* IDE on PC, load project and start developing the project. Make sure that the IC body is chosen as SPCE040A or SPCE060A/061A correctly.

Jumper / Switch Setup

Switch/Jump Name	Description	Note
D2	POWER LED.	
D3	SLEEP LED.	Light up when in sleep.
J1, J2	IOA0~IOA7	
J4, J5	IOA8~IOA15	
J8, J9	IOB0~IOB7	
J11, J12	IOB8~IOB15	
J3	Fuse pins for security option. Connect PFUSE to +5V and PVIN to GND for 1 second to burn out the fuse.	Please also refer to the S1 dip-switch(S1-3).
J6	ICE interface connector.	Connect to PC parallel port through Sunplus probe for program download.
J7	V2VREF: The 2V voltage output pin of the internal 2V regulator. VEXTREF: The external top reference voltage input signal pin for A/D Line_In.	
J10	AUD1(DAC1) output jack.	
J15	AUD2(DAC2) output jack.	
J13	SIO serial interface connector.	SCK=IOB0, SDA=IOB1, VDD=VDDIO.
J16	Power input for power adapter.	9V ~ 12V > 500 uA.
J17	Power input for 5V level power supply. It will further go through the SPY0029 to generate 3.3V to the VDD.	
J18	Power input for 3V level power supply.	
J19	Jumper switch for I/O's power supply VDDIO.	Set VDDIO=5V level or VDDIO=3V level.
J20	SLEEP LED jumper. Remove jumper to disable the SLEEP LED D3.	
J21	SLEEP signal pin, high means IC is sleeping.	
J22	Jumper between the output of 7805 and the input of SPY0029	Usually connected. Remove this jumper only if you want to feed VDD(to J18) and VDDIO(to J17) with different voltage inputs.
P1	UART(PC RS232 Com port connector).	Also refer to the S1 dip-switch.
R8	Adjust AUD1 volume when S3 is in SPY0030.	
R13	Adjust AUD2 volume when S3 is in SPY0030.	
S2	RESET key	
S3	Transistor/SPY0030 audio driver output selection	
U2, U8	SPY0030 SUNPLUS audio driver	
U9	SPY0029 SUNPLUS voltage regulator.	Bonding optioned as 3.3V output
X1	MIC input	

S1	ON (Pull Up)	OFF (Push Down)
S1-1	RX: Connect IOB7 through U6 (IC232) to P1's pin2	IOB7: Set IOB7 as normal I/O
S1-2	TX: Connect IOB10 through U6 (IC232) to P1's pin3	IOB10: Set IOB10 as normal I/O
S1-3	PFUSE: Connect IC's PFUSE pin to J3	NORMAL: Disconnect PFUSE to J3
S1-4	ICE	FREERUN

- Switch to FREERUN when SPCE chip runs internally and disconnect s with ICE.
- To burn out the security fuse, switch S1-3 to PFUSE and then connect +5V to J3 PFUSE and GND to J3 PVIN.

20.3 SPCE061 code downloading application circuit

SPCE061A is a MTP type MCU, featuring its flash ROM. The purpose of this section is to show how user can use a minimal set of pins on SPCE061A to download program code onto the SPCE061A flash ROM. In addition, user can opt to "burn" the fuse to activate the security function by supplying an additional 5V power between PFUSE and PVIN after the code is successfully downloaded. The code in SPCE061A flash ROM will become executable-only.

In order to launch a download session, the SPCE061A needs to be connected to *μ'nSP®* IDE via ICE through the Sunplus probe. Moreover, some parts on SPCE061A also need to be activated such as its PLL for system clock, logic circuit and I/O. The ICE on SPCE needs to be connected to *μ'nSP®* IDE via ICESDA, ICESCK and ICE pin. The VDD and VSS on the probe cable need to be supplied from an external power. Furthermore, after downloading program to SPCE061A and unplug the Sunplus probe, the ICESCK pin should be connected to ground through 100Kohm resistor. Otherwise there will be a current leakage during sleep mode due to the floating status of the ICESCK pin.

The voltage operating range on I/Os is determined by I/O power, VDDIO and VSSIO. The valid voltage operating range on I/Os can be set via VDDIO as user's wish. For simplicity of downloading only, user can use 3.3V for both VDD and VDDIO.

Minimal set of pins required for SPCE061A downloading: 6 pads
(VDD 3.3V, VSS: GND, VDDIO: 3.3V or 5V)

For Probe

Pad 5	:	VDD
Pad 7	:	ICE
Pad 8	:	ICESCK
Pad 9	:	ICESDA
Pad 10	:	VSS

For I/O power supply

Pad 56: VDDIO

The pads above forms the minimal set necessary for program download.

20.4 Security option

In addition, the security option is also for user convenience. In order to make flash executable-only, user only needs to connect 2 pins, PVIN and PFUSE to 5V for about 1 second after the code downloading is complete. It is not necessary for any other pins to be connected while burning. After the fuse is burned, the SPCE061A chip will no longer be readable or programmable and only executable.

Flash burning

Pad 20: PFUSE (5V)

Pad 11: PVIN (0V)

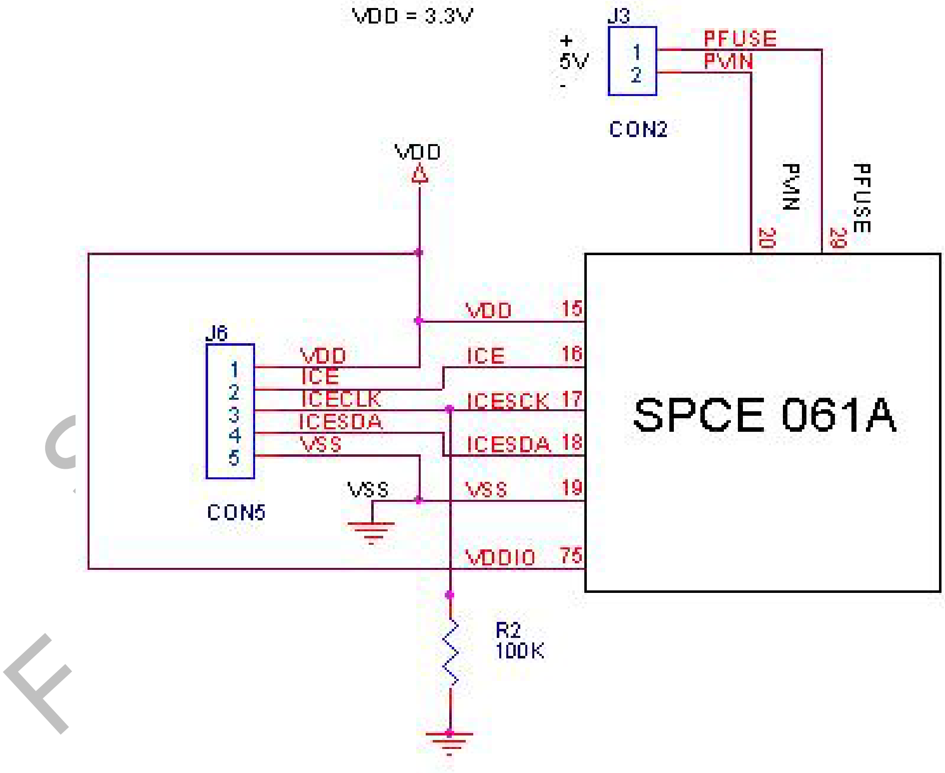
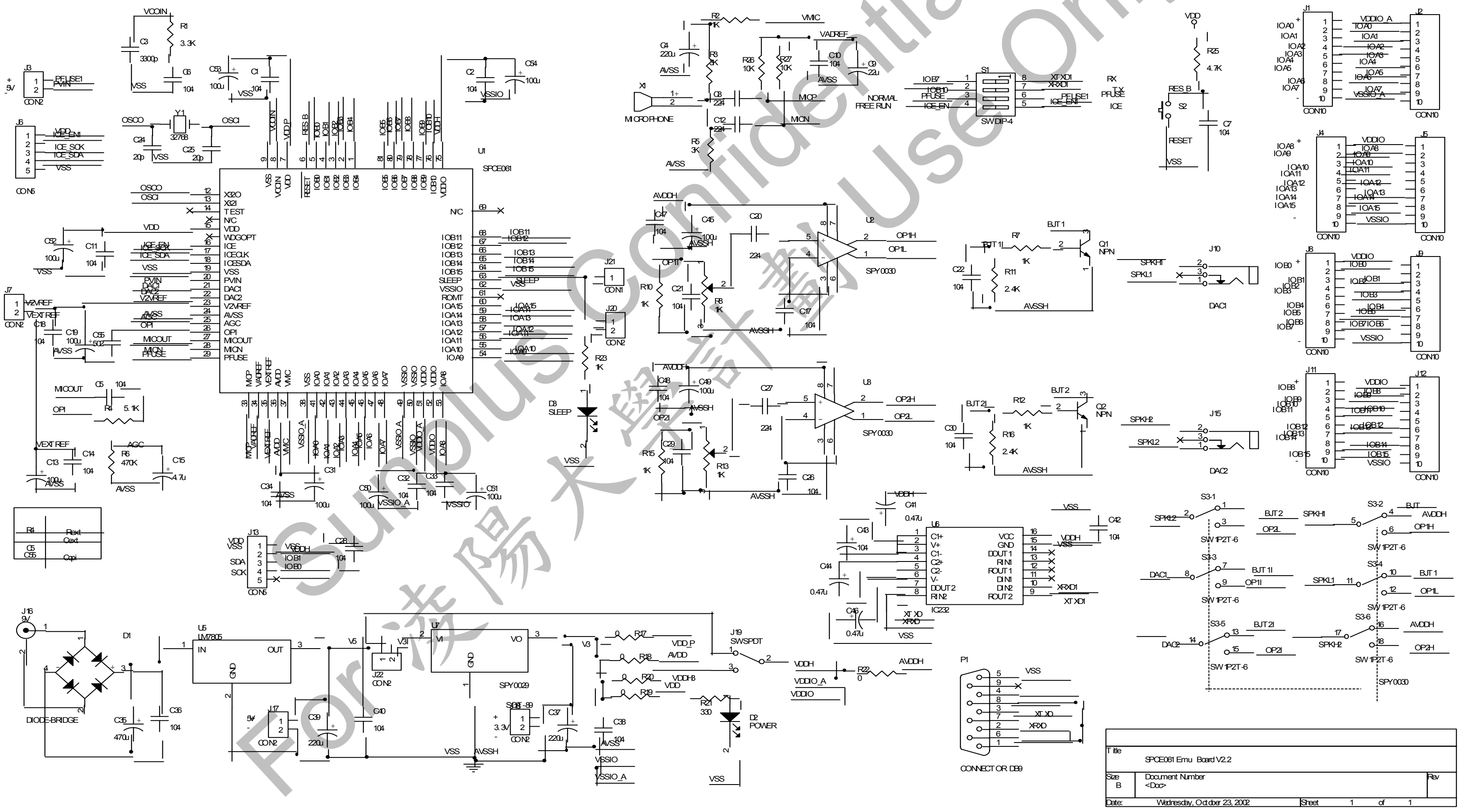


Fig. SPCE061A Download Circuit diagram

20.5 SPCE040A/060A/061A EMU board v2.2 schematics



21 APPENDIX B. Reserved Words

The followings are reserved words for *μ'nSP*® except the assembly language commands are not listed here.

```
1. Words with prefix of "__sn_" are reserved.
2. _BREAK, _FIQ, _RESET, _IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _IRQ6, _IRQ7, __subf2,
__blockcopy, __cmpf2, __common, __cvf2i1, __cvf2i2, __cvf2u1, __cvf2u2, __cvi1f2,
__cvi2f2, __divf2, __divi1, __divi2, __divu1, __divu2, __lshiu1, __lshiu2, __modi1,
__modi2, __modu1, __modu2, __mulf2, __muli2, __mulu1, __mulu2, __negf2, __rshi1, __rshi2,
__rshu1, __rshu2, __addf2
```