SUNPLUS

SPCE SACM MS02 User's Manual v1.0

05/16/2003

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

1.1 Document History

Revision	Date	By	Remark
V1.0	03/14/2003	Arthur Shieh	MS02

1.2 Library History

Revision	Date	By	Remark
V1.0	05/05/2003	Pao-Hwa Hsieh Arthur Shieh	MS02 (8 channel) / MS02 Lite(4 Channel) Version
V0.9	02/24/2003	Adamcar Tseng	MS02 8bit version

2 Type of Speech Compression Algorithm

2.1 Summary

Audio:

Present Algorithm Title	Data rate	Application
SACM-A1600	12/ 16 /24 Kbps	Audio
SACM-A2000	16 / 20 /24 Kbps	Audio
SACM-A3200	32 /36/40/44/48 Kbps	Audio
SACM-A3200 2Ch	32 /36/40/44/48 Kbps	Audio (2 Channel)

Speech:

Present Algorithm Title	Data rate	Application
SACM-S200	0.8K / 0.9K / 1K / 1.2K / 1.4K / 1.6K / 1.8k / 2K / 2.4K / 2.8 Kbps	Speech
SACM-S240/S120	2.4 Kbps	Speech
SACM-S480/S720	4.8 / 7.2 Kbps	Speech
SACM-S530	5.3K / 5.96K / 6.63K / 7.29K / 7.95 Kbps	Speech

Melody:

Present Algorithm Title	Type	Channel	Application
SACM-MS01	FM	6 (4 FM+ 2 ADPCM)	Music Synthesizer
SACM-MS02	**Wave Table**	**4 (SPCE500A) / 8 (SPCE061A)**	**Music Synthesizer**

Recording:

Present Algorithm Title	Data rate	Channel	Application
SACM-A3200	32 / 36 / 40 / 44 / 48 Kbps	1	Recording
SACM-A3200 2Ch	32 / 36 / 40 / 44 / 48 Kbps	2	Recording
SACM-DVR for A2000	16Kbps data rate with 8Kbps sample rate from ADC for recording	1	Recording

2.2 Naming convention

SACM-Xnnn

SACM: Speech Audio Coding Method

X = A: Audio

S: Speech

MS: Melody

nnn = Data rate (for X=A or S)

= Synthesizer type (for X = MS); 01 = FM, 02 = Wave table.

DVR: Digital Voice Recording

Example:

SACM A2000 stands for Sunplus audio algorithm with nominal data rate 20Kbps.The actual data rate will depend on the options that algorithm provides and the sampling rate adopted.

3 Basics for SACM MS02

MS02 is Sunplus' MIDI music synthesizer. The backbone technology, wave-table allows the rich presentation of MIDI music to fit on an SPCE embedded system via SACM MS02. Musical Instrument Digital Interface (MIDI) is a widely adopted music format, which is easy to access and convenient to process for users. User simply needs to use Sunplus music tool, Sunmidiar, to convert the MIDI songs into the MS02 song and tone color resource files required for the SACM MS02 playback. The MIDI music can then be played in users' applications through SACM MS02. Please be aware that due to the embedded system concern, not all MIDI events are parsed and processed in MS02. The polyphonic channel numbers, song attribute (tempo, score and etc.) and DAC play rate are 3 major factors on CPU loading.

3.1 Dynamic-allocated Polyphonic channels

MS02 can support up to 8 polyphonic channels, each of which can deliver an intact note envelope without being intervened by another note in most cases by taking advantage of dynamic allocation. Due to the performance issue, MS02 can supports up to 8 polyphonic channels on SPCE061A, and it is advised that user use fewer than 6 polyphonic channels (typical 4 channels) on SPCE500A. For concurrent algorithm applications, the use of 4 channels MS02 along with other speech/audio algorithm is recommended.

When a MIDI song is playing by MS02, the channel dynamic-allocation mechanism will assign a note to an available channel when a note-on event comes up. The next note will be allocated to another available channel automatically. In case that all channels are taken, certain rules will be applied to decide which channel shall be cut off. This case can be avoided in advance when composing the MIDI.

3.2 Tone color library

MS02 also grants users the option to make the trade-off between music quality and memory size. The size of tone color library implies the sound quality of instruments used. Sunmidiar offers an extensive assistance for developers from tone color editing to MIDI analyzing and converting. For details, please refer to Sunmidiar user manual. Users can make custom tone colors on their own by the Sunplus patch editor. The tone color library can be changed at run time. Due to the reliability issue, tone color library does not support manual mode.

3.3 Auto / Manual Mode

MS02 supports manual mode for MIDI Song. The manual mode for tone color library is not supported. User can

place the song data in the external memory storage and access the song data by manual mode. The manual mode call back functions can be referred at sacm_user.asm. Users have to implement the memory access functions based on the storage type.

3.4 Play Rate, Service Rate and Pitch Range

MS02 also offers a great flexibility that allows users adjust the play rate, which is the frequency by which PCM data is delivered to DAC. The play rate adjusting, implemented via passing parameters, can make trade-offs between CPU loading and sound quality. The benefit users can get from this feature is that this trade-off can carry out without the need to re-make the tone color library. That is, the initial sampling rate won't affect the play rate. For CPU loading information please refer to appendix.

In SACM-MS02, service rate is 3 times the play rate. Service rate means the frequency of processing the sound data.

The certain section in an instrument tone color can synthesized a certain pitch range.

The Pitch Range is determined by the following formula.

Sample_rate / service_rate * pow(2 , (max_pitch - base_pitch)/12) < 1

The pitch range of a certain section has an upper limit, maximal pitch range, but no minimal pitch. However, if the pitch synthesized is deviated from the base pitch too far, the sound fidelity to the instrument will be degraded.

3.5 Sampling rate

There is no limitation to SACM MS02 tone color sampling rate. The only concern is the CPU loading. We would advise coding engineers that the sampling rate should be bounded in between 8KHz to 25KHz to ensure the availability of 8 polyphonic channels.

3.6 CPU loading

The CPU loading estimation for SACM-MS02 is an ad hoc. It varies by the factors, channel numbers, song attribute (tempo, score and etc.) and DAC play rate. If channel number is higher, CPU loading is also higher. The faster the tempo, the higher the CPU loading. The higher DAC play rate, the higher CPU loading. Please refer to appendix for CPU loading information. For SACM-MS02, the CPU loading can be measured by taking the service loop time and ISR time into account. The MIDI composing techniques and hardware settings can help to lower the CPU loading and to grant user applications more flexibility.

SACM-MS02 features

- Up to 8 dynamic channels
- Play rate (DAC rate) options: 8K, 10K, 12K, 16K, 20K, 24K, 28K, 32K, 36K, 40 KHz
- Tempo / Key adjustable at playback
- Instrument / Tone color run-time change
- Auto / Manual Mode supported.
- MIDI compatible

SACM-MS02 Technical basics

- Variable sampling rate for tone color from 8KHz to 25KHz.
- Foreground/background Service loop, 1 timer required for playback
- ROM requirement: 4596 words (TEXT : 3954, CODE: 642)
- RAM requirement: 291 words
- RAM section shared via ORAM section: OVERLAP_MS02_RAM_BLOCK

4 API for SACM-MS02

4.1 Hardware Dependent Functions Function: Initializes SACM-MS02

4.1.1 Function: Initialize SACM-MS02 module, sets Interrupt source, Timer, and play mode (Manual/Auto mode) before playing

Syntax:

C: int SACM_MS02_Initial(int Play_Rate, int Channel)

ASM: R1 = [Play_Rate]

R2 = [Channel]

Call F_SACM_MS02_Initial

Parameters:

Play_Rate 0: DAC for 8K play rate
1: DAC for 10K play rate
2: DAC for 12K play rate
3: DAC for 16K play rate
4: DAC for 20K play rate
5: DAC for 24K play rate
6: DAC for 28K play rate
7: DAC for 32K play rate
8: DAC for 36K play rate
9: DAC for 40K play rate

Channel : 0~8 for SPCE061A/060A/040A
0~4 for SPCE500A/380A/250A/120A

Return Value:

N/A

Library: <SacmVxx.LIB>

Remark:

1. This function initializes the decoder of MS02. It also initializes the system clock, Timer A/Timer B, DAC and enables the Timer A FIQ / IRQ1 or Timer B FIQ / IRQ2. User can select which timer to use accordingly in F_SACM_MS02_Init_.
2. The hardware setting is opened for user's reference(see F_SACM_MS02_Init_: function in sacmVxx.asm) .
3. All channels will be enabled after initialization

4. The play rate of SACM_MS02 is the interrupt triggered by timer.
5. This function utilizes a register, R_InterruptStatus (spce.asm), to keep track of interrupt setting with user's program for SPCE500A. It uses P_INT_Mask instead for SPCE 061A/060A

4.2 Service Loop Functions: Service loop to fill SACM-MS02 song data to sequencer

4.2.1 Function: Foreground service loop:

Syntax:

C: void SACM_MS02_ServiceLoop(void);

ASM: call F_SACM_MS02_ServiceLoop

Parameters: None

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. In this function, the MIDI events are parsed and processed for sequencer.

4.3 Playback Functions: Playback control

4.3.1 Function: Begin to play an SACM_MS02 melody

Syntax:

C: int SACM_MS02_Play(int Song_Index, int Channel, int Ramp_Set)

ASM: R1 = [Song_Index]

R2 = [Channle]

R3 = [Ramp_Set]

Call F_SACM_MS02_Play

Parameters:

Song_Index: 0 - max speech index : Auto Mode

-1 : Manula Mode

Channel:

1: To DAC1

2: To DAC2

3: To both DAC1 and DAC2

Ramp_Set

0: Disable both ramp up and down

1: Enable ramp up only

2: Enable ramp down only

3: Enable both ramp up and down

Return Value: N/A

Library: <SacmVxx.LIB>

Remark:

1. The SACM-MS02 provides eight dynamic channels ,each of which maps to a MIDI channel .
2. The song data can be played externally by user function.
3. The Song_Index is the speech sequence of T_SACM_MS02_SongTable in resource.asm.
4. The F_ISR_Service_SACM_MS02, must be hooked to the corresponding _FIQ:/_IRQ1/_IRQ2 label before using this function.
5. The Timer interrupt is working at the rate specified by the initial function, SACM_MS02_Initial(int Play_Rate, int Channel) .

Example: Plays a SACM-MS02 melody with auto end in auto mode.

(a). In main.c:

```
#include "sacmV33.h"

int SongIndex = 0;

Main()
{
    SACM_MS02_Initial(MS02_DAC_16K,4);        // DAC 16K, 4 Channels
    SACM_MS02_Play(SongIndex,DAC1+DAC2, Ramp_UpDn_On)//Play 1st speech on both DAC1 &2
    while(SACM_MS02_Status()&0x01)
     {
          SACM_MS02_ServiceLoop();
     }
}
```

(b). In isr.asm:

```
_IRQ1:
        push R1, R5 to [sp];

        call F_ISR_Service_SACM_MS02;     // MS02 ISR service loop

        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop R1, R5 from [sp];
        RETI;
```

4.3.2 Function: Stop playing SACM-MS02 melody

Syntax:

C: void SACM_MS02_Stop(void)

ASM: Call F_SACM_MS02_Stop

Parameters: None

Return Value: None

Library: <SacmVxx.LIB>

Remark: This function will not change the interrupt setting.

4.3.3 Function: Pause playing SACM-MS02 melody

Syntax:

C: void SACM_MS02_Pause(void)

ASM: Call F_SACM_MS02_Pause

Parameters: None

Return Value: None

Library: <SacmVxx.LIB>

Remark:

4.3.4 Function: Resume the paused SACM-MS02 melody

Syntax:

C: void SACM_MS02_Resume(void);

ASM: Call F_SACM_MS02_Resume

Parameters: None

Return Value: None

Library: <SacmVxx.LIB>

Remark:

4.3.5 Function: Main volume control for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_Volume(int Volume_Index)

ASM: R1 = [Volume_Index]

Call F_SACM_MS02_Volume

Parameters:

Volume_Index: [0..15], 0:Min volume, 15:Max volume

Return Value: None

Library: <SacmVxx.LIB>

Remark:

4.3.6 Function: Obtain the status of the SACM_MS02 module

Syntax:

C: unsigned int SACM_MS02_Status(void);

ASM: Call F_SACM_MS02_Status

[Return_Value] = R1

Parameters: None

Return Value:

bit 0: 0: Song ended
1: Song Playing

bit 1: 0: Song not Paused
1: Song Paused

bit 2: 0: DAC 1 MS02 output disable
1: DAC1 MS02 output enable
bit 3 : 0: DAC2 MS02 output disable
1: DAC2 MS02 output enable
bit 4: 0: DAC Ramp up disable
1: DAC Ramp up enable
bit 5: 0: DAC Ramp down disable
1: DAC Ramp down enable
bit 6: 0: Manual
1: Auto
bit 7: 0: Stereo Disable
1:Stereo Enable

Library: <SacmVxx.LIB>

Remark: None

4.3.7 Function: Enable one of channels for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ChannelOn(int MIDI_Channel)

ASM: R1 = [MIDI_Channel]
Call F_SACM_MS02_ChannelOn

Parameters:

MIDI_Channel : [0..15]

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. The MIDI channel information can be knew from the original MIDI information. User can refer to the original MIDI for channel setting and polyphony details.

4.3.8 Function: Disable one of channels for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ChannelOff(int MIDI_Channel)

ASM: R1 = [Channel]
Call F_SACM_MS02_ChannelOff

Parameters:

MIDI_Channel: [0..15]

Return Value: None

Library: <SacmVxx.LIB>

Remark:

4.3.9 Function: Hold one of channels for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_HoldChannel(int MIDI_Channel)

ASM: R1 = [Channel]

Call F_SACM_MS02_HoldChannel

Parameters:

MIDI_Channel: [0..15]

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. For the channel held, the note events coming up from this channel will not be allocated dynamically and always go into the designated channel.

4.3.10 Function: Release one of channels for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ReleaseChannel(int MIDI_Channel)

ASM: R1 = [Channel]

Call F_SACM_MS02_ReleaseChannel

Parameters:

MIDI_Channel: [0..15]

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. For the channel held, this function allows the note events designated to this channel to be allocated dynamically again.

4.3.11 Function: Change the instrument on one of the SACM-MS02 channels

Syntax:

C: SACM_MS02_ ChangeInstru (int MIDI_Channel, int Instrument,)

ASM: R1 = [MIDI_Channel]

R2 = [Instrument]

call F_SACM_MS02_ ChangeInstru

Parameters:

MIDI_Channel : [0..15];

Instrument: [0..Max. Instrument]

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. The instrument argument is the mapped instrument index. It is decided when the MIDI and tone color are exported from Sunmidiar.

4.3.12 Function: Key Change for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_KeyShift (int Key_Shift_Index)

ASM: R1 = [Key_Shift_Index]

call F_ SACM_MS02_KeyShift

Parameters:

Key_Shift_Index: number of keys to shift

Return Value: None

Library: <SacmVxx.LIB>

Remark: 1. The key shift range is bound by the current pitch and maximal pitch that the tone color can synthesize.

4.3.13 Function: Tempo Change for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ChangeTempo(int TempoFactor)

ASM: R1 = [TempoFactor]

call F_ SACM_MS02_ChangeTempo

Parameters:

TempoFactor : [-14..13] -14~ -1 slow; 0: normal; 0~13 fast

Return Value: None

Library: <SacmVxx.LIB>

Remark: 1. The tempo is changed in a comparative manner by this function. The MIDI tempo event is not supported yet.

2. The tempo factor setting to tempo change table as follows

Tempo Factor	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Multiplier	2.86	2.50	2.17	1.92	1.72	1.54	1.39	1.27	1.22	1.16	1.12	1.08	1.03	1.00
Tempo Factor	**-1**	**-2**	**-3**	**-4**	**-5**	**-6**	**-7**	**-8**	**-9**	**-10**	**-11**	**-12**	**-13**	**-14**
Multiplier	0.95	0.91	0.87	0.83	0.77	0.71	0.67	0.63	0.59	0.53	0.45	0.40	0.36	0.33

Example: If original tempo is 100 BPM, and user sets the Tempofactor as 10, the new tempo will be 100 * 1.92 = 192.

If original tempo is 100 BPM, and user sets the Tempofactor as -10, the new tempo will be 100 * 0.53 = 53.

4.3.14 Function: Reset to Default Tempo for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ResetTempo()

ASM: call F_SACM_MS02_ResetTempo

Parameters: N/A

Return Value: N/A

Library: <SacmVxx.LIB>

Remark:

4.3.15 Function: Change instrument in a MIDI channel for SACM-MS02 melody playing

Syntax:

C: void SACM_MS02_ChangeInstru(int MIDI_Channel , int Instrument)

ASM: R1 = [MIDI_Channel];
R2 = [Instrument];
call F_SACM_MS02_ChangeInstru;

Parameters: MIDI_Channel : [0..15] MIDI channel
Instrument : [0..127] MIDI instrument

Return Value: N/A

Library: <SacmVxx.LIB>

Remark: 1. This function allows user to change instrument at run time but the selected instrument must also be included in the tone color library.

4.3.16 Function: Change instrument set (tone color library) for SACM-MS02

melody playing

Syntax:

C: void SACM_MS02_ChangeInstruSet(int InstruSetIndex)

ASM: R1 = [InstruSetIndex];

call F_ SACM_MS02_ChangeInstruSet;

Parameters: InstruSetIndex : The index of Instrument Set in tone color library

Return Value: N/A

Library: <SacmVxx.LIB>

Remark: 1. This function allows user to change instrument set at run time. The index is the sequence in T_SACM_MS02_InstrumentSet in resource.asm.

4.3.17 Function: Play an note when SACM_MS02 is running

Syntax:

C: void SACM_MS02_PlayNote (int MIDI_Channel, int Pitch, int Velocity , int Duration)

ASM: R1 = [MIDI_Channel];

R2 = [Pitch];

R3 = [Velocity];

R4 = [Duration];

call F_SACM_MS02_Play

Parameters:

MIDI_Channel : [0 ..15];

Pitch : [0..127];

Velocity : [0..127]

Duration : [0..65535] tickers; where a ticker typically 5ms long.

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. For more details on MIDI, please refer to sources on Web or publications on MIDI.
2. MIDI channel decides the MIDI channel by which the note is delivered and, as a result, also decides which instrument tone color will carry out the note.
3. MIDI Pitch is from 0 to127. The lowest note name is then C0 (note number 0), and the highest possible note name is G10 (note number 127). Not all MIDI notes can be played. Each tone color has its pitch range and limited by the formulas, Sample_rate / service_rate * pow(2 , (max_pitch - base_pitch)/12) < 1 . User can also use the help of Sunmidiar to test the pitch range. Please refer to

Sunmidiar for details.

4. Velocity is a factor to the output volume of the note played. It affects the volume of the note played and meanwhile the volume of other channels is unchanged.
5. To make this API functional, user have to compose a dummy MIDI with the desired tone color and then play this dummy song in MS02 before this API is called.
6. The note played will be dynamically allocated to one of the polyphonic channels by the same rules in other MS02 polyphonic channels.
7. If the MIDI channel specified is not used, the default channel will be set as channel 0.

4.4 ISR Functions: Interrupt service routine for SACM-MS02

The ISR service routine will synthesize the song data from service loop subroutine (SACM_MS02_ServiceLoop). The audio output will be sent to DAC.

Syntax:

C: Void ISR_Service_SACM_MS02(Void);

ASM: F_ISR_Service_SACM_MS02

Parameters: None

Return Value: None

Library: <SacmVxx.LIB>

Remark:

1. ISR service for SACM-MS02 .
2. Hook the interrupt service routine on one of FIQ TMA, IRQ1 TMA or IRQ2 TMB .
3. The play rate of in SACM-MS02 can offer 10 options, 8KHz, 10 KHz,,12 KHz,16 KHz,20 KHz, 24 KHz, 28 KHz, 32 KHz, 36 KHz, and 40 KHz. User needs to select the desired play rate in SACM_MS01_Initial() and to set the interrupt in F_ SACM_MS01_Initial_.
4. A flag in the F_ISR_Service_SACM_MS02 will block the code after stopping playing. Using this function with user's function in the same FIQ is possible

EX:

```
_IRQ1:
        call F_ISR_Service_SACM_MS02;     // MS02 ISR service loop
        call    F_User_ISR
        reti
```

EX:

```
IRQ1()
{
```

```
        ISR_Service_SACM_MS02();      // MS02 ISR service loop
}
```

5. The F_ISR_Service_SACM_MS02 must be hooked on the _FIQ / IRQ1/ IRQ2: label (see isr.asm in MS02 example) before use this function.

4.5 User Functions: for SACM-MS02 playback

4.5.1 Function: Song event notification

Syntax:

C: User-defined

ASM: R1 = Event_Value

F_SACM_MS02_SongEvent

Parameters: R1 = return value

bit[15:12] = 0x0 : End Event

bit[15:12] = 0x1 : Note Event

bit[15:12] = 0x2 : Instrument Event

bit[15:12] = 0x3 : Volume Event

bit[15:12] = 0x4 : Pan Event

bit[15:12] = 0x6 : Tempo Event

bit[15:12] = 0x7 : Pitch Bend Event

bit[15:12] = 0x8 : Other Controller Event

bit[11:8] Channel 0-7

End Event

bit[7:0] Unused

Note Event:

bit[7:0] Pitch

Instrument Event:

bit[7:0] GM Instrument

Volume Event

bit[7:0] Volume

Pan Event

bit[7:0] Pan

Tempo Event

bit[7:0] Unused

Extending Argument R2 bit[15:0] beat per minute

Pitch Bend Event

bit[7:0] Unused

Extending Argument R2 bit[15:14] zero

Extending Argument R2 bit[13:7] pitchMsb

Extending Argument R2 bit[6:0] pitchLsb

Other Controller Event

bit[15:0] Unused

Return Value: None

Library: <SacmVxx.LIB>, sacm_user.asm

Remark:

1. User can implement the function based on the event value passed in. The action should not take too long to block the melody playing.
2. When library calls back this function, the parameter in R1 carries different composition of information about the event details depending on the event types indicated in [b15..b12] of R1. R2 also serve as an auxiliary argument if the event is a tempo or pitch bend event.

   bit[15:12] = 0x0 : End Event

   bit[15:12] = 0x1 : Note Event

   bit[15:12] = 0x2 : Instrument Event

   bit[15:12] = 0x3 : Volume Event

   bit[15:12] = 0x4 : Pan Event

   bit[15:12] = 0x6 : Tempo Event

   bit[15:12] = 0x7 : Pitch Bend Event

   bit[15:12] = 0x8 : Other Controller Event
3. If user prefers to implement the function in C programming languish, user can call a C

function from this F_SACM_MS02_SongEvent. The price would be the extra overhead. However, this function should not block program flow too long; otherwise the MS02 will not play smoothly.

4.5.2 Function: Set start address for SACM-MS02 song data.

Syntax:

C: USER_SetStartAddr()

ASM: call F_USER_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: <SacmVxx.LIB>, sacm_user.asm

Remark: 1. Manual mode only.

2. User implements the function based on the storage type. The memory interface has to be constructed in advance if user intends to access the data from external storage, e.g. SRAM, FLASH.

4.5.3 Function: Fetch the SACM-MS02 song data from user 's storage

Syntax:

C: User-defined

ASM: call F_USER_MS02_GetSongData

Parameters: User-defined

Return Value: R1 = data

Library: <SacmVxx.LIB> , sacm_user.asm

Remark:

1. Manual mode use only
2. User implements the function based on the storage type. The memory interface has to be constructed in advance if user intends to access the data from external storage, e.g. SRAM, FLASH or mask ROM.
3. The SACM MS02 library will fetch a word data through this function where R1 hosts the data.

5 Hardware dependent API in SACMVxx.asm (Open source)

5.1 Ramp Functions

5.1.1 Function: Ramp up DAC1

Syntax:

C: void SP_RampUpDAC1(void)

ASM: call F_RampUpDAC1

Parameters: None

Return Value: None

Library: opened in sacmVxx.asm

Remark: Program must stop sending data to DAC1 while executing this function.

5.1.2 Function: Ramp up DAC2

Syntax:

C: void SP_RampUpDAC2(void)

ASM: call F_RampUpDAC2

Parameters: None

Return Value: None

Library: opened in sacmVxx.asm

Remark: Program must stop sending data to DAC2 while executing this function.

5.1.3 Function: Ramp up DAC1

Syntax:

C: void SP_RampDnDAC1(void)

ASM: call F_RampDnDAC1

Parameters: None

Return Value: None

Library: opened in hardware.asm

Remark: Programmers must stop sending data to DAC1 while executing this function.

5.1.4 Function: Ramp up DAC2

Syntax:

C: void SP_RampDnDAC2(void)

ASM: call F_RampDnDAC2

Parameters: None

Return Value: None

Library: opened in sacmVxx.asm

Remark: Programmers must stop sending data to DAC2 while executing this function.

5.2 Queue Functions

5.2.1 Function: Initial Queue

Syntax:

ASM: call F_SP_InitQueue

Parameters: None

Return Value: A word song data

Library: opened in sacmVxx.asm

Remark: 1. Manual use only

2. For each song compression module, e.g. SACM_A2000, SACM_S240, it has its own function, named F_SP_InitQueue_A2000, F_SP_InitQueue_S240; therefore, making two independent queues for each module is possible.

5.2.2 Function: Obtain a word from queue

Syntax:

ASM: call F_SP_ReadQueue

Parameters: None

Return Value: A word data

Library: opened in sacmVxx.asm

Remark: 1. Manual use only

2. Same as remake2 of F_SP_InitQueue

5.2.3 Function: Fill a word to queue

Syntax:

ASM: F_SP_WriteQueue

Parameters: None

Return Value: None

Library: opened in sacmVxx.asm

Remark: 1. Manual use only

2. Same as remake2 of F_SP_InitQueue

5.3 Memory Access Functions

5.3.1 Function(C & ASM): Fetch song data from resource in ROM (See *μ'nsp Assembly Manual* for details)

Syntax:

C: unsigned int GetResource(long)

ASM:

F_GetResource

Parameters: None

Return Value: A word song data

Library: opened in sacmVxx.asm

Remark:

1. Manual use only

2. Long type parameter includes 16-bit address and 6-bit page index.

6 How to adapt your old project for new library

6.1 The SACM project architecture

Since SACMV32, the library has some minor changes over its architecture so that the project architecture has a minor adjustment as well.

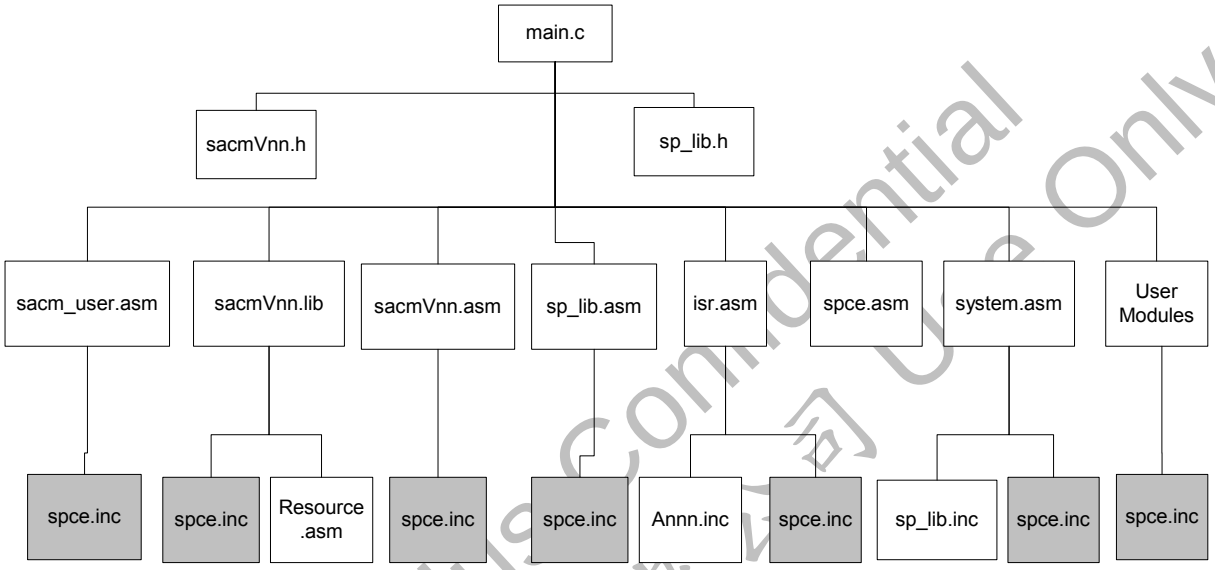


Fig. A typical SACM project architecture

In the SACM examples, user shall see an architecture like the demonstration above. User will also notice that the hardware.asm and key.asm (if applicatble) are disappeared. In the hardware.asm, there are 3 types of information included, SPCE port definition, SACM related functions (initializations, queue and hardware dependent functions), and SPCE dependent APIs. For the purpose of modulization, it is split since SACMv32.lib.

Hardware.asm is now split and some header files are also arranged into 3 modules

(1) spce.inc: SPCE port definition, spce.asm: R_InterruptStatus for SPCE500A to keep tack of interrupt setting status.

(2) SACMVnn.asm: Library initializations, queue functions and ramp up/down hardware dependent functions.
SACMVnn.h: C function declarations for SACM APIs.
Annn.inc (s200.inc, s240, inc, s480.inc, s530.inc, A1600.inc, A2000.inc, A3200.inc, A32002Ch.inc, MS02 and DVR.inc): Assembly function declarations for each algorithm.

(3) sp_lib.asm: General APIs for SPCE, key scan and I/O configuration function.

6.2 Step-by-step procedure

For a programmer to update old projects to under new SACM library structure, the procedures are

(1). Find each line, ".include hardware.inc" inside assembly files in project .

(2) Check the files where the line, ".include hardware.inc", presents .

(2.1) If SPCE port definition is used in the module, then add ".include spce.inc" on the top of the file.

(2.2) If any key function is used in the module, then add ".include sp_lib.inc" on the top of the file.

(2.3) If any SACM library function is used in the module, then add ".include Annn.inc" on the top of the file. Where Annn.inc can be s200.inc, s240,inc, s480.inc, s530.inc, A1600.inc, A2000.inc, A3200.inc , A32002Ch.inc, MS02.inc or DVR.inc.

(2.4) If R_InterruptStatus is used, add " .external R_InterruptStatus" to the top of the file.

(3) Find each line, "#include "hardware.h" " inside C files in project.

(4) Check the file where the line, "#include "hardware.h" ", presents.

(2.1) If SPCE port definition is used in the module, then add "#include "spce.h" " on the top of the file.

(2.2) If any key function is used in the module, then add "#include "sp_lib.h" " on the top of the file.

(2/3) If any SACM library function is used in the module, then add "#include "SACMVnnn.h" " on the top of the file.

(5) Remove hardware.asm, hareware.inc, key.asm(if applicable), key.inc (if applicable) from project and add in source files , sacmVnn.asm, spce.asm, sp_lib.asm, sacm_user.asm(if applicable), and header files, spce.inc, Annn.inc and sacmVnn.h to project

(6) Open spce.inc to check out the "Body_Type" definition on the top of the file. Change the definition according to the body used.

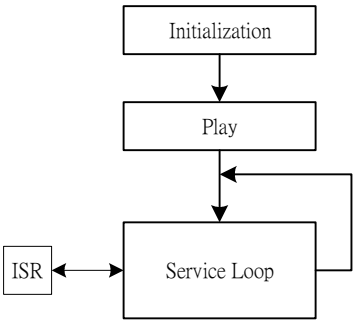
(7) Check the project content, and see if the library API is still supported in new library structure. If not, modify the program structure according to the examples (foreground/background, auto mode/manual mode and SACM_user.asm).

User will see that the new SACM library structure is more accessible. It will take some efforts to change form old version project to new one. User can contact Sunplus representative for technical support through Sunplus web site, http://www.sunplus.com.tw.

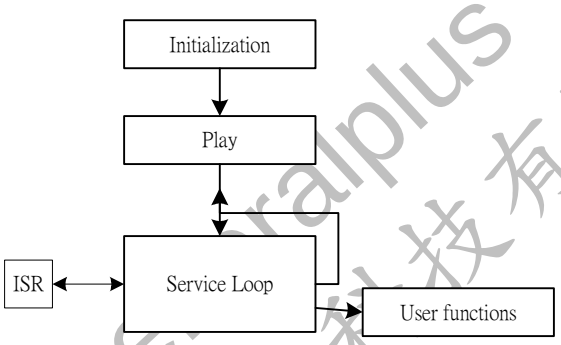
7 How to use the SACM library

7.1 The programming flow in auto mode

(a). Auto Mode



(b). The programming flow in manual mode



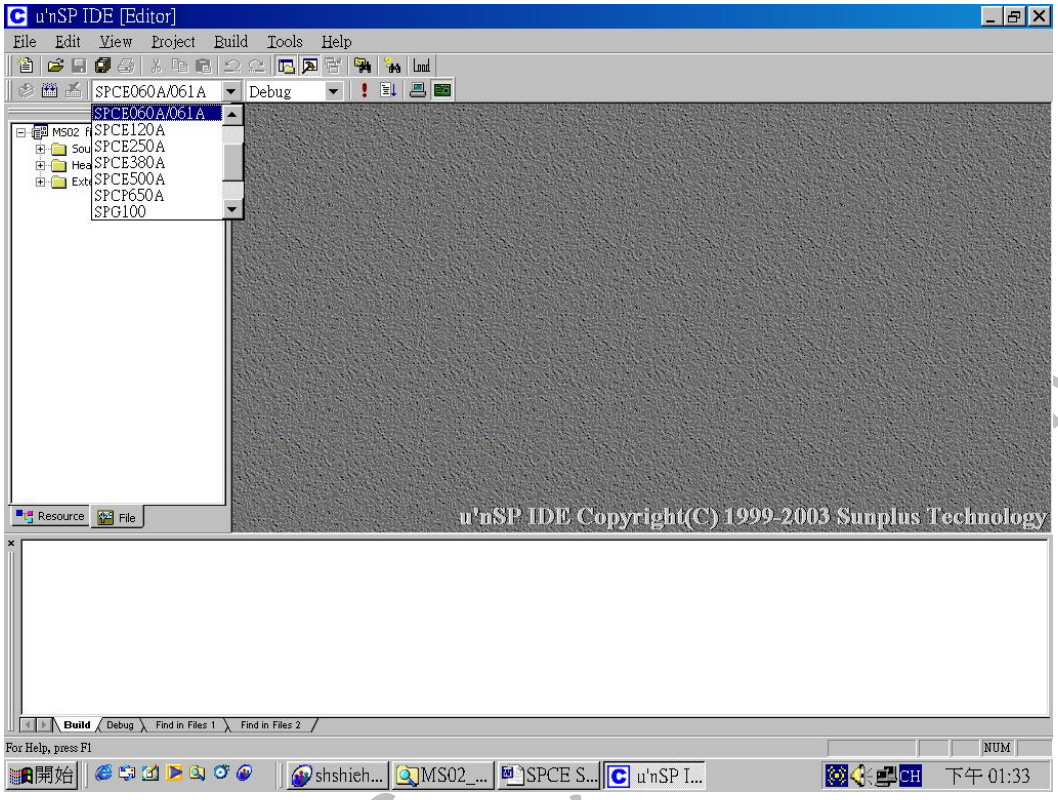
7.2 Quick start: Step-by-Step Instructions

The easiest way to start your SACM application is start from a sample project enclosed in the SACM library package. Here we provide the step-by-step instructions for users to launch the project development.

7.2.1 Open a sample project

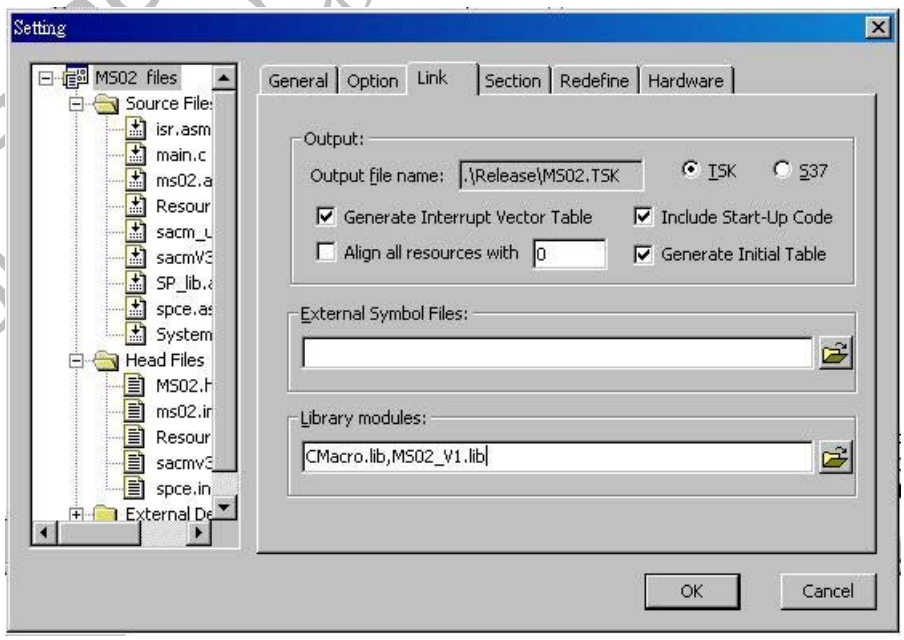
Open the sample project by the IC body and the SACM library you wanted to use.

And make sure that the body type is correct.



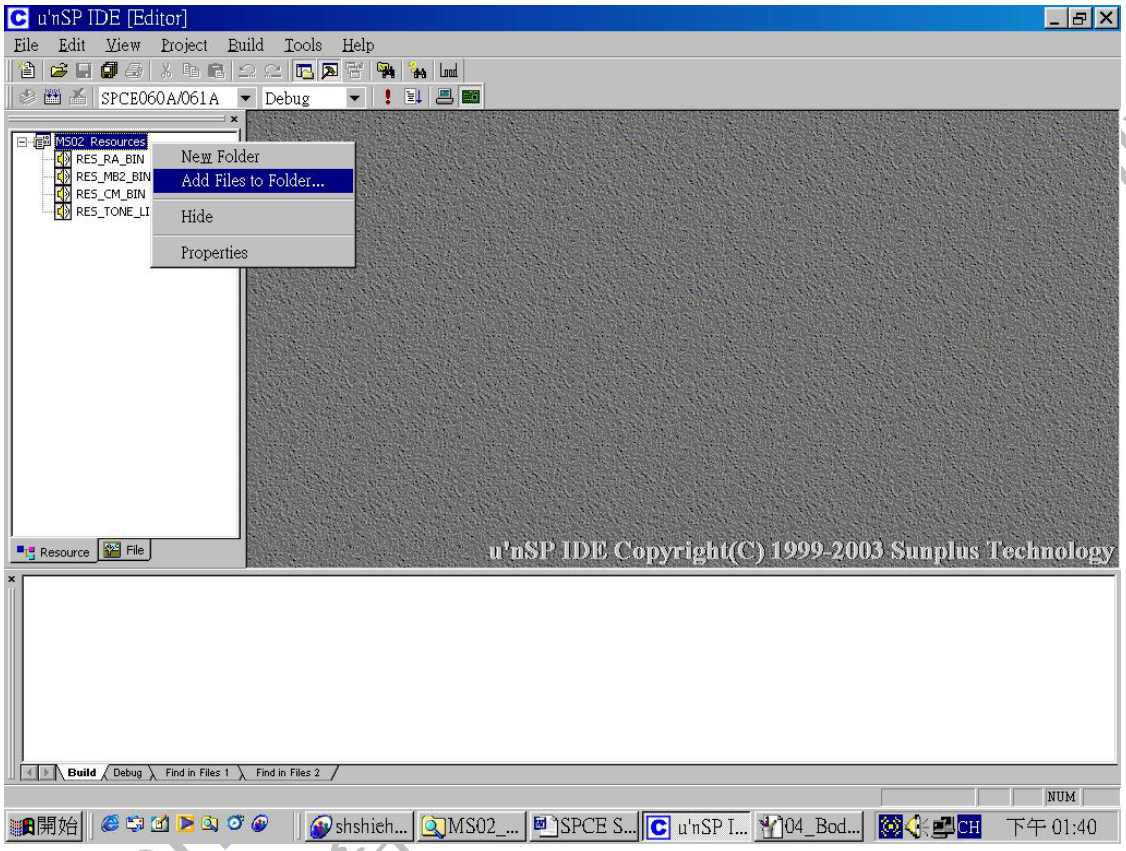
7.2.2 Link the libraries to user's program

Go to **[Project] -> [Setting] -> [Link]** and Check if library is correctly typed here, i.e. sacmvXXXX.lib to the library modules text box.



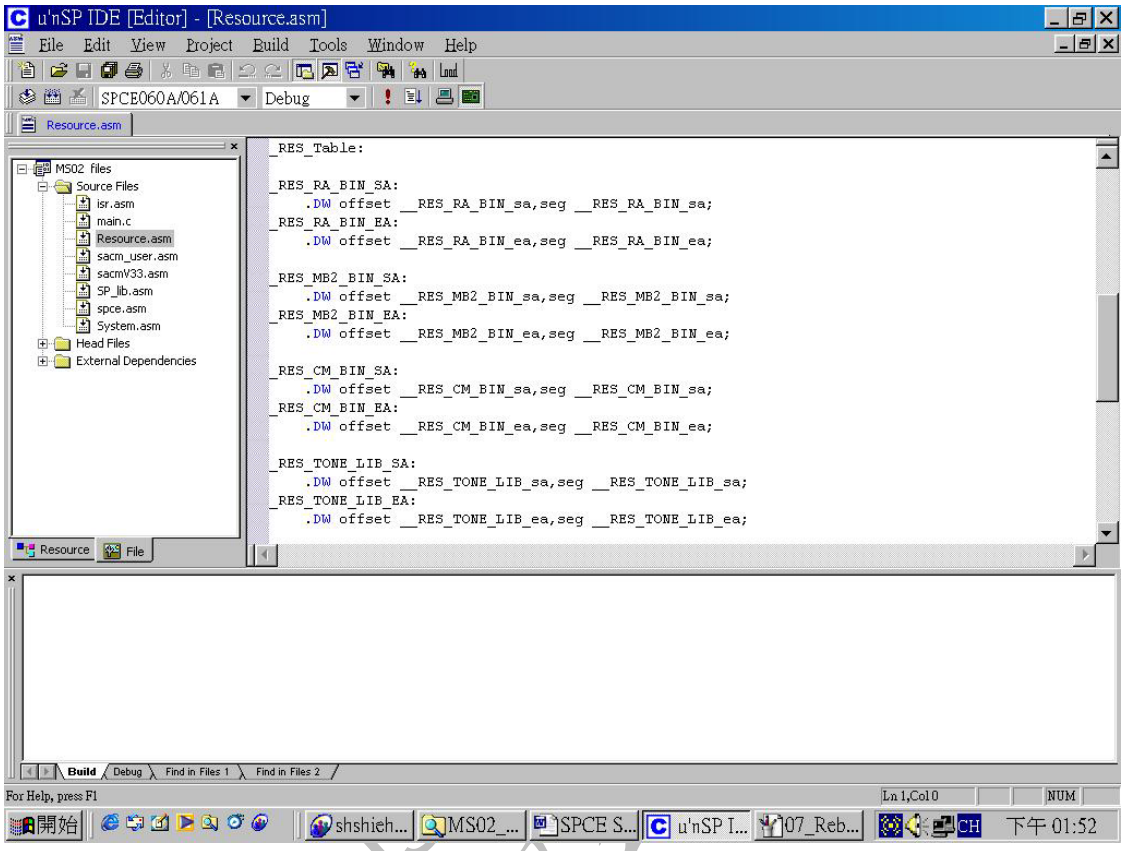
7.2.3 Add resources to user's program

- Click on resource tab in work space window
- move mouse to Porject Resources(in this case, MS02 Resources)
- Click on right button and choose "Add Files to Folder"
- Add compressed audio or speech file into the folder.



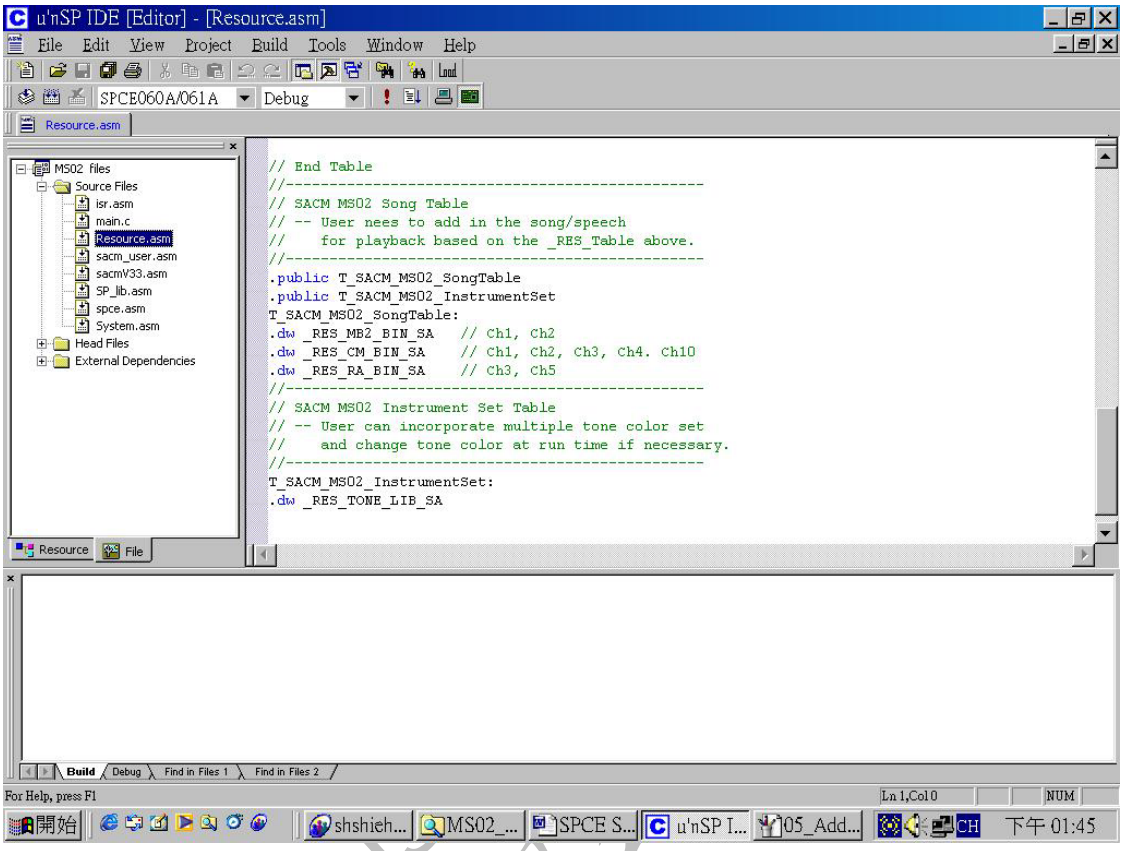
7.2.4 Rebuild project

After user rebuild the project, the user will see the resource table has been generated with updated resource information in the resource.asm.



7.2.5 Arrange Song/Speech sequence in the Song/Speech table in resource.asm

User will needs to arrange the speech /song sequence in the T_SACM_MS02_SongTable and the sequence of tone color library in T_SACM_MS02_InstrumentSet.



7.3 A simple example

Main.C

```
#include "sacmv33.h"

Main()
{
    SACM_MS02_Initial(MS02_DAC_28K,8);                    // Initialization for playing
    SACM_MS02_Play(Song_Index, DAC1+DAC2, Ramp_UpDn_On);
    While(1){
        SACM_MS02_ServiceLoop();          // Fill Data to Queue
    }
    Return 0;
}
```

Note: The Song Index is the sequence in speech table, i.e. T_SACM_MS02_SpngTable in resource.asm.

ISR.ASM

```
_IRQ1:
        push R1, R5 to [sp];

        call F_ISR_Service_SACM_MS02;     // MS02 ISR service loop

        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop R1, R5 from [sp];
        RETI;
```

Resource.asm

```
//-------------------------------------------------
// SACM MS02 Song Table
// -- User nees to add in the song/speech
//       for playback based on the _RES_Table above.
//-------------------------------------------------
.public T_SACM_MS02_SongTable
.public T_SACM_MS02_InstrumentSet
T_SACM_MS02_SongTable:
.dw _RES_MB2_BIN_SA
.dw _RES_CM_BIN_SA
.dw _RES_RA_BIN_SA

//-------------------------------------------------
// SACM MS02 Instrument Set Table
// -- User can incorporate multiple tone color set
//       and change tone color at run time if necessary.
//-------------------------------------------------
T_SACM_MS02_InstrumentSet:
.dw _RES_TONE4_BIN_SA
```

R_InterruptStuaus - A public interrupt control register in spce.asm for SPCE500A

Programmers may share the interrupt source with SACM library, A register, R_InterrruptStatus is reserved for sharing the interrupt source. This register records the status of occupied interrupt by library; therefore, it is an interface to identify which interrupt is used by library. If a content of "0x2000" is in the R_InterruptStatus, it means the Timer A FIQ is being used by library subroutines at this moment. For example, the R_InterruptStatus will be changed from "0x0000" to "0x2000" when SACM_MS02_Play() is called. In contrast, SACM_MS02_Stop() will not change the INT setting from "0x2000" to "0x0000" when SACM_MS02 is called. Every interrupt setting must follow the rule to share the interrupt resource. The following is an example to enable IRQ4 while SACM_MS02 speech is playing.

```
R1 = 0;                          // At beginning, R_InterruptStatus = 0x0000
Call   F_SACM_MS02_Play          // The R_InterruptStatus change to 0x2000 by
```

```
                                    // F_SACM_A2000_Play, TimerA FIQ enable and the speech playing.
…
.if BODY_TYPE == SPCE061A
    R1 = [P_INT_Mask]           // Get current interrupt setting status from P_INT_Mask if for SPCE061A
.endif
.if BODY_TYPE == SPCE500A
    R1 = [R_InterruptStatus]    // Get current interrupt setting status from R_InterruptStatus if for SPCE500A
.endif                          // Enable IRQ2 at this moment
R1 |= 0x0400                    // Set Timer B IRQ2
[R_InterruptStatus] = R1        // Update R_InterruptStatus to 0x2400
[P_INT_Ctrl] = R1                   // Set interrupt control port 0x7010
```

For SPCE061, a new hardware port P_INT_Mask(702DH) serves the purpose as well and it is advised for users to take advantage of it. . In **SPCE.inc**. The BODY_TYPE definition determines the SPCE body type and as a result program can know whether R_InterruptStatus or P_INT_Mask should be refered in sacmVxx.asm.

8 How to use Sunmidiar to export the MIDI and Tone color library

Sunplus Sunmidiar offers a great convenience for users to process MIDI data and generates the required melody binary file and tone color library. For SACM-MS02 on SPCE, Sunmidiar can walk user through the tone color making to MIDI processing. Sunmidiar also supports ICE downloading to facilitate composing MS02 applications. For the Sunmidiar operation details, please refer to "Sunmidiar user manual".

Here are the easy step-by-step instructions

1. Open a new project

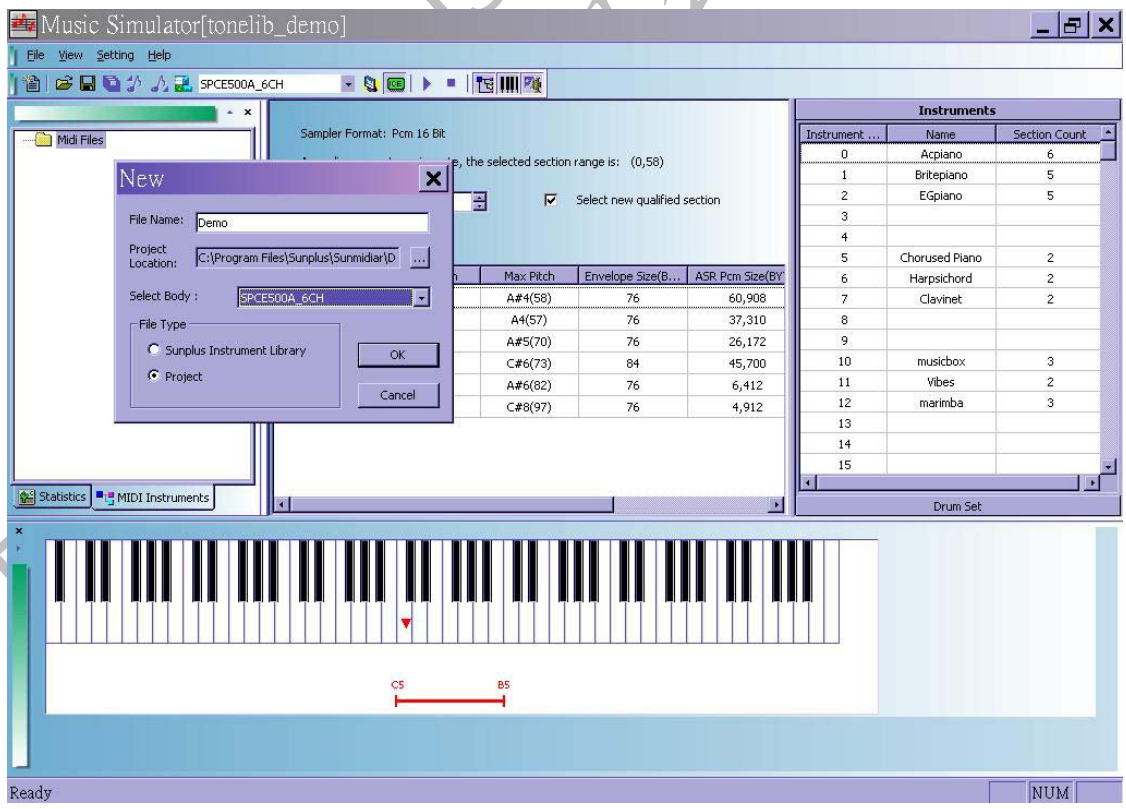
1.1 menu--> file -> new --> dialogue pop up

1.2 Type in user project name

1.3 Select project path,

1.4 Set body type as SPCE061

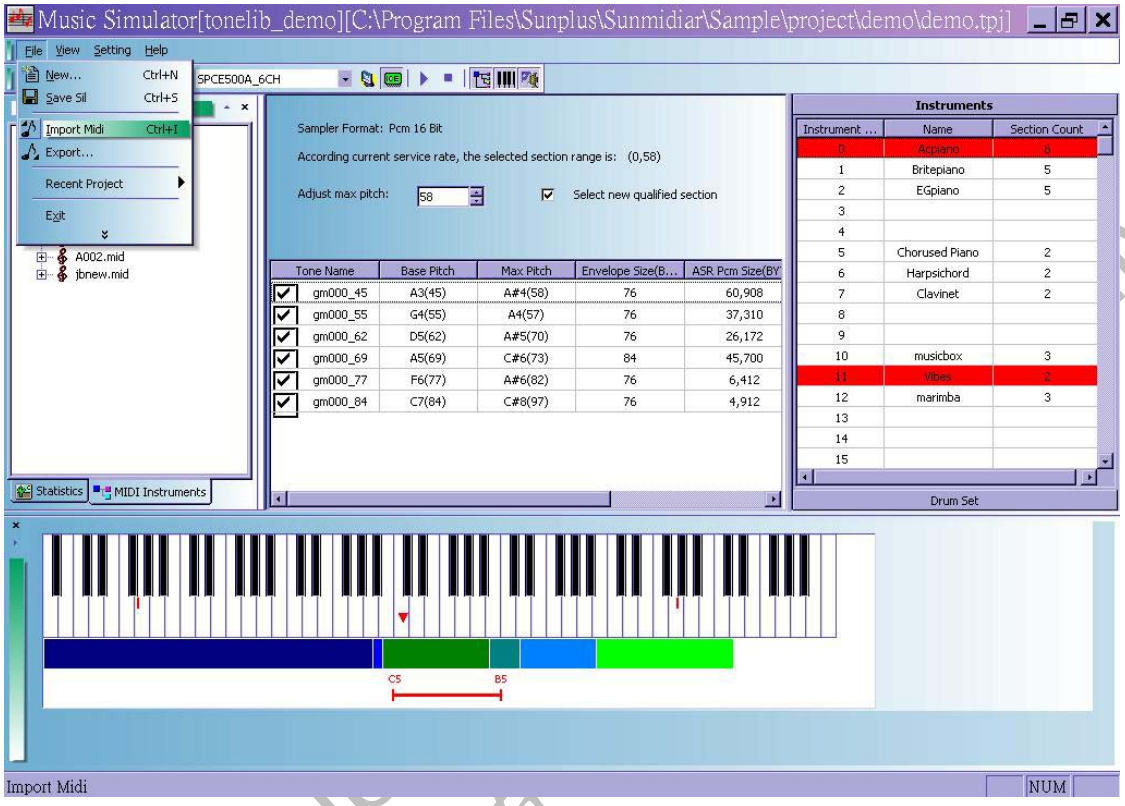
1.5 Check file type as project



2. Import MIDI

2.1 menu--> import MIDI --> dialogue pop up

2.2 choose file type as midi and select the MIDI file.

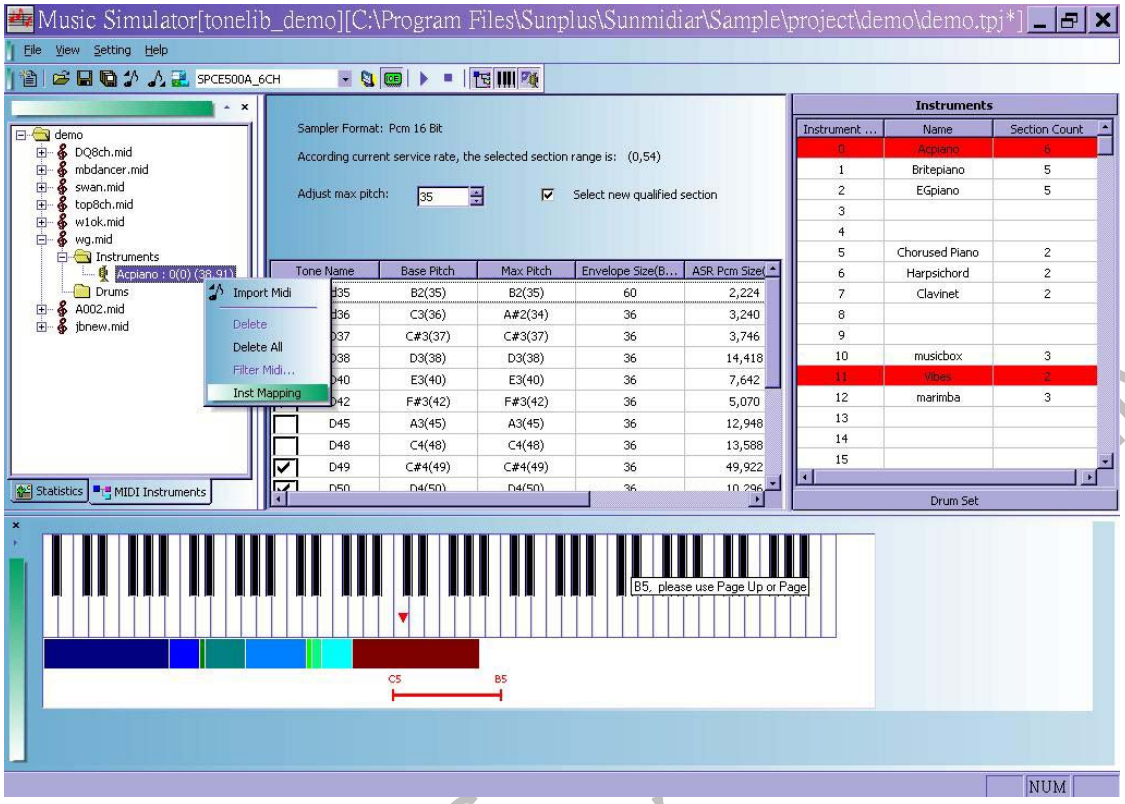


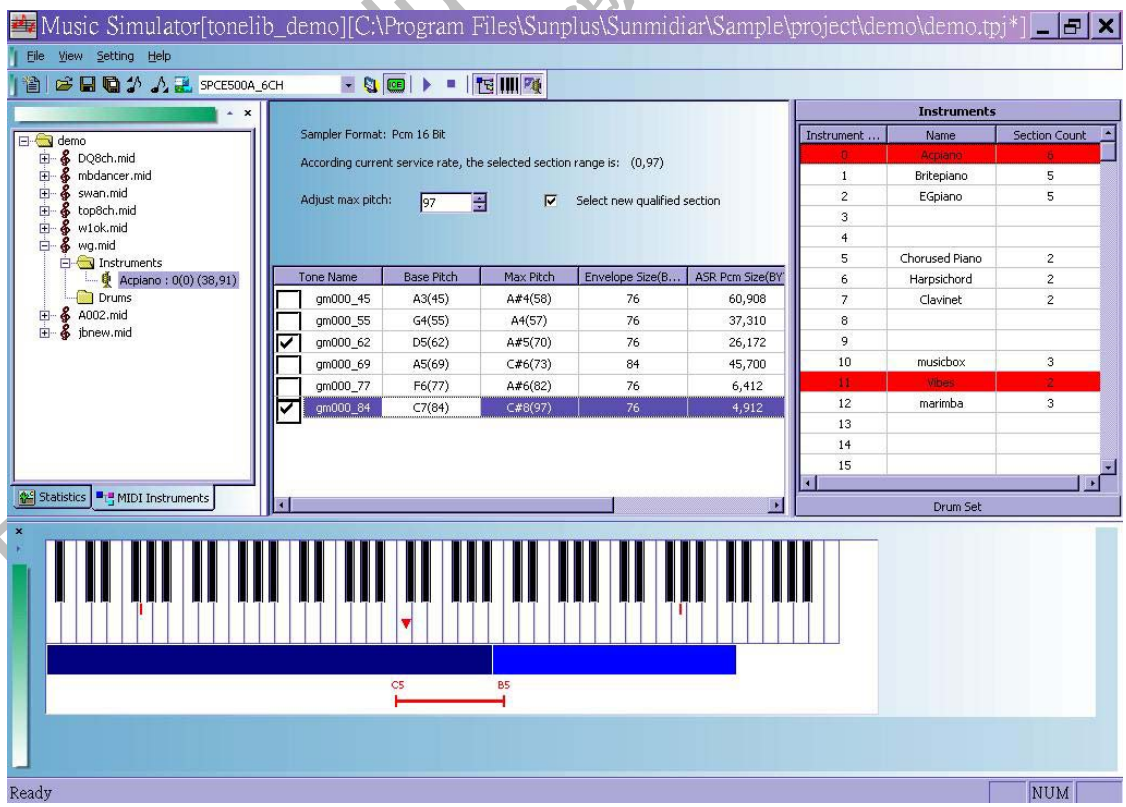
3. Edit MIDI

3.1 Go to the MIDI instruments window on the left and unfold the MIDI, instruments and drums

3.2 Check out if all instruments and drums are mapped to a instrument that is provided in the right window(Instruments and drum set), Instrument mapping can be done by left click on instrument under MIDI(e.g. Acpiano) , choose the Inst_Mapping and map it to an existing instrument in the right window

3.3 check the edit window in the middle and see if the each section of each instrument is well adjusted to cover the pitch range shown on the piano key below. The pitch range is shown by 2 short, vertical, red lines on the lower piano key.





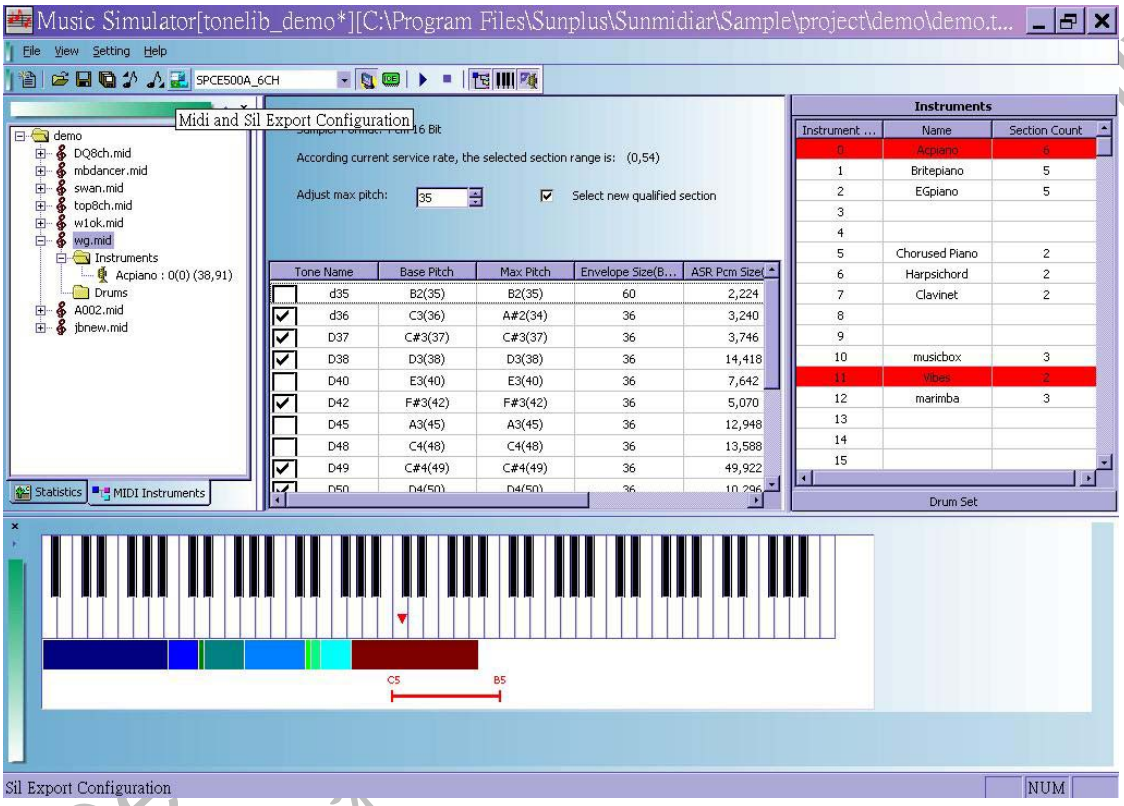
4. Export MIDI binary file and tone color library

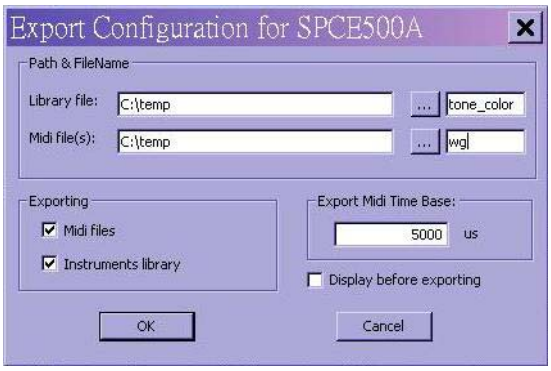
4.1 chick on the seventh icon, "MIDI and Sil export configurations" on the tool bar

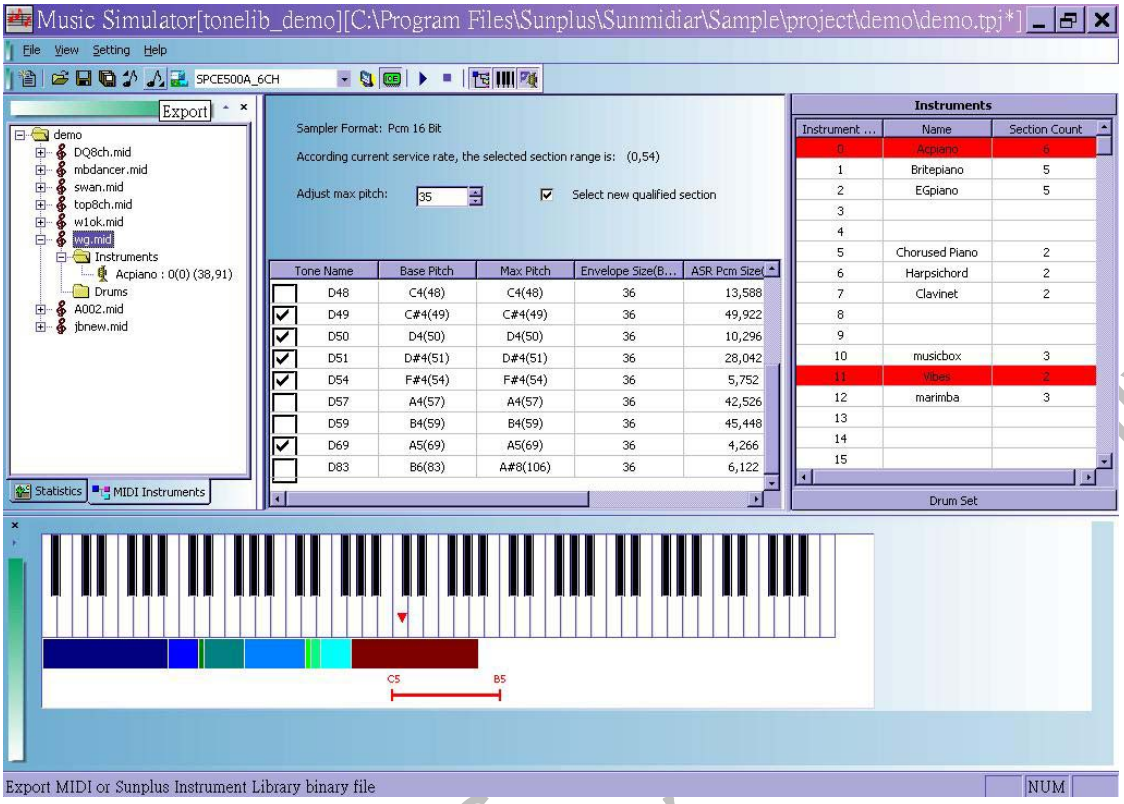
4.2 Select the export folder and give names to the export file. Make sure that in Exporting group box, both MIDI files and instruments library are checked.

4.3 Click on OK and then click on PC icon on tool bar. The PC icon is on the right of the IC body (SPCE061A) list box

4.4 Chick on the sixth icon , "Export" on tool bar to export binary files.







5. Done

The Song binary file and tone library file can now be used in MS02 project.

9 API V.S. RESOURCE

The following list shows the hardware resources used by SACM APIs. For example, the value of hardware port, such as [P_TimerA_Ctrl], [P_TimerA_Data]…etc, will be modified by SACM_A2000_Initial(). The [P_INT_Ctrl] and [P_TimerA_Data] will be changed by SACM_A2000_Play(), …etc. Moreover, all hardware requests in an API are public for user's reference in sacmv32.asm. For instance, the function F_SACM_A2000_Init_ is hardware setting for F_SACM_A2000_Initial.

Hardware (Unit: Word)

Functions	Used Register
SACM_A1600_Initial() SACM_A2000_Initial() SACM_A3200_Initial() SACM_2Ch_A3200_Initial() SACM_S480_Initial() SACM_S530_Initial() SACM_S240_Initial() SACM_S200_Initial() SACM_MS01_Initial() SACM_MS02_Initial() SACM_DVR_Initial()	[P_SystemClock] [P_TimerA_Ctrl] / [P_TimerB_Ctrl] [P_TimerA_Data] / [P_TimerB_Data] [P_DAC_Ctrl] [P_INT_Clear]
SACM_A1600_Play() SACM_A2000_Play() SACM_A3200_Play() SACM_A3200_Ch1_Play() SACM_A3200_Ch2_Play() SACM_S530_Play() SACM_S480_Play() SACM_S240_Play() SACM_S200_Play() SACM_MS01_Play() SACM_MS02_Play() SACM_MS01_InitDecoder() SACM_DVR_Play() SACM_DVR_InitDecoder()	[P_INT_Ctrl] [P_INT_Clear] [P_TimerA_Data]
SACM_DVR_Record() SACM_DVR_InitEncoder()	[P_ADC_Ctrl] [P_TimerA_Data] [P_INT_Ctrl]
SACM_A1600_Stop() SACM_A2000_Stop() SACM_A3200_Stop()	[P_ADC_Ctrl]

Functions	Used Register
SACM_A3200_Ch1_Stop() SACM_A3200_Ch2_Stop() SACM_S530_Stop() SACM_S480_Stop() SACM_S240_Stop() SACM_S200_Stop() SACM_MS01_Stop() SACM_MS02_Stop() SACM_DVR_Stop()	
SACM_A1600_ServiceLoop() SACM_A2000_ServiceLoop() SACM_S530_ServiceLoop() SACM_S480_ServiceLoop() SACM_S240_ServiceLoop() SACM_S200_ServiceLoop() SACM_MS01_ServiceLoop() SACM_MS02_ServiceLoop() SACM_MS01_Decoder() SACM_DVR_ServiceLoop() SACM_DVR_Encoder() SACM_DVR_Decoder()	[P_INT_Ctrl] [P_INT_Clear]
F_ISR_Service_SACM_A1600 F_ISR_Service_SACM_A2000 F_ISR_Service_SACM_A3200 F_ISR_Service_SACM_A3200_Ch1 F_ISR_Service_SACM_A3200_Ch2 F_ISR_Service_SACM_S530 F_ISR_Service_SACM_S480 F_ISR_Service_SACM_S240 F_ISR_Service_SACM_S200 F_FIQ_Service_SACM_MS01 F_ISR_Service_SACM_MS02 F_ISR_Service_SACM_DVR*	[P_DAC1] [P_DAC2] [P_ADC]

10 Resources List of SACM algorithm

10.1 TABLE 1: RAM Size (Unit: Decimal Word)

	IRAM	ISRAM	RAM	SRAM	ORAM	OSRAM
A1600					371	
A2000	-	-	-	-	763	-
A3200					21	
A3200 2Ch					41	
S530					357	
S480/S720	-	-	-	-	425	-
S240	-	-	-	-	145	-
S200					1024	
MS01	4	-	-	-	60	63
MS02		**-**	**-**	**-**	**291**	
DVR	-	-	-	-	1551	-

Note: DVR includes both A2000 Encoder and A2000 Decoder algorithm

10.2 TABLE 2: ROM Size (Unit: Decimal Word)

	TEXT	CODE	DATA	USER DEFINE
A1600		~7.8K		
A2000	-	~3.4K	-	-
A3200		~0.9K		
A3200 2Ch		~1.4K		
S530		~8.7K		
S480/S720	-	~2.7K	-	-
S240	-	~2.3K	-	-
S200		~7.5K		
MS01	-	~4K	-	-
MS02	**642**	**3954**	**-**	**-**
DVR	-	~6.2K	-	-

10.3 TABLE 3: Hardware Resources VS Library

	Interrupt	Timer Setting	Audio
A1600	TMA FIQ	16 KHz	DAC
A2000	TMA FIQ	16 KHz	DAC
A3200	TMA FIQ	8 KHz ~12KHz	ADC/DAC
A3200 2Ch	TMA IRQ1, TMB IRQ2	8 KHz ~12KHz	ADC/DAC
S530	TMA FIQ	8 KHz~12 KHz	DAC
S480/S720	TMA FIQ	16 KHz	DAC
S240	TMA FIQ	20 KHz	DAC
S200	TMA FIQ	16 KHz	DAC
MS01	TMA FIQ, TMB IRQ2, 1KHz IRQ4	16KHz/20KHz/24KHz +1KHz(ADPCM)	DAC
MS02	TMA IRQ1	8K/10K/12K/16K/20K/24K/ 28K/32K/36K/40 KHz	DAC
DVR	TMA FIQ	16 KHz(Play) /20 KHz(Rec)	ADC/DAC

10.4 TABLE 4: CPU Usage Rate (approximate)

	SPCE500A CPU Usage Rate (24 MHz)	SPCE061A CPU Usage Rate (49 MHz)
A1600	52%	20%
A2000	62%	24%
A3200	16%	6%
A3200 2Ch	32%	12%
S530	39%@8KHz(5.3Kbps) 48%@10KHz(6.6Kbps) 57% @12KHz(7.9Kbps)	14%@8KHz(5.3Kbps) 19%@10KHz(6.6Kbps) 24% @12KHz(7.9Kbps)
S480/S720	31%@8KHz(4.8Kbps) 34% @8KHz(7.2Kbps)	12%@8KHz(4.8Kbps) 13% @8KHz(7.2Kbps)
S240	55%	22%
S200	90%	40%
MS01	32% 1Ch@16KHz + 32% 2Ch ADPCM 38% 2Ch@16KHz + 32% 2Ch ADPCM 44% 3Ch@16KHz + 32% 2Ch ADPCM 52% 4Ch@16KHz + 32% 2Ch ADPCM 47% 1Ch@24KHz + 32% 2Ch ADPCM 57% 2Ch@24KHz + 32% 2Ch ADPCM 67% 3Ch@24KHz + 32% 2Ch ADPCM 77% 4Ch@24KHz + 32% 2Ch ADPCM	13% 1Ch@16KHz + 12% 2Ch ADPCM 16% 2Ch@16KHz + 12% 2Ch ADPCM 19% 3Ch@16KHz + 12% 2Ch ADPCM 22% 4Ch@16KHz + 12% 2Ch ADPCM 20% 1Ch@24KHz + 12% 2Ch ADPCM 24% 2Ch@24KHz + 12% 2Ch ADPCM 29% 3Ch@24KHz + 12% 2Ch ADPCM 34% 4Ch@24KHz + 12% 2Ch ADPCM
DVR	84%	36%

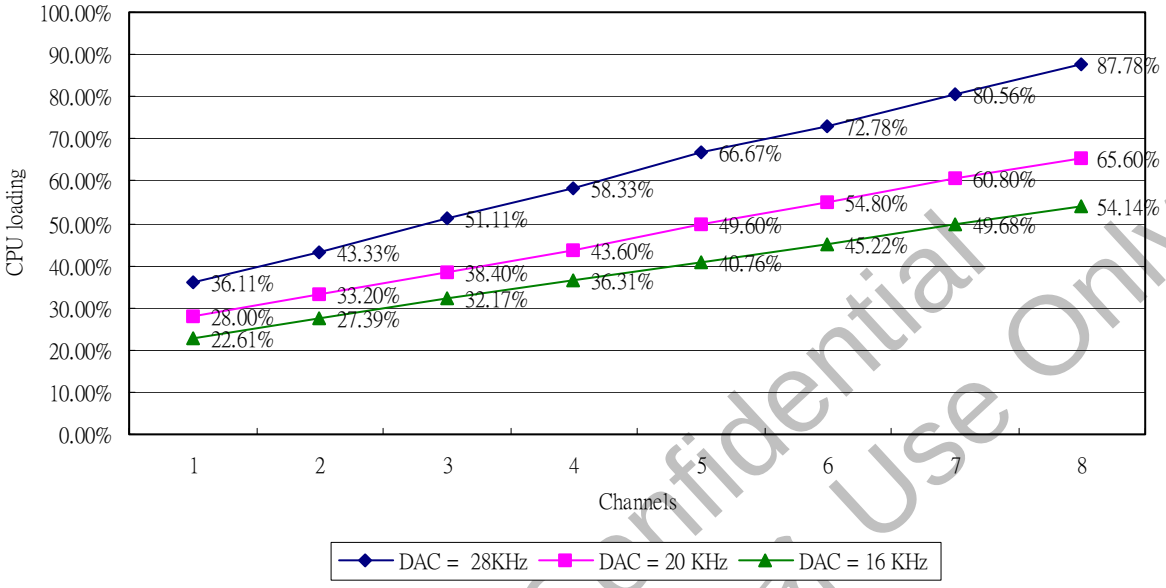
Note: DVR is including both A2000 Encoder and A2000 Decoder algorithm

MS02 (Maximum : 8 channels) Tempo = 250

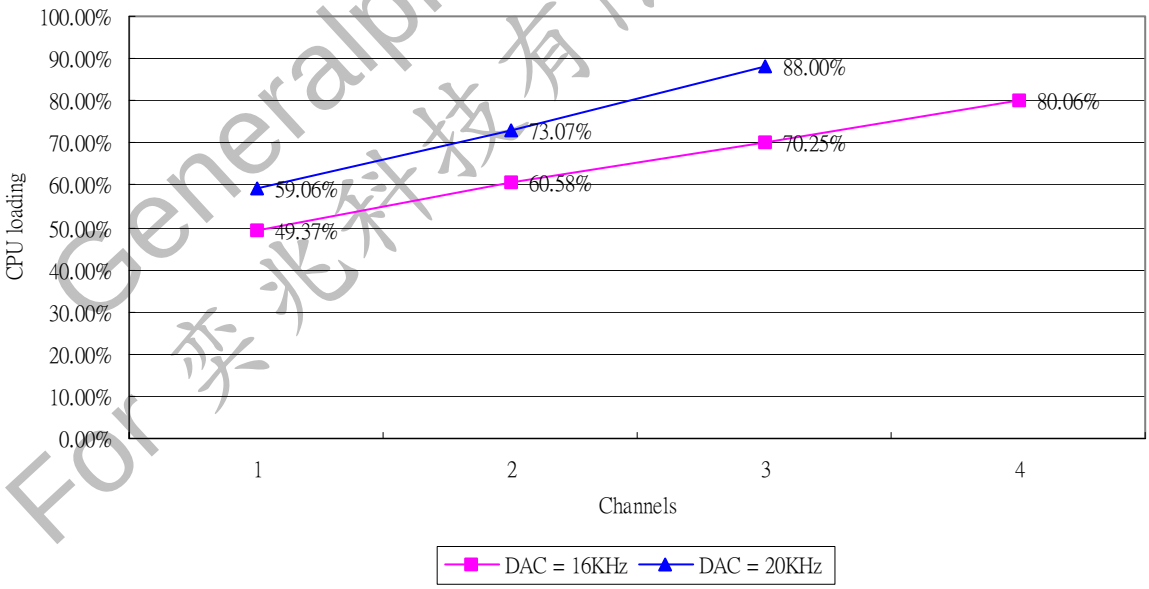
SPCE061A CPU loading for MS02 at 49.152MHz			
Channels	DAC = 28KHz	DAC = 20 KHz	DAC = 16 KHz
1	36.11%	28.00%	22.61%
2	43.33%	33.20%	27.39%
3	51.11%	38.40%	32.17%
4	58.33%	43.60%	36.31%
5	66.67%	49.60%	40.76%
6	72.78%	54.80%	45.22%
7	80.56%	60.80%	49.68%
8	87.78%	65.60%	54.14%

SPCE500A CPU loading for MS02 at 24.576MHz		
Channels	DAC = 20KHz	DAC = 16KHz
1	59.06%	49.37%
2	73.07%	60.58%
3	88.00%	70.25%
4	N/A	80.06%

CPU loading MS02 on SPCE061A



CPU loading for MS02 on SPCE500A



MS02 Lite (Maximum : 4 channels) Tempo = 250, DAC = 20KHz

Channels	SPCE061A at 49.152MHz	SPCE500A at 24.576MHz
1	22.80%	47.62%
2	28.60%	60.12%
3	33.20%	74.21%
4	39.20%	85.71%

Note:

1. MS02.lib is an 8 channel MS02 library; MS02_Lite is a 4channel MS02 library specifically for concurrent algorithms. User can opt to MS02 or MS02_Lite for SPCE061A or SPCE500A based on the application need. Fro example, for 4-channel application, MS02 Lite is more desirable than MS02 since it takes less CPU loading. For SPCE500, it is possible to run MS02.lib at up to 6 channels if the tempo is slow and DAC rate is set low. The measure of actual CPU load should be an ad hoc.
2. The CPU loading of MS02_Lite.lib is lighter than MS02.lib in the same condition but it can deliver only up to 4 polyphonic channels.
3. CPU loading is measured by timing the service loop and the ISR service routine and then user calculate its percentage in the certain amount of time.

4.

10.5 TABLE 5: Timing Limitation (approximate)

	Service Loop Time Limit (24MHz,u'nSP v1.0)	Service Loop Time Limit (49MHz,u'nSP v1.1)
A1600	6 ms / 16 ms	1.8 ms / 16 ms
A2000	1.92ms / 4ms	0.64 ms / 4ms
A3200	20 us / 125 us	8 us / 125 us
A3200 2Ch	40 us / 125 us	16 us / 125 us
S530	2. 4ms / 7.5 ms (5.3Kbps) 2.4ms / 5.8 ms (6.6Kbps) 2.4ms / 5.0 ms (7.2Kbps)	0.88ms / 7.5 ms (5.3Kbps) 0.90ms / 6 ms (6.6Kbps) 0.96ms / 5 ms (7.2Kbps)
S480	1.4ms / 7.4 ms	0.52ms / 7.5 ms
S720	1.2 ms / 5.2ms	0.5ms / 5ms
S240	17 ms / 40ms	16ms / 100ms
S200	13 ms / 15ms	4.3ms / 15ms
MS01	1.6ms / 84ms (tempo = 320)	64us / 22ms (tempo = 320)
DVR	39ms / 48 ms	12ms / 48 ms

Note:

1. The first number is the time taken to execute service loop, second number is the maximum interval in which the program have to execute service loop. These figures are measured with SACM v32.lib
2. For example, the F_SACM_A2000_ServiceLoop must be called each 1.33ms in user's main loop.

```
main()
{
    SACM_MS02_Initial(MS02_DAC_28K, 6);      // Play rate 28KHz, 6 channels
    SACM_MS02_Play(0,3,3);                   // Song 0, DAC1+DAC2,Ramp up + Ramp Down
    While(1)
    {
        User_Function();
        …
        SACM_MS02_ServiceLoop(); <= Go here in each 50 us
    }
}
```

MS02 Service loop time limit

Service loop Time Limit on SPCE061A			
Channels	DAC = 28KHz	DAC = 20 KHz	DAC = 16 KHz
1	11us / 36us	11.2us / 50us	11.2us / 62.8us
2	14us / 36us	14us / 50us	14us /62.8us
3	17us / 36us	16.8us / 50us	16.8us / 62.8us
4	19.8us / 36us	19.6us / 50us	19.6us / 62.8us
5	23us / 36us	22.8us / 50us	22.8us / 62.8us
6	25.6us / 36us	25.6us / 50us	25.6us / 62.8us
7	28.6us / 36us	28.8us / 50us	28.8us / 62.8us
8	31.4us / 36us	31.6us / 53us	31.6us / 62.8us

Service loop Time Limit on SPCE500A		
Channels	DAC = 20KHz	DAC = 16KHz
1	28.8us / 50.8us	28.8us/63us
2	36.4us / 50.8us	36us/62.4us
3	44us / 50us	43.2us/62.5us
4		50us/62.4us

MS02 Lite Service loop time limit

MS02 Lite Service loop Time Limit (DAC = 20KHz)		
Channels	SPCE061A at 49.152MHz	SPCE500A at 24.576MHz
1	8.4us / 50us	22us / 50us
2	11.5us / 50us	28.8us / 50us
3	14us / 50us	36.4us / 50us
4	17.2us / 50us	43.2us / 50us

Note: The service-loop timing varies as the DAC rate changes.

10.6 TABLE 6: Name of Overlap RAM in the library

	Overlap SRAM definition	Overlap RAM definition
A1600	OVERLAP_A1600_SRAM_BLOCK(ORAM)	OVERLAP_A1600_RAM_BLOCK(ORAM)
A2000	OVERLAP_A2000_SRAM_BLOCK(ORAM)	OVERLAP_A2000_RAM_BLOCK(ORAM)
A3200	-	OVERLAP_A3200_RAM _BLOCK(ORAM)
2Channel A3200	OVERLAP_A3200_2CH_SRAM_BLOCK (ORAM)	-
S530	-	OVERLAP_S530_RAM_BLOCK(ORAM)
S480	-	OVERLAP_S480_RAM_BLOCK(ORAM)
S240	-	OVERLAP_S240_RAM_BLOCK(ORAM)
S200	-	OVERLAP_S200_RAM_BLOCK(ORAM)
MS01	OVERLAP_MS01_SRAM_BLOCK (OSRAM)	OVERLAP_MS01_RAM_BLOCK(ORAM)
MS02		OVERLAP_MS02_RAM_BLOCK(ORAM)
DVR	OVERLAP_DVR_SRAM_BLOCK(ORAM)	OVERLAP_DVR_RAM_BLOCK(ORAM)

Note: see *.map in detail