SPCE SACM Library User's Manual v3.0

(For SACM V40)

03/30/2004

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

1.1 Document History

Revision	Date	By	Remark
V3.0	03/30/2004	Ray Cheng	Add DVR1600 (A1600 encoder and decoder) Add speed control function in A1600, S480/S720, S530, S200 and DVR1600 Remove A2000, A3200, S240
V2.0	12/26/2002	Arthur Shieh	Add A1600, A3200, S200, S530 Add new features to MS01
V1.0	12/26/2001	Andy Hsu Revised by Michael Lin	New release

1.2 Library History

Revision	Date	By	Remark
V40a	03/30/2004	Ray Cheng	SACM library is separated to several individual libraries. Include A1600, A1600_SC, S480/S720, S480_SC/S720_SC, S530, S530_SC, S200_SC, MS01, DVR1600, DVR1600_SC. Where SC = speed control.
V32f	05/16/2003	Arthur Shieh Rison Lo	Modified FIR_MOV style Add C ISR API
V32e	04/16/2003	Arthur Shieh Rison Lo	Add MS01+S200, S240, S480,S530 Modify SACM library for concurrent
V32	12/26/2002	Arthur Shieh, Adamcar Tseng & Andy Hsu	Add A1600, A3200, S200, S530 New Auto/manual mode New background service loop New MS01 features.(Play note, Event….)
V1.0	05/27//2001	Andy	Set .OSRM section for 4 algorithms
V0.9B	05/20//2001	Andy	Add Ramp Up/Dn control for speech playing
V0.9A	04/12//2001	Andy	Add Manual Mode for SACM-A2000
V0.8A	05/08/2001	Andy	Add Queue Interface for DVR(Record/Playback)
V0.3A	01/11/2001	Andy	SACM-DVR new functions
V0.2B	12/14/2000	Andy	New version for A2000, S480/720, S240, MS01, DVR (With PC Play function
V0.2A	12/14//2000	Andy	New version for A2000, S480/720, S240, MS01, DVR (No PC Play function)
V0.1	11/15//2000	Andy	Fix a2000 ending bug, add speech status function
V0.0	09/20//2000	Andy	First release

2 Type of Speech Compression Algorithm

2.1 Summary

Audio

Present Algorithm Title	Data rate	Application
SACM-A1600	10 / 12 / 14 / 16 / 20 / 24 Kbps	Audio
SACM-A1600-SC	10 / 12 / 14 / 16 / 20 / 24 Kbps	Audio with Speed Control

Speech

Present Algorithm Title	Data rate	Application
SACM-S200-SC	0.8K / 0.9K / 1K / 1.2K / 1.4K / 1.6K / 1.8k / 2K / 2.4K / 2.8K / 3.36Kbps	Speech
SACM-S480/S720	4.8 / 7.2 Kbps	Speech
SACM-S480/S720-SC	4.8 / 7.2 Kbps	Speech with Speed Control
SACM-S530	5.3K / 5.96K / 6.63K / 7.29K / 7.95 Kbps	Speech
SACM-S530	5.3K / 5.96K / 6.63K / 7.29K / 7.95 Kbps	Speech with Speed Control

Melody

Present Algorithm Title	Data rate	Channel	Application
SACM-MS01	N/A	6 (4 FM+ 2 ADPCM)	Music Synthesizer

Recording

Present Algorithm Title	Data rate	Application
SACM-DVR1600	10 / 12 / 14 / 16 / 20 / 24 Kbps	Recording. A1600 decoder and encoder.
SACM-DVR1600-SC	10 / 12 / 14 / 16 / 20 / 24 Kbps	Recording with Speed Control. A1600 decoder and encoder. Speed control only for decoder.

2.2 Naming convention

SACM-Xnnn [-SC]

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.

SC: Speed Control.

DVR: Digital Voice Recording

Example: SACM A1600 stands for Sunplus audio algorithm with nominal data rate of 16Kbps.The actual data rate depends on the options provided and the sampling rate adopted.

2.3 Algorithm supported by SPCE

Due to the RAM size and CPU speed limitations, some SPCE series cannot support all SACM algorithms. The following table shows the available SACM algorithm to SPCE series.

Algorithm	SPCE500A/380A/250A/120A	SPCE040A/060A/061A	SPCE1070A/1080A
SACM-A1600	YES	YES	YES
SACM-A1600-SC	NO	YES	NO
SACM-S530	YES	YES	YES
SACM-S530-SC	NO	YES	NO
SACM-S480/S720	YES	YES	YES
SACM-S480/S720-SC	NO	YES	NO
SACM-S200-SC	NO	YES	NO
SACM-MS01	YES	YES	YES
SACM-DVR1600	NO	YES	NO
SACM-DVR1600-SC	NO	YES	NO

2.4 Difference between SACM V40 and SACM V32

Unlike SACM v32, which includes several libraries as a whole, the SACM v40 library is separated into 10 subsets and each corresponds to a different SACM algorithm. That is, each of these subsets is an individual SACM library. You just include whatever you need into your program project. They are SACM_A1600_V40a.lib, SACM_A1600_SC_V40a.lib, SACM_S530_V40a.lib, SACM_S530_SC_V40a.lib, SACM_S480_V40a.lib, SACM_S480_SC_V40a.lib, SACM_S200_SC_V40a.lib, SACM_MS01_V40a.lib, SACM_DVR1600_V40a.lib and SACM_DVR1600_SC_V40a.lib, where "SC" means Speed Control. You can change the playing speed without altering the pitch of the sound. That is, A1600, S480, S530, S200 and DVR1600 support speed control function. SACM V40 library adds DVR1600 (A1600 encoder and decoder), but removes the S240, A2000, A3200 and DVR (A2000 encoder).

For SACM v32, there are different sets of APIs for Auto mode and Manual mode while in SACM v40 library. They share the same API architecture between Auto mode and Manual mode but controlled by the parameter of the playback API. Please refer to the SACMxxxx API's description in the following chapters for details. It is very convenient for users to develop their programs and easy to manage their projects.

Please note that in a project using SACM v40, user cannot include one algorithm with speed control (Ex:SACM_S530_SC.lib) and the same algorithm without speed control(Ex:SACM_S530.lib) simultaneously.

Because they will have the same APIs only except speed control API and will not be able to compile successfully. For example, user cannot link SACM_S480_V40.lib and SACM_S480_V40_SC.lib simultaneously in one project. If users want to have speed control function in a project that originally do not have, they just remove the original library SACM_XXXX_V40x.lib which without speed control function and add the SACM_XXXX_SC_V40x.lib into the u'nSP IDE and then you can use speed control API in your program.

Important: The file format of SACM V40 library is different from SACMV32. Please use appropriate encoder tools. Please refer to the following list.

Algorithm	Encode Tool
SACM-A1600	DVR1600.exe
SACM-DVR1600	DVR1600.exe
SACM-S480/S720	S485372C.exe + Add_Header.exe
SACM-S530	S485372C.exe + Add_Header.exe
SACM-S200	S200.exe
SACM-MS01	Midi2Pop.exe, scfm.exe

3 Auto mode vs. Manual mode

For SACM playback, auto mode and manual mode are distinguished by the ways library fetches data. In auto mode playback, library fetches the speech data automatically from internal ROM. In manual mode playback, users have to handle the speech/song data fetch byte-by-byte, word-by-word or block-by-block. Manual mode can work with external devices such as CPU + external memory or CPU + CPU system configuration topologies.

For auto mode and manual mode designation, the auto mode and manual mode are designated in the play function. For example, SACM_S200_Play(-1, DAC1+DAC2,Ramp_Up+Ramp_Dn); –1 as speech index is for manual mode and therefore the algorithm initialization would not have to specify the operating mode.

In new version of SACM library, the new manual mode requires the sacm_XXXX_user.asm, where XXXX is algorithm name. For example, sacm_A1600_user.asm. User has to implement the functions of F_USER_XXXX_SetStartAddr , F_USER_XXXX_GetData and F_USER_XXXX_WriteData (only for DVR1600) in order to execute manual mode. These functions are call-back functions for library to access memory storage.

For DVR1600, user can find the call-back function, F_USER_DVR1600_WriteData in sacm_DVR1600_user.asm to be implemented. User can actually hook DVR1600 to various types of external memory storage for audio recording through the call-back functions, F_USER_DVR1600_WriteData and F_USER_DVR1600_GetData. The DVR1600 can be viewed as the encoder of A1600 and thus user can use A1600 manual mode to playback DVR1600 recording.

4 Service loop

In new SACM library, it supports both foreground and background service loops. In the old fashion foreground service loop, users have to put service loop in main. Main loop will keep entering the service loop. Inside the SACM service loop, there will be a mechanism to determine if any task should be carried on. Some overheads are produced inevitably. The amount of overhead varies depends on the payload of CPU.

For background service loop, the program checks if the service-loop tasks should take place right after exiting interrupt. It means, program checks the demand for service loop tasks more efficiently and therefore grants users better flexibility to arrange user-defined tasks in main. Since the background service loop eliminates some unnecessary overheads, the background service loop programming can save more CPU resources and boost its throughput.

Both methods can facilitate building a SACM project by user.

Example:

<u>Foreground service loop</u>:

In main.c :

```
int main()
{
        System_Initial();
        SACM_A1600_Initial();
        SACM_A1600_Play (0, DAC1+DAC2, Ramp_Up+Ramp_Dn);
        while(1)
        {
                System_ServiceLoop();                   // Service loop for watchdog clear
                SACM_A1600_ServiceLoop();               // Foreground Service loop
        } // end of while(1)
       return 0;
} // end of main
```

In isr.asm:

```
_FIQ:
        push R1, R5 to [SP];                    // save registers
        call F_ISR_Service_SACM_A1600;          // interrupt service routine
        R1 = C_FIQ_TMA                          // clear interrupt.
```

```
    [P_INT_Clear] = R1;
    pop R1, R5 from [SP];                   // restore registers
    reti;
```

Background service loop:

In main.c :

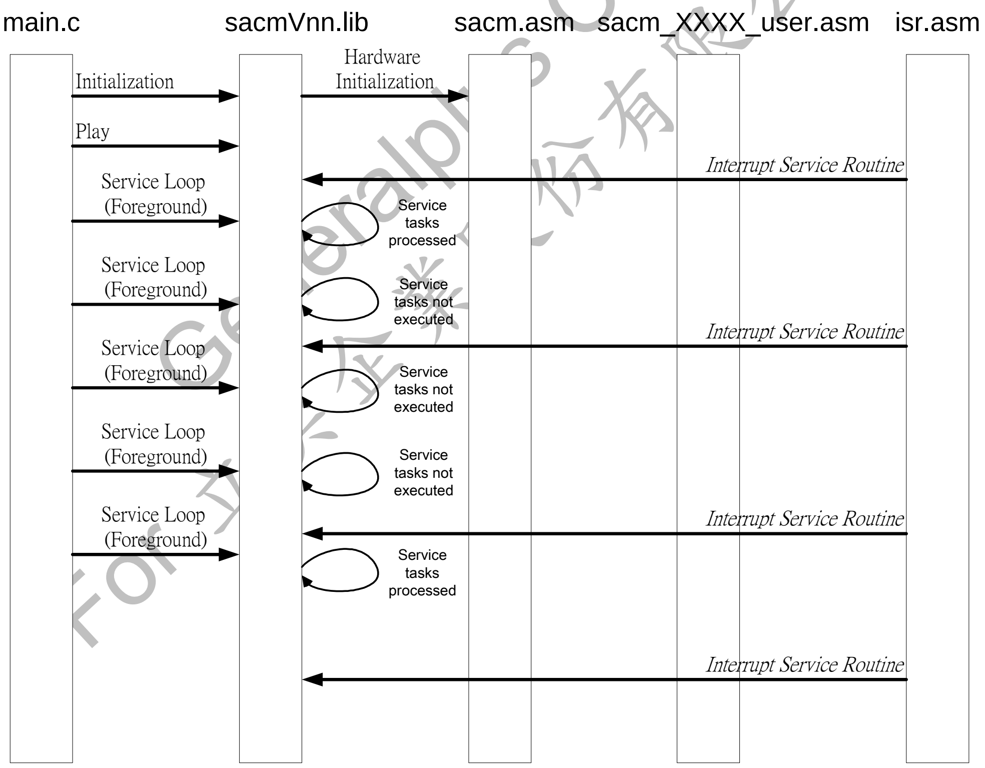
```
int main()
{
    System_Initial();
    SACM_A1600_Initial();
    SACM_A1600_Play (0,DAC1+DAC2,Ramp_Up+Ramp_Dn);
    while(1)
    {
        System_ServiceLoop();               // Service loop for watchdog clear
    } // end of while(1)
    return 0;
} // end of main
```

In isr.asm:

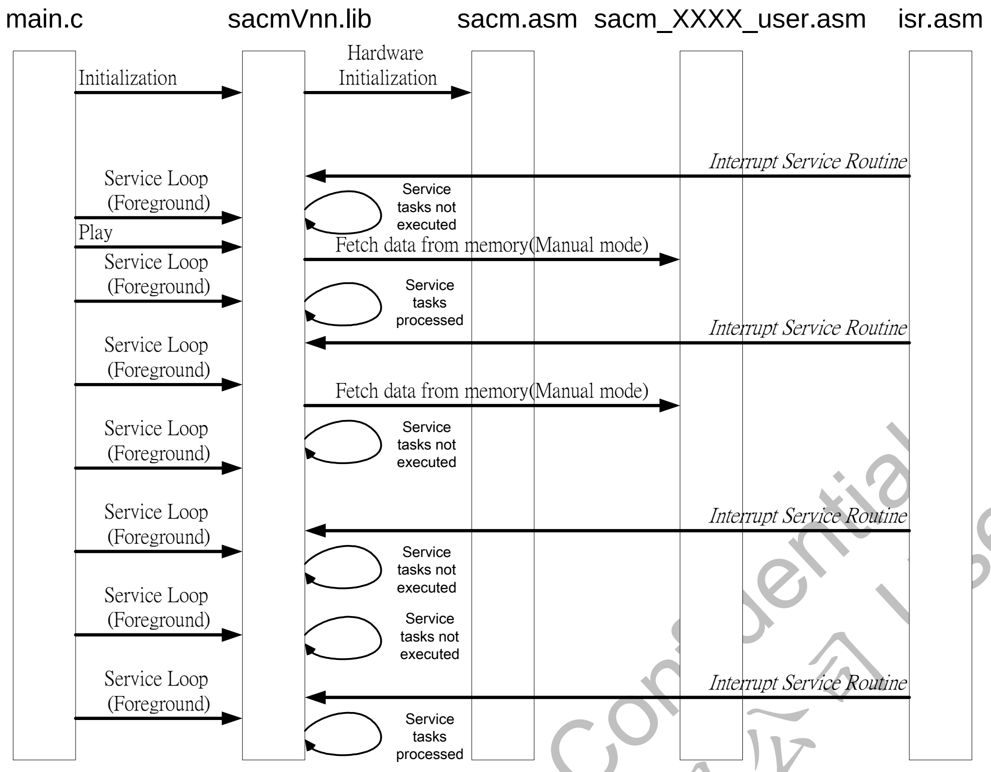
```
_FIQ:
    push R1, R5 to [SP];                    // save registers
    call F_ISR_Service_SACM_A1600;          // interrupt service routine
    // --------------------------------------------------------------
    // User interrupt routine must be placed before this line.
    // --------------------------------------------------------------
    R1 = F_SACM_A1600_ServiceLoop_ISR;      // Get function Address of background service loop
                                            // It restores registers r1~ r5 internally.
    push R1 to [SP];                        // push address to stack for PC to refer
    push SR to [SP];                        // push SR to stack
    R1 = C_FIQ_TMA;                         // clear interrupt.
    [P_INT_Clear] = R1;
    reti;
```

For SACMV40:

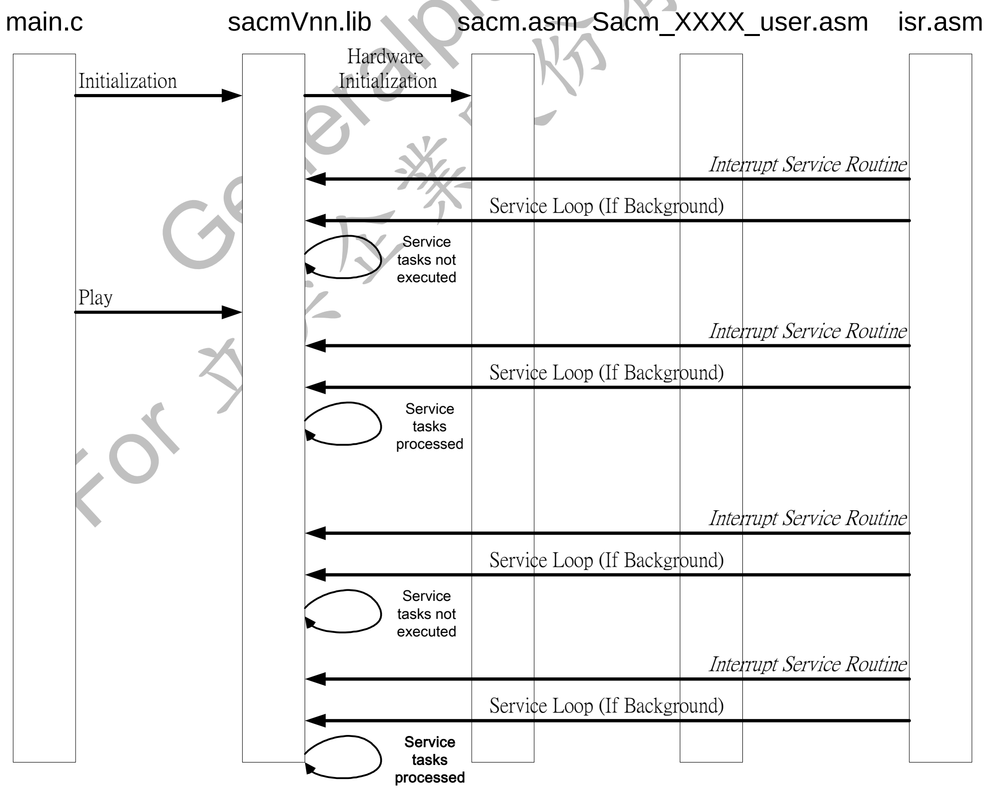
	Foreground Service Loop	Background Service Loop
S200	YES	YES
S480/S720	YES	YES
S480/S720-SC	YES	YES
S530	YES	YES
S530-SC	YES	YES
A1600	YES	YES
A1600-SC	YES	YES
DVR1600	YES	NO
DVR1600-SC	YES	NO
MS01	YES	YES


main.c
sacmVnn.lib
sacm.asm
sacm_XXXX_user.asm
isr.asm
Initialization
Hardware Initialization
Play
Interrupt Service Routine
Service Loop (Foreground)
Service tasks processed
Service Loop (Foreground)
Service tasks not executed
Interrupt Service Routine
Service Loop (Foreground)
Service tasks not executed
Service Loop (Foreground)
Service tasks not executed
Interrupt Service Routine
Service Loop (Foreground)
Service tasks processed
Interrupt Service Routine

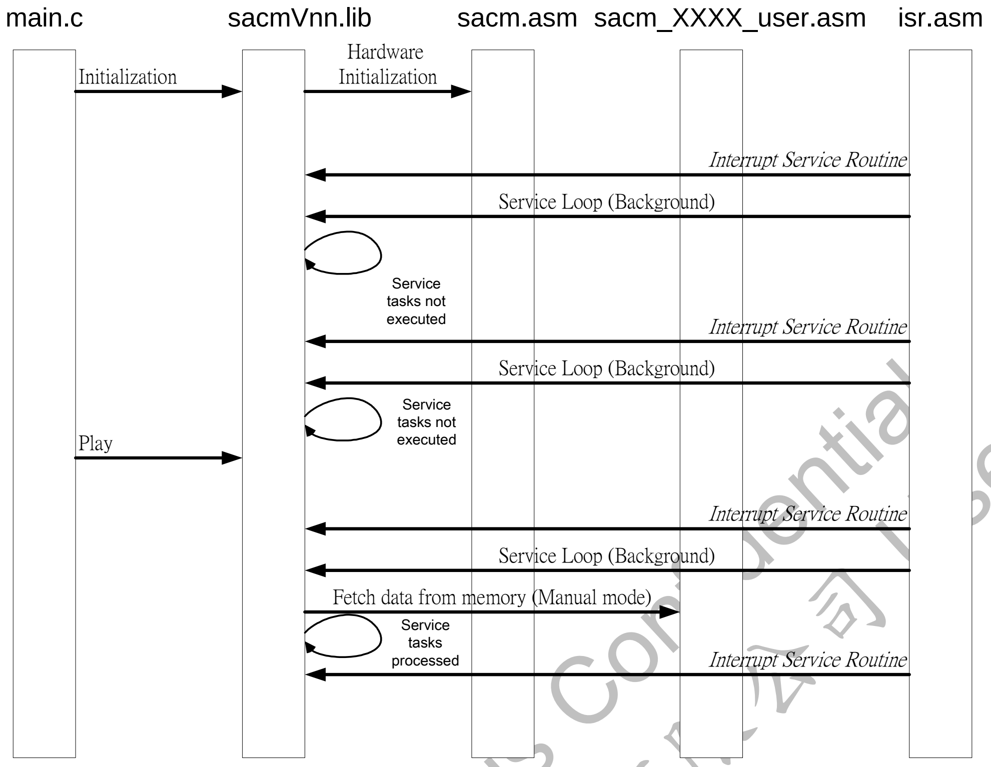
Timing diagram: Auto mode, Foreground service loop



Timing diagram: Manual mode, Foreground service loop



Timing diagram: Auto mode, Background service loop


main.c
sacmVnn.lib
sacm.asm
sacm_XXXX_user.asm
isr.asm
Initialization
Hardware Initialization
Interrupt Service Routine
Service Loop (Background)
Service tasks not executed
Interrupt Service Routine
Service Loop (Background)
Service tasks not executed
Play
Interrupt Service Routine
Service Loop (Background)
Fetch data from memory (Manual mode)
Service tasks processed
Interrupt Service Routine

Timing diagram: Manual mode, Background service loop

5 Memory Allocation

For each SACM algorithm, it is necessary to use a size of RAM blocks for encoding or decoding purposes. The RAM space taken can be shared among algorithms or with user application by aligning the RAM blocks manually. The memory allocation manifest can be found as the name "project_name.map" in the directory, "release" or "debug". The u'nSP IDE(v1.6 or later) also provides a convenient tool, memory map, to graphically list the memory space taken by each module, section, public function and variable.

The principle of sharing RAM is that as long as the algorithms or applications are not active simultaneously. The RAM block can be shared through ORAM or OSRAM section declaration. For details of RAM/ORAM/OSRAM section, please refer to "Sunplus u'nSP Assembly Tools User's Manual".

If user plans to build an application with both speech (S200, S480 or S530) and background music (MS01) up and running at the same time, the RAM allocation would also have to be separated as the section names suggest. In newer version of SACM library (V40 or later), each speech algorithm has a unique section name, which guarantees the simultaneity of speech and music synthesizer.

If user chooses more than one algorithm in the same project but the program is not going to run more than one algorithm at the same time, the advantage of ORAM section is to allow user to share the same physical memory block among different algorithms. User can use u'nSP IDE (Project→ Setting→ redefine) to align the ORAM address. If user is not satisfied with the manual allocation and wants to return to default compiler arrangement, users only have to delete the file, "project_name.lik" in project folder, and rebuild all projects. The memory allocation will be realigned based on default compiler rules.

The RAM block section definitions as follows:

Table: Name and size of Overlap RAM in the library

Overlap RAM definition		
Algorithm	**Overlap RAM Label**	**Size (word)**
A1600	**OVERLAP_DVR1600_RAM_BLOCK	319 (0x13F)
	OVERLAP_A1600_API_BLOCK	2 (0x2)
	**OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
*A1600-SC	**OVERLAP_DVR1600_RAM_BLOCK	781 (0x30D)
	OVERLAP_A1600_API_BLOCK	2 (0x2)
	**OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
	**OVERLAP_DVR1600_SPEED_RAM_BLOCK	529 (0x211)
DVR1600	OVERLAP_DVR1600_RAM_BLOCK	354 (0x162)
	OVERLAP_DVR1600_API_BLOCK	351 (0x23)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x 88)
*DVR1600-SC	OVERLAP_DVR1600_RAM_BLOCK	786 (0x312)
	OVERLAP_DVR1600_API_BLOCK	35 (0x23)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
	OVERLAP_DVR1600_SPEED_RAM_BLOCK	529 (0x211)
S530	OVERLAP_S530_RAM_BLOCK	169 (0xA9)
	OVERLAP_S530_API_BLOCK	2 (0x2)
	OVERLAP_S530_DM_BLOCK	228 (0xE4)
*S530-SC	OVERLAP_S530_RAM_BLOCK	613 (0x265)
	OVERLAP_S530_API_BLOCK	2 (0x2)
	OVERLAP_S530_DM_BLOCK	228 (0xE4)
	OVERLAP_S530_SPEED_RAM_BLOCK	529 (0x211)
S480	OVERLAP_S480_RAM_BLOCK	170 (0xAA)
	OVERLAP_S480_API_BLOCK	2 (0x2)
	OVERLAP_S480_DM_BLOCK	292 (0x124)
*S480-SC	OVERLAP_S480_RAM_BLOCK	614 (0x266)
	OVERLAP_S480_API_BLOCK	2 (0x2)
	OVERLAP_S480_DM_BLOCK	292 (0x124)
	OVERLAP_S480_SPEED_RAM_BLOCK	529 (0x211)
*S200-SC	OVERLAP_S200_RAM_BLOCK	718 (0x2CE)
	OVERLAP_S200_API_BLOCK	2 (0x2)
	OVERLAP_S200_DM_BLOCK	500 (0x1F4)
MS01	OVERLAP_MS01_RAM_BLOCK	318 (0x13E)
	OVERLAP_MS01_DM_BLOCK	130 (0x82)

*:XXXX_SC means that algorithm support speed control function.

**: A1600 is decoder of DVR1600. So, some A1600 overlap RAM labels are the same as DVR1600.

6 API for SACM-A1600

6.1 Hardware Dependent Function: Initializes SACM-A1600

6.1.1 Function: Initialize the A1600 library

Syntax:

C: void SACM_A1600_Initial(void)

ASM: call F_SACM_A1600_Initial

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A1600 decoder. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ with 16KHz sample rate.
2. The hardware setting is opened for user's reference (see F_SP_SACM_A1600_Init_ function in sacmvxxx.asm).
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program if for SPCE500A. It uses P_INT_Mask instead if for SPCE 061A/060A

6.2 Service Loop Functions: Service loop for SACM-A1600 decoding

6.2.1 Function: Foreground service loop:

Syntax:

C: void SACM_A1600_ServiceLoop(void);

ASM: call F_SACM_A1600_ServiceLoop

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

6.2.2 Function: Background service loop:

Syntax:

C: N/A

ASM: call F_SACM_A1600_ServiceLoop_ISR

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the A1600 code is allocated in page 0 to assure that program flow can proceed to F_SACM_A1600_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register. Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to "Sunplus u'nSP Assembly Tools User Manual" for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_A1600;          // Interrupt service routine.
 R1 = F_SACM_A1600_ServiceLoop_ISR;      // Background Service loop
                                         // for SACM A1600 playing
 push R1 to [SP];                        // push function address to stack
 push SR to [SP];                        // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

6.3 Playback Functions: Playback control

6.3.1 Function: Play a SACM-A1600 speech

Syntax:

C: void SACM_A1600_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM: R1 = Speech_Index

R2 = Channel

R3 = Ramp_Set

call F_SACM_A1600_Play

Parameters:

Speech_Index: -1: Manual Mode

0 ~ max. of speech index: Auto Mode

Channel: 1: To DAC1 only

2: To DAC2 only

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 ramp down

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark:

1. The data rate of SACM-A1600 can be compressed by 10/12/14/16/20/24Kbps. The data rate is selected by decoder automatically.
2. The Speech_Index is the speech sequence of T_SACM_A1600_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_A1600_SetStartAddr and F_USER_A1600_GetData in sacm_A1600_user.asm on users' own.
3. The F_ISR_Service_SACM_A1600 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16KHz after calling this subroutine.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

6.3.2 Function: Stop playing SACM-A1600 speech

Syntax:

C: void SACM_A1600_Stop(void);

ASM: call F_SACM_A1600_Stop

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: This function will not change the interrupt setting.

6.3.3 Function: Pause playing SACM-A1600 speech

Syntax:

C: void SACM_A1600_Pause(void);

ASM: call F_SACM_A1600_Pause

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: None

6.3.4 Function: Resume paused SACM-A1600 speech

Syntax:

C: void SACM_A1600_Resume(void);

ASM: call F_SACM_A1600_Resume

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: None

6.3.5 Function: Change the volume of SACM-A1600

Syntax:

C: void SACM_A1600_Volume(int Volume_Index)

ASM: R1 = Volume_Index
call F_SACM_A1600_Volume

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

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: None

6.3.6 Function: Get the status of the SACM-A1600 module

Syntax:

C: unsigned int SACM_A1600_Status(void);

ASM: call F_SACM_A1600_Status
[Retrun_Value] = R1

Parameters: None

Return Value: bit 0: 0: Speech ended
1: Speech Playing
bit 1-15: Reserved

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark: None

6.3.7 Function: Change the speed index of SACM-A1600

Syntax:

C: void SACM_A1600_Speed(int Speed_Index)

ASM: R1 = [Speed_Index]
call F_SACM_A1600_Speed

Parameters: Speed_Index: [-12..12], 0:Normal speed, -12:Min speed, 12:Max speed

Return Value: None

Library: <Sacm_A1600_SC_Vxxx.LIB>

Remark:

1. Min speed (Speed_Index = -12) is 1/2 of Normal speed (Speed_Index = 0).
2. Max speed (Speed_Index = 12) is twice of Normal speed (Speed_Index = 0).

6.4 ISR Functions: Interrupt service routine for SACM-A1600

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_A1600_Init_, in sacmvxxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM: call F_ISR_Service_SACM_A1600

Parameters: None

Return Value: None

Library: <Sacm_A1600_Vxxx.LIB>, <Sacm_A1600_SC_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)
2. The F_ISR_Service_SACM_A1600 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to place user-define function in the same FIQ or IRQ.

EX:

```
_FIQ:
        push  R1, R5 to [SP];
        call    F_ISR_Service_SACM_A1600
        call    F_User_ISR
        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop    R1, R5 from [SP]
        reti
```

6.5 User Functions: for SACM-A1600 playback in manual mode

6.5.1 Function: Set start address for SACM-A1600 speech data.

Syntax:

C: USER_A1600_SetStartAddr (,,)

ASM: call   F_USER_A1600_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_A1600_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. ROM, FLASH.

6.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: N/A

ASM: R1 = the start address of buffer to write

R2 = the length of data

call   F_USER_A1600_GetData

Parameters: R1 = the start address of buffer to write

R2 = the length of data

Return Value: User-defined

Library: sacm_A1600_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. ROM, FLASH.
3. This function is call-back function for SACM A1600 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

Example:

Play a SACM-A1600 speech with Auto mode or Manual mode.

(a). In main.c:

```
#include    "sacmv40.h"
#define     Manual_Mode_Index          -1
#define     Manual                     0
#define     Auto                       1
#define     DAC1                       1
#define     DAC2                       2
#define     Ramp_Up                    1
#define     Ramp_Dn                    2
int         Mode;
int         SpeechIndex = 0;
main()
{
    Mode = Auto;                                  // Play mode setting
    if(Mode == Auto)                              // Auto mode play
    {
        SACM_A1600_Initial();                     // Initialization
        SACM_A1600_Play(SpeechIndex, DAC1+DAC2, Ramp_Up+Ramp_Dn);// Play 1st speech
        While(1)
        {
            SACM_A1600_ServiceLoop();             // Service loop for decode
        } // end of While(1)
    } // end of if(Mode == Auto)
    if(Mode == Manual)                            // Manual mode play
    {
        SACM_ A1600_Initial();                    // Initialization
        USER_A1600_SetStartAddr(SpeechIndex);     // Set start address of speech data
        SACM_A1600_Play(Manual_Mode_Index, DAC1+DAC2, Ramp_Up+Ramp_Dn);
```

```
                {                                            // Manual mode play speech
                        SACM_A1600_ServiceLoop();            // Service loop for decode
                } // end of SACM_A1600_Play
        } // end of if(Mode == Manual)
} // end of main()
```

(b). In ism.asm:

```
_FIQ:
        push  R1, R5 to [SP];                                // push registers
        call    F_ISR_Service_SACM_A1600;                    // ISR
        R1 = C_FIQ_TMA;
        [R_INT_Clear] = R1;                                  // clear TimerA FIQ interrupt flag
        pop    R1, R5 from [SP];                             // pop registers
        reti;                                                // return
```

(c). In sacm_A1600_user.asm: (manual mode only)

```
_USER_A1600_SetStartAddr:
    // defined by user
    // F_USER_A1600_SetStartAddr are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;

F_USER_A1600_GetData:
    // defined by user
    // F_USER_A1600_GetData are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;
```

7 API for SACM-S200

7.1 Hardware Dependent Function: Initializes SACM-S200

7.1.1 Function: Initialize the S200 library

Syntax:

C: void SACM_S200_Initial (void);

ASM: call F_SACM_S200_Initial

Parameters: N/A

Return Value: N/A

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark:

1. This function initializes the SACM-S200 decoder. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ with 16KHz sample rate.
2. The hardware setting is opened for user's reference (see F_SP_SACM_S200_Init_: function in sacmvxxx.asm).
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program for SPCE500A. It uses P_INT_Mask instead for SPCE 061A/060A

7.2 Service Loop Functions: Service loop for SACM-S200 decoding

7.2.1 Function: Foreground service loop

Syntax:

C: void SACM_S200_ServiceLoop(void);

ASM: call F_SACM_S200_ServiceLoop

Parameters: None

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

7.2.2 Function: Background service loop:

Syntax:

C: N/A

ASM: Call F_SACM_S200_ServiceLoop_ISR

Parameters: N/A

Return Value: N/A

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the S200 code is allocated in page 0 to assure that program flow can proceed to F_SACM_S200_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register.   Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to “Sunplus u’nSP Assembly Tools User Manual” for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_S200;          // Interrupt service routine.
 R1 = F_SACM_S200_ServiceLoop_ISR;      // Background Service loop
                                        // for SACMS200 playing
 push R1 to [SP];                       // push function address to stack
 push SR to [SP];                       // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

7.3 Playback Functions: Playback control

7.3.1 Function: Play a SACM-S200 speech

Syntax:

C: void   SACM_S200_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM: R1 = Speech_Index

R2 = Channel

R3 = Ramp_Set

call F_SACM_S200_Play

Parameters:

Speech_Index:	-1: Manual Mode 0 – max. of speech index: Auto Mode
Channel:	1: To DAC1 only 2: To DAC2 only 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: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark:

1. The data rate of SACM-S200 can be compressed by 0.8K~3.36K Kbps. The data rate is determined at encoding and selected by the decoder automatically at decoding. The data rate can be 0.8K, 0.9K, 1K, 1.2K, 1.4K, 1.6K, 1.8K, 2K, 2.4K, 2.8K or 3.36Kbps.
2. The Speech_Index is the speech sequence of T_SACM_S200_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_S200_SetStartAddr and F_USER_S200_GetData in sacm_S200_user.asm on users’ own.
3. The F_ISR_Service_SACM_S200 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16KHz after calling this subroutine.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

7.3.2 Function: Stop playing SACM-S200 speech

Syntax:

C: void SACM_S200_Stop(void);

ASM: call F_SACM_S200_Stop

Parameters: None

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: This function will not change the interrupt setting.

7.3.3 Function: Pause playing SACM-S200 speech

Syntax:

C:	void SACM_S200_Pause(void);
ASM:	call F_SACM_S200_Pause
Parameters:	None
Return Value:	None
Library:	<Sacm_S200_SC_Vxxx.LIB>
Remark:	None

7.3.4 Function: Resume paused SACM-S200 speech

Syntax:

C:	void SACM_S200_Resume(void);
ASM:	call F_SACM_S200_Resume
Parameters:	None
Return Value:	None
Library:	<Sacm_S200_SC_Vxxx.LIB>
Remark:	None

7.3.5 Function: Change the volume of SACM-S200

Syntax:

C:	void SACM_S200_Volume(int Volume_Index)
ASM:	R1 = Volume_Index call F_SACM_S200_Volume
Parameters:	Volume_Index: [0..15], 0:Min volume, 15:Max volume
Return Value:	None
Library:	<Sacm_S200_SC_Vxxx.LIB>
Remark:	None

7.3.6 Function: Get the status of the SACM-S200 module

Syntax:

C:	unsigned int SACM_S200_Status(void);
ASM:	call F_SACM_S200_Status [Retrun_Value] = R1

Parameters: None

Return Value: bit 0: 0: Speech ended

1: Speech Playing

bit 1-15: Reserved

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: None

7.3.7 Function: Change the speed index of SACM-S200

Syntax:

C: void SACM_S200_Speed(int Speed_Index)

ASM: R1 = [Speed_Index]

call F_SACM_S200_Speed

Parameters:

Speed_Index: [-2..2], 0:Normal speed, -2:Min speed, 2:Max speed

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark:

1. Min speed (Speed_Index = -2) is 1/2 of Normal speed (Speed_Index = 0).
2. Max speed (Speed _Index = 2) is twice of Normal speed (Speed_Index = 0).

7.3.8 Function: Set constant pitch index of SACM-S200

Syntax:

C: void SACM_S200_Pitch0(int Pitch0_Index)

ASM: R1 = [Pitch0_Index]

call F_SACM_S200_Pitch0

Parameters:

Pitch0_Index: [-2..2], 0:Normal pitch, -2:lowest pitch, 2:highest pitch

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: None

7.3.9 Function: Change the pitch index of SACM-S200

Syntax:

C: void SACM_S200_Pitch1(int Pitch1_Index)

ASM: R1 = [Pitch1_Index]
call F_SACM_S200_Pitch1

Parameters:
Pitch1_Index: [-2..2], 0:Normal pitch, -2:lowest pitch, 2:highest pitch

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: None

7.3.10 Function: Change voiced index of SACM-S200

Syntax:

C: void SACM_S200_Voice(int Voice_Index)

ASM: R1 = [Voice_Index]
call F_SACM_S200_Voice

Parameters:
Voice_Index: [-2..2], 0:Normal, -2:pure unvoiced, 2:pure voiced

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark: None

7.4 ISR Functions: Interrupt service routine for SACM-S200

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_S200_Init_, in sacmvxxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM: call F_ISR_Service_SACM_S200

Parameters: None

Return Value: None

Library: <Sacm_S200_SC_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)
2. The F_ISR_Service_SACM_S200 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to place

user-define function in the same FIQ or IRQ.

EX:

```
_FIQ:
        push  R1, R5 to [SP];
        call   F_ISR_Service_SACM_S200
        call   F_User_ISR
        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop   R1, R5 from [SP]
        reti
```

7.5 User Functions: for SACM-S200 playback in manual mode

7.5.1 Function: Set start address for SACM-S200 speech data.

Syntax:

C: USER_S200_SetStartAddr (,,)

ASM: call F_USER_S200_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_S200_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. ROM, FLASH.

7.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: N/A

ASM: R1 = the start address of buffer to write

R2 = the length of data

call F_USER_S200_GetData

Parameters: R1 = the start address of buffer to write

R2 = the length of data

Return Value: User-defined

Library: sacm_S200_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. ROM, FLASH.
3. This function is call-back function for SACM S200 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

Example:

Play a SACM-S200 speech with Auto mode or Manual mode.

(a). In main.c:

```
#include    "sacmv40.h"
#define     Manual_Mode_Index        -1
#define     Manual                   0
#define     Auto                     1
#define     DAC1                     1
#define     DAC2                     2
#define     Ramp_Up                  1
#define     Ramp_Dn                  2
int         Mode;
int         SpeechIndex = 0;
main()
{
    Mode = Auto;                                // Play mode setting
    if(Mode == Auto)                            // Auto mode play
    {
        SACM_S200_Initial();                    // Initialization
        SACM_S200_Play(SpeechIndex, DAC1+DAC2, Ramp_Up+Ramp_Dn);// Play 1st speech
        While(1)
        {
            SACM_S200_ServiceLoop();            // Service loop for decode
        } // end of While(1)
    } // end of if(Mode == Auto)
    if(Mode == Manual)                          // Manual mode play
    {
        SACM_S200_Initial();                    // Initialization
        USER_S200_SetStartAddr(SpeechIndex);    // Set start address of speech data
```

```
                SACM_S200_Play(Manual_Mode_Index,DAC1+DAC2,Ramp_Up+Ramp_Dn);
                {                                       // Manual mode play speech
                    SACM_S200_ServiceLoop();            // Service loop for decode
                } // end of SACM_S200_Play
            } // end of if(Mode == Manual)
        } // end of main()
```

(b). In ism.asm:

```
_FIQ:
            push  R1, R5 to [SP];                       // push registers
            call    F_ISR_Service_SACM_S200;            // ISR
            R1 = C_FIQ_TMA;
            [R_INT_Clear] = R1;                         // clear TimerA FIQ interrupt flag
            pop    R1, R5 from [SP];                    // pop registers
            reti;                                       // return
```

(c). In sacm_S200_user.asm: (manual mode only)

```
_USER_S200_SetStartAddr:
        // defined by user
        // F_USER_S200_SetStartAddr are Manual mode use only. User implements this 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. ROM, FLASH.
        …
        retf;

F_USER_S200_GetData:
        // defined by user
        // F_USER_S200_GetData are Manual mode use only. User implements this 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. ROM, FLASH.
        …
        retf;
```

8 API for SACM-S480/S720

8.1 Hardware Dependent Function: Initializes SACM-S480/S720

8.1.1 Function: Initialize SACM-S480/S720 library

Syntax:

C: void SACM_S480_Initial(void);

ASM: call F_SACM_S480_Initial

Parameters: None

Return Value: None

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark:

1. This function initializes the decoder of SACM-S480. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ at 16KHz sample rate.
2. The hardware setting is opened for user's reference (see F_SP_SACM_S480_Speed_Init_: function in sacmvxxx.asm).
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program if for SPCE500A. It uses P_INT_Mask instead if for SPCE 061A/060A

8.2 Service Loop Functions: Service loop for SACM-S480 decoding

8.2.1 Function: Foreground service loop

Syntax:

C: void SACM_S480_ServiceLoop(void);

ASM: call F_SACM_S480_ServiceLoop

Parameters: None

Return Value: None

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

8.2.2 Function: Background service loop:

Syntax:

C: N/A

ASM: Call F_SACM_S480_ServiceLoop_ISR

Parameters: N/A

Return Value: N/A

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the S480 code is allocated in page 0 to assure that program flow can proceed to F_SACM_S480_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register. Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to "Sunplus u'nSP Assembly Tools User Manual" for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_S480;           // Interrupt service routine.
 R1 = F_SACM_S480_ServiceLoop_ISR;       // Background Service loop
                                         // for SACM S480 playing
 push R1 to [SP];                        // push function address to stack
 push SR to [SP];                        // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

8.3 Playback Functions: Playback control

8.3.1 Function: Play a SACM-S480/S720 speech

Syntax:

C: void SACM_S480_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM: R1 = Speech_Index

R2 = Channel

R3 = Ramp_Set

call F_SACM_S480_Play

Parameters:

Speech_Index:	-1: Manual Mode 0 – max. of speech index: Auto Mode
Channel:	1: To DAC1 only 2: To DAC2 only 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: None

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark:

1. The data rate of SACM-S480 can be compressed by 4.8/7.2Kbps. The data rate is selected by decoder automatically.
2. The Speech_Index is the speech sequence of T_SACM_S480_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_S480_SetStartAddr and F_USER_S480_GetData in sacm_S480_user.asm on users' own.
3. The F_ISR_Service_SACM_S480 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16KHz after calling this subroutine.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

8.3.2 Function: Stop playing SACM-S480/S720 speech

Syntax:

C: void SACM_S480_Stop(void);

ASM: call F_SACM_S480_Stop

Parameters: None

Return Value: None

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark: This function will not change the interrupt setting.

8.3.3 Function: Pause playing SACM-S480/S720 speech

Syntax:

C:	void SACM_S480_Pause(void);
ASM:	call F_SACM_S480_Pause
Parameters:	None
Return Value:	None
Library:	<Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>
Remark:	None

8.3.4 Function: Resume paused SACM-S480/S720 speech

Syntax:

C:	void SACM_S480_Resume(void);
ASM:	call F_SACM_S480_Resume
Parameters:	None
Return Value:	None
Library:	<Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>
Remark:	None
Example:	None

8.3.5 Function: Change the volume of SACM-S480/S720

Syntax:

C:	void SACM_S480_Volume(int Volume_Index)
ASM:	R1 = [Volume_Index] call F_SACM_S480_Volume
Parameters:	Volume_Index: [0..15], 0:Min volume, 15:Max volume
Return Value:	None
Library:	<Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>
Remark:	None

8.3.6 Function: Get the status of the SACM-S480/S720 module

Syntax:

C:	unsigned int SACM_S480_Status(void);
ASM:	call F_SACM_S480_Status

[Retrun_Value] = R1

Parameters: None

Return Value: bit 0: 0: Speech ended

1: Speech Playing

bit 1-15: Reserved

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark: None

8.3.7 Function: Change the speed index of SACM-S480/S720

Syntax:

C: void SACM_S480_Speed(int Speed_Index)

ASM:

```
R1 = [Speed_Index]
call    F_SACM_S480_Speed
```

Parameters: Speed_Index: [-12..12], 0:Normal speed, -12:Min speed, 12:Max speed

Return Value: None

Library: <Sacm_S480_SC_Vxxx.LIB>

Remark: Min speed (Speed_Index = -12) is 1/2 of Normal speed (Speed_Index = 0). Max speed (Speed _Index = 12) is twice of Normal speed.

8.4 ISR Functions: Interrupt service routine for SACM-S480/S720

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_S480_Init_, in sacmvxxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM:

```
call    F_ISR_Service_SACM_S480
```

Parameters: None

Return Value: None

Library: <Sacm_S480_Vxxx.LIB>, <Sacm_S480_SC_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)
2. The F_ISR_Service_SACM_S480 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to place user-define function in the same FIQ or IRQ

EX:

```
_FIQ:
        push  R1, R5 to [SP];
        call    F_ISR_Service_SACM_S480
        call    F_User_ISR
        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop    R1, R5 from [SP]
        reti
```

8.5 User Functions: for SACM-S480/S720 playback in manual mode

8.5.1 Function: Set start address for SACM-S480/S720 speech data.

Syntax:

C: USER_S480_SetStartAddr (,,)

ASM: call F_USER_S480_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_S480_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. ROM, FLASH.

8.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: N/A

ASM: R1 = the start address of buffer to write
R2 = the length of data
call F_USER_S480_GetData

Parameters: R1 = the start address of buffer to write
R2 = the length of data

Return Value: User-defined

Library: sacm_S480_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. ROM, FLASH.
3. This function is call-back function for SACM S480 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

Example:

Play a SACM-S480 speech with Auto mode or Manual mode.

(a). In main.c:

```
#include     "sacmv40.h"
#define      Manual_Mode_Index        -1
#define      Manual                   0
#define      Auto                     1
#define      DAC1                     1
#define      DAC2                     2
#define      Ramp_Up                  1
#define      Ramp_Dn                  2
int          Mode;
int          SpeechIndex = 0;
main()
{
     Mode = Auto;                                  // Play mode setting
     if(Mode == Auto)                              // Auto mode play
     {
          SACM_S480_Initial();                     // Initialization
          SACM_S480_Play(SpeechIndex, DAC1+DAC2, Ramp_Up+Ramp_Dn);  // Play 1st speech
          While(1)
          {
               SACM_S480_ServiceLoop();            // Service loop for decode
          } // end of While(1)
     } // end of if(Mode == Auto)
     if(Mode == Manual)                            // Manual mode play
     {
          SACM_S480_Initial();                     // Initialization
          USER_S480_SetStartAddr(SpeechIndex);     // Set start address of speech data
          SACM_S480_Play(Manual_Mode_Index,DAC1+DAC2,Ramp_Up+Ramp_Dn);
```

```
            {                                           // Manual mode play speech
                SACM_S480_ServiceLoop();                // Service loop for decode
            } // end of SACM_S480_Play
        } // end of if(Mode == Manual)
    } // end of main()
```

(b). In ism.asm:

```
_FIQ:
        push  R1, R5 to [SP];                           // push registers
        call    F_ISR_Service_SACM_S480;                // ISR
        R1 = C_FIQ_TMA;
        [R_INT_Clear] = R1;                             // clear TimerA FIQ interrupt flag
        pop    R1, R5 from [SP];                        // pop registers
        reti;                                           // return
```

(c). In sacm_S480_user.asm: (manual mode only)

```
_USER_S480_SetStartAddr:
    // defined by user
    // F_USER_S480_SetStartAddr are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;

F_USER_S480_GetData:
    // defined by user
    // F_USER_S480_GetData are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;
```

9 API for SACM-S530

9.1 Hardware Dependent Function: Initializes SACM-S530

9.1.1 Function: Initialize SACM-S530 library

Syntax:

C: void SACM_S530_Initial(void);

ASM: call F_SACM_S530_Initial

Parameters: None

Return Value: None

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark:

1. This function initializes the decoder of SACM-S530. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ at 16KHz sample rate.
2. The hardware setting is opened for user's reference (see F_SACM_S530_Init_: function in sacmvxxx.asm).
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program if for SPCE500A. It uses P_INT_Mask instead if for SPCE 061A/060A

9.2 Service Loop Functions: Service loop for SACM-S530 decoding

9.2.1 Function: Foreground service loop

Syntax:

C: void SACM_S530_ServiceLoop(void);

ASM: call F_SACM_S530_ServiceLoop

Parameters: None

Return Value: None

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

9.2.2 Function: Background service loop:

Syntax:

C: N/A

ASM: Call F_SACM_S530_ServiceLoop_ISR

Parameters: N/A

Return Value: N/A

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the S530 code is allocated in page 0 to assure that program flow can proceed to F_SACM_S530_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register. Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to "Sunplus u'nSP Assembly Tools User Manual" for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_S530;          // Interrupt service routine.
 R1 = F_SACM_S530_ServiceLoop_ISR;      // Background Service loop
                                        // for SACM S530 playing
 push R1 to [SP];                       // push function address to stack
 push SR to [SP];                       // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

9.3 Playback Functions: Playback control

9.3.1 Function: Play a SACM-S530 speech

Syntax:

C: void SACM_S530_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM: R1 = Speech_Index

R2 = Channel

R3 = Ramp_Set

call F_SACM_S530_Play

Parameters:

	Speech_Index:	-1: Manual Mode
		0 – max. of speech index: Auto Mode
	Channel:	1: To DAC1 only
		2: To DAC2 only
		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: None

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark:

1. The data rate of SACM-S5300 can be compressed by 5.3 Kbps when sampling is 8KHz. The data rate is selected by decoder automatically at decoding. The data rate can vary along with the sampling rate. That is, data rate can be 5.3 Kbps, 5.96 Kbps, 6.63 Kbps, 7.29 Kbps and 7.95 Kbps when the sampling rate is 8KHz, 9KHz,10KHz, 11KHz, or 12KHz.
2. The Speech_Index is the speech sequence of T_SACM_S530_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_S530_SetStartAddr and F_USER_S530_GetData in sacm_S530_user.asm on users’ own.
3. The F_ISR_Service_SACM_S530 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16/18/20/22/24 KHz after calling this subroutine. Please refer to F_SP_SACM_S530_Init_ in sacmvxxx.asm for detail.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

9.3.2 Function: Stop playing SACM-S530 speech

Syntax:

C: void SACM_S530_Stop(void);

ASM: call F_SACM_S530_Stop

Parameters: None

Return Value: None

Library:	<Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>
Remark:	This function will not change the interrupt setting.

9.3.3 Function: Pause playing SACM-S530 speech

Syntax:

C:	void SACM_S530_Pause(void);
ASM:	call F_SACM_S530_Pause
Parameters:	None
Return Value:	None
Library:	<Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>
Remark:	None

9.3.4 Function: Resume paused SACM-S530 speech

Syntax:

C:	void SACM_S530_Resume(void);
ASM:	call F_SACM_S530_Resume
Parameters:	None
Return Value:	None
Library:	<Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>
Remark:	None

9.3.5 Function: Change the volume of SACM-S530

Syntax:

C:	void SACM_S530_Volume(int Volume_Index)
ASM:	R1 = [Volume_Index] call F_SACM_S530_Volume
Parameters:	Volume_Index: [0..15], 0:Min volume, 15:Max volume
Return Value:	None
Library:	<Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>
Remark:	None

9.3.6 Function: Get the status of the SACM-S530 module

Syntax:

C: unsigned int SACM_S530_Status(void);

ASM: call F_SACM_S530_Status

[Retrun_Value] = R1

Parameters: None

Return Value: bit 0: 0: Speech ended

1: Speech Playing

bit 1-15: Reserved

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark: None

9.3.7 Function: Change the speed index of SACM-S530 while playing

Syntax:

C: void SACM_S530_Speed(int Speed_Index)

ASM: R1 = [Speed_Index]

call F_SACM_S530_Speed

Parameters: Speed_Index: [-12..12], 0:Normal speed, -12:Min speed, 12:Max speed

Return Value: None

Library: <Sacm_S530_SC_Vxxx.LIB>

Remark:

1. Min speed (Speed_Index = -12) is 1/2 of Normal speed (Speed_Index = 0).
2. Max speed (Speed _Index = 12) is twice of Normal speed (Speed_Index = 0).

9.4 ISR Functions: Interrupt service routine for SACM-S530

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_S530_Init_, in sacmvxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM: Call F_ISR_Service_SACM_S530

Parameters: None

Return Value: None

Library: <Sacm_S530_Vxxx.LIB>, <Sacm_S530_SC_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)

2. The F_ISR_Service_SACM_S530 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to place user-define function in the same FIQ or IRQ

EX:

```
    _FIQ:
            push  R1, R5 to [SP];
            call    F_ISR_Service_SACM_S530
            call    F_User_ISR
            R1 = C_IRQ1_TMA;
            [P_INT_Clear] = R1;
            pop    R1, R5 from [SP]
            reti
```

9.5 User Functions: for SACM-S530 playback in manual mode

9.5.1 Function: Set start address for SACM-S530 speech data.

Syntax:

C: USER_S530_SetStartAddr (,,)

ASM: call F_USER_S530_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_S530_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. ROM, FLASH.

9.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: User-defined

ASM:

```
R1 = the start address of buffer to write
R2 = the length of data
call    F_USER_S530_GetData
```

Parameters: R1 = the start address of buffer to write

R2 = the length of data

Return Value: User-defined

Library: sacm_S530_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. ROM, FLASH.
3. This function is call-back function for SACM S530 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

Example:

Play a SACM-S530 speech with Auto mode or Manual mode.

(a). In main.c:

```
#include    "sacmv40.h"
#define     Manual_Mode_Index       -1
#define     Manual                  0
#define     Auto                    1
#define     DAC1                    1
#define     DAC2                    2
#define     Ramp_Up                 1
#define     Ramp_Dn                 2
int         Mode;
int         SpeechIndex = 0;
main()
{
    Mode = Auto;                                // Play mode setting
    if(Mode == Auto)                            // Auto mode play
    {
        SACM_S530_Initial();                    // Initialization
        SACM_S530_Play(SpeechIndex, DAC1+DAC2, Ramp_Up+Ramp_Dn);// Play 1st speech
        While(1)
        {
            SACM_S530_ServiceLoop();            // Service loop for decode
        } // end of While(1)
    } // end of if(Mode == Auto)
    if(Mode == Manual)                          // Manual mode play
```

```
        {
            SACM_S530_Initial();                      // Initialization
            USER_S530_SetStartAddr(SpeechIndex);      // Set start address of speech data
            SACM_S530_Play(Manual_Mode_Index,DAC1+DAC2,Ramp_Up+Ramp_Dn);
            {                                         // Manual mode play speech
                SACM_S530_ServiceLoop();              // Service loop for decode
            } // end of SACM_S530_Play
        } // end of if(Mode == Manual)
    } // end of main()
```

(b). In ism.asm:

```
_FIQ:
        push  R1, R5 to [SP];                         // push registers
        call    F_ISR_Service_SACM_S530;              // ISR
        R1 = C_FIQ_TMA;
        [R_INT_Clear] = R1;                           // clear TimerA FIQ interrupt flag
        pop   R1, R5 from [SP];                       // pop registers
        reti;                                         // return
```

(c). In sacm_S530_user.asm: (manual mode only)

```
_USER_S530_SetStartAddr:
    // defined by user
    // F_USER_S530_SetStartAddr are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;

F_USER_S530_GetData:
    // defined by user
    // F_USER_S530_GetData are Manual mode use only. User implements this 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. ROM, FLASH.
    …
    retf;
```

10 API for SACM-MS01

10.1 Hardware Dependent Function: Initializes SACM-MS01

10.1.1 Function: Initialize SACM-MS01 library

Syntax:

C: void SACM_MS01_Initial(void)

ASM: Call F_SACM_MS01_Initial

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark:

1. This function initializes the decoder of MS01. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ at the sample rate on 16KHz.
2. The hardware setting is opened for user's reference (see F_SP_SACM_MS01_Init_: function in sacmvxxx.asm) .
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program for SPCE500A. It uses P_INT_Mask instead for SPCE 061A/060A

10.2 Service Loop Functions: Service loop for SACM-MS01 decoding

10.2.1 Function: Foreground service loop:

Syntax:

C: void SACM_MS01_ServiceLoop(void);

ASM: call F_SACM_MS01_ServiceLoop

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

10.2.2 Function: Background service loop:

Syntax:

C: N/A

ASM: Call F_SACM_MS01_ServiceLoop_ISR

Parameters: N/A

Return Value: N/A

Library: <Sacm_MS01_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the MS01 code is allocated in page 0 to assure that program flow can proceed to F_SACM_MS01_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register.   Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to "Sunplus u'nSP Assembly Tools User Manual" for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_S480;          // Interrupt service routine.
 R1 = F_SACM_S480_ServiceLoop_ISR;      // Background Service loop
                                        // for SACM S480 playing
 push R1 to [SP];                       // push function address to stack
 push SR to [SP];                       // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

10.3 Playback Functions: Playback control

10.3.1 Function: Play a SACM-MS01 melody

Syntax:

C: void   SACM_MS01_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM: R1 = Speech_Index

R2 = Channel

R3 = Ramp_Set

call F_SACM_S530_Play

Parameters:

Speech_Index:	-1: Manual Mode
	0 – max. of speech index: Auto Mode
Channel:	1: To DAC1 only
	2: To DAC2 only
	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: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark:

1. The SACM-MS01 provides six channels melody (0,1 ch are drum channels, 2,3,4,5 are FM synthesizer channels).
2. The Speech_Index is the speech sequence of T_SACM_MS01_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_MS01_SetStartAddr and F_USER_MS01_GetData in sacm_MS01_user.asm on users' own.
3. The F_ISR_Service_SACM_MS01 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16KHz after calling this subroutine.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

10.3.2 Function: Stop playing SACM-MS01 melody

Syntax:

C: void SACM_MS01_Stop(void)

ASM: call F_SACM_MS01_Stop

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark: This function will not change the interrupt setting.

10.3.3 Function: Pause playing SACM-MS01 melody

Syntax:

C: void SACM_MS01_Pause(void)

ASM: call F_SACM_MS01_Pause

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark: None

10.3.4 Function: Resume paused SACM-MS01 melody

Syntax:

C: void SACM_MS01_Resume(void);

ASM: call F_SACM_MS01_Resume

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark:

10.3.5 Function: Change the volume of SACM-MS01

Syntax:

C: void SACM_MS01_Volume(int Volume_Index)

ASM:
```
R1 = [Volume_Index]
call    F_SACM_MS01_Volume
```

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

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark: None

10.3.6 Function: Get the status of the SACM-MS01 module

Syntax:

C: unsigned int SACM_MS01_Status(void);

ASM:
```
call    F_SACM_MS01_Status
[Return_Value] = R1
```

Parameters: None

Return Value: bit 0: 0: Speech ended

1: Speech Playing

bit 1~15: Reserved

Library: <Sacm_MS01_Vxxx.LIB>

Remark: None

MS01 Built–in Tone Color List

ZZ	SACM-MS01 Tone colors	Suggested Pitch Range	Performance
00	Piano	1, ~ 7’	1, ~ 7’
01	Marimba	1, ~ 1”#	The higher pitch, the higher volume
02	Music Box	7. ~ 2”#	7. ~ 2”#
03	Guitar	1, ~ 2”#	3, ~ 2”#
04	Cello 1	1, ~ 2”#	1, ~ 2”#
05	Violin	1, ~ 7’	1, ~ 1’
06	French Horn	5, ~ 2”#	5, ~ 2”#
07	Flute 1	1, ~ 1”#	1, ~ 1”#
08	E. Piano	1, ~ 2”#	The higher pitch, the higher volume
09	Harpsichord	1, ~ 6#	1, ~ 6#
0A	Clav.	1, ~ 2’#	1, ~ 2’#
0B	Vibraphone	3. ~ 3’	3. ~ 3’
0C	Bell	4. ~ 7’	4. ~ 7’
0D	Nylon-str. Gt.	1, ~ 1”#	1, ~ 1”#
0E	Steel-str. Gt.	1, ~ 7	1, ~ 1
0F	Jazz Gt.	1, ~ 1”#	1, ~ 1”#
10	Clean Gt.	1, ~ 3	1, ~ 7.
11	Distortion Gt.	1, ~ 6#	1, ~ 7.
12	Bass	1, ~ 7	1, ~ 7
13	Banjo 1	1, ~ 7	1, ~ 7
14	Banjo 2	1, ~ 4#	1, ~ 7.
15	Violin 2	2. ~ 1’	2. ~ 1’
16	Viola	2. ~ 1’	2. ~ 1’
17	Cello 2	1, ~ 5’	1, ~ 1’
18	Contrabass	1, ~ 1’	1, ~ 1’
19	Trumpet	1, ~ 7’	1, ~ 6#
1A	Oboe 1	1, ~ 3’	1, ~ 1’
1B	Oboe 2	1, ~ 3’	1, ~ 2#
1C	English Horn	1, ~ 1”	1, ~ 7
1D	Clarinet	1, ~ 7’	1, ~ 2’#
1E	Piccolo	1, ~ 1”#	1, ~ 1”#
1F	Flute 2	1, ~ 1”#	1, ~ 1”#
20	Recorder	1, ~ 1”#	1, ~ 5’#
21	Whistle	1: ~ 2”#	1: ~ 2”#
22	SFX	1: ~ 2”#	1: ~ 2”#

- , Octave 0
- : Octave 1
- . Octave 2

•<space>	Octave 3
• '	Octave 4
• "	Octave 5
• ^	Octave 6
• #	sharp mark

Octave0 = {C0, C0#, D0, D0#, E0, F0, F0#, G0, G0#, A0, A0#, B0}**,**

Octave1 = {C1, C1#, D1, D1#, E1, F1, F1#, G1, G1#, A1, A1#, B1}**,**

Octave6 = {C6, C6#, D6, D6#, E6, F6, F6#, G6, G6#, A6, A6#, B6}

10.4 ISR Functions: Interrupt service routine for SACM-MS01

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_MS01_Init_, in sacmvxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM: F_ISR_Service_SACM_MS01

Parameters: None

Return Value: None

Library: <Sacm_MS01_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)
2. The F_ISR_Service_SACM_MS01 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to place user-define function in the same FIQ or IRQ

EX:

```
_FIQ:
        push  R1, R5 to [SP];
        call    F_ISR_Service_SACM_MS01
        call    F_User_ISR
        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop    R1, R5 from [SP]
        reti
```

10.5 User Functions: for SACM-MS01 playback in manual mode

10.5.1 Function: Set start address for SACM-MS01 speech data.

Syntax:

C: USER_MS01_SetStartAddr (,,)

ASM: call F_USER_ MS01_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_MS01_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. ROM, FLASH.

10.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: N/A

ASM: R1 = the start address of buffer to write
R2 = the length of data
call F_USER_MS01_GetData

Parameters: R1 = the start address of buffer to write
R2 = the length of data

Return Value: User-defined

Library: sacm_MS01_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. ROM, FLASH.
3. This function is call-back function for SACM MS01 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

Example:

Play a SACM-MS01 melody with Auto mode or Manual mode.

(a). In main.c:

```
        #include    "sacmv40.h"
        #define     Manual_Mode_Index        -1
        #define     Manual                   0
        #define     Auto                     1
        #define     DAC1                     1
        #define     DAC2                     2
        #define     Ramp_Up                  1
        #define     Ramp_Dn                  2
        int         Mode;
        int         SpeechIndex = 0;
        main()
        {
            Mode = Auto;                                    // Play mode setting
            if(Mode == Auto)                                // Auto mode play
            {
                SACM_MS01_Initial();                        // Initialization
                SACM_MS01_Play(SpeechIndex, DAC1+DAC2, Ramp_Up+Ramp_Dn);// Play 1st speech
                While(1)
                {
                    SACM_MS01_ServiceLoop();                // Service loop for decode
                } // end of While(1)
            } // end of if(Mode == Auto)
            if(Mode == Manual)                              // Manual mode play
            {
                SACM_MS01_Initial();                        // Initialization
                USER_MS01_SetStartAddr(SpeechIndex);        // Set start address of speech data
                SACM_MS01_Play(Manual_Mode_Index,DAC1+DAC2,Ramp_Up+Ramp_Dn);
                {                                           // Manual mode play speech
                    SACM_MS01_ServiceLoop();                // Service loop for decode
                } // end of SACM_MS01_Play
            } // end of if(Mode == Manual)
        } // end of main()
```

(b). In ism.asm:

```
_FIQ:
            push  R1, R5 to [SP];                           // push registers
            call  F_ISR_Service_SACM_MS01;                  // ISR
```

```
          R1 = C_FIQ_TMA;
          [R_INT_Clear] = R1;                      // clear TimerA FIQ interrupt flag
          pop    R1, R5 from [SP];                 // pop registers
          reti;                                    // return
```

(c). In sacm_MS01_user.asm: (manual mode only)

```
_USER_MS01_SetStartAddr:
     // defined by user
     // F_USER_MS01_SetStartAddr are Manual mode use only. User implements this 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. ROM, FLASH.
     …
     retf;

F_USER_MS01_GetData:
     // defined by user
     // F_USER_MS01_GetData are Manual mode use only. User implements this 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. ROM, FLASH.
     …
     retf;
```

11 API of SACM-DVR1600

11.1 Hardware Dependent Function: Initializes SACM-DVR1600

11.1.1 Function: Initialize SACM-DVR1600 library

Syntax:

C:	void SACM_DVR1600_Initial(void)
ASM:	call F_SACM_DVR1600_Initial
Parameters:	None
Return Value:	None
Library:	<Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. This function initializes the decoder of SACM-DVR1600. It also initializes the system clock, Timer A, DAC and enables the Timer A FIQ at the sample rate on 16KHz.
2. The hardware setting is opened for user's reference (see F_SP_SACM_DVR1600_Init_ in sacmvxxx.asm).
3. This function utilizes a register, R_InterruptStatus (spce.asm), to work with user's program for SPCE500A. It uses P_INT_Mask instead for SPCE 061A/060A.
4. In SACM-DVR1600, the external memory module is required to store the recording data. External memory module can apply GPIO or Sunplus serial interface to connect to external memory. The memory access interface must implement the F_USER_DVR1600_GetData, F_USER_DVR1600_WriteData and User_DVR1600_SetStartAddr in sacm_DVR1600_user.asm.
5. The decode engine is the same as that of SACM_A1600.

11.2 Service Loop Functions: S Service loop for SACM-DVR1600 decoding

11.2.1 Function: Foreground service loop

Syntax:

C:	void SACM_DVR1600_ServiceLoop(void)
ASM:	call F_SACM_DVR1600_ServiceLoop
Parameters:	None
Return Value:	None
Library:	<Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: Foreground service loop has to be placed in main loop.

11.2.2 Function: Background service loop

Syntax:

C: N/A

ASM: Call F_SACM_DVR1600_ServiceLoop_ISR

Parameters: None

Return Value: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. Background service loop is placed in either FIQ or IRQ and it must be placed before leaving the FIQ or IRQ. Please refer to the following example.
2. Please make sure that the DVR1600 code is allocated in page 0 to assure that program flow can proceed to F_SACM_DVR1600_ServiceLoop_ISR correctly.
3. This background service loop will pop the R1-R5 register. Program does not have to pop the registers before reti.
4. The advantage users can get from background service loop is that program can reduce unnecessary overhead in main and as a result program can yield better performance in terms of throughput.
5. Example: By this manner of programming, the program allows the background service loop to take place right after leaving the FIQ/IRQ and meanwhile the next interrupt can still intrude the service loop even when the background service loop is still running. Please refer to “Sunplus u’nSP Assembly Tools User Manual” for the detail of the argument passing in library routine.

```
_FIQ:
 push R1, R5 to [SP];
 call F_ISR_Service_SACM_DVR1600;          // Interrupt service routine.
 R1 = F_SACM_DVR1600_ServiceLoop_ISR;      // Background Service loop
                                           // for SACM A1600 playing
 push R1 to [SP];                          // push function address to stack
 push SR to [SP];                          // push SR to stack
 R1 = C_FIQ_TMA;
 [P_INT_Clear] = R1;
 reti;
```

11.3 Playback Functions: Record/Playback control

11.3.1 Function: Start recording data from MIC to external memory module

Syntax:

C:	void SACM_DVR1600_Rec(int RceMonitor, int ADC_Channel, int Bit_Rate)

ASM:	R1 = RceMonitor
	R2 = ADC_Channel
	R3 = Bit_Rate
	call F_SACM_DVR_Record

Parameters: RceMonitor : 0: RceMonitorOff, ADC data not sent to DAC

1: RceMonitorOn, ADC data sent to DAC

ADC_Channel: 0: Microphone; 1~7 Line-in for SPCE040A/060A/061A;

0: Microphone; 1 Line-in for SPCE120A/250A/380A/500A;

Bit_Rate :

0 : BIT_RATE_10K

1 : BIT_RATE_12K

2 : BIT_RATE_14K

3 : BIT_RATE_16K

4 : BIT_RATE_20K

5 : BIT_RATE_24K

Return Value: N/A

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. When SACM_DVR1600_Rec is called, it will call, from inside, the F_SP_SwitchChannel in sacmvxxx.asm according to the ADC channel specified.

11.3.2 Function: Play DVR1600 speech

Syntax:

C:	void SACM_DVR1600_Play(int Speech_Index, int Channel, int Ramp_Set)

ASM:	R1 = Speech_Index
	R2 = Channel
	R3 = Ramp_Set
	call F_ SACM_DVR1600_Play

Parameters:

Speech_Index:	-1: Manual Mode
	0 – max. of speech index: Auto Mode
Channel:	1: To DAC1 only
	2: To DAC2 only
	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: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. The data rate of SACM-DVR1600 can be compressed by 10/12/14/16/20/24Kbps. The data rate is selected by the decoder automatically.
2. The Speech_Index is the speech sequence of T_SACM_DVR1600_SpeechTable in resource.asm. For manual mode playback, the Speech_Index should be set as –1. And user should implement the _USER_DVR1600_SetStartAddr and F_USER_DVR1600_GetData in sacm_DVR1600_user.asm on users’ own.
3. The F_ISR_Service_SACM_DVR1600 can be hooked on the _FIQ:, _IRQ1:, or _IRQ2: label (see isr.asm for details) before using this function.
4. The interrupt service routine (ISR) is working on 16KHz after calling this subroutine.
5. About the details of Ramp Up/Down function, please refer to section 12.1 Ramp Functions.

11.3.3 Function: Stop recording/playback

Syntax:

C: void SACM_DVR1600_Stop(void);

ASM: call F_SACM_DVR1600_Stop

Parameters: N/A

Return Value: N/A

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: This function will not change the interrupt setting.

11.3.4 Function: Pause currently playing SACM-DVR1600 speech

Syntax:

C: void SACM_DVR1600_Pause(void);

ASM: call F_SACM_DVR1600_Pause

Parameters: None

Return Value: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: None

11.3.5 Function: Resume paused SACM-DVR1600 speech

Syntax:

C: void SACM_DVR1600_Resume(void);

ASM: Call F_SACM_DVR1600_Resume

Parameters: None

Return Value: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: None

11.3.6 Function: Change the volume of SACM-DVR1600

Syntax:

C: void SACM_DVR1600_Volume(int Volume_Index)

ASM: R1 = Volume_Index

call F_SACM_DVR1600_Volume

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

Return Value: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: None

11.3.7 Function: Get the status from SACM-DVR1600 module

Syntax:

C: unsigned int SACM_DVR1600_Status(void);

ASM: call F_SACM_DVR1600_Status

[Retrun_Value] = R1

Parameters: N/A

Return Value:

bit 0: 0: Stop

1: Recording

bit 1: 0: Stop

1: Playing

bit 2 – bit 15 is reserved

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark: For both auto and manual modes

11.3.8 Function: Change the speed index of SACM-DVR1600

Syntax:

C: void SACM_DVR1600_Speed(int Speed_Index)

ASM: R1 = [Speed_Index]

call F_SACM_DVR1600_Speed

Parameters:

Speed_Index: [-12..12], 0:Normal speed, -12:Min speed, 12:Max speed

Return Value: None

Library: <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. Min speed (Speed_Index = -12) is 1/2 of Normal speed (Speed_Index = 0).
2. Max speed (Speed _Index = 12) is twice of Normal speed(Speed_Index = 0).

11.4 ISR Functions: Interrupt service routine for SACM-DVR1600 playback / recording

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. It can be placed in FIQ or IRQ1 or IRQ2 depending on Timer used. The initial function, F_SP_SACM_DVR1600_Init_, in sacmvxxx.asm must also be updated as well.

Syntax:

C: N/A

ASM: Call F_ISR_Service_SACM_DVR1600

Parameters: None

Return Value: None

Library: <Sacm_DVR1600_Vxxx.LIB>, <Sacm_DVR1600_SC_Vxxx.LIB>

Remark:

1. This function is used in assembly only and it can be hooked on the _FIQ, _IRQ1 or _IRQ2: label. (See isr.asm for details)
2. The F_ISR_Service_SACM_DVR1600 will not take up any time to process the Interrupt routine except minor overheads if the program is not playing. It is possible for users to

place user-define function in the same FIQ or IRQ

EX:

```
    _FIQ:
        push  R1, R5 to [SP];
        call    F_ISR_Service_SACM_DVR1600
        call    F_User_ISR
        R1 = C_IRQ1_TMA;
        [P_INT_Clear] = R1;
        pop    R1, R5 from [SP]
        reti
```

3. The ISR is working on 16KHz when playback, 32KHz when recording.

11.5 User Functions: for SACM-DVR1600 playback in manual mode

11.5.1 Function: Set start address for SACM-DVR1600 speech data.

Syntax:

C: USER_DVR1600_SetStartAddr (,,)

ASM: call F_USER_DVR1600_SetStartAddr

Parameters: User-defined

Return Value: User-defined

Library: sacm_DVR1600_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.

11.5.2 Function: Read the speech data from user's storage and write to buffer

Syntax:

C: N/A

ASM: R1 = the start address of buffer to write

R2 = the length of data

call F_USER_DVR1600_GetData

Parameters: R1 = the start address of buffer to write

R2 = the length of data

Return Value: User-defined

Library: sacm_DVR1600_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.
3. This function is call-back function for SACM DVR1600 library to read speech data from memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to get.

11.5.3 Function: Read the speech data from buffer and write to user's storage

Syntax:

C: N/A

ASM: R1 = the start address of buffer to read

R2 = the length of data

call F_USER_DVR1600_WriteData

Parameters: R1 = the start address of buffer to read

R2 = the length of data

Return Value: User-defined

Library: sacm_DVR1600_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.
3. This function is call-back function for SACM A1600 library to write encoded data to memory storage. Where R1 is the beginning address of internal buffer and R2 is the data length that library wants to write.

Example:

SACM-DVR1600 record and playback.

(a). In main.c:

```
#include    "sacmv40.h"
#define     Manual_Mode_Index       -1
#define     DAC1                    1
#define     DAC2                    2
```

```
#define     Ramp_Up                     1
#define     Ramp_Dn                     2
#define     RceMonitorOff               0
#define     RceMonitorOn                1
int         BIT_RATE      = BIT_RATE_16K;    // BIT_RATE_(10K, 12K, 14K, 16K, 20K, 24K)
int         ADC_Channel   = Mic;             // 0~7 (Mic, Line_In1 ~ Line_In6)
int         SpeechIndex   = 0;
int         Key;
main()
{
    while(1)
    {
        Key = SP_GetCh();
        switch(Key)
        {
            case 0x0100:                        // IOA8+Vcc: Record
                USER_DVR1600_SetStartAddr(4);   // REC skip 4 Bytes for length header
                BIT_RATE = BIT_RATE_16K;        // Bit rate = 16K bps
                ADC_Channel = Mic;              // ADC channel : Mic_In
                SACM_DVR1600_Rec(RceMonitorOff, ADC_Channel, BIT_RATE); // Start recode
                break;
            case 0x0200:                        // IOA9+Vcc: Stop recording/playback
                SACM_DVR1600_Stop();            // Stop recording/playback
                break;
            case 0x0400:                        // IOA10+Vcc: Play Manual mode Speech
                USER_DVR1600_SetStartAddr(0);   // Set start address
                SACM_DVR1600_Play(Manual_Mode_Index,DAC1+DAC2,Ramp_Up+Ramp_Dn);
                                                // Manual mode play
                break;
            default:
                break;
        } // end of switch(Key)
        SACM_DVR1600_ServiceLoop();             // Service loop for SACM playing
        System_ServiceLoop();                   // Service loop for Key Scanning
    } // end of while(1)
} // end of main()
```

(b). In ism.asm:

```
_FIQ:
          push  R1, R5 to [SP];                        // push registers
          call    F_ISR_Service_SACM_DVR1600;          // ISR
          R1 = C_FIQ_TMA;
          [R_INT_Clear] = R1;                          // clear TimerA FIQ interrupt flag
          pop    R1, R5 from [SP];                     // pop registers
          reti;                                        // return
```

(c). In sacm_DVR1600_user.asm: (manual mode only)

```
_USER_DVR1600_SetStartAddr:
     // defined by user
     // F_USER_DVR1600_SetStartAddr are Manual mode use only. User implements this 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.
     …
     retf;

F_USER_DVR1600_GetData:
     // defined by user
     // F_USER_DVR1600_GetData are Manual mode use only. User implements this 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.
     …
     retf;

F_USER_DVR1600_WriteData:
     // defined by user
     // F_USER_DVR1600_WriteData are Manual mode use only. User implements this 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.
     …
     retf;
```

12 Hardware dependence API in SACMVxx.asm (Open source)

12.1 Ramp Functions

In current type DAC, the digital input range is 0x0000~0xFFFF and the corresponding result analog range is 0~3mA (or 2mA by setting). The middle value is 0x8000(1.5mA or 1mA). In order to avoid unnecessary power consumption, we will set the output of current DAC to 0x0000(i.e. 0mA) when sound is not playing or especially before the system enters sleep. If we set the DAC output from 0x8000 to 0x0000 suddenly, a "burst sound" will be generated due to the sudden change of the DAC value. Therefore, we need to reduce DAC value from what it is to 0x0000 gradually and smoothly. In other words, the "Ramp Down" process. Similarly, before playing a sound whose data usually starts from 0x8000, users have to "Ramp Up" the current DAC from 0x0000 to 0x8000 gradually to avoid the "burst sound" (a sound occurs suddenly without expectation). There are many methods to achieve Ramp Up/Ramp Down process. Here only we provide a Ramp Up/Ramp Down method which is easy to be implemented. Users can rewrite these Ramp Up/Ramp Down functions using other methods.

12.1.1 Function: Ramp up DAC1

Syntax:

C: void SP_RampUpDAC1(void)

ASM: call F_SP_RampUpDAC1

Parameters: None

Return Value: None

Library: sacmvxx.asm

Remark: Programmers must stop sending data to DAC1 while executing this function. Make sure there is no interrupt routine still sending data to DAC1now since it will destroy the process of DAC1 Ramp up process and probably cause noise (usually "burst sound") to audio output.

12.1.2 Function: Ramp up DAC2

Syntax:

C: void SP_RampUpDAC2(void)

ASM: call F_SP_RampUpDAC2

Parameters: None

Return Value: None

Library: sacmvxx.asm

Remark: Programmers must stop sending data to DAC2 while executing this function. Make sure there is no interrupt routine still sending data to DAC1now since it will destroy the process of DAC1 ramp up process and probably cause noise (usually "burst sound") to audio output.

12.1.3 Function: Ramp down DAC1

Syntax:

C: void SP_RampDnDAC1(void)

ASM: call F_SP_RampDnDAC1

Parameters: None

Return Value: None

Library: sacmvxx.asm

Remark: Programmers must stop sending data to DAC1 while executing this function. Make sure there is no interrupt routine still sending data to DAC1now, since it will destroy the process of DAC1 Ramp up process and probably cause noise (usually "burst sound") to audio output.

12.1.4 Function: Ramp down DAC2

Syntax:

C: void SP_RampDnDAC2(void)

ASM: call F_SP_RampDnDAC2

Parameters: None

Return Value: None

Library: sacmvxx.asm

Remark: Programmers must stop sending data to DAC2 while executing this function. Make sure there is no interrupt routine still sending data to DAC1now since it will destroy the process of DAC1 Ramp up process and probably cause noise (usually "burst sound") to audio output.

12.2 AD/DA Function

12.2.1 Function: Switch SPCE ADC channel for recording

Syntax:

C: void SP_SwitchChannel(int ADC_Channel);

ASM: R1 = [ADC_Channel]
Call F_SP_SwitchChannel

Parameters:

ADC_Channel: 0: Microphone, 1~7 for Line-in for SPCE040A/060A/061A

0: Microphone, 1 for Line-in for SPCE120A/250A/380A/500A

Return Value: None

Library: sacmvxx.asm

Remark: None

12.2.2 Function: Get ADC data for recording

Syntax:

ASM: Call F_SACM_DVR1600_GetADC

Parameters: None

Return Value: ADC data from SPCE hardware

Library: sacmvxx.asm

Remark:

This hardware dependent function will implement different code segment based on the BODY_TYPE setting. For details, please refer to spce.inc and sacmvxx.asm.

12.2.3 Function: Send data to DAC1

Syntax:

ASM: R4 = 16-bit unsigned PCM data

Call F_SACM_XXXX_SendDAC1

Parameters: R4 = 16-bit unsigned PCM data

Return Value: None

Library: sacmvxx.asm

Remark:

1. This hardware dependent function will implement different code segment based on the BODY_TYPE setting. For detail, please refer to spce.inc and sacmvxx.asm.
2. XXXX means algorithm title, e.g. S200, S480, S530, MS01, A1600, DVR1600…

12.2.4 Function: Send data to DAC2

Syntax:

ASM: R4 = 16-bit unsigned PCM data

Call F_SACM_XXXX_SendDAC2

Parameters: R4 = 16-bit unsigned PCM data

Return Value: None

Library: sacmvxx.asm

Remark:

1. This hardware dependent function will implement different code segment based on the BODY_TYPE setting. For detail, please refer to spce.inc and sacmvxx.asm.
2. XXXX means algorithm title, e.g. S200, S480, S530, MS01, A1600, DVR1600…

13 How to adapt your old project for new library

13.1 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.

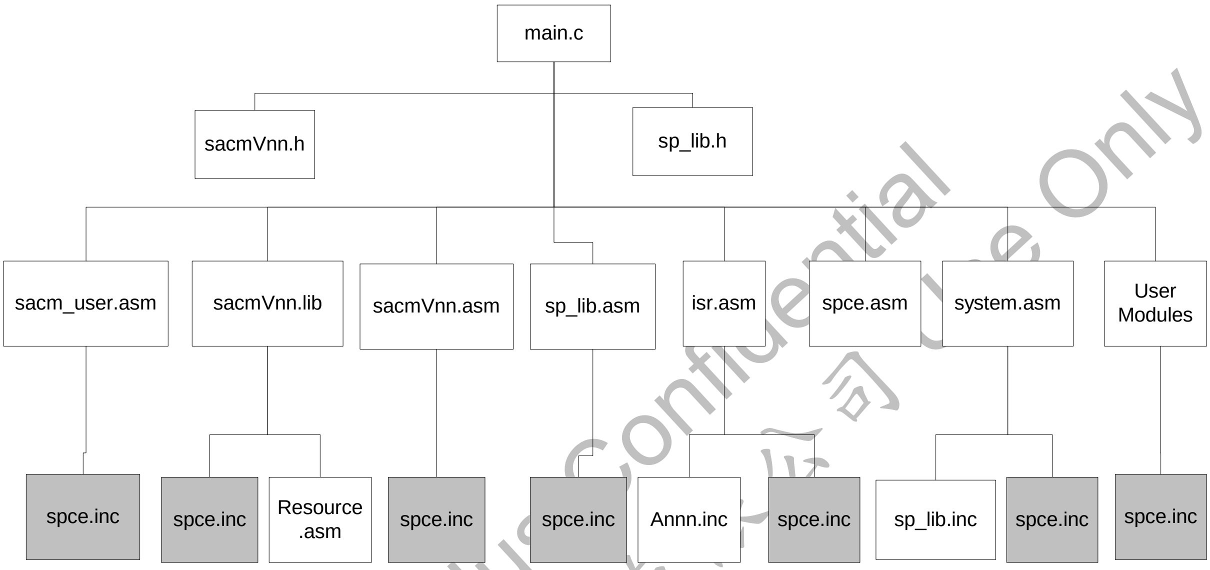


Fig. A typical SACM project architecture

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 applicable) 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.

xxxx.inc (s200.inc, s480.inc, s530.inc, a1600.inc, ms01 and DVR1600.inc): Assembly function declarations for each algorithm.

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

13.2 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 xxxx.inc" on the top of the file.

Where xxxx.inc can be s200.inc, s480.inc, s530.inc, A1600.inc, ms01.inc or DVR1600.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 "sacmvxx.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 , sacmVxx.asm, spce.asm, sp_lib.asm, sacm_user.asm(if applicable), and header files, spce.inc, xxxx.inc and sacmVxx.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.

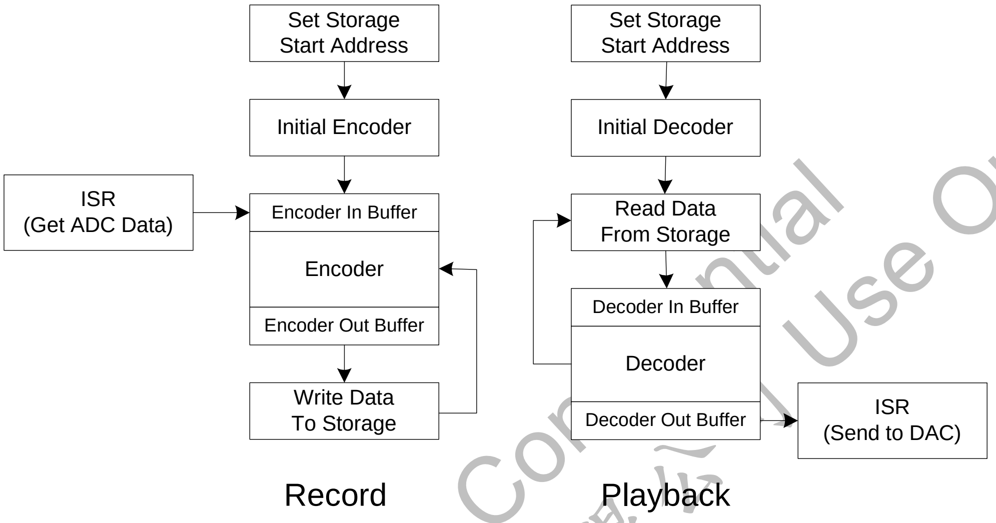
(7) Remove the old library and link the new library in [Project] → [Setting] → [Link]

(8) 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_XXXX_user.asm).

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

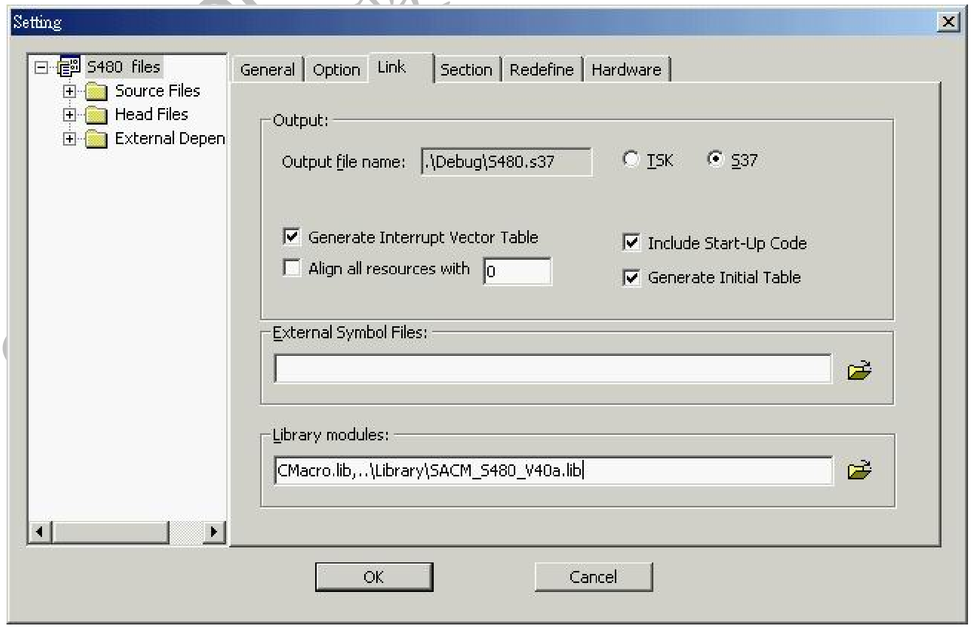
14 How to use the speech library

14.1 The programming flow



14.2 Link the libraries to user's program

[Project] → [Setting] → [Link] and add library, i.e. sacmv40a.lib to the library modules text box.



Output:

Output file name: .\Debug\S480.s37 ○ TSK ● S37

☑ Generate Interrupt Vector Table

□ Align all resources with 0

☑ Include Start-Up Code

☑ Generate Initial Table

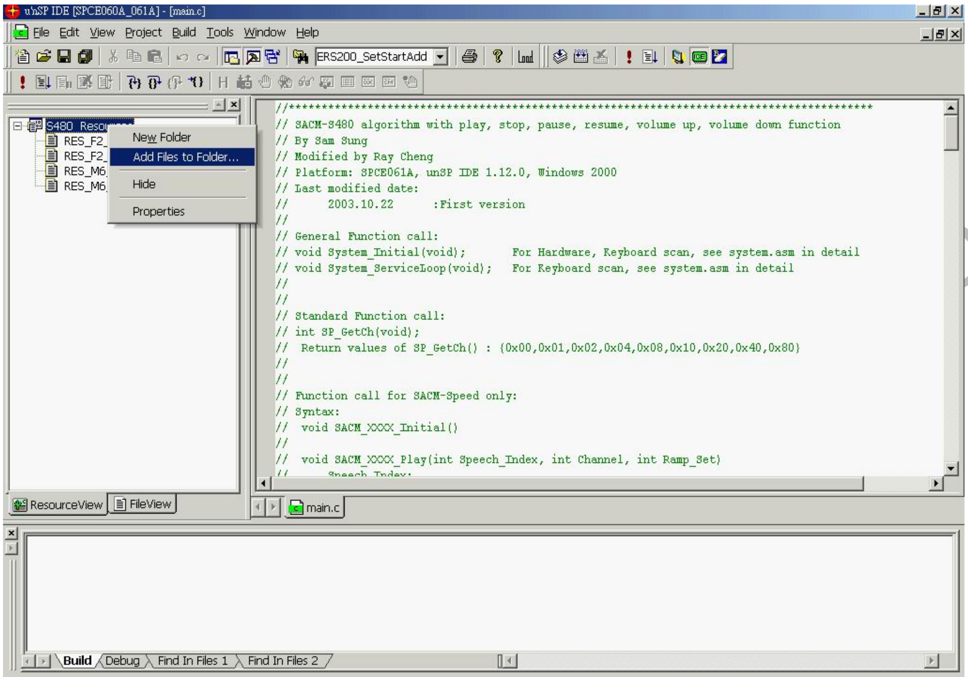
External Symbol Files:

Library modules: CMacro.lib,..\Library\SACM_S480_V40a.lib

Note: If there is any C program in user's project, user must add Cmacro.lib in library modules.

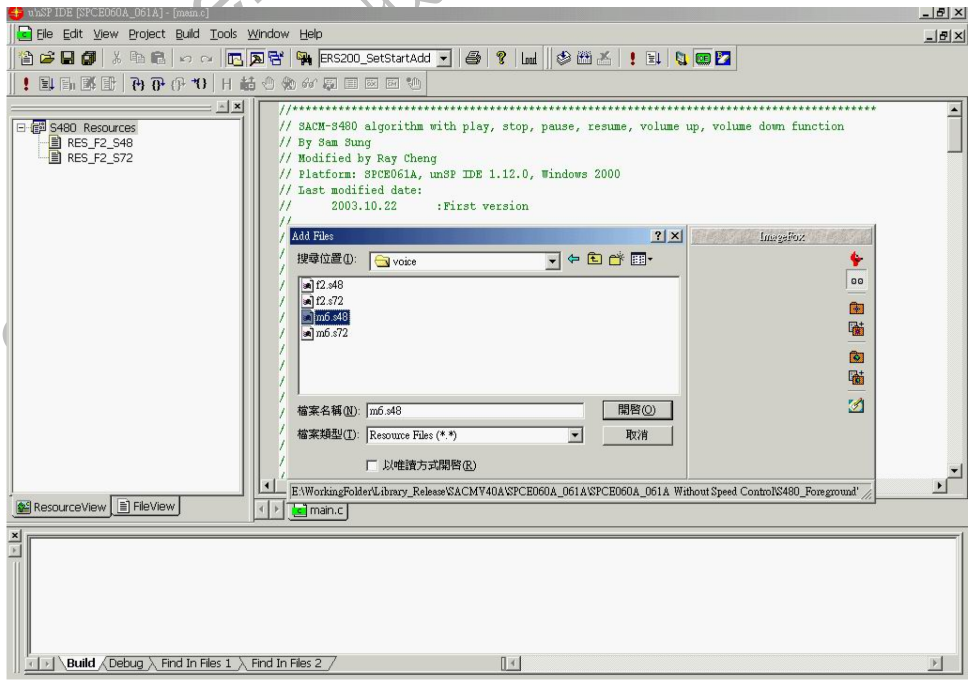
14.3 Adding resources

For adding new resources, user can click on the resource tab in the workspace window, and add in the resource from the file dialogue popped up. Then check out the resource.asm for the speech table for SACM library and edit the speech sequence for users' application.



```
//*****************************************************************************************
// SACM-S480 algorithm with play, stop, pause, resume, volume up, volume down function
// By Sam Sung
// Modified by Ray Cheng
// Platform: SPCE061A, unSP IDE 1.12.0, Windows 2000
// Last modified date:
//      2003.10.22      :First version
//
// General Function call:
// void System_Initial(void);       For Hardware, Keyboard scan, see system.asm in detail
// void System_ServiceLoop(void);   For Keyboard scan, see system.asm in detail
//
//
// Standard Function call:
// int SP_GetCh(void);
//  Return values of SP_GetCh() : {0x00,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}
//
//
// Function call for SACM-Speed only:
// Syntax:
//  void SACM_XXXX_Initial()
//
//  void SACM_XXXX_Play(int Speech_Index, int Channel, int Ramp_Set)
//      Speech_Index:
```

Resource Window:



```
//*****************************************************************************************
// SACM-S480 algorithm with play, stop, pause, resume, volume up, volume down function
// By Sam Sung
// Modified by Ray Cheng
// Platform: SPCE061A, unSP IDE 1.12.0, Windows 2000
// Last modified date:
//      2003.10.22      :First version
//
```

搜尋位置(I): voice

f2.s48
f2.s72
m6.s48
m6.s72

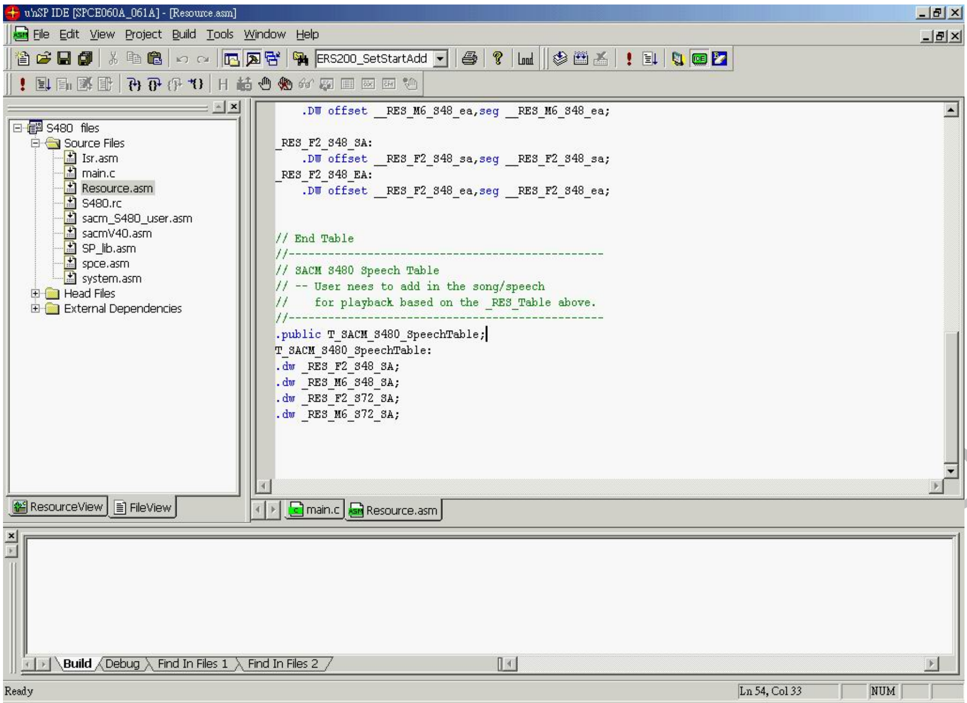
檔案名稱(N): m6.s48

檔案類型(T): Resource Files (*.*)

□ 以唯讀方式開啓(R)

E:\WorkingFolder\Library_Release\SACMV40A\SPCE060A_061A\SPCE060A_061A Without Speed Control\S480_Foreground'

Adding resources:



```
    .DW offset __RES_M6_S48_ea,seg __RES_M6_S48_ea;

_RES_F2_S48_SA:
    .DW offset __RES_F2_S48_sa,seg __RES_F2_S48_sa;
_RES_F2_S48_EA:
    .DW offset __RES_F2_S48_ea,seg __RES_F2_S48_ea;


// End Table
//-----------------------------------------------
// SACM S480 Speech Table
// -- User nees to add in the song/speech
//    for playback based on the _RES_Table above.
//-----------------------------------------------
.public T_SACM_S480_SpeechTable;
T_SACM_S480_SpeechTable:
.dw _RES_F2_S48_SA;
.dw _RES_M6_S48_SA;
.dw _RES_F2_S72_SA;
.dw _RES_M6_S72_SA;
```

Edit the Speech table (e.g. T_SACM_S480_SpeechTable for SACM S480)

Note: The _RES_F2_S48_SA means the "**S**"tart "**A**"ddress of "**RES**"ource file F2.S48.

14.4 A simple example

Main.C

```
Main()
{
        SACM_A1600_Initial();                          // Initialization for playing
        SACM_A1600_Play(Speech_Index, DAC1+DAC2, Ramp_UpDn_On);
        While(1)
        {
                SACM_A16000_ServiceLoop();
        }
        Return 0;
}
```

Note: The Speech Index is the sequence in speech table, i.e. T_SACM_A1600_SpeechTable in resource.asm.

ISR.ASM

```
        _FIQ:
        push R1, R5 to [SP];
        call F_FIQ_Service_SACM_A1600;                // Interrupt Service
        R1 = C_FIQ_TMA;                               // Clear Interrupt flag
        [P_INT_Clear] = R1;
        pop R1, R5 from [SP];
        reti;
```

Resource.asm

```
T_SACM_A1600_SpeechTable:
        .DW Speech_Addr1                              // Speech index = 0
        .DW Speech_Addr2                              // Speech index = 1
```

14.5 Quick instructions

The easiest way to start your own SACM project is to start from a sample project in SACM library package. Then you can simply insert your application code into the sample project.

Instructions:

1. Open the sample project that contains the SACM algorithm you need.
2. Modified the Hardware setting in spce.inc and sacmvxx.asm in necessary. (e.g. BODY_TYPE, C_SystemClock, C_A1600_Timer_Setting, F_SP_SACM_A1600_Init_)
3. Rebuild project
4. Add user resources into the project resources, rebuild project, check ROM allocation and edit Speech Tables. (e.g. T_SACM_A1600_SpeechTable)
5. Rebuild project.
6. Rewrite main.c and rebuild project to test the speech files you just added. (e.g. MaxSpeechNum)
7. Add application code to the project in either C or assembly and modified the main.c (main.asm) for flow control.

14.6 Interrupt Status Register

Programmers may share the interrupt source with SACM library, A register, R_InterrruptStatus is a public interrupt control register in spce.asm for SPCE120A/250A/380A500A. This register is reserved for sharing the interrupt source and it 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_A1600_Play() is called. In contrast, SACM_A1600_Stop() will not change the INT setting from “0x2000” to “0x0000” when SACM_A1600 is called. Every interrupt setting must follow the rule to share the interrupt resource. The following is an example to enable IRQ4 while SACM-A1600 speech is playing.

```
R1 = 0;                          // At beginning, R_InterruptStatus = 0x0000
call    F_SACM_A1600_Play        // The R_InterruptStatus change to 0x2000 by
                                 // F_SACM_A1600_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
                                 // SPCE040A/060A/061A
.endif
.if BODY_TYPE == SPCE500A
    R1 = [R_InterruptStatus]     // Get current interrupt setting status from R_InterruptStatus if for
                                 // SPCE120A/250A380A/500A
.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 SPCE040A/060A/061A, 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.

15 How to setup concurrent algorithms in your application

For users who would like to design an application with 2 algorithms up and running, there are 2 issues to consider, real-time and resources.

Resource issue is quite straight forward, and it is about if IC can afford 2 algorithm running at the same time in terms of CPU performance, RAM, ROM and interrupts. The resource information of each individual algorithm can be referred in the appendix of this document. For RAM allocation, there is a technique, ORAM section, which is also covered in this document and can be applied at users' convenience. Currently, only SPCE060A/061A/040A can support concurrent algorithm applications. For interrupt issue, since each of the SACM algorithms requires one or more interrupts to deliver audio output to DAC, the concurrent algorithms will either takes up 2 interrupts or share the same interrupts at the same interrupt frequency. User is free to choose either user 1 timer or 2 timers for concurrent algorithms.

The real time issue is about the service loop and ISR. In each algorithm, certain decode process has to be carried out in a certain of period. If the decode process cannot be executed successfully, it will cause malfunction of algorithm. User has to refer to the service loop timing requirement information in the appendix. It is advised that for the concurrent algorithm applications, user can use one foreground service loop and one background service loop to allow the background service loop to be executed timely. If user chose both to be with foreground service loop, user is advised to handle the service loop sequence with discretion.

After the evaluation based on real-time and resource issues, user may still need to try the possible combinations for optimal presentation of the SACM algorithms.

Instructions:

Auto Mode (MS01 + A1600):

In SPCE060/061, SACM MS01 can play with other algorithms such as A1600, S200, S480, and S530. Below is an example showing how to play ms01 and A1600 (background) simultaneously.

Step 1: Decide what FIQ or IRQ service routine your service loop want to hooks on. It is suggested that hooks your ms01 service loop and A1600 service loop on FIQ Timer A at the same interrupt frequency.

Step 2: Modified the parameter setting in sacmv40.asm. First check F_SP_SACM_A1600_Init_

a. Set the P_SystemClock in Sacm_A1600_Init_ to choose system clock.

b. Because A1600 service loop is on FIQ timer A, select the clock source of timer A in P_TimerA_Ctrl and timer setting in P_TimerA_Data.

c. Turn off ADC and Mic by setting P_ADC_Ctrl, if not use.

d. Assign DAC channels to your speech and melody . For example, if user wants to play A1600 on DAC1 and melody on DAC2, user shall write 0x00A0 to P_DAC_Ctrl.

e. Enable timer A FIQ for A1600 service loop.

Below is an example of initial A1600.

EX:

```
F_SP_SACM_A1600_Init_:
        R1 = C_SystemClock;              // CPU Clock setting
        [P_SystemClock]=R1;
        R1 = 0x0030;                     // TimerA CKA=Fosc/2 CKB=1 Tout:off
        [P_TimerA_Ctrl] = R1;
        R1 = C_A1600_Timer_Setting;      // TimerA setting
        [P_TimerA_Data] = R1;
        R1 = 0x0002;                     // Disable ADC, disable MIC_In
        [P_ADC_Ctrl] = R1;
        R1 = 0x00A0;                     // Latch DAC1 by Timer A; Latch DAC2 by Timer A;
        [P_DAC_Ctrl] = R1;

        R1 = 0xffff;
        [P_INT_Clear] = R1;              // Clear interrupt occupied events

        .if BODY_TYPE == SPCE061A
        R1 = [P_INT_Mask];
        .endif
        .if BODY_TYPE == SPCE500A
        R1 = [R_InterruptStatus];
        .endif

        R1 |= C_FIQ_TMA;                 // Enable Timer B FIQ
        [R_InterruptStatus] = R1;
        [P_INT_Ctrl] = R1;
        retf
```

Step 3: Check the F_SP_SACM_MS01_Init_ for system clock, and timer A setting.

F_SP_SACM_MS01_Init_:

EX:

```
     R1 = C_SystemClock;                    // CPU Clock setting
    [P_SystemClock]=R1;
    R1 = 0x0030;                            // TimerA CKA=Fosc/2 CKB=1 Tout:off
    [P_TimerA_Ctrl] = R1;
    R1 = C_MS01_Timer_Setting;              // TimerA setting
    [P_TimerA_Data] = R1;
     R1 = 0x0002;                           // Disable ADC, disable MIC_In
     [P_ADC_Ctrl] = R1;
    R1 = 0x00A0;                            // Latch DAC1 by Timer A; Latch DAC2 by Timer A;
    [P_DAC_Ctrl] = R1;

    R1 = 0xffff;
    [P_INT_Clear] = R1;                     // Clear interrupt occupied events

    .if BODY_TYPE == SPCE061A
    R1 = [P_INT_Mask];
    .endif
    .if BODY_TYPE == SPCE500A
    R1 = [R_InterruptStatus];
    .endif

    R1 |= C_FIQ_TMA;                        // Enable Timer B FIQ
    [R_InterruptStatus] = R1;
    [P_INT_Ctrl] = R1;
     retf
```

Step 4: In order to play two algorithms simultaneously and correctly, the service loop routines of MS01 and A1600 are set according to user's choice in step1.

EX:

```
  _FIQ:
     push R1,R5 to [SP];
    R1 = C_FIQ_TMA;
    test R1,[P_INT_Ctrl];
    jne L_FIQ_TimerA;                          // Timer A FIQ entrance
    R1 = C_FIQ_TMB;
    test R1,[P_INT_Ctrl];
```

```
         jne L_FIQ_TimerB;                        // Timer B FIQ entrance
L_FIQ_TimerA:
        //----------------------------------------------------------------------------------------------------
        // hook Timer A FIQ subroutine here and define it to be external
        call      F_ISR_Service_SACM_A1600;          //A1600   FIQ   TMA Service
        call      F_ISR_Service_SACM_MS01;           // MS01   FIQ_TMA Service
        //----------------------------------------------------------------------------------------------------
        R1 = F_SACM_A1600_ServiceLoop_ISR;       // Background Service loop for SACM1600
        push R1 to [SP];
        push SR to [SP];

        R1 = C_FIQ_TMA;
         [P_INT_Clear] = R1;
        reti;
L_FIQ_TimerB:
        R1 = C_FIQ_TMB;
        [P_INT_Clear] = R1;
        pop R1, R5 from [sp];
        reti;
```

Step 5: In the main.c, initialize the SACM-MS01 and SACM-A1600. Users can arrange the songs and instruments in T_SACM_MS01_SpeechTable, and T_SACM_MS01_DrumTable in resource.asm at you own. The speech table for A1600 is T_SACM_A1600_SpeechTable.

EX:

```
main()
{
        System_Initial();
        SACM_A1600_Initial();                                    // Initial A1600
        SACM_MS01_Initial();                                     // Initial MS01
        SACM_A1600_Play(0,DAC1,Ramp_Up+Ramp_Dn);                 // Play speech on DAC2
        SACM_MS01_Play(0,DAC2,Ramp_Up+Ramp_Dn);                  // Play melody on DAC1
        while(1)
        {
                SACM_MS01_ServiceLoop();                         // Service loop for MS01
        }
}
```

Step 6: Add user application code and it is done.

Manual Mode: (MS01+ A1600)

In manual mode, users can read speech data from internal ROM or external memory. User implements USER_XXXX_SetStartAddr and F_USER_XXXX_GetData 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. ROM, FLASH.

Step 1: Decide what FIQ or IRQ service routine your service loop want to hooks on. It is suggested that hooks your ms01 service loop and A1600 service loop on FIQ Timer A at the same interrupt frequency.

Step2 : Modify USER_A1600_SetStartAddr, F_USER_A1600_GetData, USER_MS01_SetStartAddr and F_USER_MS01_GetData. User implements these functions based on the external storage type. For example, the speech data are stored in internal ROM area.

```
//--------------------------------------------------------------------------------------------------
//-- Procedure: _USER_A1600_SetStartAddr
//-- Syntax: USER_A1600_SetStartAddr(int)
//-- Parameter: R1 = PlayIndex
//-- Return: NONE
//-- Description: This API allows users to set the beginning address
//                      to fetch data. This address can be either a ROM address
//                      or a external storage address. User would have to modify
//                      the function body based on the application's need.
//----------------------------------------------------------------------------------------------------
_USER_A1600_SetStartAddr: .proc
F_USER_A1600_SetStartAddr:
        push R1, R2 to [sp];
        R1 += T_SACM_A1600_SpeechTable;
        R1 = [R1];
        R2 = [R1 ++];
        [R_A1600_Resouce_BS] = R2;
        R1 = [R1];
        R1 = R1 LSL 4;
        R1 = R1 LSL 4;
        R1 = R1 LSL 2;
        [R_A1600_Resouce_DS] = R1;
        pop R1, R2 from [sp];
```

```
          retf
          .endp

//---------------------------------------------------------------------------------------------------
//-- Procedure: _USER_MS01_SetStartAddr
//-- Syntax: USER_MS01_SetStartAddr(int)
//-- Parameter: R1 = PlayIndex
//-- Return: NONE
//-- Description: This API allows users to set the beginning address
//                          to fetch data. This address can be either a ROM address
//                          or a external storage address. User would have to modify
//                          the function body based on the application's need.
//---------------------------------------------------------------------------------------------------
_USER_MS01_SetStartAddr:  .proc
F_USER_MS01_SetStartAddr:
          push R1, R2 to [sp];
          R1 += T_SACM_MS01_SpeechTable;
          R1 = [R1];
          R2 = [R1 ++];
          [R_Resouce_BS] = R2;
          R1 = [R1];
          R1 = R1 LSL 4;
          R1 = R1 LSL 4;
          R1 = R1 LSL 2;
          [R_Resouce_DS] = R1;
          pop R1, R2 from [sp];
          retf
          .endp

//---------------------------------------------------------------------------------------------------
//-- Function: F_USER_A1600_GetData
//-- Parameter: R1 = the address to store ()
//                       R2 = the to store
//-- Return: Block data (can be a word, a byte or an array)
//-- Description: This function called by library to fetch data blocks
//                          for playback. The Library needs a data block specified
//                          by R1,the start address, and R2, the length of the block.
//                          Uaser would have to compose the function body
//                          based on the storage type to fulfill this demand from library.
//-------------------------------------------------------------------------------------------------------
F_USER_A1600_GetData:             .proc
        //... user implement this interface to get external data
          push R1, R4 to [SP];
          SR = [R_A1600_Resouce_DS];
          R3 = [R_A1600_Resouce_BS];
?L_Get_Loop:
          cmp R2, 0;
          jz ?L_End;
          R4 = D:[R3++];
          [R1++] = R4;
          R2 -= 1;
          cmp R3, 0;
          jnz ?L_Get_Loop;
          SR += 0x0400;
          [R_A1600_Resouce_DS] = SR;
          jmp ?L_Get_Loop;
?L_End:
          [R_A1600_Resouce_BS] = R3;
          pop R1, R4 from [SP];
          retf;
          .endp

//-------------------------------------------------------------------------------------------------------
//-- Function: F_USER_MS01_GetData
//-- Parameter: R1 = the address to store ()
//                       R2 = the to store
//-- Return: Block data (can be a word, a byte or an array)
//-- Description: This function called by library to fetch data blocks
//                          for playback. The Library needs a data block specified
//                          by R1,the start address, and R2, the length of the block.
//                          Uaser would have to compose the function body
//                          based on the storage type to fulfill this demand from library.
//-------------------------------------------------------------------------------------------------------
F_USER_MS01_GetData:              .proc
```

```
        //... user implement this interface to get external data
        push R1, R4 to [SP];
        SR = [R_Resouce_DS];
        R3 = [R_Resouce_BS];
?L_Get_Loop:
        cmp R2, 0;
        jz ?L_End;
        R4 = D:[R3++];
        [R1++] = R4;
        R2 -= 1;
        cmp R3, 0;
        jnz ?L_Get_Loop;
        SR += 0x0400;
        [R_Resouce_DS] = SR;
        jmp ?L_Get_Loop;
?L_End:
        [R_Resouce_BS] = R3;
        pop R1, R4 from [SP];
        retf;
.endp
```

Step 3: Modified the parameter setting in sacmv40.asm. First check F_SP_SACM_A1600_Init_

a. Set the P_SystemClock in Sacm_A1600_Init_ to choose system clock.

b. Because A1600 service loop is on FIQ timer A, select the clock source of timer A in P_TimerA_Ctrl and timer setting in P_TimerA_Data.

c. Turn off ADC and Mic by setting P_ADC_Ctrl, if not use.

d. Assign DAC channels to your speech and melody . For example, if user wants to play A1600 on DAC1 and melody on DAC2, user shall write 0x00A0 to P_DAC_Ctrl.

e. Enable timer A FIQ for A1600 service loop.

Below is an example of initial A1600.

EX:

```
F_SP_SACM_A1600_Init_:
             R1 = C_SystemClock;                 // CPU Clock setting
            [P_SystemClock]=R1;
            R1 = 0x0030;                         // TimerA CKA=Fosc/2 CKB=1 Tout:off
            [P_TimerA_Ctrl] = R1;
            R1 = C_A1600_Timer_Setting;          // TimerA setting
```

```
        [P_TimerA_Data] = R1;
         R1 = 0x0002;                        // Disable ADC, disable MIC_In
         [P_ADC_Ctrl] = R1;
        R1 = 0x00A0;                         // Latch DAC1 by Timer A; Latch DAC2 by Timer A;
        [P_DAC_Ctrl] = R1;

        R1 = 0xffff;
        [P_INT_Clear] = R1;                  // Clear interrupt occupied events

        .if BODY_TYPE == SPCE061A
        R1 = [P_INT_Mask];
        .endif
        .if BODY_TYPE == SPCE500A
        R1 = [R_InterruptStatus];
        .endif

        R1 |= C_FIQ_TMA;                     // Enable Timer B FIQ
        [R_InterruptStatus] = R1;
        [P_INT_Ctrl] = R1;
         retf
```

Step 4: Check the F_SP_SACM_MS01_Init_ for system clock, and timer A setting.

F_SP_SACM_MS01_Init_:

EX:

```
         R1 = C_SystemClock;                 // CPU Clock setting
        [P_SystemClock]=R1;
        R1 = 0x0030;                         // TimerA CKA=Fosc/2 CKB=1 Tout:off
        [P_TimerA_Ctrl] = R1;
        R1 = C_MS01_Timer_Setting;           // TimerA setting
        [P_TimerA_Data] = R1;
         R1 = 0x0002;                        // Disable ADC, disable MIC_In
         [P_ADC_Ctrl] = R1;
        R1 = 0x00A0;                         // Latch DAC1 by Timer A; Latch DAC2 by Timer A;
        [P_DAC_Ctrl] = R1;

        R1 = 0xffff;
        [P_INT_Clear] = R1;                  // Clear interrupt occupied events
        .if BODY_TYPE == SPCE061A
        R1 = [P_INT_Mask];
        .endif
        .if BODY_TYPE == SPCE500A
        R1 = [R_InterruptStatus];
        .endif

        R1 |= C_FIQ_TMA;                  // Enable Timer B FIQ
        [R_InterruptStatus] = R1;
        [P_INT_Ctrl] = R1;
         retf
```

Step 5: In order to play two algorithms simultaneously and correctly, the service loop routines of MS01 and A1600 are set according to user's choice in step1.

EX:

```
      _FIQ:
        push R1,R5 to [SP];
       R1 = C_FIQ_TMA;
       test R1,[P_INT_Ctrl];
       jne L_FIQ_TimerA;                      // Timer A FIQ entrance
       R1 = C_FIQ_TMB;
       test R1,[P_INT_Ctrl];
       jne L_FIQ_TimerB;                      // Timer B FIQ entrance
L_FIQ_TimerA:
      //------------------------------------------------------------------------------------------------------
      // hook Timer A FIQ subroutine here and define it to be external
     call      F_ISR_Service_SACM_A1600;           //A1600    FIQ    TMA Service
      call      F_ISR_Service_SACM_MS01;           // MS01    FIQ_TMA Service
      //------------------------------------------------------------------------------------------------------
      R1 = F_SACM_A1600_ServiceLoop_ISR;       // Background Service loop for SACM1600
      push R1 to [SP];
      push SR to [SP];

      R1 = C_FIQ_TMA;
       [P_INT_Clear] = R1;
      reti;
```

```
L_FIQ_TimerB:
        R1 = C_FIQ_TMB;
        [P_INT_Clear] = R1;
        pop R1, R5 from [sp];
        reti;
```

Step 6: In the main.c, initialize the SACM-MS01 and SACM-A1600. Users can arrange the songs and instruments in T_SACM_MS01_SpeechTable, and T_SACM_MS01_DrumTable in resource.asm at you own. The speech table for A1600 is T_SACM_A1600_SpeechTable.

EX:

```
main()
{
        System_Initial();
        SACM_A1600_Initial();                                   // Initial A1600
        SACM_MS01_Initial();                                    // Initial MS01
        USER_A1600_SetStartAddr(SpeechIndex);                   // Set start address of A1600 speech data
        SACM_A1600_Play(0,DAC1,Ramp_Up+Ramp_Dn);                // Play speech on DAC2
        USER_MS01_SetStartAddr(SongIndex);                      // Set start address of MS01 music data
        SACM_MS01_Play(0,DAC2,Ramp_Up+Ramp_Dn);                 // Play melody on DAC1
        while(1)
        {
                SACM_MS01_ServiceLoop();                        // Service loop for MS01
        }
}
```

Step 7: Add user application code and it is done.

16 Resources List of SACM algorithm

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

	IRAM	ISRAM	RAM	SRAM	ORAM	OSRAM
A1600	-	-	-	-	457	-
S530	-	-	-	-	399	-
S480/S720	-	-	-	-	464	-
MS01	-	-	-	-	448	-
DVR1600	-	-	-	-	525	-
A1600_SC	-	-	-	-	1448	-
S530_SC	-	-	-	-	1372	-
S480_SC/S720_SC	-	-	-	-	1437	-
S200_SC	-	-	-	-	1220	-
DVR1600_SC	-	-	-	-	1486	-

Note: DVR1600 includes both A1600 Encoder and A1600 Decoder algorithm

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

	TEXT	CODE	DATA	USER DEFINE
A1600		~3.5K		
S530		~8.3K		
S480/S720		~1.5K		
MS01		~3.4K		
DVR1600		~4.5K		
A1600_SC		~5.0K		
S530_SC		~9.8K		
S480_SC/S720_SC		~3.8K		
S200_SC		~7.5K		
DVR1600_SC		~5.5K		

16.3 TABLE 3: Hardware Resources VS Library

	Interrupt	Timer Setting	Audio
A1600	TMA FIQ	16 KHz	DAC
S530	TMA FIQ	16 KHz~24 KHz	DAC
S480/S720	TMA FIQ	16 KHz	DAC
MS01	TMA FIQ	16 KHz	DAC
DVR1600	TMA FIQ	16 KHz(Play) /32 KHz(Rec)	ADC/DAC

	Interrupt	Timer Setting	Audio
A1600_SC	TMA FIQ	16 KHz	DAC
S530_SC	TMA FIQ	16 KHz~24 KHz	DAC
S480_SC/S720_SC	TMA FIQ	16 KHz	DAC
S200_SC	TMA FIQ	16 KHz	DAC
DVR1600_SC	TMA FIQ	16 KHz(Play) /32 KHz(Rec)	ADC/DAC

16.4 TABLE 4: CPU Usage Rate (approximate)

	SPCE500A at 24 MHz	SPCE061A at 24M Hz	SPCE061A at 49 MHz
A1600	52%	38%	19%
S530	58%@8KHz(5.3Kbps) 75%@10KHz(6.6Kbps) N/A @12KHz(7.9Kbps)	46%@8KHz(5.3Kbps) 56%@10KHz(6.6Kbps) 65% @12KHz(7.9Kbps)	21%@8KHz(5.3Kbps) 28%@10KHz(6.6Kbps) 34% @12KHz(7.9Kbps)
S480/S720	50%@8KHz(4.8Kbps) 52% @8KHz(7.2Kbps)	35%@8KHz(4.8Kbps) 39% @8KHz(7.2Kbps)	20%@8KHz(4.8Kbps) 21% @8KHz(7.2Kbps)
S200	N/A	78%	37%
MS01	67% 1Ch@16KHz 78% 2Ch@16KHz 88% 3Ch@16KHz N/A 4Ch@16KHz	50% 1Ch@16KHz 59% 2Ch@16KHz 68% 3Ch@16KHz 76% 4Ch@16KHz	24% 1Ch@16KHz 28% 2Ch@16KHz 31% 3Ch@16KHz 36% 4Ch@16KHz
DVR1600	N/A	81%	39%

16.5 TABLE 5: Timing Limitation (approximate)

	Service Loop Time Limit
A1600	16 ms
S530	7.5 ms (5.3Kbps) 6.0 ms (6.6Kbps) 5.0 ms (7.2Kbps)
S480	7.5 ms
S720	5.0ms
S200	12.5ms
MS01	16ms
DVR1600	16ms

Note:

1. The number is the maximum interval in which the program have to execute service loop. These figures are measured with SACM v40 library
2. For example, the F_SACM_A1600_ServiceLoop must be called each 16ms in user's main loop. Otherwise, It

may cause noise to audio output.

```
main()
{
        SACM_A160_Initial();
        SACM_A1600_Play(0,3,3);
        While(1)
        {
                User_Function();
                …
                SACM_A1600_ServiceLoop();        <= Go here in each 16ms
        }
}
```

16.6 TABLE 6: Name of Overlap RAM in the library

Table: Name and size of Overlap RAM in the library

Overlap RAM definition		
Algorithm	**Overlap RAM Label**	**Size (word)**
A1600	OVERLAP_DVR1600_RAM_BLOCK	319 (0x13F)
	OVERLAP_A1600_API_BLOCK	2 (0x2)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
A1600-SC	OVERLAP_DVR1600_RAM_BLOCK	781 (0x30D)
	OVERLAP_A1600_API_BLOCK	2 (0x2)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
	OVERLAP_DVR1600_SPEED_RAM_BLOCK	529 (0x211)
DVR1600	OVERLAP_DVR1600_RAM_BLOCK	354 (0x162)
	OVERLAP_DVR1600_API_BLOCK	351 (0x23)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x 88)
DVR1600-SC	OVERLAP_DVR1600_RAM_BLOCK	786 (0x312)
	OVERLAP_DVR1600_API_BLOCK	35 (0x23)
	OVERLAP_DVR1600_DM_BLOCK	136 (0x88)
	OVERLAP_DVR1600_SPEED_RAM_BLOCK	529 (0x211)
S530	OVERLAP_S530_RAM_BLOCK	169 (0xA9)
	OVERLAP_S530_API_BLOCK	2 (0x2)
	OVERLAP_S530_DM_BLOCK	228 (0xE4)
S530-SC	OVERLAP_S530_RAM_BLOCK	613 (0x265)
	OVERLAP_S530_API_BLOCK	2 (0x2)
	OVERLAP_S530_DM_BLOCK	228 (0xE4)
	OVERLAP_S530_SPEED_RAM_BLOCK	529 (0x211)

S480	OVERLAP_S480_RAM_BLOCK	170 (0xAA)
	OVERLAP_S480_API_BLOCK	2 (0x2)
	OVERLAP_S480_DM_BLOCK	292 (0x124)
S480-SC	OVERLAP_S480_RAM_BLOCK	614 (0x266)
	OVERLAP_S480_API_BLOCK	2 (0x2)
	OVERLAP_S480_DM_BLOCK	292 (0x124)
	OVERLAP_S480_SPEED_RAM_BLOCK	529 (0x211)
S200-SC	OVERLAP_S200_RAM_BLOCK	718 (0x2CE)
	OVERLAP_S200_API_BLOCK	2 (0x2)
	OVERLAP_S200_DM_BLOCK	500 (0x1F4)
MS01	OVERLAP_MS01_RAM_BLOCK	318 (0x13E)
	OVERLAP_MS01_DM_BLOCK	130 (0x82)

Where XXXX_SC means that algorithm support speed control function.