GPCM SACM Library User's Manual

1.3 – Oct. 17, 2024

No.19, Industry E. Rd. IV, Hsinchu Science Park, Hsinchu City 30077, Taiwan, R.O.C.

Tel: 886-3-666-2118 Fax: 886-3-666-2117 Web: www.generalplus.com

1 Table of Content

2 Revision History

Revision	Date	by	Remark
V1.3	10/17/2024	Kim Huang	1. Add the SACM-A2000 introduction (new chapters 9 & 10) 2. Modify the SACM-A1801 chapter & update the Library, 3. Add MS02 Infinite loop Enable/Disable function description 4. Add MS02 IO Event function description 5. Add the GPCM3 CPU usage rate & resource
V1.2	08/03/2023	Kim Huang	1. Added 9 functions of the MS02PN algorithm.
V1.1	06/20/2021	Kim Huang	1. Add DVRPCM description. 2. Add MS02_PN(Play Note) description. 3. MS02 library functions revised. 4. Added union method in Memory Allocation chapter to declare shared RAM. 5. Update the resource list.
V1.0	09/15/2020	Kim Huang	2. Add A3400Pro & Ch2 with event description 3. MS02 library function revised, mainly add the settings for Max. MIDI Ch number and relevant descriptions
V0.9	07/27/2020	Kim Huang	First edition

3 Type of Speech Compression Algorithm

3.1 Summary

Audio

Present Algorithm Title	Data Rate	Sample Rate	Application
SACM-A1801	7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps	16KHz	Speech and audio
SACM-A1801 Ch2	7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps	16KHz	Speech and audio
SACM-A3400Pro	34 Kbps(4-bit)/ 42 Kbps(5-bit)	8KHz	Speech and audio
SACM-A3400Pro Ch2	34 Kbps(4-bit)/ 42 Kbps(5-bit)	8KHz	Speech and audio

Melody

Present Algorithm Title	Type	Channel	Sample Rate	Application
SACM-MS02	Wave Table	1 ~ 32	12K/ 16K/ 20K/ 24K/ 32KHz	Music Synthesizer
SACM-MS02PN	Wave Table	1 ~ 16	8K/ 10K/12K/ 16K/ 20K/ 24K/ 28K/ 32K/ 36KHz/ 40K	Play Note or Drum

Recording

Present Algorithm Title	Data Rate	Sample Rate	Application
SACM-DVR1801	7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps	16KHz	Audio Recording

3.2 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

DVR: Digital Voice Recording

Example: SACM A1801 is the Generalplus's audio algorithm with normal data rate of 16Kbps. The actual data rate depends on the options provided and the sampling rate adopted.

4 Service Loop

In SACM library, the primary job of service loop function is to execute the decode process. The service loop function is located inside main loop, which will keep proceeding the service loop. Inside the SACM service loop, there is a mechanism to determine if any task should be carried on. Some overheads are produced inevitably. The amount of overhead may vary based on the payload of CPU.

Example:

In main.c:

```
int main()
{
    SystemInit();
    SPIFC_Open();                                   // SPIFC initial
    SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
    SPIFC_AutoMode(SPIFC_4IO_ENHANCE_MODE);
    SPIFC_TimingFineTune();                         // Calibrate SPIFC clock timing

    A1801Num = GetA1801Num();
    SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer);
    SACM_A1801_Play(GetA1801StartAddr(A1801PlayIdx), A1801_DAC_CH0 , A1801_AUTO_RAMP_UP +
                    A1801_AUTO_RAMP_DOWN);
    while(1)
    {
        SACM_A1801_ServiceLoop();
    } // end of while(1)
    return 0;
} // end of main
```

In isr.c:

1. **DMA ISR:**
   (a) DMA ISR service is able to send the SACM-decoded speech data to DAC_out. The size of 320-byte is called a frame size. When a frame-size has been sent out, the DMA INT will be triggered and notify SACM service loop function to proceed decoding next frame.
   (b) The xxx_CB_Init() inside SACM_xxx_User.c, e.g. A1801_CB_Init(), allows user to determine to which channel, Ch0 or Ch1, send the decoded speech data as well as which timer to be used. The default DAC Ch0 is DMA0 and Ch1 is DMA1; The default timer is TM0.

```
void A1801_CB_Init(const SACM_A1801_WORKING_RAM *A1801WorkingRam)
{
  mA1801DacCh0DmaHandle = DMA0;
  mA1801DacCh1DmaHandle = DMA1;
  mA1801DacTimerHandle = TM0;
```

(c) DMA ISR Service adopts the indirect addressing to assign DMA_IRQHandler() to which SACM function being executed. In A1801_CB_StartPlay() inside SACM_xxx_User.c, it calls DMA_InstallIsrService() and pass the SACM_A1801_DmaIsrService() in a pointer form.

```
void A1801_CB_StartPlay(const SACM_A1801_WORKING_RAM *A1801WorkingRam)
{
  DacAutoRampUp(SACM_A1801_GetStatus());                                    // DAC

  if((SACM_A1801_GetStatus() & A1801_ENABLE_DAC_CH0_FLAG) != 0)
  {
    DMA_Init(mA1801DacCh0DmaHandle, DMA_REQSEL_DAC_CH0, DMA_SRC_DATA_16B, DM
    DMA_InstallIsrService(mA1801DacCh0DmaHandle, SACM_A1801_DmaIsrService);
    DMA_EnableInt(mA1801DacCh0DmaHandle);
```

(d) In DMA_InstallIsrService(), mDmaCh0IsrServiceFunPtr()=SACM_A1801_DmaIsrService();

```
void DMA_InstallIsrService(DMA_TYPE_DEF *DmaHandle, void (*DmaIsrServiceFunc)(void))
{
  if(DmaHandle == DMA0)
  {
    mDmaCh0IsrServiceFunPtr = DmaIsrServiceFunc;
  }
  else if(DmaHandle == DMA1)
  {
    mDmaCh1IsrServiceFunPtr = DmaIsrServiceFunc;
  }
  else if(DmaHandle == DMA2)
  {
    mDmaCh2IsrServiceFunPtr = DmaIsrServiceFunc;
  }
}
```

(e) In DMA_IRQHandler(), DMA0/ 1/ 2 ISR will execute the corresponding functions, e.g. mDmaCh0IsrServiceFunPtr() = SACM_A1801_DmaIsrService();

```
void DMA_IRQHandler()
{
  if(((DMA_INT->INTSTS & DMA_INTSTS_DMA0_DONE_INT_FLAG) != 0) &&
  {
    if(mDmaCh0IsrServiceFunPtr == NULL)
    {
    else
    {
      mDmaCh0IsrServiceFunPtr();
    }
```

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 Keil\ Output\ lst directory or in the G+IDE\ bin\ Debug directory.

The capability of sharing RAM is relied on whether variety of algorithms or applications are active simultaneously. If an application only plays one SACM speech algorithm (e.g. only one algorithm, either A1801 or A3400, will be played), RAM share may be adopted to save RAM capacity. If an application contains an SACM speech algorithm (e.g. A1801 or A3400) with a background music (MS02) playing at the same time, RAM allocation must be declared separately, name recommended as WorkingRam.

1. Example of separating RAM
   (a) In the following example, A1801 Ch1 & Ch2 Algorithms will be playback concurrently; Working RAM needs to be declared separately:

```
__attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801WorkingRam;
__attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801Ch2WorkingRam;
__attribute__ ((aligned (4))) SACM_A1801_TEMP_RAM A1801TempBuffer;

A1801Num = GetA1801Num();
SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer);
SACM_A1801_CH2_Initial(&A1801Ch2WorkingRam, &A1801TempBuffer);
```

2. Example of sharing RAM
   (a) 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, different algorithms are allowed sharing the same physical memory block.
   (b) The following section introduces how to configure RAM share using Union + Struct.
      (1) The syntax of Union is similar to the syntax of struct. The major difference between these two is that the variables declared inside union are memory-shared basis. The size of union is determined by the largest value of the data type. For example, if we declare a short (2-byte) and a double (8-byte) inside union, the size of the union is an 8-byte data structure.
      (2) The structure of Union is suitable for integrating different codes which are not being executed simultaneously such as Pitch tracking and A1801 functions. Thus, the RAM required by these two functions can be separately declared in Union.
      (3) The member of Union can be composed of Struct. In the following example, Lib_RAM_Allocate Union is composed of three Structs: PT_RAM, A18_MS02_RAM, and A18_VC_RAM.
      (4) PT_RAM is the RAM used by Pitch Tracking function; it will not be executed along with A1801, MS02, and VC3 concurrently. Thus, they can be separately declared and sharing the same RAM

space.

```
typedef union Lib_RAM_Allocate
{
  struct PT_RAM
  {
  uint8_t PtWorkingRam[sizeof(PT_WORKING_RAM)];
  int16_t PtEvent[C_MAX_EVENT*4];
//  uint8_t MS02PNWorkingRam[sizeof(SACM_MS02PN_WORKING_RAM)];
//  uint8_t MS02PNKernelRam[MS02PN_DEC_MEM_SIZE];
  }PT_RAM;

  struct A18_MS02_RAM
  {
  uint8_t DVR1801_Working_RAM[sizeof(SACM_DVR1801_WORKING_RAM)];
  uint8_t DVR1801_temp_RAM[sizeof(SACM_DVR1801_TEMP_RAM)];
  uint8_t MS02_Lib_RAM[sizeof(SACM_MS02_WORKING_RAM)];
  uint8_t MS02KernelRam[MS02_DEC_Memory];
  }A18_MS02_RAM;

  struct A18_VC_RAM
  {
  uint8_t DVR1801_Working_RAM[sizeof(SACM_DVR1801_WORKING_RAM)];
  uint8_t DVR1801_temp_RAM[sizeof(SACM_DVR1801_TEMP_RAM)];
  uint8_t Vc3ApiWorkingRam[sizeof(SACM_VC3_API_WORKING_RAM)];
  uint8_t Vc3KernelWorkingRam[sizeof(SACM_VC3_KERNEL_RAM)];
  }A18_VC_RAM;
}Lib_RAM_Allocate;
```

(5) In the previous example, because both A1801 and MIDI will be executed concurrently, they cannot share the same RAM space. As a result, we can combine the RAM spaces that these two functions need and declare it as a Struct. The similarity applies to A18_VC_RAM; both A1801 and VC3 will be executed at the same time so that we can declare them together. Thus, Union will find the largest A18_VC_RAM struct from these three structs for the RAM space of the Union to be declared.

```
struct A18_MS02_RAM
{
uint8_t DVR1801_Working_RAM[sizeof(SACM_DVR1801_WORKING_RAM)];
uint8_t DVR1801_temp_RAM[sizeof(SACM_DVR1801_TEMP_RAM)];
uint8_t MS02_Lib_RAM[sizeof(SACM_MS02_WORKING_RAM)];
uint8_t MS02KernelRam[MS02_DEC_Memory];
}A18_MS02_RAM;
```

(c) The following section depicts how to invoke RAM from Union.

(1) For example, declare a RAM space named LibRAM with the structure as Lib_RAM_Allocate Union

```
Aligned4B Lib_RAM_Allocate LibRAM;
```

(2) Invoke syntax is the same as Stuct; for example, the syntax to invoke DVR1801_Working_RAM of A18_MS02_RAM stuct from LibRAM is as follows:

(SACM_DVR1801_WORKING_RAM*)LibRAM.A18_MS02_RAM.DVR1801_Working_RAM;

(3) Make Dvr1801WorkingRam pointer points to the DVR1801_Working_RAM and use the declared RAM in Lib_RAM_Allocate Union.

```
Aligned4B SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam = (SACM_DVR1801_WORKING_RAM*)LibRAM.A18_MS02_RAM.DVR1801_Working_RAM;
Aligned4B SACM_DVR1801_TEMP_RAM *Dvr1801TempBuffer = (SACM_DVR1801_TEMP_RAM*)LibRAM.A18_MS02_RAM.DVR1801_temp_RAM;
Aligned4B SACM_MS02_WORKING_RAM *MS02WorkingRam = (SACM_MS02_WORKING_RAM*)LibRAM.A18_MS02_RAM.MS02_Lib_RAM;
Aligned4B uint8_t *MS02KernelRam = LibRAM.A18_MS02_RAM.MS02KernelRam;

SACM_DVR1801_Initial(Dvr1801WorkingRam, Dvr1801TempBuffer);
SACM_DVR1801_Play((int16_t *)0x04200000, DVR1801_DAC_CH0, DVR1801_AUTO_RAMP_UP + DVR1801_AUTO_RAMP_DOWN);
```

6 API for SACM-A1801

6.1 Hardware Dependent Function: SACM-A1801 initialization

6.1.1 Function: Initializes the A1801 library

Syntax:

C: void SACM_A1801_Initial(SACM_A1801_WORKING_RAM *A1801WorkingRam, SACM_A1801_TEMP_RAM *pA1801TempBuffer, void *A1801PcmBuffer);

Parameters:

*A1801WorkingRam: A1801 library working RAM address.

*pA1801TempBuffer: A1801 temp. buffer address.

* A1801PcmBuffer: A1801 Pcm A/B buffer address.

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A1801 decoder. It also initializes the Timer 0, DAC and enables the DMA IRQ with 16KHz sample rate.
2. The hardware setting is opened for user's reference (see A1801_CB_Init function in SACM_A1801_User.c).

Example:

1. First, declare two RAM spaces: SACM_A1801_WORKING_RAM and SACM_A1801_TEMP_RAM in struct type and named as A1801WorkingRam and A1801TempBuffer respectively.

```
__attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM  A1801WorkingRam;
__attribute__ ((aligned (4))) SACM_A1801_TEMP_RAM  A1801TempBuffer;
__attribute__ ((aligned (4))) SACM_A1801_PCM_BUFFER A1801PcmBuffer;
```

2. Pass the struct addresses of A1801WorkingRam, A1801PcmBuffer and A1801TempBuffer into SACM_A1801_Initial().

```
SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer, &A1801PcmBuffer);
```

6.2 Service Loop Functions: Service loop for SACM-A1801 decoding process

6.2.1 Function: Service loop for SACM-A1801 decoding process

Syntax:

C: void SACM_A1801_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

Example:

```
/*---------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------
int main(void)
{
  SystemInit();

  while(1)
  {
    WDT_Clear();
    SACM_A1801_ServiceLoop();
    KeyScan_ServiceLoop();

    ScanedKey = KeyScan_GetCh();
    switch(ScanedKey)
    {
    if((PlayCon != 0) && (SACM_A1801_Check_Con() == 0))
    {
      SACM_A1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++), A1801_DAC_CH0,
      A1801PlayIdx = A1801PlayIdx > SpeechNum ? 1 : A1801PlayIdx;
    }
  }
```

6.3 Playback Functions: Playback control

6.3.1 Function: Plays a SACM-A1801 speech

Syntax:

C: void SACM_A1801_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr: Pass into A1801PlayIdx and call GetA1801StartAddr () to obtain

a song's start address.

DacChannelSel: A1801_DAC_CH0

A1801_DAC_CH1

AutoRampUpDownCtrl : A1801_AUTO_RAMP_DISABLE

A1801_AUTO_RAMP_UP

A1801_AUTO_RAMP_DOWN

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. The data rate of SACM-A1801 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling rate is 16KHz. The data rate is determined at encode and selected by decoder automatically.
2. The A1801PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A1801Num = 0;
int16_t A1801PlayIdx = 1;
int16_t A1801Ch2PlayIdx = 1;
```

3. User may send out the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A1801_CB_Init() (in SACM_A1801_User.c).
4. Ramp-Up/Dn function may be opted by user whether ramp up (A1801_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801PlayIdx = 1;
…
SACM_A1801_Play(GetA1801StartAddr(A1801PlayIdx), A1801_DAC_CH0 ,
A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);
```

6.3.2 Function: Plays a continued SACM-A1801 speech

Syntax:

C: void SACM_A1801_Play_Con (int16_t *SpeechDataStartAddr, uint8_t DacChannelSel,

uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	pass to 1801PlayIdx and call GetA1801StartAddr() to obtain the start address of a song
DacChannelSel:	A1801_DAC_CH0
	A1801_DAC_CH1
AutoRampUpDownCtrl :	A1801_AUTO_RAMP_DISABLE
	A1801_AUTO_RAMP_UP
	A1801_AUTO_RAMP_DOWN

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay after current speech ends.
2. The data rate of SACM-A1801 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling is 16KHz. The data rate is determined at encoding and selected by decoder automatically.
3. The A1801PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
int main(void)
{
  uint32_t ScanedKey;
  uint16_t A1801Num = 0;
  int16_t A1801PlayIdx = 1;
  int16_t A1801Ch2PlayIdx = 1;
  int8_t PlayCon = 0;
```

4. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A1801_CB_Init() (in SACM_A1801_User.c).
5. Ramp-Up/Dn function may be opted by user whether ramp-up (A1801_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set

them disabled at A1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801PlayIdx = 1;
…
if((PlayCon != 0) && (SACM_A1801_Check_Con() == 0))
    {
        SACM_A1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++),
        A1801_DAC_CH0, A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);

        A1801PlayIdx = A1801PlayIdx > A1801Num ? 1 : A1801PlayIdx;
     }
```

6.3.3 Function: Checks a continued SACM-A1801 speech

Syntax:

C: int SACM_A1801_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up

1: Continued speech has set up

Library: < A1801_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value becomes 0 after continued speech plays.

6.3.4 Function: Stops a playing SACM-A1801 speech

Syntax:

C: void SACM_A1801_Stop (void);

Parameters: None

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. This function is a call-back function of SACM A1801 library when playing A1801 stops. It will perform Ramp-Dn function (if A1801_AUTO_RAMP_DOWN is enabled) and turn off the corresponding DMA. User's function is allowed to be implemented here too.

6.3.5 Function: Pause SACM-A1801 speech playback

Syntax:

C: void SACM_A1801_Pause (void);

Parameters: None

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark: None

6.3.6 Function: Resumes a paused SACM-A1801 speech

Syntax:

C: void SACM_A1801_Resume(void);

Parameters: None

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark: None

6.3.7 Function: Changes the volume of SACM-A1801

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for A1801 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*--------------------------------
 * User Definition Area
 *--------------------------------
#define     VOL_CONTROL_NUM     (4)
```

2. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.

   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as

needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

3. Note.1: APP_SwVolCtrl_VolProcess is located in A1801_CB_DecodeProcess (SACM_A1801_User.c) to process volume.

```
void A1801_CB_DecodeProcess(const SACM_A1801_WORKING_RAM *A1801WorkingRam, int16_t *DstBufAddr,
){
  SACM_A1801_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(0, DstBufAddr, A1801_FRAME_SIZE);
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;
```

6.3.8 Function: Gets the status of the SACM-A1801 module

Syntax:

C: uint16_t SACM_A1801_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A1801_Vxxx.LIB>

Remark: None

6.4 ISR Functions: DMA Interrupt service routine for SACM-A1801

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, A1801_CB_Init, in SACM_A1801_User.c must also be updated as well.

Syntax:

C: void SACM_A1801_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. DMA is able to send the A1801-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A1801_ DmaIsrService to notify SACM service loop function for decoding next frame of data.

2. Inside SACM_A1801_DmaIsrService, it will execute A1801_CB_SendDac_DmaIsr(in SACM_xxx_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

6.5 User Functions: for SACM-A1801 playback

6.5.1 Function: Call back user's function for SACM-A1801 initialization

Syntax:

C: void A1801_CB_Init(const SACM_A1801_WORKING_RAM *A1801WorkingRam);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function of SACM A1801 library for A1801 initialization.

User can determine which channel to be used (Ch0 or Ch1), and which DMA and timer to be used. The default timer is TM0.

6.5.2 Function: Call back user's function when SACM-A1801 starts playing

Syntax:

C: void A1801_CB_StartPlay(const SACM_A1801_WORKING_RAM *A1801WorkingRam);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function of SACM A1801 library when starting play a speech, users can implement their own functions in it.

6.5.3 Function: Call back user's function when SACM-A1801 stops playing

Syntax:

C: void A1801_CB_StopPlay(const SACM_A1801_WORKING_RAM *A1801WorkingRam);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function of SACM A1801 library when playing speech stops. If A1801_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

6.5.4 Function: Call back user's function when pauses a playing SACM-A1801 speech

Syntax:

C: void A1801_CB_Pause(const SACM_A1801_WORKING_RAM *A1801WorkingRam);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function of SACM A1801 library when pause playing a speech, users can implement their own functions in it.

6.5.5 Function: Call back user's function when resumes a paused SACM-A1801 speech

Syntax:

C: void A1801_CB_Resume(const SACM_A1801_WORKING_RAM *A1801WorkingRam);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function of SACM A1801 library when resuming from a paused speech; users can implement their own functions in it.

6.5.6 Function: Gets the A1801 speech data from user's storage and write to buffer

Syntax:

C: void A1801_CB_GetData(const SACM_A1801_WORKING_RAM *A1801WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function for SACM A1801 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one Frame (= 320) of data length.

6.5.7 Function: Call back user's function by A1801 library to decode data

Syntax:

C: void A1801_CB_DecodeProcess(const SACM_A1801_WORKING_RAM *A1801WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_A1801_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

6.5.8 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void A1801_CB_SendDac_DmaIsr(const SACM_A1801_WORKING_RAM *A1801WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A1801WorkingRam: A1801 library working RAM address.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A1801_User.c

Remark:

1. This function is a call-back function for SACM A1801 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
2. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: Play a SACM-A1801 speech.

(a). In main.c:

```
/*---------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------*/
    #include "GPCMFx.h"
    #include "SACM_A1801_User.h"
    #include "APP_SwVolumeControl.h"

/*---------------------------------------------------------------------------
```

```
 * Gobal Variable Declaration Area
 *-------------------------------------------------------------------------------------*/
        #if defined(__CC_ARM)
        __align(4) SACM_A1801_WORKING_RAM A1801WorkingRam;
        __align(4) SACM_A1801_TEMP_RAM A1801TempBuffer;
        __align(4) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
        #elif defined(__GNUC__)
        __attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801WorkingRam;
        __attribute__ ((aligned (4))) SACM_A1801_TEMP_RAM A1801TempBuffer;
        __attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
        #endif

        uint16_t SpeechNum = 0;
        int16_t A1801PlayIdx = 1;
        int8_t PlayCon = 0;
        int8_t SwVolGain = 9;
/*-------------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------------*/
        int main(void)
        {
           SystemInit();

           SPIFC_Open();
           SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
           SPIFC_AutoMode(SPIFC_2IO_MODE);
            if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
               {
                     while(1);    // SPIFC clock timing calibration fail!
               }

            SpeechNum = GetA1801Num();
            SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer);
            APP_SwVolCtrl_Init();                                                  // Software volume control init.
            SACM_A1801_Play(GetA1801StartAddr(A1801PlayIdx), A1801_DAC_CH0 , A1801_AUTO_RAMP_UP
                                 + A1801_AUTO_RAMP_DOWN);
           while(1)
```

```
        {
            WDT_Clear();
            SACM_A1801_ServiceLoop();
            …
            if((PlayCon != 0) && (SACM_A1801_Check_Con() == 0))
            {
                SACM_A1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++), A1801_DAC_CH0,
                                    A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);
                A1801PlayIdx = A1801PlayIdx > SpeechNum ? 1 : A1801PlayIdx;
            }
        }
    } // end of main()
```

7 API for SACM-A1801 CH2

7.1 Hardware Dependent Function: SACM-A1801 CH2 initialization

7.1.1 Function: Initializes the A1801 Ch2 library

Syntax:

C: void SACM_A1801_CH2_Initial(SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam, SACM_A1801_TEMP_RAM *pA1801Ch2TempBuffer, void *A1801Ch2PcmBuffer);

Parameters:

*A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

*pA1801Ch2TempBuffer: A1801 Ch2 temp. buffer address.

*A1801Ch2PcmBuffer: A1801 Ch2 Pcm A/B buffer address.

Return Value: None

Library: < A1801_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A1801 Ch2 decoder. It also initializes the Timer, DAC and enables the DMA IRQ with 16KHz sample rate.
2. The hardware setting is opened for user's reference (see A1801_CH2_CB_Init function in SACM_A1801_CH2_User.c).

Example:

1. First, declare two RAM spaces: SACM_A1801_WORKING_RAM and SACM_A1801_TEMP_RAM in struct type, and named as A1801Ch2WorkingRam and A1801TempBuffer respectively.
2. In the following example, to playback two A1801's Ch1 & Ch2 channels concurrently, working RAM needs to be declared separately, but A1801TempBuffer can be shared:

```
__attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801WorkingRam;
__attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801Ch2WorkingRam;
__attribute__ ((aligned (4))) SACM_A1801_TEMP_RAM A1801TempBuffer;
__attribute__ ((aligned (4))) SACM_A1801_PCM_BUFFER A1801PcmBuffer;
__attribute__ ((aligned (4))) SACM_A1801_PCM_BUFFER A1801Ch2PcmBuffer;
```

3. Pass the addresses of A1801Ch2WorkingRam and A1801TempBuffer struct to SACM_A1801_CH2_Initial().

```
SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer, &A1801PcmBuffer);
```

SACM_A1801_CH2_Initial(&A1801Ch2WorkingRam, &A1801TempBuffer, &A1801Ch2PcmBuffer);

7.2 Service Loop Functions: Service loop for SACM-A1801 CH2 decoding process

7.2.1 Function: Service loop for SACM-A1801 CH2 decoding process

Syntax:

C: void SACM_A1801_CH2_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A1801_CH2_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

Example:

```
/*-------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------
int main(void)
{
  SystemInit();

  while(1)
  {
    WDT_Clear();
    SACM_A1801_ServiceLoop();
    SACM_A1801_CH2_ServiceLoop();
    KeyScan_ServiceLoop();
```

7.3 Playback Functions: Playback control

7.3.1 Function: Plays a SACM-A1801 CH2 speech

Syntax:

C: void SACM_A1801_CH2_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr: Pass A1801PlayIdx and call GetA1801StartAddr () functions to obtain start address of a song.

DacChannelSel: A1801_DAC_CH0

A1801_DAC_CH1

AutoRampUpDownCtrl : A1801_AUTO_RAMP_DISABLE
A1801_AUTO_RAMP_UP
A1801_AUTO_RAMP_DOWN

Return Value: None

Library: < A1801_CH2_Vxxx.LIB >

Remark:

1. The data rate of SACM-A1801 Ch2 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling is 16KHz. The data rate is determined at encoding and selected by decoder automatically.
2. The A1801PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801Ch2PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A1801Num = 0;
int16_t A1801PlayIdx = 1;
int16_t A1801Ch2PlayIdx = 1;
```

3. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA0 and Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A1801_CH2_CB_Init() (in SACM_A1801_CH2_User.c).

4. Ramp-Up/Dn function may be opted by user whether ramp-up (A1801_AUTO_RAMP_UP) is needed when a song starts playing or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801Ch2PlayIdx = 1;
…
SACM_A1801_CH2_Play(GetA1801StartAddr(A1801Ch2PlayIdx), A1801_DAC_CH1,
A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);
```

7.3.2 Function: Plays a continued SACM-A1801 CH2 speech

Syntax:

C: void SACM_A1801_CH2_Play_Con(int16_t *SpeechDataStartAddr, uint8_t DacChannelNo, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass A1801Ch2PlayIdx and call GetA1801StartAddr () function to obtain the start address of a song.
DacChannelSel:	A1801_DAC_CH0
	A1801_DAC_CH1
AutoRampUpDownCtrl :	A1801_AUTO_RAMP_DISABLE
	A1801_AUTO_RAMP_UP
	A1801_AUTO_RAMP_DOWN

Return Value: None

Library: < A1801_CH2_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay time after current speech ends.
2. The data rate of SACM-A1801 CH2 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling is 16KHz. The data rate is determined at encoding and selected by decoder automatically.
3. The A1801Ch2PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.
4. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A1801_CH2_CB_Init() (in SACM_A1801_CH2_User.c).
5. Ramp-Up/Dn function may be opted by user whether ramp-up (A1801_AUTO_RAMP_UP) is needed when a song starts playing or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801Ch2PlayIdx = 1;
…
if((PlayCon != 0) && (SACM_A1801_CH2_Check_Con() == 0))
    {
```

```
        SACM_A1801_CH2_Play_Con (GetA1801StartAddr(A1801Ch2PlayIdx ++),
        A1801_DAC_CH1, A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);

        A1801Ch2PlayIdx = A1801Ch2PlayIdx > A1801Num ? 1 : A1801Ch2PlayIdx;
    }
```

7.3.3 Function: Checks a continued SACM-A1801 CH2 speech

Syntax:

C: uint16_t SACM_A1801_CH2_Check_Con(void);

Parameters: None

Return Value: 0: Continued speech has not set up.

1: Continued speech has set up.

Library: < A1801_CH2_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value will become 0 after continued speech plays.

7.3.4 Function: Stops a playing SACM-A1801 CH2 speech

Syntax:

C: void SACM_A1801_CH2_Stop(void);

Parameters: None

Return Value: None

Library: < A1801_CH2_Vxxx.LIB >

Remark:

1. This function is a call-back function of SACM A1801 Ch2 library when playing speech stops. If A1801_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

7.3.5 Function: Pauses a playing SACM-A1801 CH2 speech

Syntax:

C: void SACM_A1801_CH2_Pause(void);

Parameters: None

Return Value: None

Library: < A1801_CH2_Vxxx.LIB >

Remark: None

7.3.6 Function: Resumes a paused SACM-A1801 CH2 speech

Syntax:

C: void SACM_A1801_CH2_Resume(void);

Parameters: None

Return Value: None

Library: < A1801_CH2_Vxxx.LIB >

Remark: None

7.3.7 Function: Changes the volume of SACM-A1801 CH2

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for A1801 Ch2 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*--------------------------------
 * User Definition Area
 *--------------------------------
#define     VOL_CONTROL_NUM     (4)
```

2. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume. User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as needed.

```
/*---------------------------------------------
 * Table Declaration Area
 *---------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

3. Note.1:APP_SwVolCtrl_VolProcess is located in A1801_CH2_CB_DecodeProcess (SACM_A1801_CH2_User.c) to process volume.

```
void A1801_CH2_CB_DecodeProcess(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam, int16_t *DstBufAddr,
{
  SACM_A1801_CH2_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(1, DstBufAddr, A1801_FRAME_SIZE);
}
```

Example: The following code is an example of both A1801 & A1801 CH2 using software volume control.

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);      // Set A1801 CH1 software volume gain
APP_SwVolCtrl_SetVolGain(1, SwVolGain);      // Set A1801 CH2 software volume gain
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);      // Set A1801 CH1 software volume gain
APP_SwVolCtrl_SetVolGain(1, SwVolGain);      // Set A1801 CH2 software volume gain
break;
```

7.3.8 Function: Gets the status of SACM-A1801 CH2 module

Syntax:

C: uint16_t SACM_A1801_CH2_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A1801_CH2_Vxxx.LIB >

Remark: None

7.4 ISR Functions: DMA Interrupt service routine for SACM-A1801 CH2

This routine will get the decoded data from service loop subroutine and send data out to DAC for playing. The initial function, A1801_CH2_CB_Init in SACM_A1801_CH2_User.c, must also be updated as well.

Syntax:

C: void SACM_A1801_CH2_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A1801_CH2_Vxxx.LIB >

Remark:

1. DMA is able to send the A1801 Ch2-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A1801_CH2_ DmaIsrService to notify SACM service loop function for decoding next frame of data.
2. Inside SACM_A1801CH2__DmaIsrService, it will execute A1801_CH2__CB_SendDac_DmaIsr (in SACM_xxx_CH2_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

7.5 User Functions: for SACM-A1801 Ch2 playback

7.5.1 Function: Call back user's function for SACM-A1801 Ch2 initialization

Syntax:

C: void A1801_CH2_CB_Init(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function of SACM A1801 library for A1801 Ch2 initialization. User can determine which channel to be used (Ch0 or Ch1), and which DMA and timer to be used. The default timer is TM0.

7.5.2 Function: Call back user's function when SACM-A1801 CH2 starts playing

Syntax:

C: void A1801_CH2_CB_StartPlay(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function of SACM A1801 Ch2 library when starting to play a speech. Users may implement their own functions in it.

7.5.3 Function: Call back user's function when SACM-A1801 CH2 stops playing

Syntax:

C: void A1801_CH2_CB_StopPlay(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function of SACM A1801 Ch2 library when playing speech stops. If A1801_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

7.5.4 Function: Call back user's function when pauses a playing SACM-A1801 CH2 speech

Syntax:

C: void A1801_CH2_CB_Pause(const SACM_A1801_WORKING_RAM

*A1801Ch2WorkingRam);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function of SACM A1801 Ch2 library when pause playing a speech, users can implement their own functions in it.

7.5.5 Function: Call back user's function when resumes a paused SACM-A1801 CH2 speech

Syntax:

C: void A1801_CH2_CB_Resume(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function of SACM A1801 Ch2 library when resuming from a paused speech; users can implement their own functions in it.

7.5.6 Function: Gets the A1801 CH2 speech data from user's storage and write to buffer

Syntax:

C: void A1801_CH2_CB_GetData(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function for SACM A1801 Ch2 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the

destination buffer, with a total of one Frame (= 320) of data length.

7.5.7 Function: Call back user's function by A1801 CH2 library to decode data

Syntax:

C: void A1801_CH2_CB_DecodeProcess(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.
*DstBufAddr: Destination buffer address pointer.
*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_A1801_CH2_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

7.5.8 Function: Sends A1801 CH2 speech data to DAC output channel through DMA

Syntax:

C: void A1801_CH2_CB_SendDac_DmaIsr(const SACM_A1801_WORKING_RAM *A1801Ch2WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A1801Ch2WorkingRam: A1801 Ch2 library working RAM address.
*SrcDataAddr: Source buffer address pointer.
DataLen: Length of data moved

Return Value: None

Library: SACM_A1801_CH2_User.c

Remark:

1. This function is a call-back function for SACM A1801 CH2 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
2. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: To playback two A1801's Ch1 & Ch2 speeches concurrently

(a). In main.c:

```
/*---------------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------------*/
    #include "GPCMFx.h"
    #include "SACM_A1801_User.h"
    #include "SACM_A1801_CH2_User.h"
    #include "APP_SwVolumeControl.h"
/*---------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *---------------------------------------------------------------------------------*/
    #if defined(__CC_ARM)
    __align(4) SACM_A1801_WORKING_RAM A1801WorkingRam;
    __align(4) SACM_A1801_WORKING_RAM A1801Ch2WorkingRam;
    __align(4) SACM_A1801_TEMP_RAM A1801TempBuffer;
    __align(4) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
    #elif defined(__GNUC__)
    __attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801WorkingRam;
    __attribute__ ((aligned (4))) SACM_A1801_WORKING_RAM A1801Ch2WorkingRam;
    __attribute__ ((aligned (4))) SACM_A1801_TEMP_RAM A1801TempBuffer;
    __attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
    #endif

    uint16_t SpeechNum = 0;
    int16_t A1801PlayIdx = 1;
    int16_t A1801Ch2PlayIdx = 1;
    int8_t PlayCon = 0;
    int8_t SwVolGain = 9;
/*---------------------------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------------------------*/
    int main(void)
    {
      SystemInit();

      SPIFC_Open();
      SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
```

```
SPIFC_AutoMode(SPIFC_2IO_MODE);
 if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
    {
        while(1);    // SPIFC clock timing calibration fail!
    }

 SpeechNum = GetA1801Num();
 SACM_A1801_Initial(&A1801WorkingRam, &A1801TempBuffer);
 SACM_A1801_CH2_Initial(&A1801Ch2WorkingRam, &A1801TempBuffer);
 APP_SwVolCtrl_Init();                                       // Software volume control init.
 SACM_A1801_Play(GetA1801StartAddr(A1801PlayIdx), A1801_DAC_CH0 , A1801_AUTO_RAMP_UP
                + A1801_AUTO_RAMP_DOWN);
 SACM_A1801_CH2_Play(GetA1801StartAddr(A1801Ch2PlayIdx), A1801_DAC_CH1,
                A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);
while(1)
    {
        WDT_Clear();
        SACM_A1801_ServiceLoop();
        SACM_A1801_CH2_ServiceLoop();
        …
        if((PlayCon != 0) && (SACM_A1801_Check_Con() == 0))
        {
            SACM_A1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++), A1801_DAC_CH0,
                                A1801_AUTO_RAMP_UP + A1801_AUTO_RAMP_DOWN);
              A1801PlayIdx = A1801PlayIdx > SpeechNum ? 1 : A1801PlayIdx;
        }
        if((PlayCon != 0) && (SACM_A1801_CH2_Check_Con () == 0))
        {
            SACM_A1801_CH2_Play_Con (GetA1801StartAddr(A1801Ch2PlayIdx ++),
                                A1801_DAC_CH1, A1801_AUTO_RAMP_UP +
                                A1801_AUTO_RAMP_DOWN);
              A1801Ch2PlayIdx = A1801Ch2PlayIdx > SpeechNum ? 1 : A1801Ch2PlayIdx;
        }

    }
} // end of main()
```

8 API for SACM-DVR1801

8.1 Hardware Dependent Function: SACM-DVR1801 initialization

8.1.1 Function: Initializes the DVR1801 library

Syntax:

C: void SACM_DVR1801_Initial(SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam, SACM_DVR1801_TEMP_RAM *pDVR1801TempBuffer);

Parameters:

*DVR1801WorkingRam: DVR1801 library working RAM address.

*pDVR1801TempBuffer: DVR1801 temp. buffer address.

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. This function initializes the SACM-DVR1801 decoder. It also initializes the Timer 0, DAC and enables the DMA IRQ with 16KHz sample rate.
2. The hardware setting is opened for user's reference (see DVR1801_CB_Init function in SACM_DVR1801_User.c).

Example:

1. First, declare two RAM spaces: SACM_DVR1801_WORKING_RAM and SACM_DVR1801_TEMP_RAM in struct type and named as DVR1801WorkingRam and DVR1801TempBuffer respectively.

```
__attribute__ ((aligned (4))) SACM_DVR1801_WORKING_RAM  DVR1801WorkingRam;
__attribute__ ((aligned (4))) SACM_DVR1801_TEMP_RAM  DVR1801TempBuffer;
```

2. Pass the addresses of DVR1801WorkingRam and DVR1801TempBuffer struct to SACM_DVR1801_Initial() function.

```
SACM_DVR1801_Initial(&DVR1801WorkingRam, &DVR1801TempBuffer);
```

8.2 Service Loop Functions: Service loop for SACM-DVR1801 decoding process

8.2.1 Function: Service loop for SACM-DVR1801 decoding process

Syntax:

C: void SACM_DVR1801_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

Example:

```
/*---------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------
int main(void)
{
  SystemInit();

  while(1)
  {
    WDT_Clear();
    SACM_DVR1801_ServiceLoop();
    KeyScan_ServiceLoop();
```

8.3 Playback Functions: Playback control

8.3.1 Function: Plays a SACM-DVR1801 speech

Syntax:

C: void SACM_DVR1801_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass DVR1801PlayIdx and call GetA1801StartAddr() function to obtain the start address of a song.
DacChannelSel:	DVR1801_DAC_CH0 DVR1801_DAC_CH1
AutoRampUpDownCtrl :	DVR1801_AUTO_RAMP_DISABLE DVR1801_AUTO_RAMP_UP DVR1801_AUTO_RAMP_DOWN

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. The data rate of SACM-DVR1801 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling is 16KHz. The data rate is determined at encoding and selected by decoder automatically.
2. The A1801PlayIdx is the speech sequence. It will obtain a song's start address from

external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A1801Num = 0;
int16_t A1801PlayIdx = 1;
int8_t PlayCon = 0;
```

3. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA, which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA0 and Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in DVR1801_CB_Init() (in SACM_DVR1801_User.c).
4. Ramp-Up/Dn function may be opted by user whether ramp up (A1801_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at DVR1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801PlayIdx = 1;
…
SACM_DVR1801_Play(GetA1801StartAddr(A1801PlayIdx), DVR1801_DAC_CH0 ,
DVR1801_AUTO_RAMP_UP + DVR1801_AUTO_RAMP_DOWN);
```

8.3.2 Function: Plays a continued SACM-DVR1801 speech

Syntax:

C: void SACM_DVR1801_Play_Con (int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass DVR1801PlayIdx and call GetA1801StartAddr() function to obtain the start address of a song.
DacChannelSel:	DVR1801_DAC_CH0 DVR1801_DAC_CH1
AutoRampUpDownCtrl :	DVR1801_AUTO_RAMP_DISABLE DVR1801_AUTO_RAMP_UP DVR1801_AUTO_RAMP_DOWN

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech

immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay time after current speech ends.

2. The data rate of SACM-DVR1801 can be compressed by 7.2/ 9.6/ 12/ 14.4/ 16/ 20/ 24/ 32/ 40/ 44 Kbps when sampling is 16KHz. The data rate is determined at encoding and selected by decoder automatically.
3. The A1801PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A1801_FileMerger.h, generated by G+ File Merger. The default value of the A1801PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.
4.

```
int main(void)
{
  uint32_t ScanedKey;
  uint16_t A1801Num = 0;
  int16_t A1801PlayIdx = 1;
  int16_t A1801Ch2PlayIdx = 1;
  int8_t PlayCon = 0;
```

5. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA, which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA0 and Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in DVR1801_CB_Init() (in SACM_DVR1801_User.c).
6. Ramp-Up/Dn function may be opted by user whether ramp up (A1801_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A1801_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at DVR1801_AUTO_RAMP_DISABLE.

Example:

```
int16_t A1801PlayIdx = 1;
…
if((PlayCon != 0) && (SACM_DVR1801_Check_Con() == 0))
   {
      SACM_DVR1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++),
                  DVR1801_DAC_CH0, DVR1801_AUTO_RAMP_UP +
                  DVR1801_AUTO_RAMP_DOWN);
      A1801PlayIdx = A1801PlayIdx > A1801Num ? 1 : A1801PlayIdx;
    }
```

8.3.3 Function: Starts recording data from MIC to external memory

Syntax:

C: void SACM_DVR1801_Rec(int16_t *EncodeDataStartAddr, uint8_t BitRateIndex);

Parameters: *EncodeDataStartAddr: Encode data start address pointer

BitRateIndex: 0: bit rate = 7.2K bps
1: bit rate = 9.6K bps
2: bit rate = 12K bps
3: bit rate = 14.4K bps
4: bit rate = 16K bps
5: bit rate = 20K bps
6: bit rate = 24K bps
7: bit rate = 32K bps
8: bit rate = 40K bps
9: bit rate = 44K bps

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. *EncodeDataStartAddr pointer points to the start address of external memory for recording data.
2. The DVR1801_CB_StartRecord() will be called to set DMA when SACM_ DVR1800_Rec is called.

8.3.4 Function: Checks a continued SACM-DVR1801 speech

Syntax:

C: int SACM_DVR1801_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up
1: Continued speech has set up

Library: < DVR1801_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value will become 0 after continued speech plays.

8.3.5 Function: Stops a playing SACM-DVR1801 speech

Syntax:

C: void SACM_DVR1801_Stop (void);

Parameters: None

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. This function is a call-back function of SACM DVR1801 library when playing speech stops. If DVR1801_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

8.3.6 Function: Call back user's function when runs recording SACM-DVR1801 speech

Syntax:

C: void DVR1801_CB_StartRecord(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when run recording voice. Users can implement their own functions in it. In addition, it will obtain Mic_IN PCM data from DS_ADC-> DATA through DMA, and store the data in RAM buffer specified by Dvr1801WorkingRam->PcmBufPtr.

8.3.7 Function: Pauses a playing SACM-DVR1801 speech

Syntax:

C: void SACM_DVR1801_Pause (void);

Parameters: None

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark: None

8.3.8 Function: Resumes a paused SACM-DVR1801 speech

Syntax:

C: void SACM_DVR1801_Resume(void);

Parameters: None

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark: None

8.3.9 Function: Changes the volume of SACM-DVR1801

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for DVR1801 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existed channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.
2.

```
/*---------------------------------
 * User Definition Area
 *---------------------------------
#define     VOL_CONTROL_NUM     (4)
```

3. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.

   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c when needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

4. Note.1: APP_SwVolCtrl_VolProcess is located at DVR1801_CB_DecodeProcess (SACM_DVR1801_User.c) for volume control.

```
void DVR1801_CB_DecodeProcess(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstBufAddr,
{
  SACM_DVR1801_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(0, DstBufAddr, DVR1801_FRAME_SIZE);
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;
```

8.3.10 Function: Gets the status of the SACM-DVR1801 module

Syntax:

C: uint16_t SACM_DVR1801_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing
0x0002: Encode mode
0x0004: Pause
0x0008: Auto Ramp Dn Enabled
0x0010: Auto Ramp-Up Enabled
0x0020: DAC Ch0 Enabled
0x0040: DAC Ch1 Enabled
0x0100: Decode Work
0x0200: Decode End
0x0800: Stop

Library: < DVR1801_Vxxx.LIB>

Remark: None

8.4 ISR Functions: DMA Interrupt service routine for SACM-DVR1801

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, DVR1801_CB_Init, in SACM_DVR1801_User.c must also be updated as well.

Syntax:

C: void SACM_DVR1801_DmaIsrService (void);

Parameters: None

Return Value: None

Library: < DVR1801_Vxxx.LIB>

Remark:

1. DMA is able to send the speech data, which has been decoded by DVR1801, to DAC_out in a frame-size unit. When a completed frame-data has been sent, the DMA INT will be triggered and run SACM_DVR1801_ DmaIsrService to notify SACM service loop function to proceed decoding next frame.
2. In SACM_DVR1801_DmaIsrService, it runs DVR1801_CB_SendDac_DmaIsr (in SACM_xxx_User.c). This function is to set DAC output channel to DAC Ch0 or Ch1 and re-enable DMA. User can obtain the decoded speech data here.

8.5 User Functions: for SACM-DVR1801 playback & recording process

8.5.1 Function: Call back user's function for SACM-DVR1801 initialization

Syntax:

C: void DVR1801_CB_Init(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library for DVR1801 initialization. User may determine which DAC channel(Ch0 or Ch1) to use as well as which DMA and Timer(default TM0) to use. In addition, the MIC settings for recording mode such as DSADC initialization and DAGC, etc. can all be configured here.

8.5.2 Function: Call back user's function when SACM-DVR1801 starts playing

Syntax:

C: void DVR1801_CB_StartPlay(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when starting play a speech, users can implement their own functions in it.

8.5.3 Function: Call back user's function when SACM-DVR1801 stops playing

Syntax:

C: void DVR1801_CB_StopPlay(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer.

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when playing speech stops. If DVR1801_AUTO_RAMP_DOWN is enabled, Ramp-Dn will be run and DMA will be turned off. Other functions may be added here too if intending to run them at the end of song.

8.5.4 Function: Call back user's function when pauses a playing SACM-DVR1801 speech

Syntax:

C: void DVR1801_CB_Pause(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer.

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when pause playing a speech, user can implement their own functions in it.

8.5.5 Function: Call back user's function when resumes a paused SACM-DVR1801 speech

Syntax:

C: void DVR1801_CB_Resume(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when resuming from a paused speech, users can implement their own functions in it.

8.5.6 Function: Call back user's function when recording SACM-DVR1801 speech

Syntax:

C: void DVR1801_CB_StartRecord(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when running recording voice. Users can implement their own function in it and through DMA's DS_ADC-> DATA to obtain Mic_IN PCM data and store them into the RAM buffer pointed by Dvr1801WorkingRam->PcmBufPtr.

8.5.7 Function: Call back user's function when SACM-DVR1801 stops recording

Syntax:

C: void DVR1801_CB_StopRecord(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, uint32_t RecDataLen);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

RecDataLen [in]: Record data length (byte).

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function of SACM DVR1801 library when recording stops. Other functions may be added here too if intending to run them at the end of recording.

8.5.8 Function: Gets the DVR1801 speech data from user's storage and write to buffer

Syntax:

C: void DVR1801_CB_GetData(const SACM_DVR1801_WORKING_RAM *DVR1801WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *DVR1801WorkingRam [in]: Dvr1801 working RAM pointer

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function for SACM DVR1801 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one frame (320-byte) of data length.

8.5.9 Function: Call back user's function to write encoded data when recording

Syntax:

C: void DVR1801_CB_WriteData(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstDataAddr, int16_t *SrcBufAddr, uint16_t DataLen);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

*DstDataAddr [out]: Destination data address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

DataLen [in]: Data length (halfword)

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. When recording, this function is called by DVR1801 Library to write encoded data.

8.5.10 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void DVR1801_CB_SendDac_DmaIsr(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. This function is a call-back function for SACM DVR1801 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.

2. DMA move is based on one-frame size (320-byte) each time. When a complete frame data is moved, it will trigger DMA INT.

8.5.11 Function: Gets MIC ADC data and send to destination buffer through DMA

Syntax:

C: void DVR1801_CB_GetAdc_DmaIsr(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstBufAddr, uint16_t DataLen);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

DataLen [in]: Data length (halfword)

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. While recording, this function is called by DVR1801 Library to get the Mic ADC data and send to destination buffer through the DMA.
2. Data length = Frame size = 320.

8.5.12 Function: Call back user's function by DVR1801 library to decode data

Syntax:

C: void DVR1801_CB_DecodeProcess(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_DVR1801_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

8.5.13 Function: Call back user's function by DVR1801 library to encode data

Syntax:

C: void DVR1801_CB_EncodeProcess(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *Dvr1801WorkingRam [in]: Dvr1801 working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

Return Value: None

Library: SACM_DVR1801_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_DVR1801_EncodeProcess () to perform decode process. User will obtain the decoded speech data here.

Example1:

SACM-DVR1801 recording and playback.

(a). In main.c:

```
/*-------------------------------------------------------------------------------
 * Header File Included Area
 *-------------------------------------------------------------------------------*/
    #include "GPCMFx.h"
```

```
#include "SACM_DVR1801_User.h"
#include "APP_SwVolumeControl.h"

/*------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *------------------------------------------------------------------------------*/
#if defined(__CC_ARM)
__align(4) SACM_DVR1801_WORKING_RAM Dvr1801WorkingRam;
__align(4) SACM_DVR1801_TEMP_RAM Dvr1801TempBuffer;
__align(4) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
#elif defined(__GNUC__)
__attribute__ ((aligned (4))) SACM_DVR1801_WORKING_RAM Dvr1801WorkingRam;
__attribute__ ((aligned (4))) SACM_DVR1801_TEMP_RAM Dvr1801TempBuffer;
__attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
#endif

uint32_t ScanedKey;
uint16_t A1801Num = 0;
int16_t A1801PlayIdx = 1;
int8_t PlayCon = 0;
int16_t SwVolGain = 9;

/*------------------------------------------------------------------------------
 * Main Function
 *------------------------------------------------------------------------------*/
int main(void)
{
   SystemInit();

   SPIFC_Open();
   SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
   SPIFC_AutoMode(SPIFC_2IO_MODE);
    if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
      {
            while(1);    // SPIFC clock timing calibration fail!
      }
```

```
    A1801Num = GetA1801Num();
    SACM_DVR1801_Initial(&Dvr1801WorkingRam, &Dvr1801TempBuffer);

    APP_SwVolCtrl_Init();                                   // Software volume control init.
    KeyScan_Initial(&KeyScanWorkingRam);                    // key scan init
     while(1)
       {
            WDT_Clear();
            SACM_DVR1801_ServiceLoop();
            KeyScan_ServiceLoop();

            ScanedKey = KeyScan_GetCh();
            switch(ScanedKey)
            {
              case 0x01:                                            // Record
                  PlayCon = 0;
                  SACM_DVR1801_Stop();
                  SPIFC_BlockErase(0x00200000);
                  SACM_DVR1801_Initial(&Dvr1801WorkingRam, &Dvr1801TempBuffer);
                  SACM_DVR1801_Rec((int16_t *) 0x00200000, DVR1801_RECORD_BITRATE_16000);
                  break;

             case 0x02:                                             // Play recording
                  PlayCon = 0;
                  SACM_DVR1801_Stop();
                  SPIFC_AutoMode(SPIFC_2IO_MODE);
                  SACM_DVR1801_Initial(&Dvr1801WorkingRam, &Dvr1801TempBuffer);
                  SACM_DVR1801_Play((int16_t *)0x04200000, DVR1801_DAC_CH0,
                       DVR1801_AUTO_RAMP_UP + DVR1801_AUTO_RAMP_DOWN);
                  break;
          } // end of switch()
        } // end of while()
    } // end of main()
```

Example2:

Play a continued SACM-DVR1800 speech.

(a). In main.c:

```
/*---------------------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------------------*/
      #include "GPCMFx.h"
      #include "SACM_DVR1801_User.h"
      #include "APP_SwVolumeControl.h"

/*---------------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *---------------------------------------------------------------------------------------*/
      #if defined(__CC_ARM)
      __align(4) SACM_DVR1801_WORKING_RAM Dvr1801WorkingRam;
      __align(4) SACM_DVR1801_TEMP_RAM Dvr1801TempBuffer;
      __align(4) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
      #elif defined(__GNUC__)
      __attribute__ ((aligned (4))) SACM_DVR1801_WORKING_RAM Dvr1801WorkingRam;
      __attribute__ ((aligned (4))) SACM_DVR1801_TEMP_RAM Dvr1801TempBuffer;
      __attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
      #endif

      uint16_t A1801Num = 0;
      int16_t A1801PlayIdx = 1;
      int8_t PlayCon = 1;
      int16_t SwVolGain = 9;

/*---------------------------------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------------------------------*/
      int main(void)
      {
         SystemInit();
         SPIFC_Open();
         SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
         SPIFC_AutoMode(SPIFC_2IO_MODE);
          if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
             {
                  while(1);    // SPIFC clock timing calibration fail!
```

```
        }

    A1801Num = GetA1801Num();
    SACM_DVR1801_Initial(&Dvr1801WorkingRam, &Dvr1801TempBuffer);
    SACM_DVR1801_Play(GetA1801StartAddr(A1801PlayIdx++), DVR1801_DAC_CH0,
                      DVR1801_AUTO_RAMP_UP + DVR1801_AUTO_RAMP_DOWN);
    APP_SwVolCtrl_Init();                                   // Software volume control init.
     while(1)
        {
            WDT_Clear();
            SACM_DVR1801_ServiceLoop();
            …
            if((PlayCon != 0) && (SACM_DVR1801_Check_Con() == 0))
            {
               SACM_DVR1801_Play_Con(GetA1801StartAddr(A1801PlayIdx++), DVR1801_DAC_CH0,
                                     DVR1801_AUTO_RAMP_UP + DVR1801_AUTO_RAMP_DOWN);
              A1801PlayIdx = A1801PlayIdx > A1801Num ? 1 : A1801PlayIdx;
            } // end if
        } // end of while()
} // end of main()
```

9 API for SACM-A2000

9.1 Hardware Dependent Function: SACM-A2000 initialization

9.1.1 Function: Initializes the A2000 library

Syntax:

C: void SACM_A2000_Initial(SACM_A2000_WORKING_RAM *A2000WorkingRam, SACM_A2000_TEMP_RAM *pA2000TempBuffer, void *A2000PcmBuffer);

Parameters:

*A2000WorkingRam: A2000 library working RAM address.

*pA2000TempBuffer: A2000 temp. buffer address.

*A2000PcmBuffer: A2000 Pcm A/B buffer address.

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

3. This function initializes the SACM-A2000 decoder. It also initializes the Timer 0, DAC and enables the DMA IRQ with 32KHz sample rate.
4. The hardware setting is opened for user's reference (see A2000_CB_Init function in SACM_A2000_User.c).

Example:

4. First, declare three RAM spaces: SACM_A2000_WORKING_RAM , SACM_A2000_TEMP_RAM and in struct type and SACM_A2000_PCM_BUFFER named as A2000WorkingRam, A2000TempBuffer and A2000PcmBuffer respectively.

```
__attribute__ ((aligned (4))) SACM_A2000_WORKING_RAM A2000WorkingRam;
__attribute__ ((aligned (4))) SACM_A2000_TEMP_RAM A2000TempBuffer;
__attribute__ ((aligned (4))) SACM_A2000_PCM_BUFFER A2000PcmBuffer;
```

5. Pass the struct addresses of A2000WorkingRam, A2000TempBuffer and A2000PcmBuffer into SACM_A2000_Initial().

```
SACM_A2000_Initial(&A2000WorkingRam, &A2000TempBuffer, &A2000PcmBuffer);
```

9.2 Service Loop Functions: Service loop for SACM-A2000 decoding process

9.2.1 Function: Service loop for SACM-A2000 decoding process

Syntax:

C: void SACM_A2000_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

2. This function has to be placed in main loop.

Example:

```
while(1)
{
  WDT_Clear();
  SACM_A2000_ServiceLoop();
  KeyScan_ServiceLoop();

  if(SACM_A2000_CheckCpuOverload() != 0)
  {
    SACM_A2000_ClearCpuOverload();
  }
```

9.3 Playback Functions: Playback control

9.3.1 Function: Plays a SACM-A2000 speech

Syntax:

C: void SACM_A2000_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

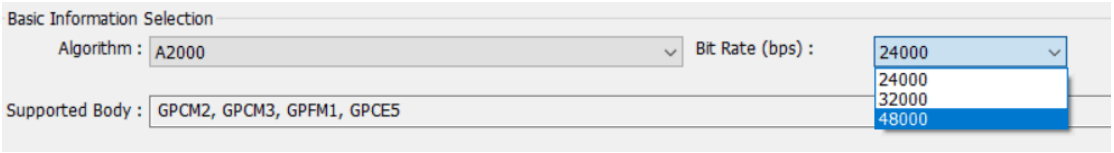
*SpeechDataStartAddr:	Pass into A2000PlayIdx and call GetA2000StartAddr () to obtain a song's start address.
DacChannelSel:	A2000_DAC_CH0 A2000_DAC_CH1
AutoRampUpDownCtrl :	A2000_AUTO_RAMP_DISABLE A2000_AUTO_RAMP_UP A2000_AUTO_RAMP_DOWN

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

5. The data rate of SACM-A2000 can be compressed by 24/ 32/ 48 Kbps when sampling rate is 32KHz. The data rate is determined at encode and selected by decoder automatically.



6. The A2000PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A2000_FileMerger.h, generated by G+ File Merger. The default value of the A2000PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A20ChNum = 0;
int16_t A2000PlayIdx = 1;
int8_t PlayCon = 0;
```

7. User may send out the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM decoded speech data to DAC_Out. Ch0 will use DMA3 and Ch1 will use DMA4 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A2000_CB_Init() (in SACM_A2000_User.c).
8. Ramp-Up/Dn function may be opted by user whether ramp up (A2000_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A2000_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A2000_AUTO_RAMP_DISABLE.

Example:

```
int16_t A2000PlayIdx = 1;
…
SACM_A2000_Play(GetA2000StartAddr(A2000PlayIdx), A2000_DAC_CH0 ,
A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
```

9.3.2 Function: Plays a continued SACM-A2000 speech

Syntax:

C: void SACM_A2000_Play_Con(int16_t *SpeechDataStartAddr, uint8_t DacChannelNo, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr: pass to A2000PlayIdx and call A2000_User_GetA2000StartAddr() to obtain the start address

	of a song
DacChannelSel:	A2000_DAC_CH0
	A2000_DAC_CH1
AutoRampUpDownCtrl :	A2000_AUTO_RAMP_DISABLE
	A2000_AUTO_RAMP_UP
	A2000_AUTO_RAMP_DOWN

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

6. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay after current speech ends.
7. The data rate of SACM-A2000 can be compressed by 24/ 32/ 48 Kbps when sampling rate is 32KHz. The data rate is determined at encode and selected by decoder automatically.
8. The A2000PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A2000_FileMerger.h, generated by G+ File Merger. The default value of the A2000PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A20ChNum = 0;
int16_t A2000PlayIdx = 1;
int8_t PlayCon = 0;
```

9. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 will use DMA3 and Ch1 will use DMA4 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A2000_CB_Init() (in SACM_A2000_User.c).
10. Ramp-Up/Dn function may be opted by user whether ramp-up (A2000_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A2000_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A2000_AUTO_RAMP_DISABLE.

Example:

```
int16_t A2000PlayIdx = 1;
…
if((PlayCon != 0) && (SACM_A2000_Check_Con() == 0))
```

```
{

    SACM_A2000_Play_Con(A2000_User_GetA2000StartAddr(A2000PlayIdx++),
A2000_DAC_CH0, A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
        A2000PlayIdx = A2000PlayIdx > A20ChNum ? 1 : A2000PlayIdx;
}
```

9.3.3 Function: Checks a continued SACM-A2000 speech

Syntax:

C: int SACM_A2000_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up

1: Continued speech has set up

Library: < A2000_Vxxx.LIB>

Remark:

3. If continued speech has set up, the return value is 1.
4. Return value becomes 0 after continued speech plays.

9.3.4 Function: Stops a playing SACM-A2000 speech

Syntax:

C: void SACM_A2000_Stop (void);

Parameters: None

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

2. This function is a call-back function of SACM A2000 library when playing A2000 stops. It will perform Ramp-Dn function (if A2000_AUTO_RAMP_DOWN is enabled) and turn off the corresponding DMA. User's function is allowed to be implemented here too.

9.3.5 Function: Pause SACM-A2000 speech playback

Syntax:

C: void SACM_A2000_Pause (void);

Parameters: None

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark: None

9.3.6 Function: Resumes a paused SACM-A2000 speech

Syntax:

C: void SACM_A2000_Resume(void);

Parameters: None

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark: None

9.3.7 Function: Changes the volume of SACM-A2000

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

4. It is a software volume control mechanism, which controls the output volume for A2000 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*--------------------------------
 * User Definition Area
 *--------------------------------
#define     VOL_CONTROL_NUM     (4)
```

5. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.
   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

6. Note.1: APP_SwVolCtrl_VolProcess is located in A2000_CB_DecodeProcess (SACM_A2000_User.c) to process volume.

```
void A2000_CB_DecodeProcess(const SACM_A2000_WORKING_RAM *A2000WorkingRam,
{
  uint32_t GpEventFlag;

  GpEventFlag = SACM_A2000_DecodeProcess(DstBufAddr, SrcBufAddr);

  APP_SwVolCtrl_VolProcess(0, DstBufAddr, A2000_FRAME_SIZE);

  if(GpEventFlag != 0)
  {
    GP_EVENTOR_AddEvent(EVENTOR_CH0);
  }
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;
```

9.3.8 Function: Gets the status of the SACM-A2000 module

Syntax:

C: uint16_t SACM_A2000_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A2000_Vxxx.LIB>

Remark: None

9.4 ISR Functions: DMA Interrupt service routine for SACM-A2000

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, A2000_CB_Init, in SACM_A2000_User.c must also be updated as well.

Syntax:

C: void SACM_A2000_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A2000_Vxxx.LIB>

Remark:

3. DMA is able to send the A2000-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A2000_ DmaIsrService to notify SACM service loop function for decoding next frame of data.

4. Inside SACM_A2000_DmaIsrService, it will execute A2000_CB_SendDac_DmaIsr(in SACM_xxx_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

9.5 User Functions: for SACM-A2000 playback

9.5.1 Function: Call back user's function for SACM-A2000 initialization

Syntax:

C: void A2000_CB_Init(const SACM_A2000_WORKING_RAM *A2000WorkingRam);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function of SACM A2000 library for A2000 initialization. User can determine which channel to be used (Ch0 or Ch1), and which DMA and

timer to be used. The default timer is TM0.

9.5.2 Function: Call back user's function when SACM-A2000 starts playing

Syntax:

C: void A2000_CB_StartPlay(const SACM_A2000_WORKING_RAM *A2000WorkingRam);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function of SACM A2000 library when starting play a speech, users can implement their own functions in it.

9.5.3 Function: Call back user's function when SACM-A2000 stops playing

Syntax:

C: void A2000_CB_StopPlay(const SACM_A2000_WORKING_RAM *A2000WorkingRam);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function of SACM A2000 library when playing speech stops. If A2000_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

9.5.4 Function: Call back user's function when pauses a playing SACM-A2000 speech

Syntax:

C: void A2000_CB_Pause(const SACM_A2000_WORKING_RAM *A2000WorkingRam);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function of SACM A2000 library when pause playing a speech, users can implement their own functions in it.

9.5.5 Function: Call back user's function when resumes a paused SACM-A2000 speech

Syntax:

C: void A2000_CB_Resume(const SACM_A2000_WORKING_RAM *A2000WorkingRam);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function of SACM A2000 library when resuming from a paused speech; users can implement their own functions in it.

9.5.6 Function: Gets the A2000 speech data from user's storage and write to buffer

Syntax:

C: void A2000_CB_GetData(const SACM_A2000_WORKING_RAM *A2000WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A2000_User.c

Remark:

2. This function is a call-back function for SACM A2000 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one Frame (= 640) of data length.

9.5.7 Function: Call back user's function by A2000 library to decode data

Syntax:

C: void A2000_CB_DecodeProcess(const SACM_A2000_WORKING_RAM *A2000WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A2000WorkingRam: A2000 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A2000_User.c

Remark:

3. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
4. This function will call SACM_A2000_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

9.5.8 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void A2000_CB_SendDac_DmaIsr(const SACM_A2000_WORKING_RAM *A2000WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A2000WorkingRam: A2000 library working RAM address.
*SrcDataAddr: Source buffer address pointer.
DataLen: Length of data moved

Return Value: None

Library: SACM_A2000_User.c

Remark:

3. This function is a call-back function for SACM A2000 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
4. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: Play a SACM-A2000 speech.

(a). In main.c:

```
/*---------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------*/
#include "GPCM3_FM1.h"
#include "SACM_A2000_User.h"
#include "APP_SwVolumeControl.h"
/*---------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *---------------------------------------------------------------------------*/
__attribute__ ((aligned (4))) SACM_A2000_WORKING_RAM A2000WorkingRam;
```

```
__attribute__ ((aligned (4))) SACM_A2000_TEMP_RAM A2000TempBuffer;
__attribute__ ((aligned (4))) SACM_A2000_PCM_BUFFER A2000PcmBuffer;
uint16_t A20ChNum = 0;
int8_t PlayCon = 0;
int16_t A2000PlayIdx = 1;
int8_t A20SwVolGain = 8;                    // for A2000


 /*-------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------*/
      int main(void)
      {
          // A2000 initialize
          A20ChNum = A2000_User_GetA2000Num();
          SACM_A2000_Initial(&A2000WorkingRam, &A2000TempBuffer, &A2000PcmBuffer);

          APP_SwVolCtrl_Init();                          // Software volume control init.
          APP_SwVolCtrl_SetVolGain(0, A20SwVolGain);
          KeyScan_Initial(&KeyScanWorkingRam);           // key scan init.

          SACM_A2000_Play(A2000_User_GetA2000StartAddr(A2000PlayIdx), A2000_DAC_CH0 ,
                                  A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
        while(1)
            {
                  WDT_Clear();
                  SACM_A2000_ServiceLoop();

                  if(SACM_A2000_CheckCpuOverload() != 0)
                  {
                        SACM_A2000_ClearCpuOverload();
                  }
                 …
               if((PlayCon != 0) && (SACM_A2000_Check_Con() == 0))
                  {
                        SACM_A2000_Play_Con(A2000_User_GetA2000StartAddr(A2000PlayIdx++),
A2000_DAC_CH0, A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
```

```
                    A2000PlayIdx = A2000PlayIdx > A20ChNum ? 1 : A2000PlayIdx;
                }
            }
        } // end of main()
```

10 API for SACM-A2000 CH2

10.1 Hardware Dependent Function: SACM-A2000 CH2 initialization

10.1.1 Function: Initializes the A2000 Ch2 library

Syntax:

C: void SACM_A2000_CH2_Initial(SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam, SACM_A2000_TEMP_RAM *pA2000Ch2TempBuffer, void *A2000Ch2PcmBuffer);

Parameters:

*A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

*pA2000Ch2TempBuffer: A2000 Ch2 temp. buffer address.

*A2000Ch2PcmBuffer: A2000 Ch2 Pcm A/B buffer address.

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark:

3. This function initializes the SACM-A2000 Ch2 decoder. It also initializes the Timer, DAC and enables the DMA IRQ with 32KHz sample rate.
4. The hardware setting is opened for user's reference (see A2000_CH2_CB_Init function in SACM_A2000_CH2_User.c).

Example:

3. First, declare two RAM spaces: SACM_A2000_WORKING_RAM and SACM_A2000_TEMP_RAM in struct type, and named as A2000Ch2WorkingRam and A2000TempBuffer respectively.
4. In the following example, to playback two A2000's Ch1 & Ch2 channels concurrently, working RAM needs to be declared separately, but A2000TempBuffer can be shared:

```
__attribute__ ((aligned (4))) SACM_A2000_WORKING_RAM A2000Ch2WorkingRam;
__attribute__ ((aligned (4))) SACM_A2000_PCM_BUFFER A2000Ch2PcmBuffer;
__attribute__ ((aligned (4))) SACM_A2000_TEMP_RAM A2000TempBuffer;
```

6. Pass the addresses of A2000Ch2WorkingRam, A2000TempBuffer and A2000Ch2PcmBuffer struct to SACM_A2000_CH2_Initial().

SACM_A2000_CH2_Initial(&A2000Ch2WorkingRam, &A2000TempBuffer, &A2000Ch2PcmBuffer);

10.2 Service Loop Functions: Service loop for SACM-A2000 CH2 decoding process

10.2.1 Function: Service loop for SACM-A2000 CH2 decoding process

Syntax:

C: void SACM_A2000_CH2_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A2000_CH2_Vxxx.LIB>

Remark:

2. This function has to be placed in main loop.

Example:

```
/*--------------------------------------------------------------
 * Main Function
 *--------------------------------------------------------------
int main(void)
{
  SystemInit();

  while(1)
  {
    WDT_Clear();
    A2000_User_A2000WithGpEventServerLoop();
    A2000_CH2_User_A2000Ch2WithGpEvent_ServerLoop();
    KeyScan_ServiceLoop();
```

10.3 Playback Functions: Playback control

10.3.1 Function: Plays a SACM-A2000 CH2 speech

Syntax:

C: void SACM_A2000_CH2_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass A2000PlayIdx and call GetA2000StartAddr () functions to obtain start address of a song.
DacChannelSel:	A2000_DAC_CH0 A2000_DAC_CH1
AutoRampUpDownCtrl :	A2000_AUTO_RAMP_DISABLE

A2000_AUTO_RAMP_UP

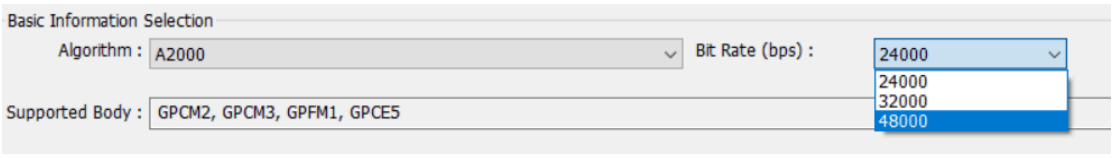
A2000_AUTO_RAMP_DOWN

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark:

5. The data rate of SACM-A2000 can be compressed by 24/ 32/ 48 Kbps when sampling rate is 32KHz. The data rate is determined at encode and selected by decoder automatically.



6. The A2000PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A2000_FileMerger.h, generated by G+ File Merger. The default value of the A2000Ch2PlayIdx = 2, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t A20ChNum = 0;
int8_t PlayCon = 0;
int16_t A2000PlayIdx = 1;
int16_t A2000Ch2PlayIdx = 2;
```

7. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA3 and Ch1 uses DMA4 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A2000_CH2_CB_Init() (in SACM_A2000_CH2_User.c).

8. Ramp-Up/Dn function may be opted by user whether ramp-up (A2000_AUTO_RAMP_UP) is needed when a song starts playing or ramp down (A2000_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A2000_AUTO_RAMP_DISABLE.

Example:

```
int16_t A2000Ch2PlayIdx = 2;
…
SACM_A2000_CH2_Play(A2000_User_GetA2000StartAddr(A2000Ch2PlayIdx),
A2000_DAC_CH1, A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
```

10.3.2 Function: Plays a continued SACM-A2000 CH2 speech

Syntax:

C: void SACM_A2000_CH2_Play_Con(int16_t *SpeechDataStartAddr, uint8_t DacChannelNo, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass A2000Ch2PlayIdx and call GetA2000StartAddr () function to obtain the start address of a song.
DacChannelSel:	A2000_DAC_CH0
	A2000_DAC_CH1
AutoRampUpDownCtrl :	A2000_AUTO_RAMP_DISABLE
	A2000_AUTO_RAMP_UP
	A2000_AUTO_RAMP_DOWN

Return Value: None

Library: < A2000_CH2_Vxxx.LIB>

Remark:

6. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay time after current speech ends.
7. The data rate of SACM-A2000 can be compressed by 24/ 32/ 48 Kbps when sampling rate is 32KHz. The data rate is determined at encode and selected by decoder automatically.
8. The A2000Ch2PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A2000_FileMerger.h, generated by G+ File Merger. The default value of the A2000Ch2PlayIdx = 2, meaning start playback from the first song. User may change the play index value in Main.c.
9. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch1 uses DMA4 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A2000_CH2_CB_Init() (in SACM_A2000_CH2_User.c).
10. Ramp-Up/Dn function may be opted by user whether ramp-up (A2000_AUTO_RAMP_UP) is needed when a song starts playing or ramp down (A2000_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A2000_AUTO_RAMP_DISABLE.

Example:

```
int16_t A2000Ch2PlayIdx = 2;
…
if((PlayCon != 0) && (SACM_A2000_CH2_Check_Con() == 0))
    {
        SACM_A2000_CH2_Play_Con
(A2000_User_GetA2000StartAddr(A2000Ch2PlayIdx ++), A2000_DAC_CH0,
A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);

        A2000Ch2PlayIdx = A2000Ch2PlayIdx > A2000Num ? 1 : A2000Ch2PlayIdx;
     }
```

10.3.3 Function: Checks a continued SACM-A2000 CH2 speech

Syntax:

C: uint16_t SACM_A2000_CH2_Check_Con(void);

Parameters: None

Return Value: 0: Continued speech has not set up.

1: Continued speech has set up.

Library: < A2000_CH2_Vxxx.LIB>

Remark:

3. If continued speech has set up, the return value is 1.
4. Return value will become 0 after continued speech plays.

10.3.4 Function: Stops a playing SACM-A2000 CH2 speech

Syntax:

C: void SACM_A2000_CH2_Stop(void);

Parameters: None

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark:

2. This function is a call-back function of SACM A2000 Ch2 library when playing speech stops. If A2000_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

10.3.5 Function: Pauses a playing SACM-A2000 CH2 speech

Syntax:

C: void SACM_A2000_CH2_Pause(void);

Parameters: None

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark: None

10.3.6 Function: Resumes a paused SACM-A2000 CH2 speech

Syntax:

C: void SACM_A2000_CH2_Resume(void);

Parameters: None

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark: None

10.3.7 Function: Changes the volume of SACM-A2000 CH2

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

4. It is a software volume control mechanism, which controls the output volume for A2000 Ch2 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*--------------------------------
 * User Definition Area
 *--------------------------------
#define     VOL_CONTROL_NUM     (4)
```

5. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.

User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

6. Note.1:APP_SwVolCtrl_VolProcess is located in A2000_CH2_CB_DecodeProcess (SACM_A2000_CH2_User.c) to process volume.

```
void A2000_CH2_CB_DecodeProcess(const SACM_A2000_WORKING_RAM *A2000Ch2
){
  uint32_t GpEventFlag;

  GpEventFlag = SACM_A2000_CH2_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(1, DstBufAddr, A2000_FRAME_SIZE);
```

Example: The following code is an example of both A2000 & A2000 CH2 using software volume control.

```
int8_t A20SwVolGain = 8;                // for A2000
int8_t A20Ch2SwVolGain =8;              // for A2000 Ch2
…
APP_SwVolCtrl_Init();                   // Software volume control init.
APP_SwVolCtrl_SetVolGain(0, A20SwVolGain);
APP_SwVolCtrl_SetVolGain(1, A20Ch2SwVolGain);
```

```
case 0x40:                                          // IOA30+VDD: Volume Up
A20SwVolGain = ++A20SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : A20SwVolGain;
APP_SwVolCtrl_SetVolGain(0, A20SwVolGain);
A20Ch2SwVolGain = ++A20Ch2SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : A20Ch2SwVolGain;
APP_SwVolCtrl_SetVolGain(1, A20Ch2SwVolGain);
break;

case 0x80:                                          // IOA31+VDD: Volume Dn
A20SwVolGain = --A20SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : A20SwVolGain;
        APP_SwVolCtrl_SetVolGain(0, A20SwVolGain);
A20Ch2SwVolGain = --A20Ch2SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : A20Ch2SwVolGain;
```

APP_SwVolCtrl_SetVolGain(1, A20Ch2SwVolGain);

break;

10.3.8 Function: Gets the status of SACM-A2000 CH2 module

Syntax:

C: uint16_t SACM_A2000_CH2_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A2000_CH2_Vxxx.LIB >

Remark: None

10.4 ISR Functions: DMA Interrupt service routine for SACM-A2000 CH2

This routine will get the decoded data from service loop subroutine and send data out to DAC for playing. The initial function, A2000_CH2_CB_Init in SACM_A2000_CH2_User.c, must also be updated as well.

Syntax:

C: void SACM_A2000_CH2_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A2000_CH2_Vxxx.LIB >

Remark:

3. DMA is able to send the A2000 Ch2-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A2000_CH2_ DmaIsrService to notify SACM service loop function for decoding next frame of data.
4. Inside SACM_A2000CH2__DmaIsrService, it will execute A2000_CH2__CB_SendDac_DmaIsr (in SACM_xxx_CH2_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

10.5 User Functions: for SACM-A2000 Ch2 playback

10.5.1 Function: Call back user's function for SACM-A2000 Ch2 initialization

Syntax:

C: void A2000_CH2_CB_Init(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function of SACM A2000 library for A2000 Ch2 initialization. User can determine which channel to be used (Ch0 or Ch1), and which DMA and timer to be used. The default timer is TM1.

10.5.2 Function: Call back user's function when SACM-A2000 CH2 starts playing

Syntax:

C: void A2000_CH2_CB_StartPlay(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function of SACM A2000 Ch2 library when starting to play a speech. Users may implement their own functions in it.

10.5.3 Function: Call back user's function when SACM-A2000 CH2 stops playing

Syntax:

C: void A2000_CH2_CB_StopPlay(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function of SACM A2000 Ch2 library when playing speech stops. If A2000_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and

turn off DMA. Other functions may be added here too if intending to run them at the end of song.

10.5.4 Function: Call back user's function when pauses a playing SACM-A2000 CH2 speech

Syntax:

C: void A2000_CH2_CB_Pause(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function of SACM A2000 Ch2 library when pause playing a speech, users can implement their own functions in it.

10.5.5 Function: Call back user's function when resumes a paused SACM-A2000 CH2 speech

Syntax:

C: void A2000_CH2_CB_Resume(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function of SACM A2000 Ch2 library when resuming from a paused speech; users can implement their own functions in it.

10.5.6 Function: Gets the A2000 CH2 speech data from user's storage and write to buffer

Syntax:

C: void A2000_CH2_CB_GetData(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

2. This function is a call-back function for SACM A2000 Ch2 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one Frame (= 640) of data length.

10.5.7 Function: Call back user's function by A2000 CH2 library to decode data

Syntax:

C: void A2000_CH2_CB_DecodeProcess(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

3. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
4. This function will call SACM_A2000_CH2_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

10.5.8 Function: Sends A2000 CH2 speech data to DAC output channel through DMA

Syntax:

C: void A2000_CH2_CB_SendDac_DmaIsr(const SACM_A2000_WORKING_RAM *A2000Ch2WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A2000Ch2WorkingRam: A2000 Ch2 library working RAM address.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A2000_CH2_User.c

Remark:

3. This function is a call-back function for SACM A2000 CH2 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
4. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: To playback two A2000's Ch1 & Ch2 speeches concurrently

(a). In main.c:

```
/*---------------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------------*/
    #include "GPCM3_FM1.h"
    #include "SACM_A2000_User.h"
    #include "SACM_A2000_CH2_User.h"
    #include "APP_SwVolumeControl.h"
/*---------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *---------------------------------------------------------------------------------*/
#define Aligned4B __attribute__ ((aligned (4)))
__attribute__ ((aligned (4))) SACM_A2000_WORKING_RAM A2000WorkingRam;
__attribute__ ((aligned (4))) SACM_A2000_WORKING_RAM A2000Ch2WorkingRam;
__attribute__ ((aligned (4))) SACM_A2000_TEMP_RAM A2000TempBuffer;
__attribute__ ((aligned (4))) SACM_A2000_PCM_BUFFER A2000PcmBuffer;
__attribute__ ((aligned (4))) SACM_A2000_PCM_BUFFER A2000Ch2PcmBuffer;

__attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
#endif
    uint16_t A20ChNum = 0;
    int8_t PlayCon = 0;
    int16_t A2000PlayIdx = 1;
    int16_t A2000Ch2PlayIdx = 2;
/*---------------------------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------------------------*/
    int main(void)
    {
```

```
// A2000 initialize
    A20ChNum = A2000_User_GetA2000Num();
    SACM_A2000_Initial(&A2000WorkingRam, &A2000TempBuffer, &A2000PcmBuffer);
    SACM_A2000_CH2_Initial(&A2000Ch2WorkingRam, &A2000TempBuffer, &A2000Ch2PcmBuffer);

    APP_SwVolCtrl_Init();
    APP_SwVolCtrl_SetVolGain(0, A20SwVolGain);
    APP_SwVolCtrl_SetVolGain(1, A20Ch2SwVolGain);

    SACM_A2000_Play(A2000_User_GetA2000StartAddr(A2000PlayIdx), A2000_DAC_CH0 ,
    A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
    SACM_A2000_CH2_Play(A2000_User_GetA2000StartAddr(A2000Ch2PlayIdx), A2000_DAC_CH1,
    A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);    while(1)
    {
        WDT_Clear();
        A2000_User_A2000WithGpEventServerLoop();
        A2000_CH2_User_A2000Ch2WithGpEvent_ServerLoop();
        KeyScan_ServiceLoop();

        if(SACM_A2000_CheckCpuOverload() != 0)
        {
            SACM_A2000_ClearCpuOverload();
        }

        if(SACM_A2000_CH2_CheckCpuOverload() != 0)
        {
            SACM_A2000_CH2_ClearCpuOverload();
        }
        …
        if((PlayCon != 0) && (SACM_A2000_Check_Con() == 0))
        {
        SACM_A2000_Play_Con(A2000_User_GetA2000StartAddr(A2000PlayIdx++),
      A2000_DAC_CH0, A2000_AUTO_RAMP_UP + A2000_AUTO_RAMP_DOWN);
        A2000PlayIdx = A2000PlayIdx > A20ChNum ? 1 : A2000PlayIdx;
        }
    }
} // end of main()
```

11 API for SACM-DVRPCM

11.1 Hardware Dependent Function: SACM-DVRPCM initialization

11.1.1 Function: Initializes the DVRPCM library

Syntax:

C: void SACM_DVRPCM_Initial(SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, int16_t *DvrPcmDatabufRam, int16_t lFrameSize);

Parameters:

*DVRPCMWorkingRam: DVRPCM library working RAM address.

*DvrPcmDatabufRam: DVRPCM A/B buffer address

lFrameSize: DVRPCM frame size

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. This function initializes the SACM-DVRPCM decoder. It also initializes Timer 0, DAC and enables the DMA IRQ with sample rate.
2. The hardware setting is opened for user's reference (see DVRPCM_CB_Init function in SACM_DVRPCM_User.c).

Example:

1. First, declare two RAM spaces: SACM_DVRPCM_WORKING_RAM and DvrDataBuffer in struct type and named as DVRPCMWorkingRam and DvrDataBuffer respectively.

```
__attribute__ ((aligned (4))) SACM_DVRPCM_WORKING_RAM DvrPcmWorkingRam;
__attribute__ ((aligned (4))) int16_t DvrDataBuffer[2 * DVRPCM_FRAME_SIZE];
```

2. Pass the addresses of DVRPCMWorkingRam and DvrDataBuffer struct to SACM_DVRPCM_Initial() function.

```
SACM_DVRPCM_Initial(&DvrPcmWorkingRam,&DvrDataBuffer[0], DVRPCM_FRAME_SIZE);
```

11.2 Service Loop Functions: Service loop for SACM-DVRPCM decoding process

11.2.1 Function: Service loop for SACM-DVRPCM decoding process

Syntax:

C: void SACM_DVRPCM_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

Example:

```
/*---------------------------------------------------------------
 * Main Function
 *---------------------------------------------------------------
int main(void)
{
  SystemInit();

  while(1)
  {
    WDT_Clear();
    SACM_DVRPCM_ServiceLoop();
    KeyScan_ServiceLoop();
```

11.3 Playback Functions: Playback control

11.3.1 Function: Plays a SACM-DVRPCM speech

Syntax:

C: void SACM_DVRPCM_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass PcmPlayIdx and call GetPcmStartAddr () function to obtain the start address of a song.
DacChannelSel:	DVRPCM_DAC_CH0 DVRPCM_DAC_CH1
AutoRampUpDownCtrl :	DVRPCM_AUTO_RAMP_DISABLE DVRPCM_AUTO_RAMP_UP DVRPCM_AUTO_RAMP_DOWN

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. The PcmPlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the PcmROM.h, generated by G+ File Merger. The default value of the PcmPlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
uint16_t PcmNum = 0;
int16_t PcmPlayIdx = 1;
int8_t PlayCon = 0;
```

2. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA, which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA0 and Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in DVRPCM_CB_Init() (in SACM_DVRPCM_User.c).
3. Ramp-Up/Dn function may be opted by user whether ramp-up (APCM_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (APCM_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at DVRPCM_AUTO_RAMP_DISABLE.

Example:

```
int16_t PcmPlayIdx = 1;
…
SACM_DVRPCM_Play(GetPcmStartAddr(PcmPlayIdx), DVRPCM_DAC_CH0 ,
DVRPCM_AUTO_RAMP_UP + DVRPCM_AUTO_RAMP_DOWN);
```

11.3.2 Function: Plays a continued SACM-DVRPCM speech

Syntax:

C: void SACM_DVRPCM_Play_Con(int16_t *SpeechDataStartAddr, uint8_t DacChannelNo, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass PCMPlayIdx and call GetAPCMStartAddr() function to obtain the start address of a song.
DacChannelSel:	DVRPCM_DAC_CH0 DVRPCM_DAC_CH1
AutoRampUpDownCtrl :	DVRPCM_AUTO_RAMP_DISABLE DVRPCM_AUTO_RAMP_UP DVRPCM_AUTO_RAMP_DOWN

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay time after current

speech ends.

2. The PcmPlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the PcmROM.h, generated by G+ File Merger. The default value of the PcmPlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.
3. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA, which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 uses DMA0 and Ch1 uses DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in DVRPCM_CB_Init() (in SACM_DVRPCM_User.c).
4. Ramp-Up/Dn function may be opted by user whether ramp up (APCM_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (APCM_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at DVRPCM_AUTO_RAMP_DISABLE.

Example:

```
int16_t APCMPlayIdx = 1;
…
if((PlayCon != 0) && (SACM_DVRPCM_Check_Con() == 0))
{
    SACM_DVRPCM_Play_Con(GetPcmStartAddr(PcmPlayIdx++),
    DVRPCM_DAC_CH0, DVRPCM_AUTO_RAMP_UP +
    DVRPCM_AUTO_RAMP_DOWN);
    PcmPlayIdx = PcmPlayIdx > PcmNum ? 1 : PcmPlayIdx;
}
```

11.3.3 Function: Starts recording data from MIC to external memory

Syntax:

C: void SACM_DVRPCM_Rec(int16_t *EncodeDataStartAddr);

Parameters: *EncodeDataStartAddr: Encode data start address pointer

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. *EncodeDataStartAddr pointer points to the start address of external memory for recording data.
2. The DVRPCM_CB_StartRecord() will be called to set DMA when SACM_ DVRPCM_Rec is called.

11.3.4 Function: Checks a continued SACM-DVRPCM speech

Syntax:

C: int SACM_DVRPCM_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up

1: Continued speech has set up

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value will become 0 after continued speech plays.

11.3.5 Function: Stops a playing SACM-DVRPCM speech

Syntax:

C: void SACM_DVRPCM_Stop (void);

Parameters: None

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. This function is a call-back function of SACM DVRPCM library when playing speech stops. If DVRPCM_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

11.3.6 Function: Call back user's function when runs recording SACM-DVRPCM speech

Syntax:

C: void DVRPCM_CB_StartRecord(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam);

Parameters: *DvrPCMWorkingRam [in]: DvrPCM working RAM pointer

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when running recording voice. Users can implement their own functions in it. In addition, it will obtain Mic_IN PCM data from DS_ADC-> DATA through DMA, and store the data in RAM buffer specified by DvrPCMWorkingRam->PcmBufPtr.

11.3.7 Function: Pauses a playing SACM-DVRPCM speech

Syntax:

C: void SACM_DVRPCM_Pause (void);

Parameters: None

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark: None

11.3.8 Function: Resumes a paused SACM-DVRPCM speech

Syntax:

C: void SACM_DVRPCM_Resume(void);

Parameters: None

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark: None

11.3.9 Function: Gets the status of the SACM-DVRPCM module

Syntax:

C: uint16_t SACM_DVRPCM_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0002: Encode mode

0x0004: Pause

0x0008: Auto Ramp Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < DVRPCM_Vxxx.LIB>

Remark: None

11.4 ISR Functions: DMA Interrupt service routine for SACM-DVRPCM

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, DVRPCM_CB_Init, in SACM_DVRPCM_User.c must also be updated as well.

Syntax:

C: void SACM_DVRPCM_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < DVRPCM_Vxxx.LIB>

Remark:

1. DMA is able to send the speech data, which has been decoded by DVRPCM, to DAC_out in a frame-size unit. When a completed frame-data has been sent, the DMA INT will be triggered and run SACM_DVRPCM_ DmaIsrService to notify SACM service loop function to proceed decoding next frame.
2. In SACM_DVRPCM_DmaIsrService, it runs DVRPCM_CB_SendDac_DmaIsr (in SACM_xxx_User.c). This function is to set DAC output channel to DAC Ch0 or Ch1 and re-enable DMA. User can obtain the decoded speech data here.

11.5 User Functions: for SACM-DVRPCM playback & recording process

11.5.1 Function: Call back user's function for SACM-DVRPCM initialization

Syntax:

C: void DVRPCM_CB_Init(const SACM_DVRPCM_WORKING_RAM *DVRPCMWorkingRam);

Parameters: *DVRPCMWorkingRam [in]: DvrPCM working RAM pointer

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library for DVRPCM initialization. User may determine which DAC channel(Ch0 or Ch1) to use as well as which DMA and Timer(default TM0) to use. In addition, the MIC settings for recording mode such as DSADC initialization and DAGC etc. can all be configured here.

11.5.2 Function: Call back user's function when SACM-DVRPCM starts playing

Syntax:

C: void DVRPCM_CB_StartPlay(const SACM_DVRPCM_WORKING_RAM *DVRPCMWorkingRam);

Parameters: *DVRPCMWorkingRam [in]: DvrPCM working RAM pointer

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when starting play a speech, users can implement their own functions in it.

11.5.3 Function: Call back user's function when SACM-DVRPCM stops playing

Syntax:

C: void DVRPCM_CB_StopPlay(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer.

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when playing speech stops. If DVRPCM_AUTO_RAMP_DOWN is enabled, Ramp-Dn will be run and DMA will be turned off. Other functions may be added here too if intending to run them at the end of song.

11.5.4 Function: Call back user's function when pauses a playing SACM-DVRPCM speech

Syntax:

C: void DVRPCM_CB_Pause(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer.

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when pausing play a speech, user can implement their own functions in it.

11.5.5 Function: Call back user's function when resumes a paused SACM-DVRPCM speech

Syntax:

C: void DVRPCM_CB_Resume(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when resuming from a paused speech; users can implement their own functions in it.

11.5.6 Function: Call back user's function when recording SACM-DVRPCM speech

Syntax:

C: void DVRPCM_CB_StartRecord(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when running recording voice. Users can implement their own function in it and through DMA's DS_ADC-> DATA to obtain Mic_IN PCM data and store them into the RAM buffer pointed by DvrPCMWorkingRam->PcmBufPtr.

11.5.7 Function: Call back user's function when SACM-DVRPCM stops recording

Syntax:

C: void DVRPCM_CB_StopRecord(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, uint32_t RecDataLen);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer
RecDataLen [in]: Record data length (byte).

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function of SACM DVRPCM library when recording stops. Other functions may be added here too if intending to run them at the end of

recording.

11.5.8 Function: Gets the DVRPCM speech data from user's storage and write to buffer

Syntax:

C: void DVRPCM_CB_GetData(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function for SACM DVRPCM library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one frame (32 words) of data length.

11.5.9 Function: Call back user's function to write encoded data when recording

Syntax:

C: uint8_t DVRPCM_CB_WriteData(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, int16_t *DstDataAddr, int16_t *SrcBufAddr, uint16_t DataLen);

Parameters: *DvrPCMWorkingRam [in]: DvrPCM working RAM pointer

*DstDataAddr [out]: Destination data address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

DataLen [in]: Data length (halfword)

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. When recording, this function is called by DVRPCM Library to write encoded data.

11.5.10 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void DVRPCM_CB_SendDac_DmaIsr(const SACM_DVRPCM_WORKING_RAM

*DvrPcmWorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. This function is a call-back function for SACM DVRPCM library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.

2. DMA move is based on one-frame size (320-byte) each time. When a complete frame data is moved, it will trigger DMA INT.

11.5.11 Function: Gets MIC ADC data and send to destination buffer through DMA

Syntax:

C: void DVRPCM_CB_GetAdc_DmaIsr(const SACM_DVRPCM_WORKING_RAM *DvrPCMWorkingRam, int16_t *DstBufAddr, uint16_t DataLen);

Parameters: *DvrPCMWorkingRam [in]: DvrPCM working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

DataLen [in]: Data length (halfword)

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. When recording, this function is called by DVRPCM Library to get the Mic ADC data and send to destination buffer through the DMA.

11.5.12 Function: Call back user's function by DVRPCM library to decode data

Syntax:

C: void DVRPCM_CB_DecodeProcess(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.

11.5.13Function: Call back user's function by DVRPCM library to encode data

Syntax:

C: void DVRPCM_CB_EncodeProcess(const SACM_DVRPCM_WORKING_RAM *DvrPcmWorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: * DvrPcmWorkingRam [in]: DvrPCM working RAM pointer

*DstBufAddr [out]: Destination buffer address pointer.

*SrcBufAddr [in]: Source buffer address pointer.

Return Value: None

Library: SACM_DVRPCM_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.

Example1:

SACM-DVRPCM recording and playback.

(a). In main.c:

```
/*-------------------------------------------------------------------------------
 * Header File Included Area
 *-------------------------------------------------------------------------------*/
    #include "GPCMFx.h"
    #include "SACM_DVRPCM_User.h"
    #include "APP_SwVolumeControl.h"

/*-------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *-------------------------------------------------------------------------------*/
    #if defined(__CC_ARM)
    __align(4) SACM_DVRPCM_WORKING_RAM DvrPcmWorkingRam;
    __align(4) int16_t    DvrDataBuffer[2 * DVRPCM_FRAME_SIZE];
    __align(4) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
    #elif defined(__GNUC__)
```

```
__attribute__ ((aligned (4))) SACM_DVRPCM_WORKING_RAM DvrPcmWorkingRam;
__attribute__ ((aligned (4))) int16_t DvrDataBuffer[2 * DVRPCM_FRAME_SIZE];
__attribute__ ((aligned (4))) KEYSCAN_WORKING_RAM KeyScanWorkingRam;
#endif

   uint32_t ScanedKey;
   uint16_t PcmNum = 0;
   int16_t PcmPlayIdx = 1;
   int8_t PlayCon = 0;
int16_t SwVolGain = 9;

/*-------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------*/
int main(void)
{
   SystemInit();

   SPIFC_Open();
   SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
   SPIFC_AutoMode(SPIFC_2IO_MODE);
    if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
      {
          while(1);    // SPIFC clock timing calibration fail!
      }

    PcmNum = GetPcmNum();
    SACM_DVRPCM_Initial(&DvrPcmWorkingRam, &DvrDataBuffer[0], DVRPCM_FRAME_SIZE);
    APP_SwVolCtrl_Init();                                          // Software volume control init.
    KeyScan_Initial(&KeyScanWorkingRam);                           // key scan init
     while(1)
      {
          WDT_Clear();
          SACM_DVRPCM_ServiceLoop();
          KeyScan_ServiceLoop();

          ScanedKey = KeyScan_GetCh();
```

```
                switch(ScanedKey)
                {
                  case 0x01:                                        // Record
                      PlayCon = 0;
                      SACM_DVRPCM_Stop();
                      SACM_DVRPCM_SetRecLength(0x10000);
                      SPIFC_BlockErase(0x00200000);
                      SPIFC_BlockErase(0x00210000);
                      SACM_DVRPCM_Rec((int16_t *) 0x00200000);
                      break;

                  case 0x02:                                        // Play recording
                      PlayCon = 0;
                      SACM_DVRPCM_Stop();
                      SPIFC_AutoMode(SPIFC_4IO_MODE);
                       SACM_DVRPCM_Play((int16_t *)0x04200000, DVRPCM_DAC_CH0,
                       DVRPCM_AUTO_RAMP_UP + DVRPCM_AUTO_RAMP_DOWN);
                      break;
               } // end of switch()
            } // end of while()
        } // end of main()
```

Example2:

Play a continued SACM-DVRPCM speech.

(a). In main.c:

```
        uint16_t PcmNum = 0;
        int16_t PcmPlayIdx = 1;
        int8_t PlayCon = 1;
        int16_t SwVolGain = 9;
/*-------------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------------*/
        int main(void)
        {
           SystemInit();
           SPIFC_Open();
```

```
SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
SPIFC_AutoMode(SPIFC_2IO_MODE);
 if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
    {
          while(1);    // SPIFC clock timing calibration fail!
    }

 PcmNum = GetPcmNum();
 SACM_DVRPCM_Initial(&DvrPcmWorkingRam, &DvrDataBuffer[0], DVRPCM_FRAME_SIZE);
 APP_SwVolCtrl_Init();                                                  // Software volume control init.

 SACM_DVRPCM_Play(GetPcmStartAddr(PcmPlayIdx), DVRPCM_DAC_CH0 ,
                        DVRPCM_AUTO_RAMP_UP + DVRPCM_AUTO_RAMP_DOWN);
 while(1)
    {
          WDT_Clear();
          SACM_DVRPCM_ServiceLoop();
          …
          if((PlayCon != 0) && (SACM_DVRPCM_Check_Con() == 0))
          {
             SACM_DVRPCM_Play_Con(GetPcmStartAddr(PcmPlayIdx++), DVRPCM_DAC_CH0,
                DVRPCM_AUTO_RAMP_UP + DVRPCM_AUTO_RAMP_DOWN);
              PcmPlayIdx = PcmPlayIdx > PcmNum ? 1 : PcmPlayIdx;
          }       } // end of while()
    } // end of main()
```

12 API for SACM-A3400Pro

12.1 Hardware Dependent Function: SACM-A3400Pro initialization

12.1.1 Function: Initializes the A3400Pro library

Syntax:

C: void SACM_A3400Pro_Initial (SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *A3400ProPcmBuf, int16_t lFrameSize)

Parameters:

*A3400ProWorkingRam: A3400Pro library working RAM address.

*A3400ProPcmBuf: A3400Pro decode data buffer address.

lFrameSize: A3400Pro frame size

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A3400Pro decoder. It also initializes the Timer, DAC and enables the DMA IRQ with 12KHz sample rate.
2. The hardware setting is opened for user's reference (see A3400Pro_CB_Init function in SACM_A3400Pro_User.c).
3. A3400Pro's frame size is 32-byte in default. The frame size can be changed, but it must be a multiple of 32-byte.

Example:

1. First, declare two RAM spaces: SACM_A3400Pro_WORKING_RAM and SACM_A3400Pro_TEMP_RAM in struct type and named as A3400ProWorkingRam and A3400ProBuffer respectively.

```
__align(4) SACM_A3400Pro_WORKING_RAM  A3400ProWorkingRam;
__align(4) int16_t          A3400ProBuffer[2 * A3400PRO_FRAME_SIZE];
```

2. Pass the struct addresses of A3400ProWorkingRam and A3400ProBuffer into SACM_A3400Pro_Initial(), and set up frame size.

```
SACM_A3400Pro_Initial(&A3400ProWorkingRam,&A3400ProBuffer[0],
A3400PRO_FRAME_SIZE);
```

12.1.2 Function: Initialize the A3400Pro Event library

Syntax:

C: void Event_Initial (void)

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A3400Pro S/W event. It also initializes the Timer1 and enables the Timer1 IRQ.
2. It initializes IO pins based on the number of IO Event and IO ports given at C_IO_EVENT_NUM in SACM_A3400PRO_User.asm. It also assigns the Timer1 interrupt frequency according to the C_PWM_FREQ parameter. If C_IO_EVENT_NUM = 0, IO Event is ignored.
3. The hardware setting is opened for user's reference (see F_Event_USER_Init in SACM_A3400Pro_User.asm).

```
#define C_IO_EVENT_NUM     4                                    // IO number for IO_Event
#define C_SW_PWM_FREQ      60                                   // PWM frequency (Unit: Hz)
#define C_SW_PWM_LEVEL     128                                  // PWM level selection (32/64/128/256)
#define C_PWM_FREQ         (C_SW_PWM_FREQ * C_SW_PWM_LEVEL)     // for PWM Service Timer Setting
```

```
const uint8_t T_EVENT_IO[25] =
{
  IOA8,
  IOA9,
  IOA10,
  IOA11,
  IOA12,
  IOA13,
  IOA14,
  IOA15,

  IOA0,
  IOA1,
  IOA2,
  IOA3,
  IOA4,
  IOA5,
  IOA6,
  IOA7,

  IOA16,
  IOA17,
  IOA18,
  IOA19,
  IOA20,
  IOA21,
  IOA22,
  IOA23,

  IOA24
};
```

12.2 Service Loop Functions: Service loop for SACM-A3400Pro decoding process

12.2.1 Function: Service loop for SACM-A3400Pro decoding process

Syntax:

C: void SACM_A3400Pro_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

12.2.2 Function: Service loop for SACM-A3400Pro event

Syntax:

C: void Event_ServiceLoop (void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.
2. Service loop will identify if there is any event and determines type of event (IO or user event) to be executed.

12.3 Playback Functions: Playback control

12.3.1 Function: Plays a SACM-A3400Pro speech

Syntax:

C: void SACM_A3400Pro_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass into A3400ProPlayIdx and call GetA3400ProStartAddr () to obtain a song's start address.
DacChannelSel:	A3400Pro_DAC_CH0 A3400Pro_DAC_CH1
AutoRampUpDownCtrl :	A3400Pro_AUTO_RAMP_DISABLE A3400Pro_AUTO_RAMP_UP

A3400Pro_AUTO_RAMP_DOWN

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. The data rate of SACM-A3400Pro can be compressed by 34Kbps (4-bit) & 42 Kbps (5-bit) when sampling rate is 8KHz. The data rate is determined at encode and selected by decoder automatically.
2. The A3400ProPlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A34_FileMerger, generated by G+ File Merger. The default value of the A3400ProPlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
int main(void)
{
  uint32_t ScanedKey;
  uint16_t A3400ProNum = 0;
  int16_t A3400ProPlayIdx = 1;
  int16_t A3400ProCh2PlayIdx = 1;
```

3. User may send out the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A3400Pro_CB_Init() (in SACM_A3400Pro_User.c).
4. Ramp-Up/Dn function may be opted by user whether ramp-up (A3400Pro_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A3400Pro_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A3400Pro_AUTO_RAMP_DISABLE.

Example:

```
int16_t A3400ProPlayIdx = 1;
…
SACM_A3400Pro_Play(GetA3400ProStartAddr(A3400ProPlayIdx),
A3400Pro_DAC_CH0 , A3400Pro_AUTO_RAMP_UP +
A3400Pro_AUTO_RAMP_DOWN);
```

12.3.2 Function: Plays a continued SACM-A3400Pro speech

Syntax:

C: void SACM_A3400Pro_Play_Con (int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	pass to A3400ProPlayIdx and call GetA3400ProStartAddr() to obtain the start address of a song
DacChannelSel:	A3400Pro_DAC_CH0
	A3400Pro_DAC_CH1
AutoRampUpDownCtrl :	A3400Pro_AUTO_RAMP_DISABLE
	A3400Pro_AUTO_RAMP_UP
	A3400Pro_AUTO_RAMP_DOWN

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay after current speech ends.
2. The data rate of SACM-A3400Pro can be compressed by 34Kbps(4-bit) & 42 Kbps(5-bit) when sampling rate is 8KHz. The data rate is determined at encoding and selected by decoder automatically.
3. The A3400ProPlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A34_FileMerger, generated by G+ File Merger. The default value of the A3400ProPlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.
4. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A3400Pro_CB_Init() (in SACM_A3400Pro_User.c).
5. Ramp-Up/Dn function may be opted by user whether ramp-up (A3400Pro_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A3400Pro_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A3400Pro_AUTO_RAMP_DISABLE.

Example:

```
int16_t A3400ProPlayIdx = 1;
…
if((PlayCon != 0) && (SACM_A3400Pro_Check_Con() == 0))
    {
        SACM_A3400Pro_Play_Con(GetA3400ProStartAddr(A3400ProPlayIdx++),
```

```
                                    A3400Pro_DAC_CH0, A3400Pro_AUTO_RAMP_UP +
    A3400Pro_AUTO_RAMP_DOWN);

        A3400ProPlayIdx = A3400ProPlayIdx > A3400ProNum ? 1 : A3400ProPlayIdx;
    }
```

12.3.3 Function: Check a continued SACM-A3400Pro speech

Syntax:

C: int SACM_A3400Pro_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up

1: Continued speech has set up

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value becomes 0 after continued speech plays.

12.3.4 Function: Stops a playing SACM-A3400Pro speech

Syntax:

C: void SACM_A3400Pro_Stop (void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. This function is a stop function of SACM A3400Pro library when playing A3400Pro stops.

12.3.5 Function: Pauses a playing SACM-A3400Pro speech

Syntax:

C: void SACM_A3400Pro_Pause (void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark: None

12.3.6 Function: Resumes a paused SACM-A3400Pro speech

Syntax:

C: void SACM_A3400Pro_Resume(void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark: None

12.3.7 Function: Changes the volume of SACM-A3400Pro

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for A3400Pro and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*---------------------------------
 * User Definition Area
 *---------------------------------
#define     VOL_CONTROL_NUM     (4)
```

2. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.
   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

3. Note.1: APP_SwVolCtrl_VolProcess is located in A3400Pro_CB_DecodeProcess (SACM_A3400Pro_User.c) to process volume.

```
void A3400Pro_CB_DecodeProcess(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr)
{
  SACM_A3400Pro_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(0, DstBufAddr, A3400PRO_FRAME_SIZE);
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;
```

12.3.8 Function: Gets the status of the SACM-A3400Pro module

Syntax:

C: uint16_t SACM_A3400Pro_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A3400Pro_Vxxx.LIB>

Remark: None

12.4 ISR Functions: DMA Interrupt service routine for SACM-A3400Pro

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, A3400Pro_CB_Init, in SACM_A3400Pro_User.c must also be updated as well.

Syntax:

C: void SACM_A3400Pro_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A3400Pro_Vxxx.LIB>

Remark:

1. DMA is able to send the A3400Pro-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A3400Pro_ DmaIsrService to notify SACM service loop function for decoding next frame of data.

2. Inside SACM_A3400Pro_DmaIsrService, it will execute A3400Pro_CB_SendDac_DmaIsr(in SACM_xxx_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

12.5 User Functions: for SACM-A3400Pro playback

12.5.1 Function: Call back user's function for SACM-A3400Pro initialization

Syntax:

C: void A3400Pro_CB_Init(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro library for A3400Pro initialization. User can determine which channel to be used (Ch0 or Ch1), and which DMA and timer to be used. The default timer is TM0.

12.5.2 Function: Call back user's function when SACM-A3400Pro starts playing

Syntax:

C: void A3400Pro_CB_StartPlay(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro library when starting to play a speech, users can implement their own functions in it.

12.5.3 Function: Call back user's function when SACM-A3400Pro stops playing

Syntax:

C: void A3400Pro_CB_StopPlay(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro library when playing speech stops. If A3400Pro_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if intending to run them at the end of song.

12.5.4 Function: Call back user's function when pauses a playing SACM-A3400Pro speech

Syntax:

C: void A3400Pro_CB_Pause(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro library when pausing play a speech, users can implement their own functions in it.

12.5.5 Function: Call back user's function when resumes a paused SACM-A3400Pro speech

Syntax:

C: void A3400Pro_CB_Resume(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro library when resuming from a paused speech, users can implement their own functions in it.

12.5.6 Function: Call back user's function when IO event starts

Syntax:

C: void F_Event_USER_IoEvtStart(void)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_User.asm

Remark:

1. This function will be called by library when IO event start.
2. The state of IO pins can be set by user.

12.5.7 unction: Call back user's function when IO event ends

Syntax:

C: void F_Event_USER_IoEvtEnd(void)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_User.asm

Remark:

1. This function will be called by library when IO event ends.
2. The state of IO pins can be set by user.

12.5.8 Function: Call back user's function when gets user event

Syntax:

C: void F_Event_USER_EvtProcess(uint16_t EventData)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_User.asm

Remark:

1. When a user event is decoded, F_Event_USER_EvtProcess will be executed.
2. User can process the user event in this function.

12.5.9 Function: Gets the A3400Pro speech data from user's storage and write to buffer

Syntax:

C: void A3400Pro_CB_GetData(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.
*DstBufAddr: Destination buffer address pointer.
*SrcDataAddr: Source buffer address pointer.
DataLen: Length of data moved

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function for SACM A3400Pro library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one Frame (= 32) of data length.

12.5.10 Function: Call back user's function by A3400Pro library to decode data

Syntax:

C: void A3400Pro_CB_DecodeProcess(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.
*DstBufAddr: Destination buffer address pointer.
*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_A3400Pro_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

12.5.11 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void A3400Pro_CB_SendDac_DmaIsr(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A3400ProWorkingRam: A3400Pro library working RAM address.
*SrcDataAddr: Source buffer address pointer.
DataLen: Length of data moved

Return Value: None

Library: SACM_A3400Pro_User.c

Remark:

1. This function is a call-back function for SACM A3400Pro library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
2. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: Play a SACM-A3400Pro speech.

(a). In main.c:

```
/*---------------------------------------------------------------------------------
 * Header File Included Area
 *---------------------------------------------------------------------------------*/
    #include "GPCMFx.h"
    #include "SACM_A3400Pro_User.h"
    #include "APP_SwVolumeControl.h"

/*---------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
```

```
*-------------------------------------------------------------------------------------*/
    #if defined(__CC_ARM)
    __align(4) SACM_A3400Pro_WORKING_RAM   A3400ProWorkingRam;
    __align(4) int16_t A3400ProBuffer[2 * A3400PRO_FRAME_SIZE];
    #elif defined(__GNUC__)
    __attribute__ ((aligned (4))) SACM_A3400Pro_WORKING_RAM     A3400ProWorkingRam;
    __attribute__ ((aligned (4))) int16_t A3400ProBuffer[2 * A3400PRO_FRAME_SIZE];
    #endif

      uint16_t A3400ProNum = 0;
      int16_t A3400ProPlayIdx = 1;
      int8_t PlayCon = 1;
      int8_t PauseFlag = 0;
      int16_t AudPwmGain = 31;
      int8_t SwVolGain = 9;

/*-------------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------------*/
    int main(void)
    {
      SystemInit();

      SPIFC_Open();
      SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
      SPIFC_AutoMode(SPIFC_2IO_MODE);
       if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
        {
            while(1);    // SPIFC clock timing calibration fail!
        }

        A3400ProNum = GetA3400ProNum();
        SACM_A3400Pro_Initial(&A3400ProWorkingRam, &A3400ProBuffer[0], A3400PRO_FRAME_SIZE);
        Event_Initial();
        APP_SwVolCtrl_Init();                                       // Software volume control init.
        SACM_A3400Pro_Play(GetA3400ProStartAddr(A3400ProPlayIdx), A3400Pro_DAC_CH0 ,
              A3400Pro_AUTO_RAMP_UP + A3400Pro_AUTO_RAMP_DOWN);
```

```
    while(1)
        {
            WDT_Clear();
            SACM_A3400Pro_ServiceLoop();
            Event_ServiceLoop();
            …
            if((PlayCon != 0) && (SACM_A3400Pro_Check_Con() == 0))
            {
                SACM_A3400Pro_Play_Con(GetA3400ProStartAddr(A3400ProPlayIdx++),
                A3400Pro_DAC_CH0, A3400Pro_AUTO_RAMP_UP +
                A3400Pro_AUTO_RAMP_DOWN);
               A3400ProPlayIdx = A3400ProPlayIdx > A3400ProNum ? 1 : A3400ProPlayIdx;
            }
        }
} // end of main()
```

13 API for SACM-A3400Pro CH2

13.1 Hardware Dependent Function: SACM-A3400Pro Ch2 initialization

13.1.1 Function: Initializes the A3400Pro Ch2 library

Syntax:

C: void SACM_A3400Pro_CH2_Initial (SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam, int16_t *A3400ProCh2PcmBuf, int16_t lFrameSize)

Parameters:

*A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

*A3400ProCh2PcmBuf: A3400Pro Ch2 decode data buffer address.

lFrameSize: A3400Pro Ch2 frame size

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

4. This function initializes the SACM-A3400Pro Ch2 decoder. It also initializes the Timer, DAC and enables the DMA IRQ with 12KHz sample rate.
5. The hardware setting is opened for user's reference (see A3400Pro_CH2_CB_Init function in SACM_A3400Pro_CH2_User.c).
6. A3400Pro Ch2's frame size is 32-byte in default. The frame size can be changed, but it must be a multiple of 32-byte.

Example:

1. First, declare two RAM spaces: SACM_A3400Pro_WORKING_RAM and SACM_A3400Pro_CH2_TEMP_RAM in struct type and named as A3400ProCh2WorkingRam and A3400ProCh2Buffer respectively.

```
__align(4) SACM_A3400Pro_WORKING_RAM  A3400ProCh2WorkingRam;
__align(4) int16_t          A3400ProCh2Buffer[2  *  A3400PRO_CH2_FRAME_SIZE];
```

2. Pass the struct addresses of A3400ProCh2WorkingRam and A3400ProBuffer into SACM_A3400Pro_CH2_Initial(), and set up the frame size.

```
SACM_A3400Pro_CH2_Initial(&A3400ProCh2WorkingRam,&A3400ProBuffer[0],
A3400PRO_CH2_FRAME_SIZE);
```

13.1.2 Function: Initializes the A3400Pro Ch2 Event library

Syntax:

C: void Event_CH2_Initial (void)

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. This function initializes the SACM-A3400Pro Ch2 S/W event. It also initializes the Timer2 and enables the Timer2 IRQ.
2. It initializes IO pins based on the number of IO Event and IO ports given at C_IO_EVENT_NUM in SACM_A3400PRO_CH2_User.asm. It also assigns the Timer1 interrupt frequency according to the C_PWM_FREQ parameter. If C_IO_EVENT_NUM = 0, IO Event is ignored.
3. The hardware setting is opened for user's reference (see F_Event_CH2_USER_Init in SACM_A3400Pro_CH2_User.asm).

```
#define C_IO_EVENT_NUM     4                                    // IO number for IO_Event
#define C_SW_PWM_FREQ      60                                   // PWM frequency (Unit: Hz)
#define C_SW_PWM_LEVEL     128                                  // PWM level selection (32/64/128/256)
#define C_PWM_FREQ         (C_SW_PWM_FREQ * C_SW_PWM_LEVEL)     // for PWM Service Timer Setting
```

```
const uint8_t T_EVENT_CH2_IO[25] =
{
  IOA16,
  IOA17,
  IOA18,
  IOA19,
  IOA20,
  IOA21,
  IOA22,
  IOA23,

  IOA0,
  IOA1,
  IOA2,
  IOA3,
  IOA4,
  IOA5,
  IOA6,
  IOA7,

  IOA8,
  IOA9,
  IOA10,
  IOA11,
  IOA12,
  IOA13,
  IOA14,
  IOA15,

  IOA24
};
```

13.2 Service Loop Functions: Service loop for SACM-A3400Pro Ch2 decoding process

13.2.1 Function: Service loop for SACM-A3400Pro Ch2 decoding process

Syntax:

C: void SACM_A3400Pro_CH2_ServiceLoop(void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.

13.2.2 Function: Service loop for SACM-A3400Pro Ch2 event

Syntax:

C: void Event_CH2_ServiceLoop (void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. This function has to be placed in main loop.
2. Service loop will identify if there is any event and determines type of event (IO or user event) to be executed.

13.3 Playback Functions: Playback control

13.3.1 Function: Plays a SACM-A3400Pro Ch2speech

Syntax:

C: void SACM_A3400Pro_CH2_Play(int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	Pass into A3400ProCh2PlayIdx and call GetA3400ProStartAddr () to obtain a song's start address.
DacChannelSel:	A3400Pro_DAC_CH0 A3400Pro_DAC_CH1
AutoRampUpDownCtrl :	A3400Pro_AUTO_RAMP_DISABLE A3400Pro_AUTO_RAMP_UP

A3400Pro_AUTO_RAMP_DOWN

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. The data rate of SACM-A3400Pro Ch2 can be compressed by 34Kbps(4-bit) & 42 Kbps(5-bit) when sampling rate is 8KHz. The data rate is determined at encode and selected by decoder automatically.
2. The A3400ProCh2PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A34_FileMerger, generated by G+ File Merger. The default value of the A3400ProCh2PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.

```
int main(void)
{
  uint32_t ScanedKey;
  uint16_t A3400ProNum = 0;
  int16_t A3400ProPlayIdx = 1;
  int16_t A3400ProCh2PlayIdx = 1;
```

3. User may send out the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A3400Pro_CH2_CB_Init() (in SACM_A3400Pro_CH2_User.c).
4. Ramp-Up/Dn function may be opted by user whether ramp-up (A3400Pro_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A3400Pro_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A3400Pro_AUTO_RAMP_DISABLE.

Example:

```
int16_t A3400ProCh2PlayIdx = 1;
…
SACM_A3400Pro_CH2_Initial(&A3400ProCh2WorkingRam, &A3400ProCh2Buffer[0],
A3400PRO_CH2_FRAME_SIZE);
```

13.3.2 Function: Plays a continued SACM-A3400Pro Ch2 speech

Syntax:

C: void SACM_A3400Pro_CH2_Play_Con (int16_t *SpeechDataStartAddr, uint8_t DacChannelSel, uint8_t AutoRampUpDownCtrl);

Parameters:

*SpeechDataStartAddr:	pass to A3400ProCh2PlayIdx and call GetA3400ProStartAddr() to obtain the start address of a song
DacChannelSel:	A3400Pro_DAC_CH0
	A3400Pro_DAC_CH1
AutoRampUpDownCtrl :	A3400Pro_AUTO_RAMP_DISABLE
	A3400Pro_AUTO_RAMP_UP
	A3400Pro_AUTO_RAMP_DOWN

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. If there is a speech playing, this function doesn't start playing the continued speech immediately, but set up the environment parameters, else it will play the continued speech directly. The continued speech will play without any delay after current speech ends.
2. The data rate of SACM-A3400Pro Ch2 can be compressed by 34Kbps (4-bit) & 42 Kbps (5-bit) when sampling rate is 8KHz. The data rate is determined at encoding and selected by decoder automatically.
3. The A3400ProCh2PlayIdx is the speech sequence. It will obtain a song's start address from external SPI FLASH memory by the A34_FileMerger, generated by G+ File Merger. The default value of the A3400ProCh2PlayIdx = 1, meaning start playback from the first song. User may change the play index value in Main.c.
4. User may send the decoded speech data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded speech data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 0 is the counter for sampling rate in default. User may change the default value in A3400Pro_CH2_CB_Init() (in SACM_A3400Pro_CH2_User.c).
5. Ramp-Up/Dn function may be opted by user whether ramp up (A3400Pro_AUTO_RAMP_UP) is needed when a song is started to play or ramp down (A3400Pro_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at A3400Pro_AUTO_RAMP_DISABLE.

Example:

```
int16_t A3400ProCh2PlayIdx = 1;
…
if((PlayCon != 0) && (SACM_A3400Pro_CH2_Check_Con() == 0))
    {
```

```
SACM_A3400Pro_CH2_Play_Con(GetA3400ProStartAddr(A3400ProCh2PlayIdx++),
                    A3400Pro_DAC_CH0, A3400Pro_AUTO_RAMP_UP +
A3400Pro_AUTO_RAMP_DOWN);

    A3400ProCh2PlayIdx = A3400ProCh2PlayIdx > A3400ProNum ? 1 :
A3400ProCh2PlayIdx;
  }
```

13.3.3 Function: Checks a continued speech of SACM-A3400Pro

Syntax:

C: int SACM_A3400Pro_CH2_Check_Con(void)

Parameters: None

Return Value: 0: Continued speech has not set up

1: Continued speech has set up

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. If continued speech has set up, the return value is 1.
2. Return value becomes 0 after continued speech plays.

13.3.4 Function: Stops a playing SACM-A3400Pro Ch2speech

Syntax:

C: void SACM_A3400Pro_CH2_Stop (void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. This function is a stop function of SACM A3400Pro Ch2 library when playing A3400Pro Ch2 stops.

13.3.5 Function: Pauses a playing SACM-A3400Pro Ch2 speech

Syntax:

C: void SACM_A3400Pro_CH2_Pause (void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark: None

13.3.6 Function: Resumes a paused SACM-A3400Pro Ch2 speech

Syntax:

C: void SACM_A3400Pro_CH2_Resume(void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark: None

13.3.7 Function: Changes the volume of SACM-A3400Pro Ch2

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for A3400Pro Ch2and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existing channel is not enough, user can adjust the number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

```
/*--------------------------------
 * User Definition Area
 *--------------------------------
#define     VOL_CONTROL_NUM     (4)
```

2. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.

   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c as needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

3. Note.1: APP_SwVolCtrl_VolProcess is located in A3400Pro_CH2_CB_DecodeProcess (SACM_A3400Pro_CH2_User.c) to process volume.

```
void A3400Pro_CH2_CB_DecodeProcess(const SACM_A3400Pro_WORKING_RAM *A3400ProWorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr)
{
  SACM_A3400Pro_CH2_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(1, DstBufAddr, A3400PRO_CH2_FRAME_SIZE);
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(1, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(1, SwVolGain);
break;
```

13.3.8 Function: Gets the status of the SACM-A3400Pro Ch2 module

Syntax:

C: uint16_t SACM_A3400Pro_CH2_GetStatus(void);

Parameters: None

Return Value: 0x0001: Speech Playing

0x0004: Pause

0x0008: Auto Ramp-Dn Enabled

0x0010: Auto Ramp-Up Enabled

0x0020: DAC Ch0 Enabled

0x0040: DAC Ch1 Enabled

0x0100: Decode Work

0x0200: Decode End

0x0800: Stop

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark: None

13.4 ISR Functions: DMA Interrupt service routine for SACM-A3400Pro Ch2

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, A3400Pro_CH2_CB_Init, in SACM_A3400Pro_CH2_User.c must also be updated as well.

Syntax:

C: void SACM_A3400Pro_CH2_DmaIsrService(void);

Parameters: None

Return Value: None

Library: < A3400Pro_CH2_Vxxx.LIB>

Remark:

1. DMA is able to send the A3400Pro-decoded speech data to DAC_out. When a frame size of data has been sent completely, the DMA INT will be triggered and execute SACM_A3400Pro_CH2_ DmaIsrService to notify SACM service loop function for decoding next frame of data.

2. Inside SACM_A3400Pro_CH2_DmaIsrService, it will execute A3400Pro_CH2_CB_SendDac_DmaIsr (in SACM_xxx_User.c) that determines which channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

13.5 User Functions: for SACM-A3400Pro Ch2 playback

13.5.1 Function: Call back user's function for SACM-A3400Pro Ch2 initialization

Syntax:

C: void A3400Pro_CH2_CB_Init(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro Ch2 library for A3400Pro Ch2 initialization. User can determine which channel to be used (Ch0 or Ch1), and which DMA and timer to be used. The default timer is TM0.

13.5.2 Function: Call back user's function when SACM-A3400Pro Ch2 starts playing

Syntax:

C: void A3400Pro_CH2_CB_StartPlay(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro Ch2 library when starting play a speech, users can implement their own functions in it.

13.5.3 Function: Call back user's function when SACM-A3400Pro Ch2 stops playing

Syntax:

C: void A3400Pro_CH2_CB_StopPlay(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro Ch2 library when playing speech stops. If A3400Pro_AUTO_RAMP_DOWN is enabled, it will run Ramp-Dn function and turn off DMA. Other functions may be added here too if user intends to run them at the end of song.

13.5.4 Function: Call back user's function when pauses a playing SACM-A3400Pro Ch2speech

Syntax:

C: void A3400Pro_CH2_CB_Pause(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro Ch2 library when pausing play a speech, users can implement their own functions in it.

13.5.5 Function: Call back user's function when resumes a paused SACM-A3400Pro Ch2 speech

Syntax:

C: void A3400Pro_CH2_CB_Resume(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function of SACM A3400Pro Ch2 library when resuming from a paused speech; users can implement their own functions in it.

13.5.6 Function: Call back user's function when Ch2 IO event starts

Syntax:

C: void F_Event_CH2_USER_IoEvtStart(void)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_CH2_User.asm

Remark:

1. This function will be called by library when IO event start.
2. The state of IO pins can be set by user.

13.5.7 unction: Call back user's function when Ch2 IO event ends

Syntax:

C: void F_Event_CH2_USER_IoEvtEnd(void)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_CH2_User.asm

Remark:

1. This function will be called by library when IO event ends.
2. The state of IO pins can be set by user.

13.5.8 Function: Call back user's function when gets Ch2 user event

Syntax:

C: void F_Event_CH2_USER_EvtProcess(uint16_t EventData)

Parameters: None

Return Value: None

Library: SACM_A3400Pro_CH2_User.asm

Remark:

1. When a user event is decoded, F_Event_CH2_USER_EvtProcess will be executed.
2. User can process the user event in this function.

13.5.9 Function: Gets the A3400Pro Ch2 speech data from user's storage and write to buffer

Syntax:

C: void A3400Pro_CH2_CB_GetData(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function for SACM A3400Pro Ch2 library to read speech data from memory storage. When it is determined a song needs to be decoded, this function will be called to move from the user-specified source data area to the destination buffer, with a total of one Frame (= 32) of data length.

13.5.10 Function: Call back user's function by A3400Pro Ch2 library to decode data

Syntax:

C: void A3400Pro_CH2_CB_DecodeProcess(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam, int16_t *DstBufAddr, int16_t *SrcBufAddr);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.
*DstBufAddr: Destination buffer address pointer.
*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_A3400Pro_CH2_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, APP_SwVolCtrl_VolProcess() will be placed here for volume control process.

13.5.11 Function: Sends speech data to DAC output channel through DMA

Syntax:

C: void A3400Pro_CH2_CB_SendDac_DmaIsr(const SACM_A3400Pro_WORKING_RAM *A3400ProCh2WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *A3400ProCh2WorkingRam: A3400Pro Ch2 library working RAM address.
*SrcDataAddr: Source buffer address pointer.
DataLen: Length of data moved

Return Value: None

Library: SACM_A3400Pro_CH2_User.c

Remark:

1. This function is a call-back function for SACM A3400Pro Ch2 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.
2. DMA move is based on one-frame size each time. When a complete frame data is moved, it will trigger DMA INT.

Example: Play a SACM-A3400Pro Ch2 speech.

(a). In main.c:

```
/*---------------------------------------------------------------------------------------
 * Header File Included Area
```

```
 *-------------------------------------------------------------------------------------*/
        #include "GPCMFx.h"
        #include "SACM_A3400Pro_CH2_User.h"
        #include "APP_SwVolumeControl.h"

/*--------------------------------------------------------------------------------------
 * Gobal Variable Declaration Area
 *-------------------------------------------------------------------------------------*/
      #if defined(__CC_ARM)
      __align(4) SACM_A3400Pro_WORKING_RAM   A3400ProCh2WorkingRam;
      __align(4) int16_t    A3400ProCh2Buffer[2 * A3400PRO_CH2_FRAME_SIZE];
      #elif defined(__GNUC__)
      __attribute__ ((aligned (4))) SACM_A3400Pro_WORKING_RAM     A3400ProCh2WorkingRam;
      __attribute__ ((aligned (4))) int16_t A3400ProCh2Buffer[2 * A3400PRO_CH2_FRAME_SIZE];
      #endif

         uint16_t A3400ProNum = 0;
         int16_t A3400ProCh2PlayIdx = 1;
         int8_t PlayCon = 1;
         int8_t PauseFlag = 0;
         int16_t AudPwmGain = 31;
         int8_t SwVolGain = 9;

/*--------------------------------------------------------------------------------------
 * Main Function
 *-------------------------------------------------------------------------------------*/
        int main(void)
        {
           SystemInit();

           SPIFC_Open();
           SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV2);
           SPIFC_AutoMode(SPIFC_2IO_MODE);
            if(SPIFC_TimingFineTune() == SPIFC_CALIBRATION_FAIL)        // Calibrate SPIFC clock timing.
               {
                     while(1);    // SPIFC clock timing calibration fail!
               }
```

```
        A3400ProNum = GetA3400ProNum();
        SACM_A3400Pro_CH2_Initial(&A3400ProCh2WorkingRam, &A3400ProBuffer[0],
A3400PRO_CH2_FRAME_SIZE);
        Event_Initial();
        APP_SwVolCtrl_Init();                                   // Software volume control init.
        SACM_A3400Pro_CH2_Play(GetA3400ProStartAddr(A3400ProCh2PlayIdx), A3400Pro_DAC_CH0 ,
            A3400Pro_AUTO_RAMP_UP + A3400Pro_AUTO_RAMP_DOWN);
    while(1)
        {
            WDT_Clear();
            SACM_A3400Pro_CH2_ServiceLoop();
            Event_ServiceLoop();
            …
            if((PlayCon != 0) && (SACM_A3400Pro_CH2_Check_Con() == 0))
            {
                SACM_A3400Pro_CH2_Play_Con(GetA3400ProStartAddr(A3400ProCh2PlayIdx++),
                A3400Pro_DAC_CH0, A3400Pro_AUTO_RAMP_UP +
                A3400Pro_AUTO_RAMP_DOWN);
                A3400ProCh2PlayIdx = A3400ProCh2PlayIdx > A3400ProNum ? 1 :
A3400ProCh2PlayIdx;
            }
        }
    } // end of main()
```

14 API for SACM-MS02

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

14.1 Dynamic-allocated Polyphonic Channels

MS02 can support up to 32 polyphonic channels, each of which can deliver an intact note envelope without being intervened by another note in most cases by taking advantage of dynamic allocation. For concurrent algorithm applications, the use of up to 32 channels and playback rate 16KHz max. MS02 along with other speech/audio algorithm is recommended.

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

14.2 Tone Color Library

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

14.3 Playback Rate

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

to appendix.

1. Playback rate is configured in SACM_MS02_Initial(uint32_t Timer_SR) and SACM_MS02_Play (DAC_OUT_SR). Note that the timer's sampling rate is set in SACM_MS02_ Initial() and in SACM_MS02_Play(), it informs MS02 decode library's DAC_Out the sampling rate. These two parameters' settings must be consistent.
2. In the following example, to implement playback rate = 20KHz, assign Timer_SR = MS02_Timer_20K in SACM_MS02_ Initial() and DAC_OUT_SR = MS02_DAC_20K in SACM_MS02_Play()

```
SACM_MS02_Initial(&MS02WorkingRam, AUD_OUT_PWM, MS02_CH1, MS02_Timer_20K);
SACM_MS02_Play(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx), MS02_AUTO_RAMP_UP + MS02_AUTO_RAMP_DOWN, MS02_DAC_20K);
```

14.4 Sampling Rate

There is no limitation to SACM-MS02 tone color sampling rate. The only concern is the CPU loading. We would advise coding engineers that the sampling rate should be bounded in between 8KHz to 48KHz to ensure the availability of 16 polyphonic channels. For 32 polyphonic channels, we recommend tone color sampling rate does not exceed 24KHz.

14.5 CPU Loading

The CPU loading for SACM-MS02 is subject to change at any time. It may vary by factors, channel numbers, song attribute (tempo, score and etc.) and DAC playback rate. If channel number is higher, CPU loading is also heavier. A faster tempo and higher DAC playback rate will result in a heavier CPU loading. Please refer to appendix for CPU loading information. For SACM-MS02, the CPU loading can be measured by taking the service loop time and ISR time into account. The MIDI composing techniques and hardware settings can help to lower the CPU loading and to grant user applications more flexibility.

SACM-MS02 features

- Up to 32 dynamic channels (Note: The number of maximum MIDI channel (Default: 16 CH) can be determined by users, given in MS02dec.h. During MIDI playback, if number of channels currently in use is greater than the user-assigned maximum MIDI number, the most toward to the end playing MIDI channel will be replaced by the new instrument.

MS02_16ch.ld | SACM_MS02_User.c | MS02dec.h | KeyScan.c

```
/*--------------------------------------------------
 * Constant Definition Area
 *--------------------------------------------------
#define MS02_CH_Num      16
#define MS02_DEC_Memory 308 + (MS02_CH_Num * 68)
```

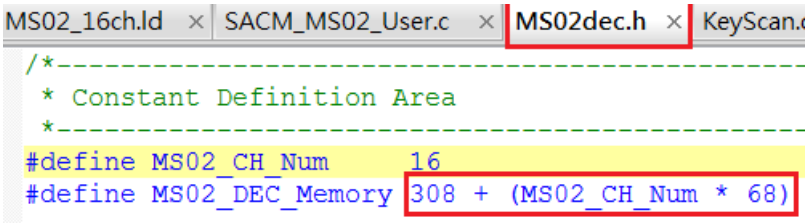
- Playback rate (DAC rate) options: 8KHz, 10KHz, 12KHz, 16KHz, 20KHz, 24KHz, 28KHz, 32KHz, 36KHz, and 40KHz (Note: A higher number of MIDI channel will result in heavier CPU

loadings, and the supported playback rate will be decreased. For example, if 32-channel is used, the maximum supported playback rate can only be 16KHz.)

- Tempo / Key adjustable at playback
- Instrument / Tone color run-time change
- To control MIDI channel ON/ OFF separately.
- Allowing multiple drums to be configured at the same channel 10, distinguishing different drums through different tracks, and setting the ON/OFF for individual drum tracks.
- MIDI compatible

SACM-MS02 Technical basics

- Variable sampling rate for tone color from 8KHz to 48KHz in 16 polyphonic channels.
- ROM requirement: About 6KB
- RAM requirement: About 3.2KB for 16-Channel & 4.4KB for 32-Channel (Note: When number of MIDI channel changes, the RAM size required by MIDI kernel library (MS02_DEC_Memory) also changes.)



14.6 Hardware Dependent Function: Initializes SACM-MS02

14.6.1 Function: Initialize SACM-MS02 library

Syntax:

C: void SACM_MS02_Initial(SACM_MS02_WORKING_RAM*, uint8_t *MS02KernelRAMPtr, uint8_t AudOutType, uint8_t DacChannelNo, uint32_t Timer_SR);

Parameters: SACM_MS02_WORKING_RAM*: MS02 working RAM pointer

*MS02KernelRAMPtr: MS02 kernel RAM pointer.

AudOutType: 0: AUD_OUT_DAC

1: AUD_OUT_PWM

DacChannelNo: 1: MS02_CH0

2: MS02_CH1

Timer_SR: Set the sampling rate of timer.

MS02_Timer_8K

MS02_Timer_10K

MS02_Timer_12K

MS02_Timer_16K

MS02_Timer_20K

MS02_Timer_24K

MS02_Timer_28K

MS02_Timer_32K

MS02_Timer_36K

MS02_Timer_40K

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. This function initializes the decoder of MS02. It also initializes the DMA, Audio output (DAC or PWM) and enables the Timer at the sample rate in user's setting.
2. This function will call MS02_CB_Init() (in SACM_MS02_User.c) for hardware setting
3. The settings in Timer_SR must be the same SampleRate parameter as in SACM_MS02_Play().

14.7 Service Loop Functions: Service loop for SACM-MS02 decoding process

14.7.1 Function: Service loop for SACM-MS02 decoding process

Syntax:

C: void SACM_MS02_ServiceLoop(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. In this function, the MIDI events are parsed and processed for sequencer.
2. This function has to be placed in main loop.

14.8 Playback Functions: Playback control

14.8.1 Function: Plays a SACM-MS02 melody

Syntax:

C: void SACM_MS02_Play(int16_t *MS02_LibPtr, int16_t *SpeechDataStartAddr, uint8_t AutoRampUpDownCtrl, int32_t SampleRate);

Parameters:

*MS02_LibPtr: Pass MS02LibIdx and call GetMS02LibStartAddr () function to

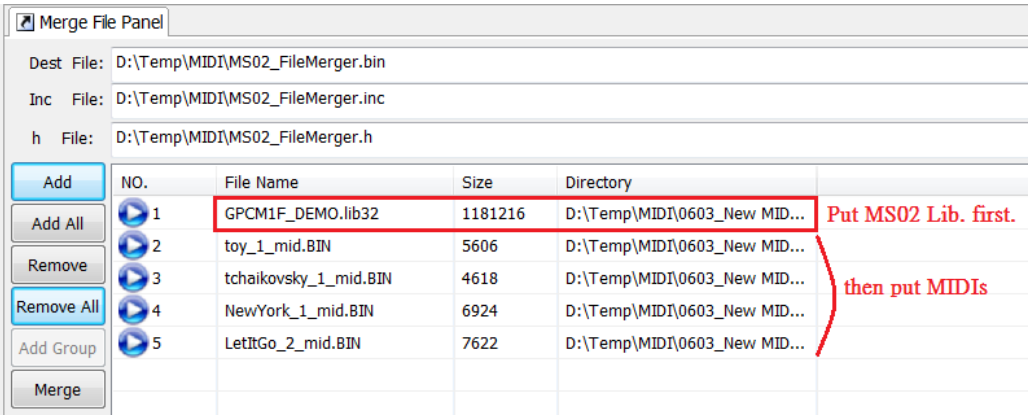
	obtain the start address of MS02 tone library.
*SpeechDataStartAddr:	Pass MidiIdx and call GetMidiStartAddr() function to obtain the start address of MIDI.
AutoRampUpDownCtrl:	0: MS02_AUTO_RAMP_DISABLE 1: MS02_AUTO_RAMP_UP 2: MS02_AUTO_RAMP_DOWN
SampleRate:	For playback 0: MS02_DAC_8K 1: MS02_DAC_10K 2: MS02_DAC_12K 3: MS02_DAC_16K 4: MS02_DAC_20K 5: MS02_DAC_24K 6: MS02_DAC_28K 4: MS02_DAC_32K 5: MS02_DAC_36K 6: MS02_DAC_40K

Return Value: None

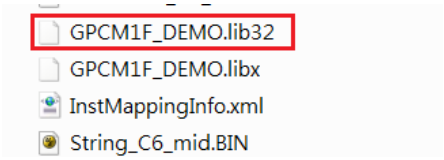
Library: <MS02_Vxxx.LIB>

Remark:

1. The MS02LibIdx and MidiIdx are the MS02 tone library Index and MIDI Index respectively. It will obtain the start address of tone and MIDI from external SPI FLASH memory by the MS02_FileMerger.h, generated by G+ File Merger. The default value of the Default MS02LibIdx = 1 and MidiIdx = 2. Thus, the order in G+File Merger, xxx.lib32 should be located front and followed by xxx_mid.bin MIDI



2. G+Midiar will generate two different MIDI tone library: xxx.lib32 and xxx.libx. GPCMF series uses xxx.lib32.



3. This function calls MS02_CB_StartPlay() for hardware settings. Users are allowed to change these default values. User may send the decoded midi data to DAC's Ch0 or Ch1. When enabling DAC's Ch0 or Ch1, it automatically enables DMA which is responsible for sending the SACM's decoded midi data to DAC_Out. Ch0 will use DMA0 and Ch1 will use DMA1 in default; Timer 1 is the counter for sampling rate in default.

4. The SampleRate parameter must be consistent with Timer_SR parameter in SACM_MS02_Initial().

5. Ramp-Up/Dn function may be opted by user whether ramp-up (MS02_AUTO_RAMP_UP) is needed when a song is started playing or ramp down (MS02_AUTO_RAMP_DOWN) when a song is ended. If both are not needed, set them disabled at MS02_AUTO_RAMP_DISABLE.

14.8.2 Function: Plays a continued SACM-MS02 melody

Syntax:

C: void SACM_MS02_Play_Con(int16_t *MS02_LibPtr, int16_t *SpeechDataStartAddr, uint8_t AutoRampUpDownCtrl, int32_t SampleRate);

Parameters:

*MS02_LibPtr:	Pass MS02LibIdx and call GetMS02LibStartAddr () function to obtain the start address of MS02 tone library.
*SpeechDataStartAddr:	Pass MidiIdx and call GetMidiStartAddr() function to obtain the start address of MIDI.
AutoRampUpDownCtrl:	0: MS02_AUTO_RAMP_DISABLE 1: MS02_AUTO_RAMP_UP 2: MS02_AUTO_RAMP_DOWN
SampleRate:	For playback 0: MS02_DAC_8K 1: MS02_DAC_10K 2: MS02_DAC_12K 3: MS02_DAC_16K

4: MS02_DAC_20K

5: MS02_DAC_24K

6: MS02_DAC_28K

4: MS02_DAC_32K

5: MS02_DAC_36K

6: MS02_DAC_40K

Return Value: None

Library: < MS02_Vxxx.LIB>

Remark:

1. If there is a melody playing, this function doesn't start playing the continued melody immediately, but set up the environment parameters, else it will play the continued melody directly. The continued melody will play without any delay after current speech ends.

14.8.3 Function: Checks a continued SACM-MS02 melody

Syntax:

C: uint16_t SACM_MS02_Check_Con(void);

Parameters: None

Return Value: 0: Continued melody has not set up

1: Continued melody has set up

Library: <MS02_Vxxx.LIB>

Remark:

1. If continued melody has set up, the return value is the 1.
2. Return value will become 0 after continued melody plays.

Example:

```
if((PlayCon != 0) && (SACM_MS02_Check_Con() == 0))
{
  MidiIdx = ++MidiIdx > MidiNum ? 2 : MidiIdx;
   SACM_MS02_Play_Con(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx),
   MS02_AUTO_RAMP_UP + MS02_AUTO_RAMP_DOWN, MS02_DAC_20K);
}
```

14.8.4 Function: Stops a playing SACM-MS02 melody

Syntax:

C: void SACM_MS02_Stop(void)

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark: When SACM-MS02 melody is stopped playing, it will run MS02_CB_StopPlay(), at SACM_MS02_User.c, and turn off DMA.

14.8.5 Function: Pauses a playing SACM-MS02 melody

Syntax:

C: void SACM_MS02_Pause(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark: None

14.8.6 Function: Resumes a paused SACM-MS02 melody

Syntax:

C: void SACM_MS02_Resume(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

14.8.7 Function: Changes the volume of SACM-MS02

Syntax:

C: void APP_SwVolCtrl_SetVolGain(int32_t VolCtrlChannel, int32_t VolGain);

Parameters: VolCtrlChannel : 0 ~ 3(Max. default value = 3, defined by user)

VolGain: 0 ~ 15

Return Value: None

Library: < APP_SwVolumeControl.c>

Remark:

1. It is a software volume control mechanism, which controls the output volume for DVR1801 and other speech algorithms. Via settings of VolCtrlChannel's parameter, it can specify which volume control channel to be used for each speech algorithm; the default channel number is 4. If the existed channel is not enough, user can adjust the

number of supported channel at VOL_CONTROL_NUM, located in APP_SwVolumeControl.h.

2.

```
/*---------------------------------
 * User Definition Area
 *---------------------------------
#define     VOL_CONTROL_NUM     (4)
```

3. VolGain: Volume Gain Index, where 0: Silence and 15: the maximum volume.

   User may adjust the 16-level volume gain table from APP_SwVolumeControl.c when needed.

```
/*-------------------------------------------
 * Table Declaration Area
 *-------------------------------------------
static uint16_t const SW_Volume_Gain_Table[] =
{
  0x0000, 0x0250, 0x0500, 0x1000,
  0x1500, 0x2000, 0x2500, 0x3000,
  0x3500, 0x4000, 0x5000, 0x6500,
  0x7D00, 0x9C00, 0xC400, 0xF500
};
```

4. Note.1: APP_SwVolCtrl_VolProcess is located at DVR1801_CB_DecodeProcess (SACM_DVR1801_User.c) for volume control.

```
void DVR1801_CB_DecodeProcess(const SACM_DVR1801_WORKING_RAM *Dvr1801WorkingRam, int16_t *DstBufAddr,
{
  SACM_DVR1801_DecodeProcess(DstBufAddr, SrcBufAddr);
  APP_SwVolCtrl_VolProcess(0, DstBufAddr, DVR1801_FRAME_SIZE);
}
```

Example:

```
int8_t SwVolGain = 9;
…
APP_SwVolCtrl_Init();          // Software volume control init.
…
case 0x20:                     // Volume Up
SwVolGain = ++SwVolGain >= MAX_VOL_GAIN ? MAX_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;

case 0x40:                     // Volume Dn
SwVolGain = --SwVolGain <= MIN_VOL_GAIN ? MIN_VOL_GAIN : SwVolGain;
APP_SwVolCtrl_SetVolGain(0, SwVolGain);
break;
```

14.8.8 Function: Gets the status of the SACM-MS02 module

Syntax:

C: uint16_t SACM_MS02_GetStatus(void);

Parameters: None

Return Value: bit 0: 0x0001: MS02 playing
0x0004: MS02 paused
0x0008: MS02 auto ramp down enabled
0x0010: MS02 auto ramp up enabled
0x0800: MS02 stop
0x1000: MS02 CPU overloading flag.
Others: Reserved

Library: <MS02_Vxxx.LIB>

Remark: None

14.8.9 Function: Enables one channel for SACM-MS02 melody playing process

Syntax:

C: void SACM_MS02_ChannelOn(int32_t MidiChan);

Parameters:

MidiChan: 0 ~ 15.

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

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

14.8.10 Function: Disables one channel for SACM-MS02 melody playing process

Syntax:

C: void SACM_MS02_ChannelOff(int32_t MidiChan);

Parameters:

MidiChan: 0 ~ 15.

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark: None

14.8.11 Function: Holds one channel for SACM-MS02 melody playing process

Syntax:

C: void SACM_MS02_HoldChannel(int32_t MidiChan);

Parameters:

MidiChan: 0 ~ 15.

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

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

14.8.12 Function: Releases one channel for SACM-MS02 melody playing process

Syntax:

C: void SACM_MS02_ReleaseChannel(int32_t MidiChan);

Parameters:

MidiChan: 0 ~ 15.

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

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

14.8.13 Function: Sets all channels status before SACM-MS02 Initialization

Syntax:

C: void SACM_MS02_SetChannelStatus(short ChannelState);

Parameters:

ChannelState: 0x0000 ~ 0xFFFF. Each bit controls one channel, where 1: ON/ 0:OFF

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. Allowing to configure all 16 MIDI Channel ON/OFF status at once. Each bit in ChannelState parameters controls one single channel independently, e.g. 0x0011(0000 0000 0001 0001) representing Ch0 and Ch4 ON.
2. Co-working with SACM_MS02_ChannelKeep_Enable(). To maintain each channel's ON/OFF status at the beginning of MIDI playback, make sure to call SACM_MS02_ ChannelKeep_Enable() and SACM_MS02_SetChannelStatus() functions before SACM_MS02_Initial().

Example:

1. Before initialization, run SetChannelStatus() and ChannelKeep_Enable() to maintain each channel On or OFF at the beginning of MIDI playback even repeated playback will not reset the status.

```
MidiNum = GetMidiNum();
SACM_MS02_SetChannelStatus(0x0002); // Ch1 ON
```

14.8.14 Function: Changes the instrument on one of the SACM-MS02 channels

Syntax:

C: void SACM_MS02_ChangeInstru(int32_t MidiChan , int32_t Instrument);

Parameters:

MidiChan: 0 ~ 15.

Instrument: 0 ~ max of Instrument

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. The instrument argument is the mapped instrument index. It is determined when the MIDI and tone color are exported from G+Midiar.

14.8.15 Function: Reset to default instruments

Syntax:

C: void SACM_MS02_ResetChanControl();

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. When a channel's instrument index has been changed via SACM_MS02_ChangeInstru(), this function can be used to resume it to the original setting.

14.8.16 Function: Key Change for SACM-MS02 melody playback process

Syntax:

C: void SACM_MS02_KeyShift(int32_t keyshift);

Parameters:

keyshift: number of keys to shift

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. keyshift: Each "+1" means raising a semitone and each "-1" means dropping a semitone. There are twelve semitones in an octave so that "+12" means raising an octave and "-12" means dropping an octave. For passing parameters, although there is almost no limit on the range (32-bit data), there is a maximum pitch limit in G+Midiar. When the passing parameter value >= Max. pitch, it will be played at the maximum pitch. (Note: It is recommended not to exceed two octaves (+/-24), if the key up/down is too large, it may sound weird.)

Example:

1. "+12" means raising an octave

```
SACM_MS02_KeyShift(12);
```

2. "-6" means dropping an 1/2 octave.

```
SACM_MS02_KeyShift(-6);
```

14.8.17 Function: Set Key Shift Status

Syntax:

C: void SACM_MS02_SetKeyChStatus(int16_t KeyChStatus);

Parameters:

KeyChStatus:0x0000 ~ 0xFFFF. Each bit controls one channel, where 1: ON(Keep)/ 0:OFF(Release)

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. When Channel control bit of KeyChStatus = 1, it means channel's key shift function is determined by SACM_MS02_SetKeyChStatus(). When the respective Control bit =0, it means the channel will not have key-shift. It will return to the initial setting.
   Ex. SACM_MS02_SetKeyChStatus(0xFFFF); // all channels' keyshift status will be retained.
2. SACM_MS02_SetKeyChStatus() must be called after SACM_MS02_Initial().

14.8.18 Function: Tempo Change for SACM-MS02 melody playback process

Syntax:

C: void SACM_MS02_ChangeTempo(int32_t TempoFactor);

Parameters:

TempoFactor: TempoFactor Index and its factor is listed in the table below:

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

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. When TempoFactor Index is higher, Tempo is faster. If a song's original tempo is 100bpm and when TempFactor is "10", the new tempo becomes 192bpm (100*1.92).

14.8.19 Function: Reset to Default Tempo for SACM-MS02 melody playback process

Syntax:

C: void SACM_MS02_ResetTempo();

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark: Associated with SACM_MS02_ChangeTempo function to resume the tempo to its original settings.

14.8.20 Function: Plays a SACM-MS02 melody note

Syntax:

C: void SACM_MS02_PlayNote(int16_t MidiChan, int16_t pitch, int16_t velocity, int16_t duration);

Parameters: MidiChan: 0 ~ 15

Pitch: 0 ~ 127

Velocity: 0 ~ 127

duration: 0 ~ 65535 tickers, where a ticker typically 5ms long.

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. Velocity is a significant parameter related with the volume (0～127). A larger number of velocity represents a higher volume. For instance, the sound of a piano will generate different volume intensity based on how hard a key is pressed.

14.8.21 Function: OKON(One Key One Note) function enabled

Syntax:

C: void SACM_MS02_OKON_Enable(int32_t MidiChan,int32_t SustainType);

Parameters: MidiChan: 0 ~ 15.

SustainType: 0: Normal duration/ 1: Long duration

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. Through MidiChan parameter to determine which channel for OKON channel.
2. Associated with SACM_MS02_PlayOKON(), make sure to run SACM_MS02_OKON_Enable() function before enabling Play OKON function.

14.8.22 Function: OKON(One Key One Note) function disabled

Syntax:

C: void SACM_MS02_OKON_Disable();

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. After SACM_MS02_OKON_Enable() is executed, this function can be used to resume the status from OKON.

14.8.23 Function: Plays one note when triggered once

Syntax:

C: void SACM_MS02_PlayOKON();

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. Each trigger plays a note. Before adopting SACM_MS02_ PlayOKON() fuction, run SACM_MS02_OKON_Enable() to configure which channel is the OKON channel.

14.8.24 Function: Sets the drum status after SACM-MS02 Initialization

Syntax:

C: void SACM_MS02_SetDrumStatus(short TrackState);

Parameters: TrackState: 0x0000 ~ 0xFFFF. Each bit controls one track, where 1: ON/ 0:OFF

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. It allows to configure multiple drum tracks (Multi-Track mechanism) and using this function to configure ON/OFF for each drum track ON/OFF.
2. TrackState is a bit control. Each bit controls drum track 1~16, for example:
   0x0001: turn ON drum track 0
   0x0002: turn ON drum track 1
   0x0003: turn ON drum track 0 and track 1
   ……..
   0xFFFF: turn ON all of drum tracks
3. Run SACM_MS02_SetDrumStatus () after SACM_MS02_Initial() to maintain each drum track ON/OFF states at beginning of MIDI playback.

Example:

```
MidiNum = GetMidiNum();
SACM_MS02_SetDrumStatus(0x000F); // Drum 0 ~ 3 ON.
```

14.8.25 Function: Set the infinite loop play

Syntax:

C: void SACM_MS02_InfiniteLoop_Enable(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. The infinite loop function will be activated if the last data of the envelope, given at G+Midiar, is an non-zero value.
2. This function shall be enabled before executing SACM_MS02_Play() function.

```
SACM_MS02_InfiniteLoop_Enable();
SACM_MS02_Play(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx)
```

3. This function shall work with SACM_MS02_InfiniteLoop_Disable () function.

14.8.26 Function: Disable the infinite loop play

Syntax:

C: void SACM_MS02_InfiniteLoop_Disable(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. This function shall be enabled before executing SACM_MS02_Play() function.

```
SACM_MS02_InfiniteLoop_Disable();
SACM_MS02_Play(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx),
```

2. This function shall work with SACM_MS02_InfiniteLoop_Enable () function.

14.9 CPU Overloading Check Function

Since MS02 decode program is processed in the main loop where also has other programs being processed, MS02 decode procedure may take longer time to wait for being executed when more programs are added into the main loop. When the MS02 decode procedure is not processed for more than the longest allowable time of MS02 A/B buffer (the longest allowable time is 1/16K * 320 = 20ms at buffer size = 320 words and sampling rate = 16KHz), the melody playback may be interrupted or loss of note. Note that calling SACM_MS02_CheckCpuOverload() and returning 1 means CPU overloaded.

14.9.1 Function: Check whether CPU overloading occurs

Syntax:

C: uint8_t SACM_MS02_CheckCpuOverload(void);

Parameters: None

Return Value: 1: CPU overloading. 0:Working fine

Library: <MS02_Vxxx.LIB>

Remark: None

14.9.2 Function: Clear MS02 CPU overloading flag

Syntax:

C: void SACM_MS02_ClearCpuOverload(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark: When calling SACM_MS02_CheckCpuOverload() and returning 1, it means CPU is overloading. In this case, we need to call SACM_MS02_ClearCpuOverload() to clear the overloading flag.

Example:

if(SACM_MS02_CheckCpuOverload() != 0)

```
{
    SACM_MS02_ClearCpuOverload();
}
```

14.10 ISR Functions: DMA Interrupt service routine for SACM-MS02

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, MS02_CB_Init, in SACM_MS02_User.c must also be updated as well.

Syntax:

C: void SACM_MS02_DmaIsrService(void);

Parameters: None

Return Value: None

Library: <MS02_Vxxx.LIB>

Remark:

1. DMA is able to send the MS02 lib-decoded MIDI data to DAC_out. When a frame size (320-byte) of data has been sent completely, the DMA INT will be triggered and execute SACM_MS02_DmaIsrService to notify MS02 service loop function to proceed decoding next frame of data.

3. In SACM_MS02_DmaIsrService, it will execute MS02_CB_SendDac_DmaIsr (in SACM_MS02_User.c) that determines which DAC channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded speech data here.

14.11 User Functions: for SACM-MS02 playback process

14.11.1 Function: Song event notification

Syntax:

C: void MS02_CB_SongEvent(short R_CallBack_Event, short R_CMD_Code);

Parameters: R_CallBackEvent: Bit[15:0] = 0x9000, only Event No. = 0x9000 is treated as user's event in order to enter this function.

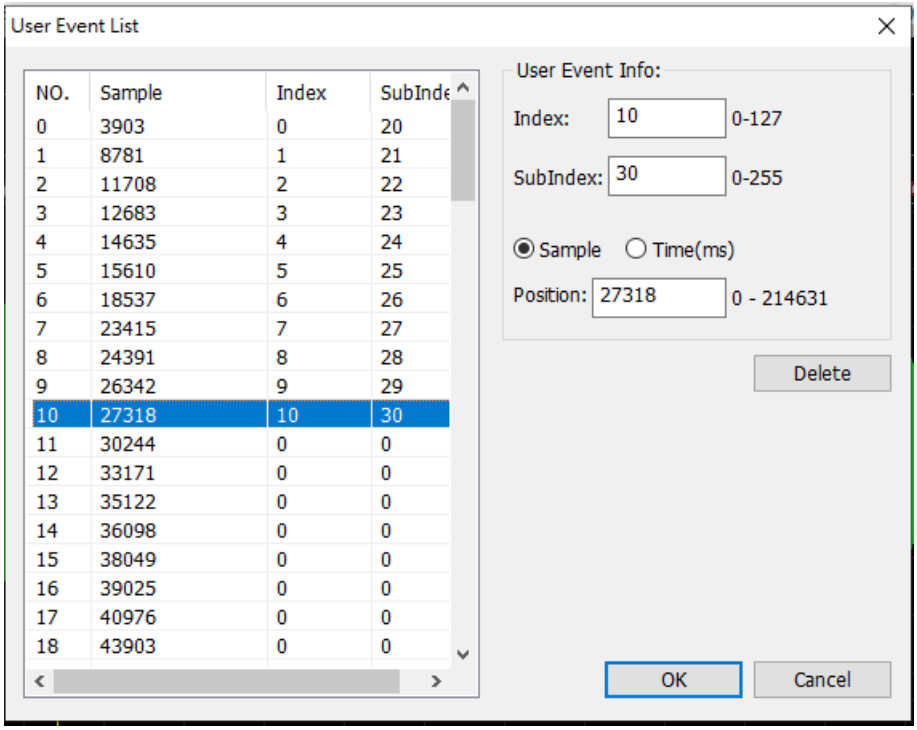
R_CMD_Code: Bit[15:8] = Main-Index, Bit[7:0] = Sub-Index

Return Value: None

Library: SACM_MS02_User.c

Remark:

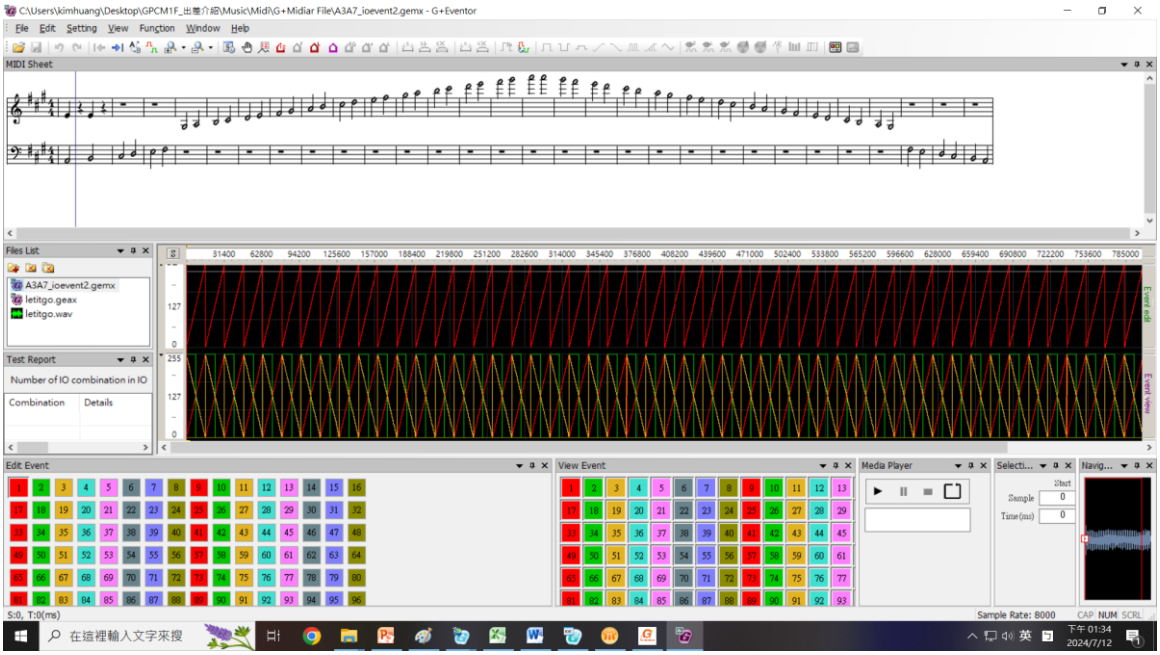
1. Insert user event(as below) via G+Eventor. It supports Index (= Main-Index) and SubIndex.



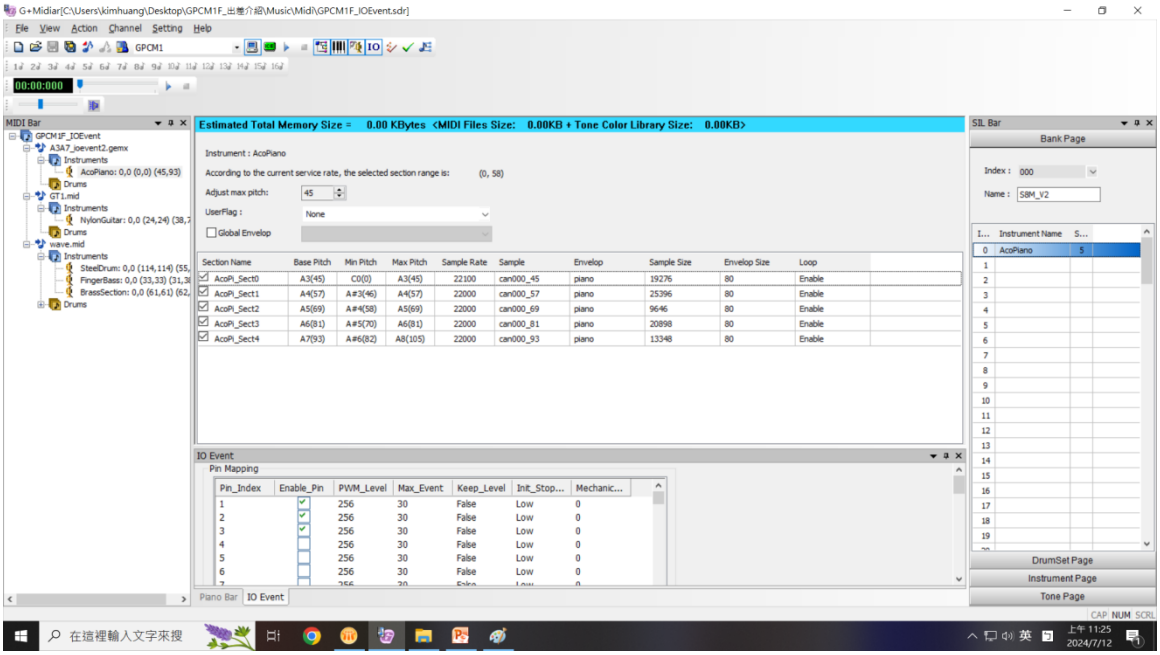
2. When decoding hits an event, it will enter the function and pass Main-Index and Sub-Index via R_CMD_Code parameter. Users can write the corresponding processing procedure in this function when the event is received.

```
/**
 * @brief
 *   Call back function for User event
 *
 * @param
 *    R_CallBackEvent: bit[15:12] = 0x9(Fix)
 *    R_CMD_Code: parameter of the callback event
 *                bit[15:8] = Main-Index
 *                bit[7:0]  = sub-Index
 * @return
 *   None.
 */
void MS02_CB_SongEvent(short R_CallBack_Event, short R_CMD_Code)
{
  //------------------------
  //CallBack Event processed by user
  //------------------------
  short mSongEvent;
  mSongEvent = R_CMD_Code;
}
```

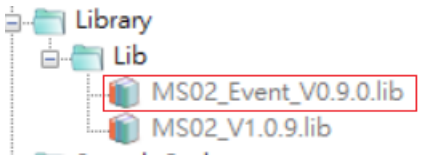
3. To add an IO event, you may use certain music creation tool such as Cakewalk to insert an event and then add an IO event in G+Eventor. After that, it will add the event and generate a .gem file.



4. Finally, we will use G+Midiar to generate a midi bin file that includes the tone library and the event(s) we insert.



5. In program settings, we should first add MS02_Event_Vx.x.lib



6. In SACM_MS02_User.c, we should:

   (a) Add GPCM_MS02_Eventor_User.h include file

SACM_MS02_User.c

```
 * Header File Included Area
 *--------------------------------------
#include "GPCM3_FM1.h"
#include "SACM_MS02_User.h"
#include "GPCM_MS02_Eventor_User.h"
#include "MS02_FileMerger.h"
#include "APP_SwVolumeControl.h"
```

(b) Declare the memory size of MS02_EVENTOR_WORKING_RAM for MS02 Event to use.

```
// For IO Event
#if defined(__CC_ARM)
__align(4) MS02_EVENTOR_WORKING_RAM GpEventorCh2WorkingRam;
__align(4) MS02_EVENTOR_CH2_SWPWM_WORKING_RAM MS02EventCh2WorkingRam;
#elif defined(__GNUC__)
__attribute__ ((aligned (4))) MS02_EVENTOR_WORKING_RAM GpEventorCh2WorkingRam;
__attribute__ ((aligned (4))) MS02_EVENTOR_CH2_SWPWM_WORKING_RAM MS02EventCh2WorkingRam;
#endif
```

(c) In MS02_CB_Init(), add MS02_EVENTOR_Initial(MS02_EVENT_CH2, &GpEventorCh2WorkingRam, (int16_t*)&MS02EventCh2WorkingRam, MS02_EVENT_IO_NUM, MS02_EVENT_SW_PWM_LEVEL), and conduct configuration.

```
void MS02_CB_Init(const SACM_MS02_WORKING_RAM *MS02WorkingRam, uint8_t AudOutType, uint8_t DacChanr
{
  mMS02DacCh0DmaHandle = DMA3;
  mMS02DacCh1DmaHandle = DMA4;
  mMS02DacTimerHandle = TM0;

  TIMER_Open(mMS02DacTimerHandle);
  TIMER_SetFreq(mMS02DacTimerHandle, Timer_SR);

  MS02_EVENTOR_Initial(MS02_EVENT_CH2, &GpEventorCh2WorkingRam, (int16_t*)&MS02EventCh2WorkingRam,
```

(d) In MS02_CB_StartPlay(), add MS02_EVENTOR_Start(MS02_EVENT_CH2, 0); enable Event function

```
void MS02_CB_StartPlay(const SACM_MS02_WORKING_RAM *MS02WorkingRam)
{
  if(AUD_OUT_TYPE == AUD_OUT_PWM)
  {
    DAC_VoltageDAC_CH0_Disable();
    DAC_VoltageDAC_CH1_Disable();
    DAC_AudioPwm_IP_Enable();
  }
  else
  {
    DAC_AudioPwm_IP_Disable();
    DAC_VoltageDAC_CH0_Enable();
    DAC_VoltageDAC_CH1_Enable();
  }
  DacAutoRampUp(SACM_MS02_GetStatus());
```

```
    if((SACM_MS02_GetStatus() & MS02_ENABLE_DAC_CH0_FLAG) == 0)
    {
      DMA_InstallIsrService(mMS02DacCh1DmaHandle, SACM_MS02_DmaIsrServi
      DMA_EnableInt(mMS02DacCh1DmaHandle);
      NVIC_EnableIRQ(DMA4_IRQn);
    }
    DMA_Trigger(mMS02DacCh1DmaHandle, (uint32_t)MS02WorkingRam->DecodeOu
    TIMER_Open(mMS02DacTimerHandle);
  }
  MS02_EVENTOR_Start(MS02_EVENT_CH2, 0);
}
```

(e) In MS02_CB_StopPlay(), add MS02_EVENTOR_Stop(MS02_EVENT_CH2, 0); enable Event function.

```
uint8_t MS02_CB_StopPlay(const SACM_MS02_WORKING_RAM *MS02WorkingRam)
{
  if((SACM_MS02_GetStatus() & MS02_PLAY_FLAG) != 0)
  {
    if((SACM_MS02_GetStatus() & MS02_ENABLE_DAC_CH0_FLAG) != 0)
    {
      DMA_Close(mMS02DacCh0DmaHandle);
    }

    if((SACM_MS02_GetStatus() & MS02_ENABLE_DAC_CH1_FLAG) != 0)
    {
      DMA_Close(mMS02DacCh1DmaHandle);
    }

    DacAutoRampDown(SACM_MS02_GetStatus());                    //
    if(AUD_OUT_TYPE == AUD_OUT_PWM)                            //
    {
      DAC_AudioPwm_IP_Disable();
    }
    MS02_EVENTOR_Stop(MS02_EVENT_CH2);
  }
```

(f) In MS02_CB_SongEvent (), add MS02_EVENTOR_ProcessEvt (MS02_EVENT_CH2, mSongEvent), and configure it to enable MS02 IO Event function.

```
void MS02_CB_SongEvent(short R_CallBack_Event, short R_CMD_Code)
{
  //-----------------------
  //CallBack Event processed by user
  //-----------------------
  short mSongEvent;
  mSongEvent = R_CMD_Code;

  mSongEvent = ((R_CMD_Code & 0x00FF)<< 8) + ((R_CMD_Code & 0xFF00)>> 8);
  MS02_EVENTOR_ProcessEvt(MS02_EVENT_CH2, mSongEvent);

}
```

14.11.2 Function: Note event notification

Syntax:

C: void MS02_CB_NoteOnEvent(short R_CMD_Code);

Parameters: R_CMD_Code:

(1) Bit[15:12] = 0x1 : Note Event

(2) Bit[11:8] = Channel 0~15

(3) Bit[7:0] = Pitch

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. When the decoding hits a note ON, it will enter the function and pass channel number and pitch information via R_CMD_Code parameter. Users can write the corresponding processing procedure in this function when the note event is received.

```
/**
 * @brief
 *   Call back function for Note on events
 *
 * @param
 *    R_CMD_Code:
 *             bit[15:12] = 0x1 : Note Event
 *             bit[11:8] Channel 0~15
 *             bit[7:0]  Pitch
 * @return
 *   None.
 */
void MS02_CB_NoteOnEvent(short R_CMD_Code)
{
  //------------------------
  // Note on Event processed by user
  //------------------------
  short mNoteEventCMD;
  if((R_CMD_Code & 0x0F00) == 0x0900)   // Ex. Only check Ch.9 note. (Note:User can change according to your needs.)
  {
    mNoteEventCMD = R_CMD_Code;
  }
}
```

14.11.3 Function: Call back user's function when SACM-MS02 starts playing

Syntax:

C: void MS02_CB_StartPlay(const SACM_MS02_WORKING_RAM *MS02WorkingRam);

Parameters: *MS02WorkingRam MS02 working ram pointer.

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function of SACM MS02 library when starting to play a MS02. It is for the Ramp-Up and DMA settings. Users can implement own functions here.

14.11.4 Function: Call back user's function when SACM-MS02 stops playing

Syntax:

C: uint8_t MS02_CB_StopPlay(const SACM_MS02_WORKING_RAM *MS02WorkingRam);

Parameters: *MS02WorkingRam MS02 working ram pointer.

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function of SACM MS02 library when playing MS02 stops. It will execute Ramp-Dn function (if MS02_AUTO_RAMP_DOWN function is enabled) and turn off the corresponding DMA. Users can implement their own functions here.

14.11.5 Function: Call back user's function when pauses a SACM-MS02 speech playback

Syntax:

C: void MS02_CB_Pause(const SACM_MS02_WORKING_RAM *MS02WorkingRam);

Parameters: *MS02WorkingRam MS02 working ram pointer.

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function of SACM MS02 library when pause playing a midi, users can implement their own functions in it.

14.11.6Function: Call back user's function when resumes a paused SACM-MS02 speech

Syntax:

C: void MS02_CB_Resume(const SACM_MS02_WORKING_RAM *MS02WorkingRam);

Parameters: *MS02WorkingRam MS02 working ram pointer.

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function of SACM MS02 library when resume from a paused midi; users can implement their own functions in it.

14.11.7Function: Gets the MS02 MIDI data from user's storage and write to buffer

Syntax:

C: void MS02_CB_GetData(const SACM_MS02_WORKING_RAM *MS02WorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *MS02WorkingRam MS02 working ram pointer.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function for SACM MS02 library to read midi data from memory storage. When MIDI determining to decode continuously, call this function and move data from the user-designated source data area to destination buffer.

14.11.8Function: Call back user's function by MS02 library to decode data

Syntax:

C: void MS02_CB_DecodeProcess(int16_t* DstBufAddr, int16_t OutBufSize);

Parameters: *MS02WorkingRam MS02 working ram pointer.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. *SrcDataAddr pointer points to the source buffer to be decoded and *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_MS02_DecodeProcess() to perform decode process. User will obtain the decoded speech data here. When a software volume control mechanism is used, SACM_MS02_SetVolumeGain will be placed here for volume control process.

14.11.9Function: Send MIDI data to DAC output channel through DMA

Syntax:

C: void MS02_CB_SendDac_DmaIsr(const SACM_MS02_WORKING_RAM *MS02WorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *MS02WorkingRam MS02 working ram pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_MS02_User.c

Remark:

1. This function is a call-back function for SACM MS02 library to send speech data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.

2. DMA move is based on one-frame size of 320 each time. When a complete frame data is moved, it will trigger DMA INT.

Example:

Play a SACM-MS02 melody.

(a). In main.c:

```
uint8_t PlayCon = 1;
uint16_t MidiNum = 0;
int16_t MidiIdx = 2;
int16_t MS02LibIdx =1;
uint8_t MS02VolGainIdx = 7;
main()
{
   SystemInit();
   SPIFC_Open();
```

```
    SPIFC_SetClkDiv(SPIFC_CLKSEL_HCLK_DIV1);
    SPIFC_AutoMode(SPIFC_4IO_ENHANCE_MODE);
    SPIFC_TimingFineTune();                          // Calibrate SPIFC clock timing

    MidiNum = GetMidiNum();
    SACM_MS02_Initial(&MS02WorkingRam, AUD_OUT_PWM, MS02_CH1, MS02_Timer_20K);
    SACM_MS02_ChannelKeep_Disable(MS02WorkingRam.MS02KernelRamPtr);
    SACM_MS02_DrumKeep_Disable(MS02WorkingRam.MS02KernelRamPtr);
    APP_SwVolCtrl_SetVolGain(0, SwVolGain);
    SACM_MS02_Play(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx),
            MS02_AUTO_RAMP_UP + MS02_AUTO_RAMP_DOWN, MS02_DAC_20K);
    while(1)
    {
        WDT_Clear();
        SACM_MS02_ServiceLoop();
        …
        if((PlayCon != 0) && (SACM_MS02_Check_Con() == 0))
        {
           MidiIdx = ++MidiIdx > MidiNum ? 2 : MidiIdx;
           SACM_MS02_Play_Con(GetMS02LibStartAddr(MS02LibIdx), GetMidiStartAddr(MidiIdx),
           MS02_AUTO_RAMP_UP + MS02_AUTO_RAMP_DOWN, MS02_DAC_20K);
        }
} // end of main()
```

15 API for SACM-MS02 PN

1. Reasons to add a new MIDI PN(Play Note):
   (a) MS02 PN Lib is capable of working independently; it does not need to be attached to MS02 Lib.
   (b) Drum Ch.10 supports multiply tracks to play several drums (Originally, MS02 Lib does not support multiple drums playback at the same time because all drums are assigned at Ch.10. If there are more than two drums (included) playing concurrently, the new one will replace the old one.)
   (c) No need of midi.bin and no length limit for midi: A standard midi lib requires a midi song and the Play Note & Drum must be attached with a midi.bin in order to be played. To simply play note or drum, a midi.bin is still needed and when midi.bin ends, play note will end too.
   (d) To play MIDI and Note concurrently: Without MIDI PN, when MS02 Lib is playing midi and note or drum at the same time, the played midi in the channel will be replaced by the instrument of the play note.
   (e) To resolve the extra RAM needed: Without MIDI PN, the channel is already assigned accordingly; for example, suppose only four channels (ch 0,1,2,9) are used, the MS02 Lib will enable all channel0 ~ 9. Thus, it will waste other unused RAMs. With the new MIDI PN, number of channel is assigned dynamically as needed. As a result, RAM space can be saved and only be used as much as needed.

15.1 Hardware Dependent Function: Initializes SACM-MS02 PN

15.1.1 Function: Initialize SACM-MS02 PN library

Syntax:

C: void SACM_MS02PN_Initial(SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam, uint8_t *MS02PNKernelRamPtr, int16_t *MS02PN_LibPtr, uint8_t DacChannelNo, uint8_t AutoRampUpDownCtrl, int32_t SampleRate);

Parameters: SACM_MS02PN_WORKING_RAM *: MS02 PN working RAM pointer

*MS02PNKernelRamPtr: MS02 PN kernel RAM pointer.

*MS02PN_LibPtr: Pass MS02LibIdx and call GetMS02PNLibStartAddr () function to obtain the start address of MS02 PN library.

DacChannelNo: 1: MS02_CH0
2: MS02_CH1

AutoRampUpDownCtrl: 0: MS02_AUTO_RAMP_DISABLE
1: MS02_AUTO_RAMP_UP
2: MS02_AUTO_RAMP_DOWN

SampleRate: For playback
0: MS02_DAC_8K
1: MS02_DAC_10K

2: MS02_DAC_12K

3: MS02_DAC_16K

4: MS02_DAC_20K

5: MS02_DAC_24K

6: MS02_DAC_28K

4: MS02_DAC_32K

5: MS02_DAC_36K

6: MS02_DAC_40K

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function initializes the decoder of MS02PN. It also initializes the DMA, and enables the Timer at the sample rate in user's setting.
2. This function will call MS02PN_CB_Init () (in SACM_MS02PN_User.c) for hardware setting.
3. The settings in Timer_SR must be the same SampleRate parameter as in SACM_MS02_Play().

15.2 Service Loop Functions: Service loop for SACM-MS02 PN decoding process

15.2.1 Function: Service loop for SACM-MS02 PN decoding process

Syntax:

C: void SACM_MS02PN_ServiceLoop(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. In this function, the MIDI events are parsed and processed for sequencer.
2. This function has to be placed in main loop.

15.3 Playback Functions: Playback control

15.3.1 Function: Plays a SACM-MS02 note

Syntax:

C: void SACM_MS02PN_PlayNote(int16_t MidiChan, int16_t pitch, int16_t velocity, int16_t

duration);

Parameters:

MidiChan:	Set MIDI channel (0 ~ 15)
pitch:	Set pitch(0 ~ 127)
velocity:	Set velocity(0 ~ 127)
duration:	0 ~ 65535 tickers, where a ticker typically 5ms long.

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. Velocity is a significant parameter related with the volume (0～127). A larger number of velocity represents the higher volume. For instance, the sound of a piano will generate different volume intensity based on how hard a key is pressed.

15.3.2 Function: Plays a SACM-MS02 drum

Syntax:

C: void SACM_MS02PN_PlayDrum(int16_t MidiChan, int16_t drumidx, int16_t velocity, int16_t duration);

Parameters:

MidiChan:	Set MIDI channel (0 ~ 15)
drumidx:	Set the drum track(0 ~ 15)
velocity:	Set velocity(0 ~ 127)
duration:	0 ~ 65535 tickers, where a ticker typically 5ms long.

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. Velocity is a significant parameter related with the volume (0～127). A larger number of velocity represents the higher volume. For instance, the sound of a piano will generate different volume intensity based on how hard a key is pressed.

15.3.3 Function: Stop a playing SACM-MS02 melody note

Syntax:

C: void SACM_MS02PN_NoteOff (int16_t MidiChan);

Parameters:

MidiChan:	Set MIDI channel (0 ~ 15)

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. One of playing MS02PN channels can be turned off. If SACM_MS02PN_Stop() is executed, all playing MS02PN channels will be turned off.

15.3.4 Function: Stops all of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_Stop (void)

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark: When SACM-MS02 melody note is stopped playing, it will run MS02PN_CB_StopPlay (), at SACM_MS02PN_User.c, and turn off DMA.

15.3.5 Function: Pauses a playing SACM-MS02 melody note

Syntax:

C: void SACM_MS02PN_Pause(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark: None

15.3.6 Function: Resumes a paused SACM-MS02 melody note

Syntax:

C: void SACM_MS02PN_Resume(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

15.3.7 Function: Gets the status of the SACM-MS02PN module

Syntax:

C: uint16_t SACM_MS02PN_GetStatus(void);

Parameters: None

Return Value: bit 0: 0x0001: MS02_PN playing

0x0004: MS02_PN paused

0x0008: MS02_PN auto ramp down enabled

0x0010: MS02_PN auto ramp up enabled

0x0800: MS02_PN stop

Others: Reserved

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark: None

15.3.8 Function: Clear the status of the SACM-MS02PN module

Syntax:

C: void SACM_MS02PN_ClearStatus(uint16_t StatusFlag);

Parameters: None

Return Value: bit 0: 0x0001: MS02_PN playing

0x0004: MS02_PN paused

0x0008: MS02_PN auto ramp down enabled

0x0010: MS02_PN auto ramp up enabled

0x0800: MS02_PN stop

Others: Reserved

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark: None

15.3.9 Function: Changes the instrument on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_ChangeInstru(int16_t MidiChan , int16_t Instrument);

Parameters:

MidiChan: 0 ~ 15.

Instrument: 0 ~ max of Instrument

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. The instrument argument is the mapped instrument index. It is determined when the MIDI and tone color are exported from G+Midiar.

15.3.10 Function: Changes the pitch bend on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_PitchBend(MidiChan , PitchBendValue);

Parameters:

MidiChan: 0 ~ 15.

PitchBendValue [in](14-bit): -8192: -2 semitones ,0: original ,8191: +2 semitones

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark: None

15.3.11 Function: Changes the velocity on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_ChangeVelocity(MidiChan , Velocity);

Parameters:

MidiChan: 0 ~ 15.

Velocity: Range: 0～127, the larger number means the louder of designated channel is and vice versa.

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function should be executed after SACM_MS02PN_PlayNote() or SACM_MS02PN_PlayDrum().
2. Velocity's default value is the velocity value given at SACM_MS02PN_PlayNote().

15.3.12 Function: Changes the ticker on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_ChangeTicker(MidiChan , TickerIdx);

Parameters:

MidiChan: 0 ~ 15.

TickerIdx: Range: 0.5ms~20ms(=Max. 40). Default: 5ms(=10), 1 step: 0.5ms

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function takes effective only after MS02PN initial.
2. The larger TickerIdx number given, the longer note playing time is.

15.3.13 Function: Set the release step on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_SetReleaseStep(MidiChan , CustomStep);

Parameters:

MidiChan: 0 ~ 15.

CustomStep: -1~-32768 (-1:slowest release, -32768:fastest release)

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function must work with SACM_MS02PN_ResetReleaseStep().
2. This function must be executed after SACM_MS02PN_PlayNote() or SACM_MS02PN_PlayDrum() in order to prevent the assigned parameters being over-written by the default values.

15.3.14 Function: Reset the release step on one of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_ResetReleaseStep(MidiChan);

Parameters: MidiChan: 0 ~ 15.

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function must work with SACM_MS02PN_SetReleaseStep ().

15.3.15 Function: Set the vibration rate on all of the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_VibrationRate (Rate);

Parameters: Rate: 0~32767

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. A larger rate value given, voice gets more trembling.
2. This function must work with SACM_MS02PN_VibrationEnable().

15.3.16 Function: Enable the vibration function of all the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_VibrationEnable(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function must work with SACM_MS02PN_VibrationDisable().

15.3.17 Function: Disable the vibration function of all the SACM-MS02PN channels

Syntax:

C: void SACM_MS02PN_VibrationDisable(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. This function must work with SACM_MS02PN_VibrationEnable().

15.4 ISR Functions: DMA Interrupt service routine for SACM-MS02PN

This routine will get the decoded data from service loop subroutine and send data to DAC for playing. The initial function, MS02PN_CB_Init, in SACM_MS02PN_User.c must also be updated as well.

Syntax:

C: void SACM_MS02PN_DmaIsrService(void);

Parameters: None

Return Value: None

Library: <MS02PN_GPCM_Vxxx.LIB>

Remark:

1. DMA is able to send the MS02 lib-decoded MIDI note data to DAC_out. When a frame size (320-byte) of data has been sent completely, the DMA INT will be triggered and execute SACM_MS02PN_DmaIsrService to notify MS02PN service loop function to proceed decoding next frame of data.

2. In SACM_MS02PN_DmaIsrService, it will execute MS02PN_CB_SendDac_DmaIsr (in SACM_MS02PN_User.c) that determines which DAC channel (Ch0 or Ch1) is assigned for DAC output channel and enables DMA again. Users will obtain decoded midi data here.

15.4.1 Function: Call back user's function when SACM-MS02PN starts playing

Syntax:

C: void MS02PN_CB_StartPlay(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. This function is a call-back function of SACM MS02PN library when starting to play a MS02 note. It is for the Ramp-Up and DMA settings. Users can implement own functions here.

15.4.2 Function: Call back user's function when SACM-MS02PN stops playing

Syntax:

C: uint8_t MS02PN_CB_StopPlay(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. This function is a call-back function of SACM MS02PN library when playing MS02 note stops. It will execute Ramp-Dn function (if MS02_AUTO_RAMP_DOWN function is enabled) and turn off the corresponding DMA. Users can implement their own functions here.

15.4.3 Function: Call back user's function when pauses a SACM-MS02PN speech playback

Syntax:

C: void MS02PN_CB_Pause(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. This function is a call-back function of SACM MS02 library when pause playing a midi note, users can implement their own functions in it.

15.4.4 Function: Call back user's function when resumes a paused SACM-MS02PN midi note

Syntax:

C: void MS02PN_CB_Resume(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. This function is a call-back function of SACM MS02PN library when resume from a paused midi; users can implement their own functions in it.

15.4.5 Function: Gets the MS02 MIDI Note data from user's storage and write to buffer

Syntax:

C: void MS02PN_CB_GetData(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam, int16_t *DstBufAddr, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

*DstBufAddr: Destination buffer address pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. This function is a call-back function for SACM MS02PN library to read midi note data from memory storage. When MIDI note determining to decode continuously, call this function and move data from the user-designated source data area to destination buffer.

15.4.6 Function: Call back user's function by MS02PN library to decode data

Syntax:

C: void MS02PN_CB_DecodeProcess(int16_t* DstBufAddr, int16_t OutBufSize);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

*DstBufAddr: Destination buffer address pointer.

OutBufSize: MS02_OUT_BUFFER_SIZE = 320

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

1. *DstBufAddr will point to destination buffer after decode.
2. This function will call SACM_MS02PN_DecodeProcess() to perform decode process. User will obtain the decoded midi note data here. When a software volume control mechanism is used, SACM_MS02PN_SetVolumeGain will be placed here for volume control process.

15.4.7 Function: Send MIDI note data to DAC output channel through DMA

Syntax:

C: void MS02PN_CB_SendDac_DmaIsr(const SACM_MS02PN_WORKING_RAM *MS02PNWorkingRam, int16_t *SrcDataAddr, uint16_t DataLen);

Parameters: *MS02PNWorkingRam MS02PN working ram pointer.

*SrcDataAddr: Source buffer address pointer.

DataLen: Length of data moved

Return Value: None

Library: SACM_MS02PN_User.c

Remark:

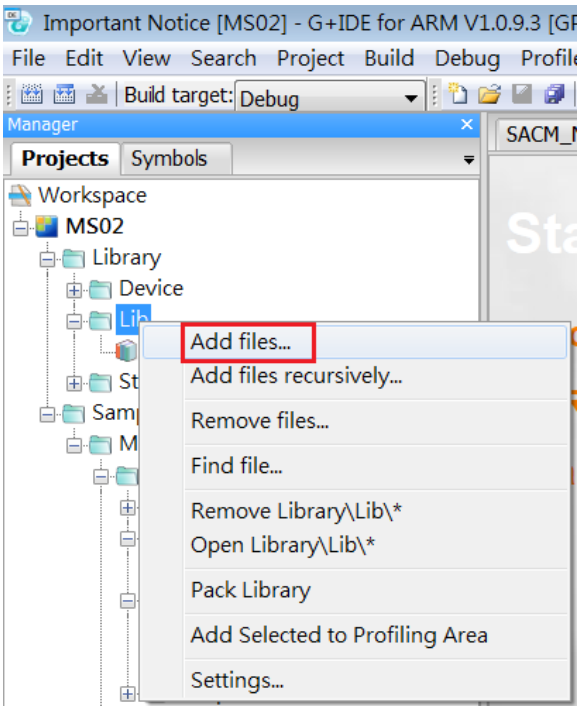
1. This function is a call-back function for SACM MS02PN library to send melody data to DAC output channel. User may set the DAC output channel to Ch0 or Ch1 and re-enable DMA again.

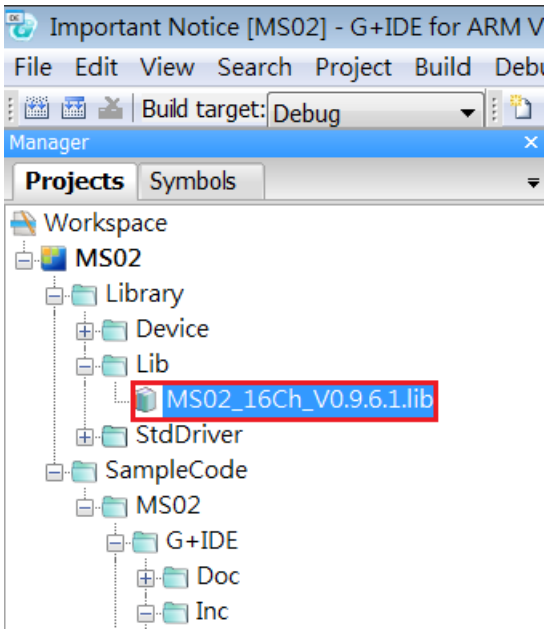
2. DMA move is based on one-frame size of 320 each time. When a complete frame data is moved, it will trigger DMA INT.

16 How to Use the Speech Library

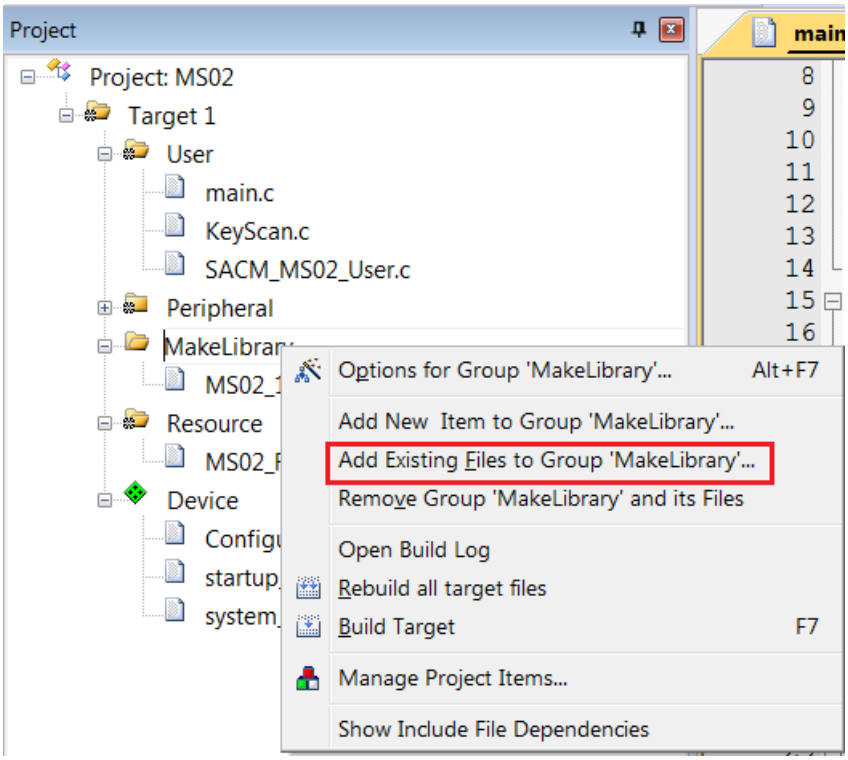
16.1 Link the libraries to User's Program

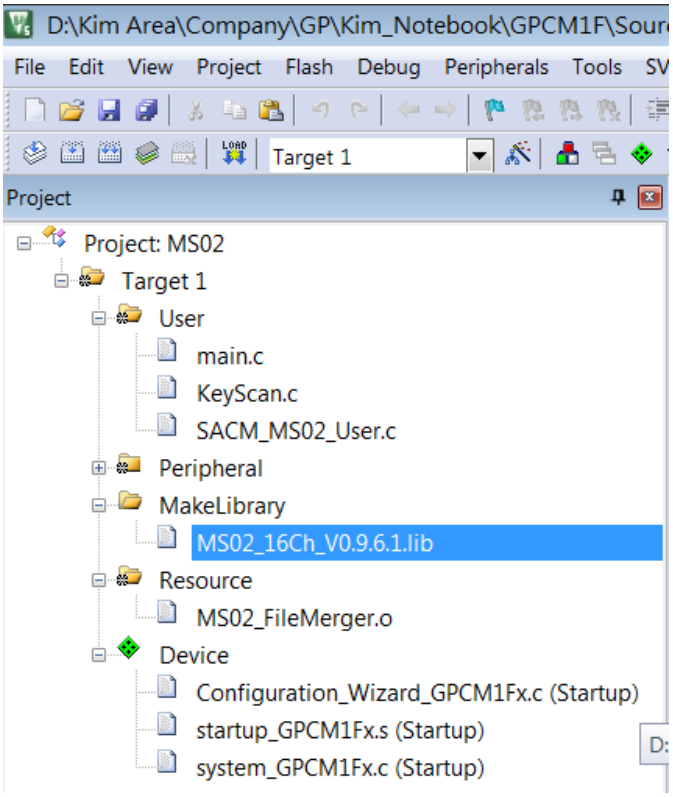
(a). In G+IDE for ARM: [Right-click on the Project tree directory] → [Add files] and add library, i.e. MS02_16Ch_Vx.x.x.lib to the Library group.





(b). In Keil: [Right-click on the Project tree directory] → [Add Existing Files to Group 'MakeLibrary'] and add library, i.e. MS02_16Ch_V0.9.6.1.lib to the MakeLibrary group.

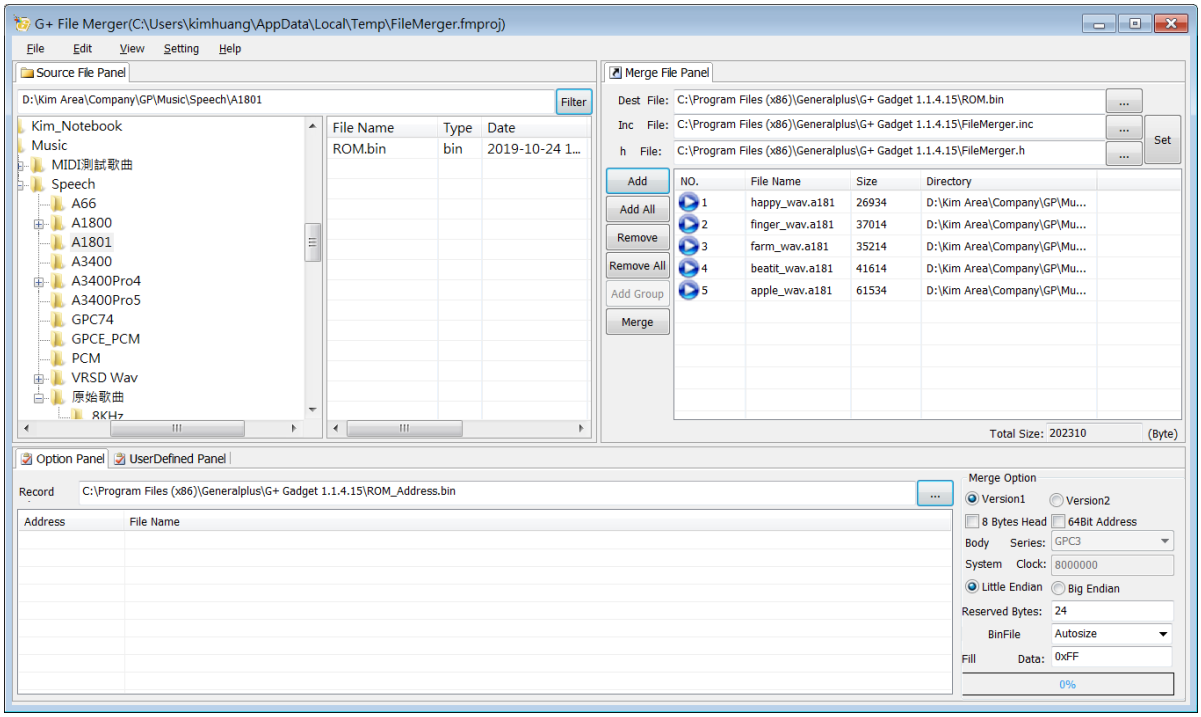




16.2 Adding Resources

(a). Before adding Speech or MIDI into project, use G+File Merger tool to merge and sort them; in addition, generate the start addresses information for each song and tone library.

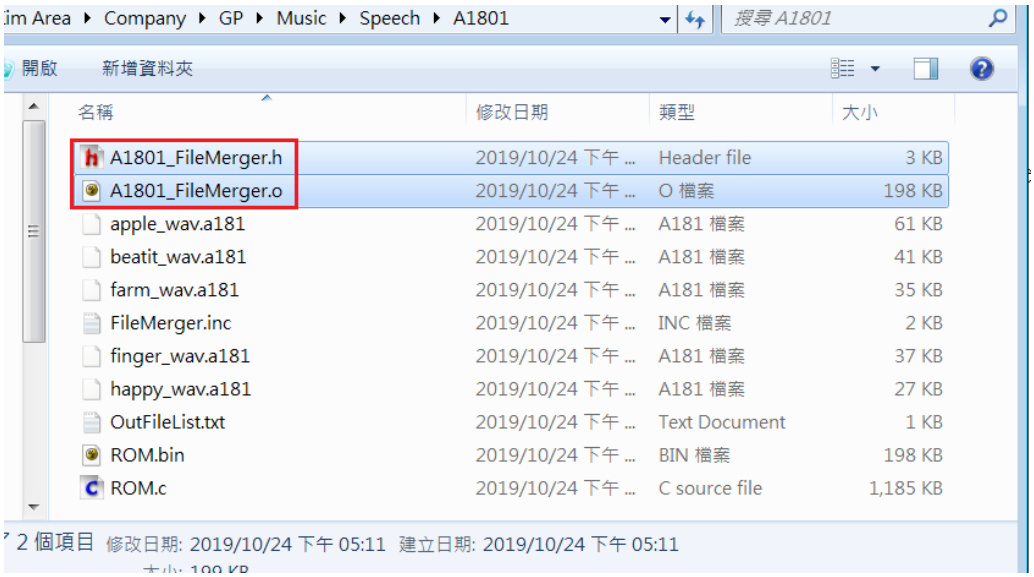
(b). G+File Merger: File Merger is capable of integrating multiple files into one single file. In general, before storing and downloading multiple songs into SPI FLASH, we will integrate and order those songs into *.bin, *.c and *.o files using File Merger.



G+File Merger GUI

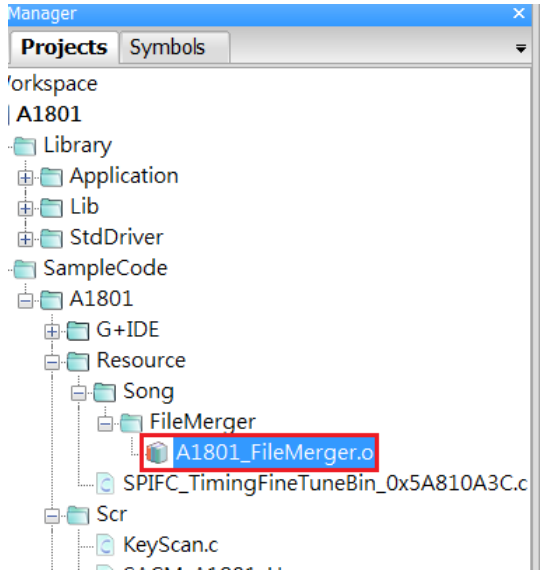
2) Through G+File Merger tool to generate *.h and *.o files for GPCMF project. Please rename the *.h and *.o files to A1801_FileMerger.h and A1801_FileMerger.o.

   Note: In a MIDI project, rename to MS02_FileMerger.h and MS02_FileMerger.o



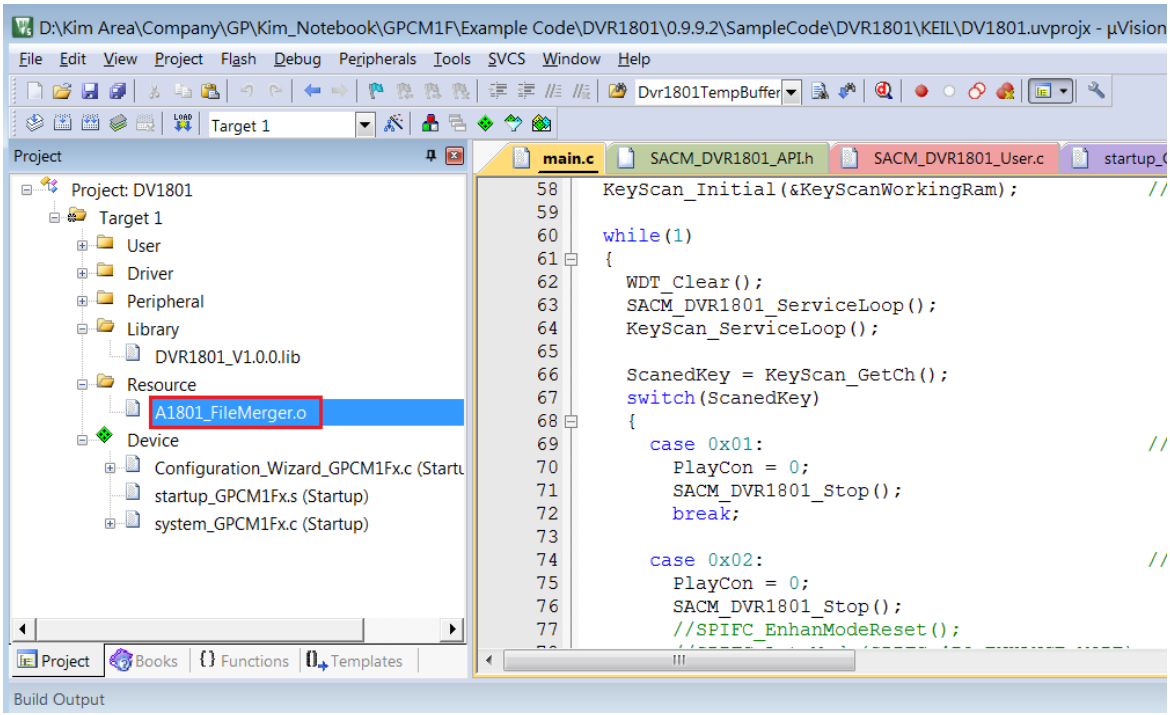
3) Copy A1801_FileMerger.h and A1801_FileMerger.o. files into Keil or G+IDE for ARM SACM project and overwrite the original contents and rebuild to replace song(s).

(c). Add A1801_FileMerger.o. file into Project. **In G+IDE for ARM: [Right-click on the Project tree directory] → [Add files]** and add A1801_FileMerger.o to the Resource group.



In Keil: [Right-click on the Project tree directory] → [Add Existing Files to Group 'MakeLibrary'] and add

A1801_FileMerger.o to the Resource group.



16.3 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:

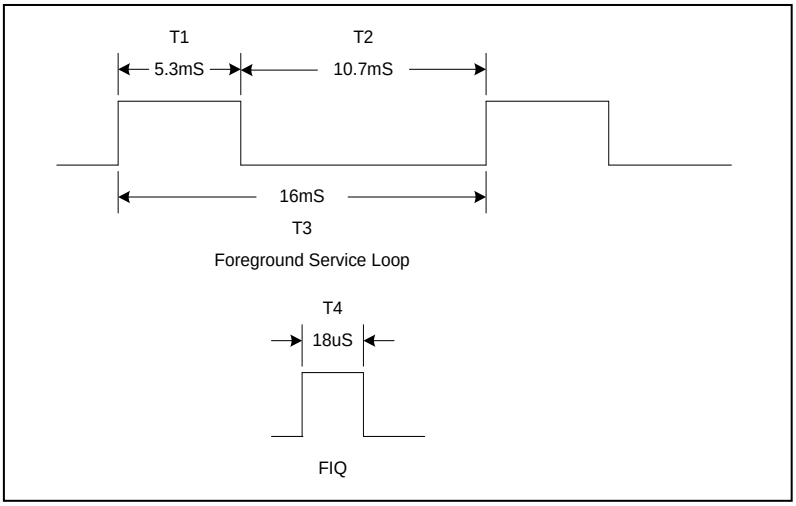
1. Open the sample project that contains the SACM algorithm you need.
2. Modified the Hardware setting in SACM_xxx_User.inc in necessary. (e.g. A1801_CB_Init)
3. Add user resources into the project resources. (e.g. A1801_FileMerger.o)
4. Rewrite main.c and rebuild project to test the speech files you just added.
5. Add application code to the project in either C and modified the main.c for flow control.

17 Resources list of SACM Algorithm

17.1 CPU Usage Rate:

GPCM1F, CPU clock = 81.92M Hz, wait state = 4 clock, Cache On							
Item	Frame Size	S.R.	T1(mS)	T2(mS)	T3(mS)	T4(uS)	CPU rate
A1801 (16K bps)	320	16000	3.2	16.8	20	13	**16.07%**
A1801 Ch2 (16K bps)	320	16000	3.2	16.8	20	13	**16.07%**
DVR1801 decode (16K bps)	320	16000	3.26	16.74	20	15.3	**16.38%**
DVR1801 encode (16K bps)	320	16000	2.56	17.44	20	11.4	**12.86%**
A2000 (32K bps)	640	32000	5	15	20	5.4	**25.03%**
A2000 Ch2 (32K bps)	640	32000	5	15	20	5.4	**25.03%**
DVRPCM	32	16000	0.033	1.967	2	4	**1.85%**
A3400Pro 4-bit (34Kbps)	32	12000	0.092	2.5747	2.66667	13	**3.94%**
A3400Pro Ch2 4-bit (34Kbps)	32	12000	0.092	2.5747	2.66667	13	**3.94%**
MS02 16Ch (16K S.R)	320	16000	~8.6	11.4	20	13	**~43%**
MS02 16Ch (20K S.R)	320	20000	~8.6	7.4	16	13	**~54%**
MS02 32Ch (16K S.R)	320	16000	~16	4	20	13	**~80%**
MS02 32Ch (20K S.R)	320	20000	~16	0	20	13	**~100%**
MS02_PN 8Ch (20K S.R)	320	20000	3.66	12.34	16	7.8	**22.92%**
MS02_PN 16Ch (20K S.R)	320	20000	7	9	16	7.8	**43.80%**

GPCM3, CPU clock = 122.88M Hz, SPIFC clock = 98.304MHz, Cache On							
Item	Frame Size	S.R.	T1(mS)	T2(mS)	T3(mS)	T4(uS)	CPU rate
A1801 (16K bps)	320	16000	2.2	17.8	20	7.4	**11.04%**
A1801 Ch2 (16K bps)	320	16000	2.2	17.8	20	7.4	**11.04%**
A2000 (20K bps)	640	20000	3.56	28.44	32	9	**11.15%**
A2000 Ch2 (20K bps)	640	20000	3.56	28.44	32	9	**11.15%**
A2000 (32K bps)	640	32000	3.5	16.5	20	9	**17.55%**
A2000 Ch2 (32K bps)	640	32000	3.5	16.5	20	9	**17.55%**
A2000 (48K bps)	640	32000	3.76	16.24	20	9	**18.85%**
A2000 Ch2 (48K bps)	640	32000	3.76	16.24	20	9	**18.85%**
A3400Pro 4-bit	32	12000	0.058	2.6087	2.6667	3.2	**2.30%**
A3400Pro Ch2 4-bit	32	12000	0.058	2.6087	2.6667	3.2	**2.30%**
MS02 8Ch (20K S.R)	320	20000	3.128	12.872	16	1.7	**19.56%**
MS02 8Ch (24K S.R)	320	24000	3.2	10.133	13.333	1.7	**24.01%**
MS02 32Ch (20K S.R)	320	20000	5.8	10.2	16	1.7	**36.26%**
MS02 32Ch (24K S.R)	320	24000	5.8	7.5333	13.333	1.7	**43.51%**
MS02_PN 8Ch (20K S.R)	320	20000	2.86	13.14	16	4	**17.90%**
MS02_PN 16Ch (20K S.R)	320	20000	5.54	10.46	16	4	**34.65%**



(1) T1: The executing time of service loop inside main Loop

(2) T2: Representing the time to execute other programs, but this time period should be deducted from the time required for the interrupt to execute the ISR since using DMA ISR processing only requires one interrupt for each frame.

(3) T3: Maximum allowable time without interruption during playback= 320 frame*(1/sampling rate). This time period is relevant to sampling rate, e.g. S.R = 16KHz = 62.5us=> T3 = 320*62.5us=20ms.

(4) T4: ISR processing time. Using DMA ISR only requires one interrupt for each frame, approximate 13us ~ 15us per interrupt.

(5) CPU Rate = (T1+T4)/T3

17.2 Resource: RAM & ROM size

(a).RAM Size:

Item	IRAM	ORAM	Stack	Total RAM (byte) (Does not include Stack size)
A1801	48	2340	~470	2388
A1801 Ch2	20	1700	~470	1720
DVR1801	108	2628	~470	2736
DVRPCM	12	172	~100	184
A3400Pro	46	192	~100	238
A3400Pro Ch2	46	192	~100	238
MS02 16Ch	9	3334	~300	3343
MS02 32Ch	9	4422	~300	4431
MS02_PN 8Ch	20	2710	~300	2730
MS02_PN 16Ch	20	3222	~300	3242

(b).ROM Size:

Item	Code	RO Data	Total ROM (byte)
A1801	7906	11408	**19314**
A1801 Ch2	2280	16	**2296**
DVR1801	14164	23584	**37748**
DVRPCM	2236	16	**2252**
A3400Pro	3560	2093	**5653**
A3400Pro Ch2	2724	45	**2769**
MS02 16Ch	6080	1134	**7214**
MS02 32Ch	6128	1134	**7262**
MS02_PN	4610	16	**4626**