MMCODECS

SRC on TMS320C55xx

Software Release Report

Software Released	Release version	Release Date
SRC	**3.1**	**28-Jan-04**

Prepared by: Srividya M. S.

Approved by: Anita Shet

Table of Contents

1 SAMPLING RATE CONVERTER ALGORITHM 3
1.1 Overview 3
1.2 Implementation Resource Usage 4
1.2.1 Performance 4
1.2.2 Testing 4
2 SRC HIGH LEVEL INTERFACE 5
2.1 ISRC Interface Design 5
2.2 xDAIS Interface APIs 6
2.2.1 SRC_TI_algNumAlloc 6
2.2.2 SRC_TI_algAlloc 6
2.2.3 SRC_TI_algInit 7
2.2.4 SRC_TI_algFree 7
2.2.5 SRC_create 7
2.2.6 SRC_delete 8
3 EXTERNAL APIS 9
3.1 SRC_setInput 9
3.2 SRC_setOutput 9
3.3 SRC_isInputBufferEmpty 9
3.4 SRC_isOutputBufferFull 10
3.5 SRC_getStatus 10
3.6 SRC_setStatus 10
3.7 SRC_getInputRate 10
3.8 SRC_getOutputRate 11
3.9 SRC_setInputRate 11
3.10 SRC_setOutputRate 11
3.11 SRC_apply 11
4 SRC OBJECT 12
4.1 ISRC_Obj Structure 12
4.2 ISRC_Fxns 12
4.3 ISRC_Fxns Struct 12
4.4 ISRC_Params Structure 12

1 SAMPLING RATE CONVERTER ALGORITHM

1.1 Overview

A Sample Rate Converter (SRC) converts digital audio signals from one signal density to another. Audio data that is sampled at a higher rate, e.g. 44.1kHz, is often stored at a lower sampling rate due to bandwidth, storage or processor limitations. This data needs to be converted to the original sampling rate upon playback to achieve the proper sound quality. Audio data may also be converted to different sampling rates in order to conform to various system configurations. An SRC performs these transformations with minimal altering of the original sound.

The SRC module has the following features:

- Efficient Stereo SRC implementation on a single TMS320C55x DSP
- Supported Sampling Frequencies (KHz): 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48
- 16^{th} Order Lagrange Interpolation for upward conversion
- 16^{th} Order Lagrange Interpolation for downward conversion
- 2^{nd} Order Elliptic LPF Filter for anti aliasing.
- Fully compliant with xdais specification.
- Fully validated on TMS320C5510 EVM using CCS ver 2.2 with the code generation tools version 2.56.
- SRC supports any to any sample rate conversion within the standard set given below:

Sampling Frequency in Hz
8000
11025
12000
16000
22050
24000
32000
44100
48000

Table 1: Supported Sampling Frequencies

NOTE: - Above specified frequencies can be converted to the same frequency without any distortion. For example a 16Khz song can be converted to 16Khz without any distortion.

1.2 Implementation Resource Usage

1.2.1 Performance

Detailed Memory and MIPS requirement is available at SRC_datasheet.doc attached along with the release.

1.2.2 Testing

Non real time testing on SRC is performed in two different phases.

1) One to measure the alias component in the output. The following example can explain how to perform the testing:

Suppose the input Sample Rate is 44.1 KHz and output rate is 8 KHz, we take a sweep sine wave with information bandwidth of 0KHz – 22.05KHz (Nyquist rate) and sample rate 44.1KHz. This signal is converted to a signal of sample rate 8Khz, then we measure the alias component in the output signal. The alias information for each conversion can be observed in the file AliasInformation.doc.

2) In phase two testing we perform listening test with a song for all combination of sample rate conversion.

2 SRC HIGH LEVEL INTERFACE

2.1 ISRC Interface Design

The given algorithm implements ISRC interface, which is derived from IALG interface. The module name is **SRC** and vendor name is **TI**. These API's are logically part of XDAIS interface framework and provided to simplify the use of given algorithm.

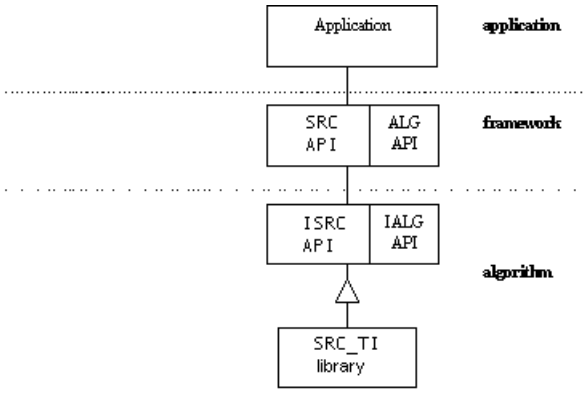

Figure 1: XDIAS Interface Design

The following is the list of the API that will be supported as part of XDAIS of the SRC interface design.

2.2 xDAIS Interface APIs

- `Int SRC_TI_algNumAlloc (void);`
- `Int SRC_TI_algAlloc(const IALG_Params * Params, IALG_Fxns ** fxns, IALG_MemRec * memTab);`
- `Int SRC_TI_algFree(IALG_Handle Handle, IALG_MemRec * mem);`
- `Int SRC_TI_algInit(IALG_Handle handle, const IALG_MemRec *mem, IALG_Handle p, IALG_Params const *params)`

2.2.1 SRC_TI_algNumAlloc

Prototype:

```
Int       SRC_TI_algNumAlloc(void)
```

Description:

This functions returns number of memory blocks to be assigned for the data memory.

2.2.2 SRC_TI_algAlloc

Prototype:

```
Int       SRC_TI_algAlloc(const IALG_Params * Params,
                          IALG_Fxns ** fxns,
                          IALG_MemRec * memTab);
```

Description:

This function fills up the field of structure. This structure fields has values about alignment, size and attribute of the different memory blocks.
The framework needs to call this function in order to find out the memory requirements of the SRC. The memory requirements will be filled in the IALG_MemRec structure array. Each IALG_MemRec structure will have the memory requirements for one block. The memTab array should have n elements if n blocks of memory have to be allocated. The base address for each of the blocks should be determined by the framework. So the base address field will not be filled in by this function.

```
typedef struct IALG_MemRec {
    Int             size;     /* size in MAU of allocation */
    Int             alignment;/* alignment requirement (MAU) */
    IALG_MemSpace   space;    /* allocation space */
    IALG_MemAttrs   attrs;    /* memory attributes */
    Void            *base;    /* base addrs of allocated buf */
} IALG_MemRec;
```

The IALG_algNumAlloc function should be called before this function is called. This is necessary because the system needs to allocate memory for the MemRec structure array,so passing it as a parameter to this function.

Parameters:

`Params` : Pointer to the Creation Parameters for this instance.

`Fxns` : fxns is a pointer to the v-table for the SRC implementation. This field is initialized by SRC_Obj().

`memTab` : An array of IALG_MemRec structures, indicating the blocks of memory for the instance's storage.

2.2.3 SRC_TI_algInit

Prototype:

```
Int  SRC_TI_algInit(SRC_TI_Handle handle,
 const IALG_MemRec *mem,
SRC_TI_Handle p,
                          SRC_TI_Params const *params)
```

Description:

This function initializes all SRC parameters.
"Handle" is the pointer to the current instance. Using the memRec structure module will set up its internal variables, to the memory blocks allocated for it.
This is called to initialize memory that has been allocated for a new SRC algorithm instance. This is called after dynamically allocating memory for the new instance. The memory to be initialized is described by a filled-in IALG_MemRec array, in the formatted by IALG_algAlloc().

2.2.4 SRC_TI_algFree

Prototype:

```
Int   SRC_TI_algFree(IALG_Handle Handle, IALG_MemRec * mem)
```

Description:

After the end of the processing, this function has to be called to free the memory allocated for an instance of the SRC.

Parameters :

`Handle` : Pointer to an active object instance.

`Mem` : An array of IALG_MemRec structures, indicating the blocks of memory for the instance's storage.

`P` : Pointer to the parent object, in the case when this SRC object is owned by a higher-level DAIS algorithm object. Otherwise NULL

`Params` : Pointer to the Creation Parameters for this instance.

Return Value:

None

2.2.5 SRC_create

Prototype:

```
ISRC_Handle      SRC_create(const ISRC_Fxns *fxns,
                            const SRC_Params *prms)
```

Description:
This dynamically creates an instance of the algorithm.

Parameters:

`fxns` : fxns is a pointer to the v-table for the SRC implementation. This field is initialized by SRC_Obj().

`prms` **:** prms is a pointer to the Creation Parameters for this instance.

2.2.6 SRC_delete

Prototype:

```
Void        SRC_delete(ISRC_Handle     handle)
```

Description:

This deletes the given dynamically created objects referenced by handle. If Handle is NULL, then SRC_delete simply returns.

Parameters:

`Handle` **:** Pointer to an active object instance.

3 EXTERNAL APIS

- `Void SRC_setInput (ISRC_Handle handle, Int *in)`
- `Void SRC_setOutput (ISRC_Handle handle, Int *out)`
- `Void SRC_apply (ISRC_Handle handle)`
- `Bool SRC_isInputBufferEmpty (ISRC_Handle handle)`
- `Bool SRC_isOutputBufferFull (ISRC_Handle handle)`
- `Void SRC_getStatus (ISRC_Handle handle, ISRC_Status *status)`
- `Void SRC_setStatus (ISRC_Handle handle, const ISRC_Status *status)`
- `LgInt SRC_getInputRate(ISRC_Handle handle)`
- `LgInt SRC_getOutputRate(ISRC_Handle handle)`
- `Void SRC_setInputRate(ISRC_Handle handle, LgInt inputRate)`
- `Void SRC_setOutputRate(ISRC_Handle handle, LgInt outputRate)`

3.1 SRC_setInput

Prototype :

```
Void        SRC_setInput(ISRC_Handle     handle, Int *in)
```

Description :

Initialize the input buffer by the parameter which we pass by *in. This function also initializes the inputbufferCounter and inputBufferEmpty by the value 0.

Parameters :

`handle` : Pointer to an active object instance.
`*in` : Value of input sample rate.

3.2 SRC_setOutput

Prototype :

```
Void        SRC_setOutput(ISRC_Handle     handle, Int *out)
```

Description :

Initialized out put buffer by the parameter which we pass by *out. It also initialized the outputBufferCounter and outputBufferFull by the value 0.

Parameters :

`handle` : Pointer to an active object instance.
`*out` : Value of output sample rate.

3.3 SRC_isInputBufferEmpty

Prototype:

```
Bool   SRC_isInputBufferEmpty(ISRC_Handle     handle)
```

Description :

This function checks the whether the inputBufferEmpty or not.

Parameters :
`Handle`: Pointer to an active object instance.

3.4 SRC_isOutputBufferFull

Prototype :
```
Bool    SRC_isOutputBufferFull(ISRC_Handle     handle)
```

Description :
This function checks the outputBufferEmpty or not.

Parameters :
`handle` : Pointer to an active object instance.

3.5 SRC_getStatus

Prototype :
```
Void   SRC_getStatus(ISRC_Handle  handle, ISRC_Status *status)
```

Description :
Returns the current run time parameters of the algorithm.

Parameters :
`handle` : Pointer to an active object instance.
`*status` : Status of active object instance.

3.6 SRC_setStatus

Prototype :
```
Void   SRC_setStatus(ISRC_Handle handle,
                     const ISRC_Status *status)
```

Description :
Set the run time parameters of the algorithm.

Parameters :
`handle` : Pointer to an active object instance.

3.7 SRC_getInputRate

Prototype :
```
LgInt    SRC_getInputRate(ISRC_Handle     handle)
```

Description :
Returns the value of input rate.

Parameters :
`handle` : Pointer to an active object instance.

3.8 SRC_getOutputRate

Prototype :
```
LgInt    SRC_setOutput(ISRC_Handle     handle)
```

Description :
Returns the value of output rate.

Parameters :
`handle` **:** Pointer to an active object instance.

3.9 SRC_setInputRate

Prototype :
```
Void  SRC_setInputRate(ISRC_Handle handle, LgInt inputRate)
```

Description :
Set the input sampling rate value.

Parameters :
`handle` **:** Pointer to an active object instance.
`inputRate` **:** Input Sampling Rate value.

3.10 SRC_setOutputRate

Prototype :
```
Void   SRC_setOutputRate(ISRC_Handle handle, LgInt outputRate)
```

Description :
Set the output sampling rate value.

Parameters :
`handle` **:** Pointer to an active object instance.
`outputRate` **:** Output sampling rate value.

3.11 SRC_apply

Prototype :
```
Void   SRC_apply(ISRC_Handle     handle)
```

Description :
This function does the actual sample rate conversion process. It takes input samples form the input buffer, does the sample rate conversion, puts the output samples in output buffer and sets the output buffer full flag. If the input buffer is empty in the middle of conversion process it sets the input buffer empty flag and quits.
If input and output sampling rate are the same it just does the buffer copy and sets the input buffer empty and output buffer full flag.

Parameters :
`handle` **:** Pointer to an active object instance.

4 SRC OBJECT

4.1 ISRC_Obj Structure

Fields :
ISRC_Fxns *fxns

4.2 ISRC_Fxns

It is a pointer to the v-table for the SRC module.
Description:
This structure defines a generic movie interface object. It has a pointer to v-table and all the data required for movie parsing. But the current implementation has no specific data so it contains only pointer to v-table.

4.3 ISRC_Fxns Struct

Declaration:

```
typedef struct ISRC_Fxns {
IALG_Fxns ialg;
Void (*setInput)    (ISRC_Handle handle, Int *in);
Void (*setOutput)   (ISRC_Handle handle, Int *out);
Void (*apply)       (ISRC_Handle handle);
Void (*getStatus)   (ISRC_Handle, ISRC_Status *);
Void (*setStatus)   (ISRC_Handle, const ISRC_Status*);
Bool (*isInputBufferEmpty)    (ISRC_Handle handle);
Bool (*isOutputBufferFull)    (ISRC_Handle handle);
LgInt     (*getInputRate)     (ISRC_Handle handle);
LgInt     (*getOutputRate)    (ISRC_Handle handle);
Void (*setInputRate)          (ISRC_Handle handle,
                               LgInt inputRate);
Void (*setOutputRate)         (ISRC_Handle handle,
                               LgInt outputRate);

} ISRC_Fxns, SRC_Fxns;
```

Description:
This is the v-table for the SRC object.

4.4 ISRC_Params Structure

Data Fields:

```
Int size;
Int inputBufferSize
Int outputBufferSize
LgInt inputRate
LgInt outputRate
Int RealtimeInput
```

Description:
This structure defines the creation parameters for ISRC instance object. The application fills the structure to describe the desired properties of the instance. The configurable creation parameters included I/O Buffer size, I/O rate and the Real time input.

Appendix A – Document change history

Ver.No.	Editor/Author	Date dd-mmm-yy	Changes made
1.0	Vipul	13th Dec 2002	
2.0	Arnab	26th Dec 2002	Supports some more frequency convertion combinations
3.0	Arnab	27th Jan 2003	Src supports any to any conversion within the standard set (8,11.025,12,16,22,24,32,44,48-KHz)
4.0	Arnab	8th Aug 2003	Code modification to increase precision.
4.1	Srividya M. S.	28th Jan 2004	Modified heading, added more description to external APIs and made software release 3.1

---------- **End Of Document** ----------