Application Report
SPRA875 – December 2002

Vocoders Integrated to RF3

Sunil J B, Binesh B — *TI Catalog Applications Team*

ABSTRACT

This application report explains how TMS320C64x™ optimized vocoders are integrated to Reference Framework Level 3 (RF3) on TEB6416. Reference Framework for eXpressDSP™ Software is a startware for developing applications that use DSP/BIOS™ and the TMS320™ DSP Algorithm Standard.

This application report is intended for developers who wish to either extend the functionalities of the existing application or use RF3 to port their applications easily. It describes the changes made within RF3 to suit the application, and how to introduce more vocoders to the existing code with minimal effort.

This guide will not provide all the details required in porting the RF3 code onto different hardware though it is likely that the modifications required will be in tune with those mentioned in this application report.

Note: A part of the source code for this application is IP protected and will not be available to external users. However, the necessary files for using the application executable is freely accessible.

Contents

TMS320C64x, eXpressDSP, DSP/BIOS, and TMS320 are trademarks of Texas Instruments.

Trademarks are the property of their respective owners.

List of Figures

List of Tables

1 Introduction

This application report is an attempt to capture the knowledge gained while integrating various wireless vocoders to the Reference Framework 3 (RF3) offered by Texas Instruments. The vocoders are very good examples to demonstrate the features of RF3 because RF3 is targeted at applications with the following properties:

- Multichannel (1-10 channels)
- Multi-algorithms (1-10 algorithms)
- XDAIS algorithms
- Implements control functionality
- Supports multiple execution rates and priorities

This application therefore makes use of RF3 v1.2 source code on TEB6416. All the speech vocoders are integrated in this one single application, and run in real time on TEB6416. RF3 V1.2, running on TEB6416, greatly simiplifies this task. The required source files in RF3 are modified or removed, to adapt to the applications requirement. Also, some new files are added in the application. All these changes are shown in Table 1 in section 3, *Changes at a Glance*.

To allow you to run any vocoder with any of the supported sampling rates, the general extension language (GEL) file is modified. This gives faster control to you for running any of the available vocoders with the desired sampling rate, and also demonstrates the usage of GEL file features to create a simple menu to pass control parameters to an application in real time.

1.1 List of Vocoders Integrated

The list of vocoders integrated include:

- GSM HR (Half Rate)
- GSM FR (Full Rate)
- GSM EFR (Enhanced Full Rate)

- TIA EVRC (Enhanced Variable Rate Codec)
- GSM NBAMR (Narrow Band Adaptive Multi Rate)
- GSM WBAMR (Wide Band Adaptive Multi Rate)

1.2 Software Components Provided

Software components provided include:

- RF3 v1.2 source code
- Additional eXpressDSP-compliant multi-rate filter code
- Library files of the vocoders integrated viz. GSM HR, FR, EFR, NBAMR, WBAMR and the TIA EVRC vocoder

2 Directory structure

The directory structure of the application, as shown in Figure 1, is as follows:

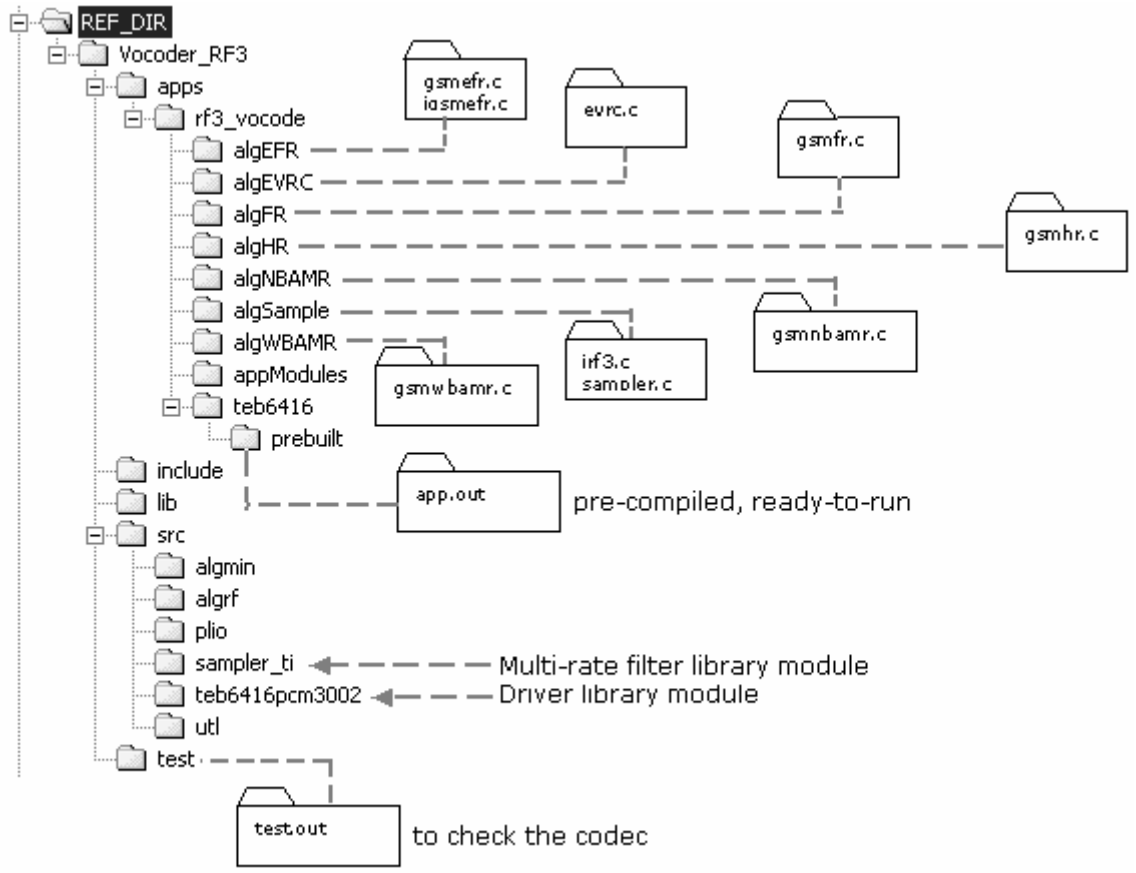


Figure 1. Directory Structure of the Application

<REF_DIR> is the directory where the application has been downloaded.

The remaining files, which have not been described, are the same as those in the RF3 structure. The <REF_DIR>\Vocode_RF3\apps\rf3_vocode\teb6416 directory has the main project file, viz. app.pjt, and related files.

The XDAIS obj/lib files of the vocoders are present in the lib folder.

<REF_DIR> also contains your guide document and the GEL file to be used on start-up, TEB_6416.gel. The details of the installation process are given in the *TMS320C6416 Test Evaluation Board (TEB) User's Guide* (SPRU561).

3 Changes at a Glance

The following are the changes made in files included in project **app.pjt.** The RF3 v1.2 source code is used, and also modified, wherever needed. See Table 1.

Table 1. List of Changes in RF3 Source

Function Name	File Name	Modification Made
Data/arrays	appThreads.c	Buffer sizes changed, added 'io_control' structure for GEL file usage
–	evrc.c	Added
–	gsmhr.c	Added
–	gsmfr.c	Added
–	gsmefr.c	Added
–	gsmnbamr.c	Added
–	gsmwbamr.c	Added
–	irf3.c	Added
–	sampler.c	Added
–	ifir.c	Removed
–	fir.c	Removed
–	ivol.c	Removed
–	vol.c	Removed
thrAudioprocInit	thrAudioproc.c	Modified to incorporate algorithm initializations
thrAudioprocRun	thrAudioproc.c	Modified to access the various algorithms
thrAudioprocSetOutputVolume	thrAudioproc.c	Deleted
thrControlIsr	thrControl.c	Modified the control structure
thrControlInit	thrControl.c	Modified to initialize new control structure
thrControlRun	thrControl.c	Deleted
thrRxSplitRun	thrRxSplit.c	Modified writer size and frame size
thrTxJoinRun	thrTxJoin.c	Modified writer size and frame size
–	app.cdb	Modified the pipe sizes, memory settings to board-specific deleted swiControl
–	thrAudioproc.h	Added new structure on vocoder details, new constants for the application
–	thrControl.h	Added data for conformity with control
–	appResources.h	Modified framesize constants

The file, app.gel, has also been modified totally in order to support the new control structure.

4 Development Details

The following sections describe how the vocoder application has been integrated on the RF3 framework. For details on the RF3 framework, see *Reference Frameworks for eXpressDSP Software: RF3, A Flexible Multichannel, Multi-Algorithm Static System* (SPRA793).

4.1 Application Requirements

A block diagram of the system is shown in Figure 2:

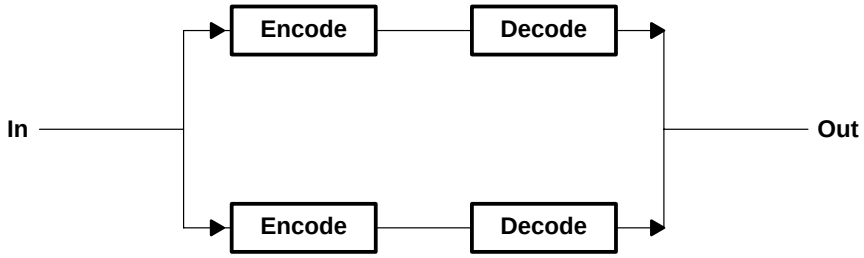


Figure 2. Basic Block Diagram

The basic flow of the application is as follows:

The input data is first passed through an encoder and then through a decoder in tandem. The idea behind this kind of data flow is that such a setup will enable you to evaluate the level of distortion that occurs, which is due to the algorithms used while coding and *not* due to any other kind of physical or informational disturbance (as in the real world).

It can be noted that the vocoders integrated operate at different data rates (essentially data sampled at different rates). Hence, there is a requirement to vary the sampling rate depending on the vocoder that is presently in use.

All vocoders require 20-ms frames, either at 8k Hz or at 16k Hz (presently only WBAMR). The PCM3002 codec on the TEB however, supplies data at 48k Hz; hence, a multi-rate filter has also been integrated to support this conversion.

4.2 Data Path

A more accurate diagram of the data path involved is illustrated in Figure 3. This figure gives the exact details of how the control thread may be invoked to “connect” the output of the multi-rate filter to any encoder. This flexibility allows you to run the vocoders at rates for which they are not designed. For example, GSM HR, which expects a data sampled at 8 Khz, may be “forced” to process data at 16 kHz or 48 kHz. As the algorithm is fast enough, no real-time misses are noticed, and you indeed have the vocoder processing data at higher rates. This provides an interesting observation on how the various coding algorithms would handle data other than speech. One exception to this rule is that GSM WBAMR will not run satisfactorily at 8 kHz, due to insufficient data in the pipe. The reason is evident, considering the pipe structure and the flow of data.

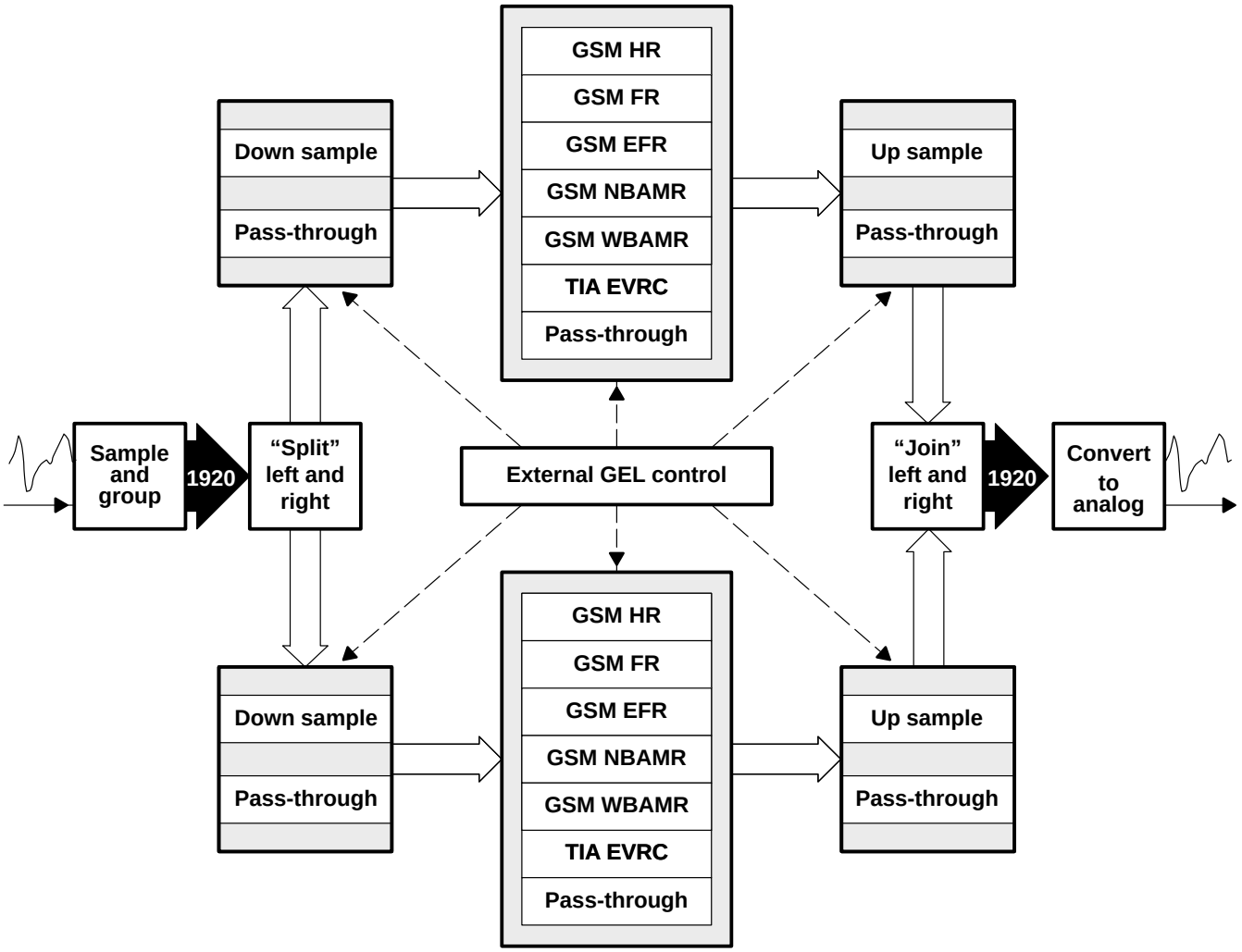


Figure 3. Data Paths With Blocks

The external GEL control logic permits you to select various vocoders and sampling rates. The arrow sizes give an idea of the amount of data transferred. In this case, since the vocoders require a 20-ms frame, and the PCM3002 being a stereo codec hardwired to operate at 48 kHz, it follows that the output of "Sample and Group" block and input to the "Convert to Analog" block is 1920 samples (which here is equivalent to a 16-bit integer) each, and the rest of the data path (arrow) size is 960 samples each, for either the Left or the Right channel.

The above data path is very similar to the default data path provided as a part of RF3 (with the FIR and VOL algorithms in the place of the vocoders and the multi-rate filters). Hence the thread structure has not been altered in this application. For a detailed description on how the threads were allocated, refer to the *Reference Frameworks for eXpressDSP Software: RF3, A Flexible, Multichannel, Multi-Algorithm, Static System* (SPRA793).

4.3 Changes in the Control Structure

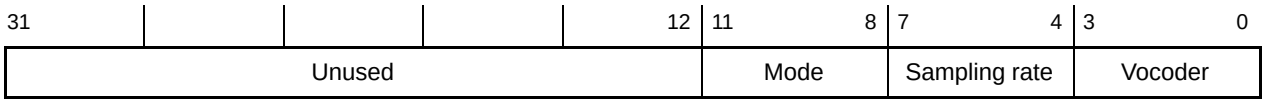
The logic involved in all control activities has been largely modified. The control is achieved in the following manner: you select an option (sampling rate/vocoder) using the GEL script, which writes the data to a particular location. The CLK object, clkControl, invokes the thrControlIsr. The polling time has been changed to 40ms considering that each frame in itself here is 20ms. The integrated switching regulator (ISR) simply reads the memory location and copies your settings onto a global copy available to the processing threads. Since there is no role or requirement of swiControl in the entire process, it has been deleted.

It was observed that an attempt to write to any C variable directly, using the GEL script, was prone to lots of problem probably due to bus conflicts. Hence, it was decided to use a memory location that was not used by the C code for any other purpose, for all communication, from the GEL script to the C code.

The locations **0x01a00750** and **0x01a00754** were experimentally selected for this purpose. These locations correspond to the **reload/link parameters for EDMA event N**.In this case, such an event is unlikely; therefore, you can safely override the usage of these locations.

The convention used for the communication is as follows:

- 0x01a00750: Sets its least-significant bit (LSB) to 1 or 0, depending on whether the GEL file has modified data or not. One indicates that you have set new values, while zero indicates that the ISR has not yet copied the values to the C variables.
- 0x01a00754: This location contains packed data pertaining to your selection of mode, sampling rate, and vocoder in the following format:



“Sampling rate” and “vocoder” fields are essentially numbers, separately defined and declared in both the GEL file and the C code with the **same** values, in the sense that if the GEL script sets the vocoder field to 1 when you select enhanced full rate (EFR), then the C code also *must* have all properties (framesize and ‘mask’) of the EFR at index 1.

Additionally, a two-level handshaking has been introduced to prevent any undesirable change of options while the thread is executing.

The first level involves GEL to thrControlIsr and back. This is achieved through the above mentioned location 0x01a00750 data. When you select the vocoder/sampling rate, the GEL sets the LSB to 1. If you attempt to modify the choice again, even while the bit is set, the new choices will overwrite the previous data. This is, however, unlikely because the choices *will* be read every 40 ms. When thrControlIsr reads the data and copies it onto C variables, it also resets this bit. In short, this handshaking ensures that the variables are copied only if the GEL has made any modification (i.,e., you have changed his selection).

The second level of handshaking exists at the processing thread, which creates a local copy of the data that thrControlIsr supplies it before processing. This again is to prevent any change in the properties in the middle of processing, which could cause undesirable program execution/ termination. A separate variable indicates whether the local copy has actually read the new values, failing which, thrControlIsr will not accept the new user settings (set through the GEL).

4.4 File-Specific Modifications and Additions

This section describes in detail all the changes made in modifying the RF3 code to suit the application.

4.4.1 thrAudioprc.h

To integrate vocoders easily, a structure has been introduced which encapsulates all relevant details of a vocoder. The structure definition, as found in this header file, is:

```
typedef struct Vocoder
{
                int in_use;
                int mode;
                int sampling_rate;
                int mask[NUM_VOCODERS+1];
                int framesize[NUM_VOCODERS+1];
                int update_complete;
} Vocoder;
```

There are only two instances of this structure throughout. One is a global instance, which acts as an interface between the GEL file input and the executing C code. This is to facilitate the two levels of handshaking described in section 4.3.

The description of the fields in the above structure is shown in Table 2.

Table 2. Field Descriptions

Field Name	Description
in_use	Specifies the *number* corresponding to the vocoder selected. As a rule, throughout the code, vocoders are “referred” through their *number*, which is an enumerated integer defined in this header file itself. In the present code, the “legal list” of vocoder *numbers* is as follows: • none 0 • GSM_EFR 1 • GSM_WBAMR 2 • GSM_FR 3 • GSM_HR 4 • GSM_AMR 5 • TIA_EVRC 6 The GEL script sets this field to any of the above numbers. The C code then decides which vocoder to use, depending on value.
mode	This field is applicable to only NBAMR and WBAMR. It has been included in the structure for integrity of data. The legal values are 0–8 (both inclusive). The GEL script ignores any value out of bounds. Presently, NBAMR mode 8 is **not** supported.
sampling_rate	Specifies the *number* corresponding to the sampling rate selected. As a rule, throughout the code, sampling rate is “referred” through the *number,* which is an enumerated integer defined in this header file itself. In the present code, the following is the “legal list” of sampling rate *numbers*. • NO_CHANGE 0 • TO_8kHz 1 • TO_16kHz 2 The GEL script sets this field to any of the above numbers. The C code then decides by what factor to down-sample/up-sample, depending on value.

Table 2. Field Descriptions (Continued)

Field Name	Description
mask	This field decides the “mask” data. The PCM3002 codec has been configured for 16-bit left-aligned ADC output. However, different vocoders require different bit precision data input, e.g., WBAMR expects 14-bit left aligned data, while HR, FR, EFR, NBAMR all require 13-bit left-aligned data, and EVRC expects 16-bit left-aligned data. Hence, the input data is ‘masked’ by forcing the required number of LSBs to zero. mask is an array of integers, which bears a one-to-one mapping with the vocoder *number*, e.g., mask[GSM_EFR] or equivalently mask[1] stores the mask of EFR, a value of 0xFFF8.
framesize	This array defines the size of a frame (in short-words) required by the vocoder. WBAMR expects 320, while the rest require 160 (all equivalent of 20-ms frames at the corresponding sampling rate). Again, as for mask, there is a one-to-one correspondence between this array and the vocoder *number*. So, framesize[GSM_EFR] has a value 160. The default framesize that exists is 960, i.e., 20ms at 48 kHz. This figure corresponds to the size of the input array prior to down-sampling an after up-sampling.
update_complete	This field is a part of the second level handshaking between the C code and the control thread, as described in section 4.3. Once the parameters have been read by thrAudioproc.c during execution, this bit is reset, indicating that the GEL values will now be written on to the structure variable.

4.4.2 thrAudioproc.c

Most features of this file have been retained with minimal modifications. The volume settings are done away with, so all volume control must be external and decided by the volume controls using the PC’s sound control and the speaker/headphone’s volume control.

The type of the *src* and *dst* buffers has been changed to int. This is because, the multichannel buffered serial port (MCBSP) and extended direct-memory access (EDMA) have been configured for 32-bit element-size. See *Writing DSP/BIOS Device Drivers for Block I/O* (SPRA802) for a more detailed explanation.

A local instance of the struct-type Vocoder has been created to maintain a “frame-local” copy of the vocoder and sampling-rate specifications. This is required, as the GEL file might change the options in the middle of the thread execution (swiControl has been removed). Hence, the options are polled before processing each frame, and that copy is used for the entire frame.

The format function is responsible for interpreting the input data in the right manner. It does the following: right-shift by 16, and store as short after masking. This is required as out of the 32 bits supplied by the codec, the 16 most-significant bits (MSBs) are meaningful data while the 16 LSBs are all zeroes (the MCBSP fills in the zeroes).

The up-down sampler has also been introduced for sampling rate conversions. The data requires some formatting because you have to pass on to the sampler the desired output sampling rate and the mask to be applied. Section 4.4.3 covers more details on the sampler used.

To run the vocoders at higher sampling rates, the vocoders are called iteratively until the entire frame is processed. One exception is WBAMR at 8 kHz, which gets a half-zero frame to process.

4.4.3 sampler.c

This is the XDAIS interface to the multi-rate filter code. This also houses the ‘format’ function (described above). The interface allows you to access the down-sampler as well as the up-sampler, which change the sampling rate and limits the data precision, using the mask passed to them. Currently, the code supports down/up sampling to/from 8k Hz and 16 kHz. It uses a simple 48-point finite impulse response (FIR) filter before/after decimation/interpolation, to prevent aliasing. The 16k-Hz up/down sampler uses a Kaiser window with $\beta = 4$.

The down-sampler accepts a 20-ms 48k-Hz frame (960 shorts), and uses the 961st and 962nd element to specify the decimation factor and mask respectively:

- in[960] = desired decimation factor. This variable is internally defined to the following settings:
  - $0 \rightarrow 6$, i.e., will down-sample to 8k Hz
  - $1 \rightarrow 3$, i.e., will down-sample to 16k Hz
- in[961] = mask to be used

The up-sampler accepts a 20-ms. 8- or 16-kHz frame, i.e., 160 or 320 shorts, and uses the 961st and 962nd element to specify the interpolation factor and mask, respectively:

- in[960] = desired interpolation factor. This variable is internally defined to the following settings:
  - $0 \rightarrow 6$, i.e., will up-sample from 8k Hz
  - $1 \rightarrow 3$, i.e., will up sample from 16k Hz
- in[961] = mask to be used

4.4.4 thrControl.c

This file has been modified to incorporate all the required changes in the control features detailed in section 4.3. Additionally, thrControlInit function was enlarged to include the default settings of the application when loaded, and also to set vocoder specific details at initialization.

The thrControlIsr routine has an additional functionality through which, when the vocoder is first selected, the sampling rate is automatically set to the nominal rate. However, by selecting the sampling rate after selecting the vocoder, it forces the vocoder to run at the new sampling rate.

Also, the polling rate for GEL changes has been increased to 40 ms, and thrControlRun has been removedn along with swiControl.

4.4.5 thrControl.h

This header has been modified to incorporate changes as required by thrControl.c.

4.4.6 gsm*.c and evrc.c

These files contain the user-friendly interface between the processing thread and the respective vocoder code. The files use the RF3 API (algrf.h) to create and activate objects. The XDAIS partial-linked object files have been included in the project, which contain the implementation of the referenced functions.

5 Introducing a New Vocoder

To introduce a new vocoder (VOC), first obtain a partial-linked object file of the XDAIS code (called Xdais_VOC64.obj). The idea behind using a partial-linked file is that it will not clash with any other symbol in the existing application.

Once done, follow these steps:

1. Create a new folder, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\algVOC.
2. Open the file, VOC_ti.h, and make the following modifications:
   - Ensure that the create, delete, init and exit functions are defined, and the VOC_TI_Params variable is declared.
   - Change the declaration of create to VOC_TI_create (IVOC_Fxns *fxns, VOC_TI_Params *params).
   - Declare the functions VOC_TI_encode_apply(IVOC_Handle, short *, short *) and VOC_TI_decode_apply(IVOC_Handle, short *, short *).
3. Now, copy VOC_ti.h to <REF_DIR>\Vocoder_RF3\include.
4. Create a file, VOC.c, in <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\algVOC. Modify the contents of VOC.c by comparing with <REF_DIR>\Vocoder_RF3\apps\ rf3_vocode\algFR\gsmfr.c.
5. Modify the functions VOC_TI_encode_apply and VOC_TI_decode_apply, to suit the new vocoder. Compare with with the functions in <REF_DIR>\Vocoder_RF3\ apps\rf3_vocode\algFR\gsmfr.c.
6. Open Code Composer Studio™, and then open <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\teb6416\app.pjt by clicking on Project → Open. Then select <REF_DIR>\Vocoder_RF3\apps\ rf3_vocode\teb6416\app.pjt
7. Add VOC.c to the project by first clicking on Project → Add Files to Project..., and select <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\algVOC\VOC.c.
8. Open thrAudioproc.h (present in the list of header files included in app.pjt), and do the following:
   - Increment the value of the constant NUM_VOCODERS.
   - Add the line, #define VOC 7, after the line, #define TIA_EVRC 6.
   - Add the line, #define VOC_MASK 0xNNNN, to define the precision of the data input required (the number of ones in 0xNNNN will be the bit precision).
   - Add the framesize as #define VOC_FRAMESIZE MMM, where MMM gives the size of the frame in short integers, as required by the VOC algorithm.
   - Add the handle, VOC_Handle algVOC in struct thrAudioproc.
   - Add the declaration extern, far IVOC_Fxns VOC_TI_IVOC, where VOC_TI_IVOC refers to the symbol which has been declared ‘global’ while partial linking.
   - Select File → Save to save the changes.

Code Composer Studio is a trademark of Texas Instruments.

9. Open the file, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\appModules\ thrAudioproc.c (also present in app.pjt), and modify the following:

   - Initialize algVOC to NULL in the definition of the thrAudioproc array (for both the channels).

   In the thrAudioprocInit() function, do the following:

   - Add the declaration, VOC_Params vocparams.
   - Initialize the algVOC variable, using VOC_TI_create(…), in exactly the same way as is done for the other vocoders.

   In the thrAudioprocRun function, make the following additions:

   - Within the first switch context, add case VOC: and the for loop as done for the preceding vocoders replacing, e.g,, GSMFR by VOC.
   - Repeat the same in the second switch context.
   - Select File → Save to save the changes

10. Open the file, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\appModules\ thrControl.c (also present in app.pjt), and modify the following:

    In thrControlIsr, add the following:

    - In the body of the branch if(last_vocode!=vocoder.in_use), add the following additional else if branch:

```
else if(vocoder.in_use == VOC)
{
        vocoder.sampling_rate = TO_<m>kHz;
        < copy the second line as it is from any other similar else~if
        construct and modify the TO_8kHz or TO_16kHz to TO_<m>kHz >
}
```

      Here, *m* represents the nominal sampling rate of the vocoder in kHz. Be sure that the filter supports this sampling rate. If not, see section 6 for a description of how to add different rates.

    In thrControlInit, add the following statements:

    vocoder.mask[VOC] = VOC_MASK;

    vocoder.framesize[VOC] = VOC_FRAMESIZE;

11. All changes to *run* the code are now complete. To change the GEL control part, do the following:

    - Open the file <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\apps\app.gel.
    - Add the statement, #include VOCODER_VOC 7, after the line, #define VOCODER_TIA_EVRC 6.

      If no input is needed from you, use 1), else use 2).

    - Duplicate the hotmenu code of GSM_FR, and replace every occurance of GSM_FR by VOC in that segment. Also, replace 8 by <m>, which is the nominal frequency in kHz at which the vocoder operates.

- Duplicate the dialog code of GSM_WBAMR and replace:

  ```
  GSM_WBAMR by VOC
  16 by <m> where m is the nominal sampling rate
  mode by <var> where <var> is the name of the parameter to be input by the user.
  0 by the minimum legal bound of <var>
  8 by the maximum legal bound of <var>
  ```

  <var> input is written at the mode location. So vocoder.mode in the C code will access <var>.

The required modifications are now complete, and the code should now be able to run the new vocoder. The user options will also be reflected through the GEL script.

6 Adding Different Sampling Rate

This section describes how to modify the multi-rate filter code. This may be required in order to support algorithms, which expect data at rates other than those currently supported (8 kHz, 16 kHz and 48 kHz). Adapting the filter to down-sample to frequencies that are not integral factors of 48 will involve a more complex change in data usage (delayLine parameter), and are not discussed here.

To use the code that is present in this application, the filter *must* be a 48-point FIR filter. This code supports only integral up/down sampling, i.e., the down sampled frequency must be an integral factor of 48, e.g., 48/*n* kHz, where *n* is an integer greater than 1. Filter coefficients can be calculated through MATLAB (fir1 is the filter used in the 16-kHz down sampling). An important point to note is that, for a desired down-sampled data rate of 48/*n* kHz, the cutoff frequency used in the FIR filter can have a maximum value of 24/*n* kHz (i.e., signal is band-limited to utmost half of the down-sampled rate, to avoid aliasing). MATLAB supplies the coefficients in floating-point format, which can be easily converted to 16Q16 format by multiplying each number by 2^{16} (=65536). The array of 48 coefficients may now be ported into the sampler code.

Follow these steps to plug in the new filter parameters:

1. Open Code Composer Studio, and then open the project file <REF_DIR>\Vocoder_RF3\src\sampler_ti\ sampler_ti64.pjt by selecting Project → Open, and select the file.
2. Open the file, <REF_DIR>\Vocoder_RF3\src\sampler_ti\state.c, which is already included in the project, and modify the following:
   - In the array, decFactor, increment the size and add *n* as the last element.
   - In the array, intFactor, increment the size and add *n* as the last element.
   - In the array, numTaps, increment the size and add 48/*n* as the last element.
   - In the array, coeff_ptr, increment the number of rows and add the array of 48 coefficients as the last row of the initialization.
3. Save the files modified by selecting File → Save All and then build the project using Project → Rebuild All. Once done, close the project.
4. Open the project file, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\teb6416\app.pjt, by selecting Project → Open, and select the file.

5. Open the file, thrAudioproc.h, which is present in the project, and add the statement, #define TO_<n>kHz 3, after the line, #define TO_16kHz 2.
6. Open the file, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\teb6416\app.gel, and make the following additions:
   - Add the statement, #define SAMPLING_RATE_TO_<n>kHz (3 << SAMPLE_OFFSET), after the line, #define SAMPLING_RATE_TO_16kHz (2 << SAMPLE_OFFSET).
   - Copy the hotmenu code of Sample_at_8kHz, and replace all occurrences of 8 by <n> in this section.
7. Open the file, <REF_DIR>\Vocoder_RF3\apps\rf3_vocode\appModules\thrAudioproc.c, and modify the following:
   - In the array, DEC_FACTOR, increment the size and add *n* as the last element.
8. Save the files using File → Save All. and rebuild the project using Project → Rebuild All.

All changes required have now been incorporated. and the application should now be able to support the new filter.

The up/down sampler operates by simply limiting the bandwidth of the signal before/after decimation/interpolation. While up-sampling as well as down-sampling, the same filter coefficients are applicable, considering the operation of the samplers.

The down-sampling is achieved by first band limiting the signal to B kHz, and then selecting every nth sample. The up-sampling is achieved by introducing n–1 zeroes between consecutive sampling, and then band limiting the output to B kHz again. Because the input is band-limited to B kHz, all signal frequencies greater than B kHz will correspond to out-of-band noise caused by the insertion of zeroes, while up-sampling. Hence, the same FIR coefficients may be used both, while up- as well as down-sampling.

7 Test Program Specification

The file <REF_DIR>\Vocoder_RF3\test\test.out may be used to check board specifications and whether or not the codec is functioning. The test.out file checks *only* the PCM3002 codec and *not* any other peripheral device. The MCBSP and EDMA are *not* used and data is just looped back internally through the codec. The codec specifications in the test program is:

DAC: 20-Bit, MSB-First, Left-Justified

ADC: 20-Bit, MSB-First, Left-Justified

DAC left-channel attenuation: 0dB

DAC right-channel attenuation: 0dB

ADC power-down mode: disabled

DAC power-down mode: disabled

ADC high-pass filter: enabled

DAC Infinite zero detection: disabled

DAC de-emphasis: disabled

Soft Mute: disabled

ADC to DAC loop back: enabled

LRCIN =1 (implies that left-channel is “H” and right channel is “L”)

As loop back is enabled, the ADC’s digital output goes straight to the DAC’s input. So ideally, the stereo input will emerge at the output end.

8 References

1. *TMS320C6416 Test Evaluation Board (TEB) Quick Start User’s Guide* (SPRU561).
2. *Reference Frameworks for eXpressDSP Software: RF3, A Flexible, Multichannel, Multi-Algorithm, Static System* (SPRA793).
3. *Writing DSP/BIOS Device Drivers for Block I/O* (SPRA802).
4. *PCM3002/PCM3003 16-/20-Bit Single-Ended Analog Input/Output STEREO AUDIO CODECs* (SBAS079).

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2002, Texas Instruments Incorporated