TEXAS INSTRUMENTS

GSM Full-Rate Voice Coding on the TMS320C62xx DSP

Application Report

September 1999

Wireless ASP Products

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Contents

List of Figures

List of Tables

List of Examples

GSM Full-Rate Voice Coding on the TMS320C62xx DSP

ABSTRACT

This document describes the GSM full-rate voice coding implementation on the TMS320C62xx processor (also called the vocoder). It discusses the system requirements and is designed for programmers who want to use the code in a basic GSM architecture.

1 Introduction

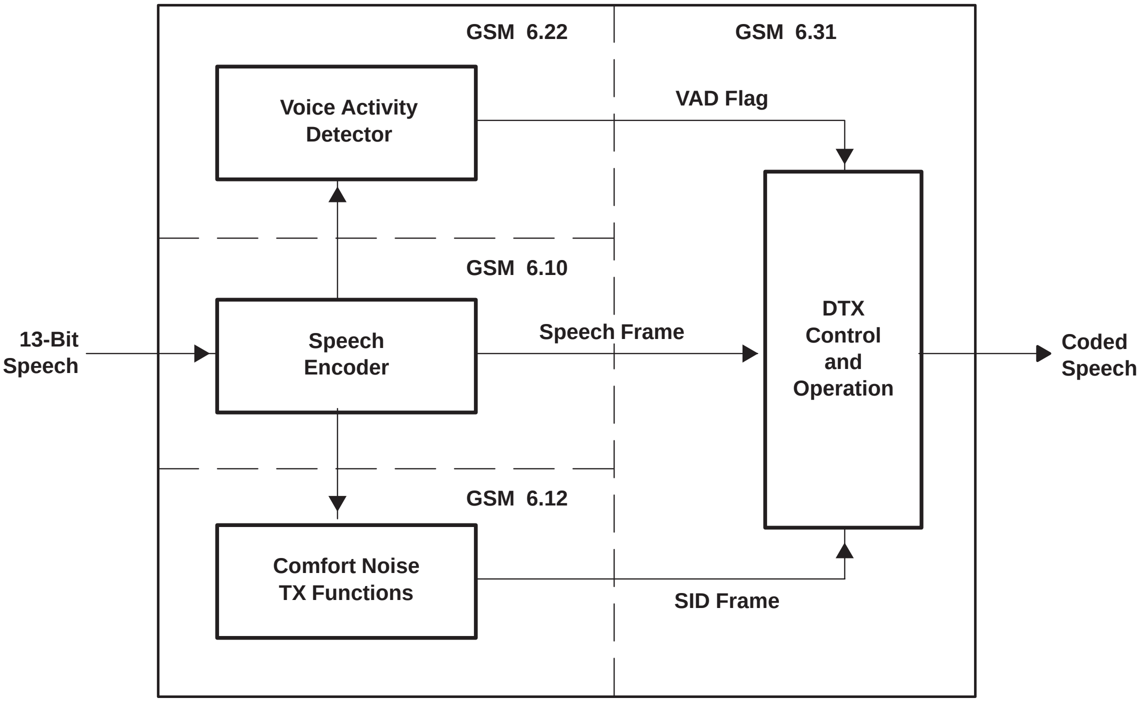
This vocoder is available as optimized C code. Of the three Global System for Mobile (GSM) voice coders, full-rate (FR), half-rate (HR), and enhanced-full-rate (EFR), the full-rate vocoder is the least CPU intensive. Even in C code, this vocoder is more efficient than the hand-optimized versions of the other two GSM voice algorithms.

This document does not describe the GSM full-rate voice-coding algorithm. For more detailed information on the algorithm, see the relevant European Telecommunications Standards Institute (ETSI) documents ETSI 300 580-(1…6).

1.1 Implementation Functions

This implementation includes all of the full-rate voice coding functions for the GSM full-rate voice coder and decoder and supports these GSM specifications: 6.01, 6.10, 6.12, 6.31, and 6.32. Bit-level precision is not specified for receiving voice activity detection (VAD) or bad frame indication (BFI), and this function is not included. VAD and discontinuous transmission (DTX) are specified at bit level for transmit and are included. Figure 1 and Figure 2 illustrate the voice coder and decoder, respectively, and show how these modules interact to provide the voice coding and decoding functions.

The voice code is based on the pseudo-Pascal language in GSM specification 6.10 that is optimized for the TMS320C62xx processor and has been verified with the tests specified.



NOTE: SID = Speech identification

Figure 1. GSM Full-Rate Speech Coder

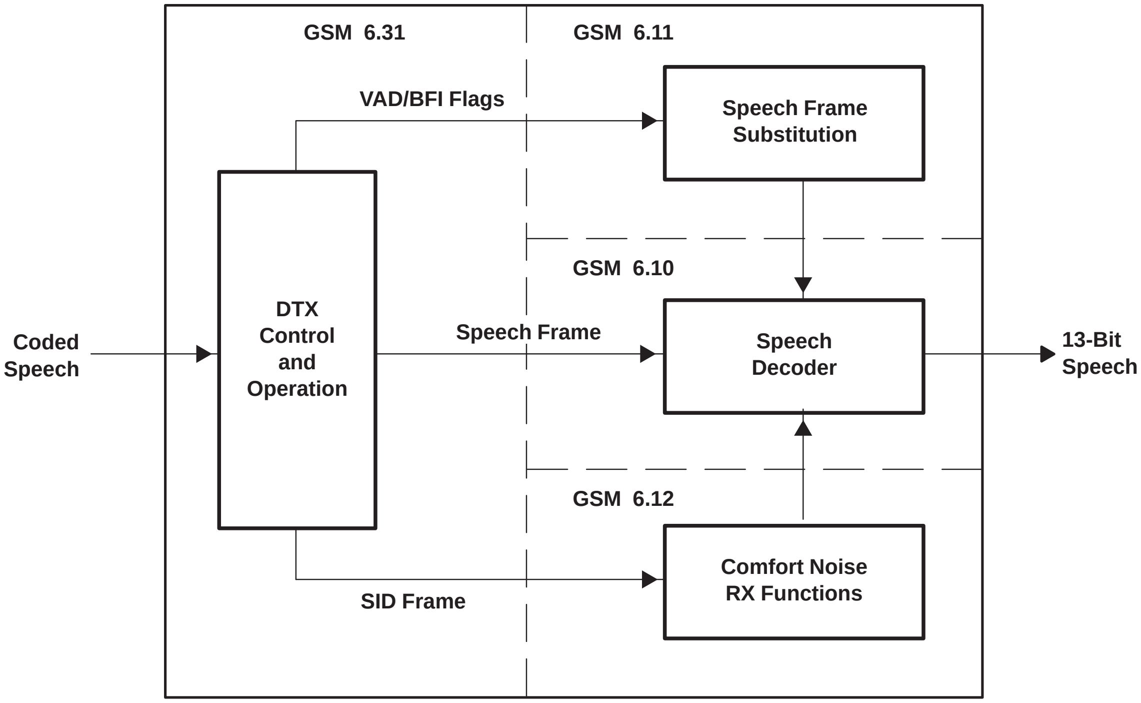


Figure 2. GSM Full-Rate Speech Decoder

1.2 Typical Implementation

A typical implementation in a GSM base station is shown in Figure 3.

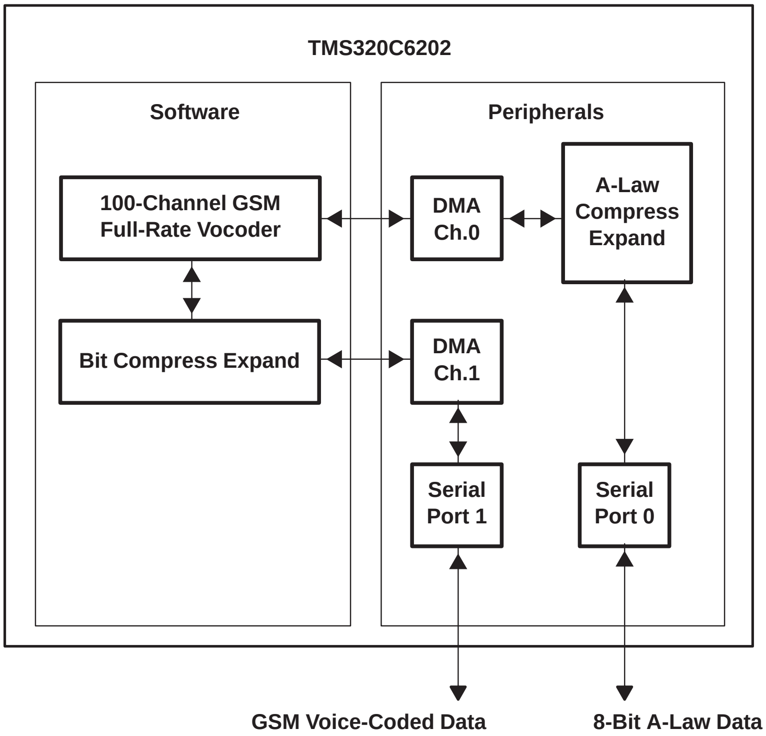


Figure 3. Typical GSM Base Station Implementation

The TMS320C6202 processor performs all of the vocoding (coding + decoding) for 100 channels of GSM at full rate. Alternatively, it may be grouped with other GSM voice coders to produce a multistandard base station. The frame format for inputting and outputting GSM- and PCM-coded (pulse code modulation) data depends on the application and is not covered in this document. Further information on typical applications using the serial and direct memory access (DMA) ports is covered in the *TMS320C6201/6701 Peripherals Reference Guide*.

2 System Requirements

C code from all the C modules must be compiled with the –mt –mh –o3 options. In addition, the correct compiler option must be used to force the compiler to do data page pointer (DP) relative addressing for array variables, as shown here:

- Compiler 2.1 –mt –mh –o3
- Compiler 3.0 –mtw –mh2147483647 –mr1 –o3

Some previous versions of the C compiler do not support some of the functions used by this code and are therefore non-functional. The code has not been tested with versions above 3.0.

2.1 Memory Allocation

Memory is allocated at link time for a single channel of vocoding (coding + decoding). Additional memory can be allocated on the heap for additional channels by calling the following routine:

```
int** address[channels] FR_Assign_GSM(channels)
```

Where:

The input parameter is the total number of channels required and it returns a pointer to an array of channel context addresses.

These addresses may be used by the Set_Channel (address) routine to switch context to that channel before calling/resetting or changing any of its variables. The first channel allocated in this array is the one linked into the .bss section. If only one channel is required, this routine need not be called, and all calls to Set_Channel (address) and Return_Channel () can be removed.

If there is insufficient heap memory for either the array of pointers or the channel data space, then some or all of the pointers will be NULL. You must ensure that there is sufficient heap space at compile time or check the values returned for error conditions. The example code allows enough heap space for the CIO and 16 channels of GSM at full rate.

2.2 Channel Switching and Interrupts

Channel switching for special operations is done by calling the routines Set_Channel (address) and Return_Channel (). Channel switching is handled automatically by the main coder and decoder routines. It is, however, necessary to handle cache enabling/freezing and interrupt disabling/enabling around the coder and decoder routines. As context is switched via the DP register, interrupts must be disabled because the DP register may point to a memory block other than that expected by a C interrupt service routine. All hand-coded assembly routines assume that interrupts are already disabled. The Handle_Ints routine can be added anywhere into the C code to provide an interrupt window. (This adds a few cycles each time that it is called.)

2.3 Execution Time Benchmarks

The execution time benchmarks were obtained with Compiler 2.1/Simulator 2.0 and are measured from the label FR_SpeechEncoder1 or FR_SpeechDecoder1 to the return address from the last subpart as shown in Table 1. The cache is enabled during speech vocoding, but is disabled during file I/O. These benchmarks do not include the CIO file input or output because in a real system, the serial ports working in conjunction with the DMA channels would probably handle this. The cycle count would be much less than it would be if CIO had been used.

Table 1. Execution Times

Processor	Channels	MHz
TMS320C6202 C only	1	2.30
TMS320C6202 C only	100	230

2.4 Program Memory Requirements

The GSM full-rate program is 26624 bytes. In addition, 17248 bytes are used by the example C code that reads and writes test vectors via the simulator CIO functions. Further optimization of the remaining C code is possible, but is not currently planned. This code is fully re-entrant and no increase in code size for the algorithm itself is required, although the control program is more complicated. The program fits inside the memory space of the TMS320C6202 processor and requires no external program memory.

2.5 Data Memory Requirements

The space required by the data memory constants is independent of the number of channels implemented, whereas the space required for channel variables is a multiple of the number of channels required (see Table 2). Other areas are based on the example code shipped with the algorithm, and are implementation-dependent. A 100-channel vocoder fits into the internal data memory of a TMS320C6202 processor.

Table 2. Data Memory Requirements

Function	Size (Bytes)
Stack	8192
Constants	1118
Channel variables	1624*n
CIO variables (from example code)	3800

3 Top-Level Routines

The top-level routines that you need to control the GSM full-rate voice coder/decoder using the GSM full-rate code are discussed in this section.

3.1 main

The C program main is used to apply the test patterns via the simulator to the GSM full-rate voice coder. It can be used as an example to generate real application code or to verify the code. It can be found the in the gsm.c file. This particular code is for a four-channel voice coder/decoder, which automatically runs all the test vector files on a limited number of channels. The number of channels is determined from #define ChMax near the beginning of the file. I/O-specific routines for this form of I/O are in the host.c file. Applications that do not use file I/O can safely remove the host.c file from that application because this code is not called elsewhere. All code not in the gsm.c or host.c file is part of the standard and must to be compiled and linked for all applications.

3.2 FR_Assign_GSM

This subroutine defines and assigns memory for the voice coders and decoders. If more than one channel is being used, then the following subroutine must be called before any other GSM full-rate routine:

```
int**= FR_Assign_GSM(Channels);
```

Where:

Channels is the number of channels required and int** is of type int* *ChannelArray.

On return, ChannelArray contains an array of address pointers the same size as that requested. In single-channel applications, this call can be skipped. The code can be found in the fr_ram.c file. When a NULL pointer is returned in any of the pointers, then insufficient heap memory is available.

3.3 FR_SpeechEncoder1...5

The following subroutines handle the encoding of a speech frame to the GSM full-rate standard:

```
FR_SpeechEncoder1 (*Channel, *Input, *Output[0]);
FR_SpeechEncoder2 (*Channel, *Input, *Output[8]);
FR_SpeechEncoder3 (*Channel, *Input, *Output[25]);
FR_SpeechEncoder4 (*Channel, *Input, *Output[42]);
FR_SpeechEncoder5 (*Channel, *Input, *Output[59]);
```

Where:

Channel is the address of the data space for this channel of voice coding/decoding.

Input is the address of the 160 samples of 13-bit 2s complement linear speech input.

Output is the destination address of the encoded frame of GSM variables.

Set_Channel and Return_Channel are handled within this routine and should not be called. In single-channel applications, Channel should be __bss__ (a TMS320C62xx compiler constant). The code can be found in the sp_codec2.c file. The GSM full-rate parameters are in an uncompressed array of shorts. Data packing is not performed and interrupts must be disabled before these routines are called.

3.4 FR_SpeechDecoder1...4

The following subroutines handle the decoding of a speech frame to the GSM full-rate standard:

```
FR_SpeechDecoder1 (*Channel, *Input, *Output);
FR_SpeechDecoder2 (*Channel, *Input, *Output[40]);
FR_SpeechDecoder3 (*Channel, *Input, *Output[80]);
FR_SpeechDecoder4 (*Channel, *Input, *Output[120]);
```

Where:

Channel is the address of the data space for this channel of voice coding/decoding.

Input is the address of a frame of expanded GSM variables.

Output is the destination address of the decoded frame of 160 samples of 13-bit 2s complement linear speech output.

Set_Channel and Return_Channel are handled within this routine and should not be called. In single-channel applications, Channel should be __bss__ (a TMS320C62xx compiler constant). The code can be found in the sp_codec2.c file. The GSM full-rate parameters are in an uncompressed array of shorts. Data unpacking is not performed and interrupts must be disabled before these routines are called.

3.5 Set_Channel

This subroutine sets the DP register to point to a specific channel. It is not usually required except at initialization, when some initialization of mode specific-variables relating to the DTX mode is required. The following subroutine may be called before channel-specific modes are examined or changed:

```
Set_Channel (*Channel);
```

Channel is the address of the data space for this channel of voice coding/decoding. It can be found in the fr_ram.c file and it is defined in the fr_ram.h file. Interrupts must be disabled before calling this routine is disabled, and they must remain disabled until after Return_Channel is called.

3.6 Return_Channel

The following subroutine complements Set_Channel by providing a means to restore the DP register to its C defaults after channel-specific modes are examined or changed:

```
Return_Channel ();
```

This subroutine can be found in the fr_ram.h file and is an inline compilation.

3.7 FR_ResetEnc

The following subroutine resets the encoder at the start of a call:

```
FR_ResetEnc(int* Context,int DTXenable)
```

Where:

Channel is the address of the data space for this channel of voice coding/decoding.

DTXenable selects if DTX is enabled (1) or disabled (0).

Set_Channel and Return_Channel are handled within this routine and should not be called. In single-channel applications, Channel should be __bss__ (a TMS320C62xx compiler constant). The code can be found in the fr_ram.c file and is defined in the fr_ram.h file.

3.8 FR_ResetDec

The following subroutine resets the decoder at the start of a call:

```
FR_ResetDec(int* Context)
```

Where:

Channel is the address of the data space for this channel of voice coding/decoding.

Set_Channel and Return_Channel are handled within this routine and should not be called. In single-channel applications, Channel should be __bss__ (a TMS320C62xx compiler constant). The code can be found in the fr_ram.c file and is defined in the fr_ram.h file.

3.9 Multichannel Setup

Example 1 shows how to use some of the previously discussed subroutines.

Example 1. Using a Subroutine

```
if (!strcmp(DTXmode,”dtx”))
{
   FR_ResetEnc(Channels[ChCount],1);
   /* enable DTX for VAD tests */
   FR_ResetDec(Channels[ChCount]);
}
else
{
   FR_Reset(Channels[ChCount],0);
   /* disable DTX for non–VAD tests */
   FR_ResetDec(Channels[ChCount]);
}
```

This code comes from the gsm.c file. It performs the initialization of the DTX mode for a new test vector file and resets the encoder and decoder at the start of a new test pattern sequence.

4 Linker Notes (LNK_FR.cmd)

The linker defines two variables that allow allocation of the memory for external variables. These variables must be defined before and after the .bss section for fr_ram.obj and vad1.obj; otherwise, memory for additional channels is incorrectly allocated. Example 2 shows how to define these variables.

Example 2. Defining Allocation Variables

```
_FR_Ram_Start = .;  /* used to allocate heap for additional
                       channels */

   fr_ram.obj (.bss)

   vad1.obj (.bss)

   rxdtx.obj (.bss)

FR_Ram_End = .;  /* used to allocate heap for additional channels */
```

Each module should be compiled separately using the –mt –mh –o3 options.

4.1 Partial_Link_FR.cmd

This link command file should be used in implementations with multiple voice coders. This command links the routines together into a single module that can be linked again later. The global definitions are reduced to the minimum required. Only global definitions beginning with FR_ remain. This command can also be used to link code from multiple sources.

4.2 Code Composer Make Files

The CC_GSM_FR.MAK and the CC_GSM_FR_PL.MAK code composer make files can be used to build a standalone full-rate voice coder or a reusable library module, respectively.

5 Simulator Notes

This section briefly discusses the simulator functions.

- Simfast.simcmd

  This simulator command file is used to verify vectors and evaluate benchmarks.

- Cont_Dual.gsm

  This file tells the CIO how to handle all of the test pattern files. For each test sequence, there are six lines; the first four lines are the filenames of the input and output files, and the fifth line indicates if that code sequence is to be run with voice activity and DTX enabled or disabled (dtx or nodtx, respectively). The last line indicates the GSM data rate, and this code it is always FULL.

  ```
  PCM Data Input
  FileGSM Data Output File
  GSM Data Input File
  PCM Data Output File
  dtx | nodtx
  FULL
  ```

  This structure is repeated for the number of test sequences that must run; at the end, the C program exits.

6 New Multichannel Functions

This section briefly describes the new multichannel functions.

6.1 Set_Channel (Address);

This function points the DP register to the local (.bss) section.

6.2 Return_Channel ();

This function restores the DP register to the default (.bss) section.

6.3 Handle-Ints();

This NULL function restores DP and handles interrupts in the middle of voice coding code. Though not actually used, it may be inserted in C code to allow more interrupts, if needed.

6.4 int**FR_Assign_GSM(int Channels);

This function assigns an array of pointers to channel spaces on the heap memory. Then, it assigns the existing .bss channel to the first address, assigns space on the heap for any additional channels, and returns a pointer to the array of pointers.

This function works with Set_Channel. If there is insufficient space on the memory heap for all channels, this function assigns the maximum number of channels and sets the pointer of the remaining channels to NULL. If there is insufficient space for the array of pointers, this function returns a NULL value.

Example 3 shows the assignment of an array of pointers to channel spaces.

Example 3. Assigning an Array of Pointers to Channel Spaces

```
   ChannelPointers = FR_Assign_GSM (NoOfChannels);
   For (Channel=0,Channel<NoOfChannels, Channel++)
{
       Set_Channel (ChannelPointers[Channel]);
       /* Channel specific “.bss” processing */
       Return_Channel ();
}
```

You must ensure that there is sufficient memory at link time, and you must check the return values for NULL at run time.

7 Compliance/Status

All of the code has been compiled with version 3.0 of the TMS320C6x C compiler for the PC and simulated with version 2.0 of the simulator for the PC, and on an evaluation module (EVM). All of the GSM full-rate voice coding DTX, VAD, and homing test vectors have passed this simulation and are designed to meet the standards described in Table 3.

Table 3. Compliance With GSM Standards

Standard	Title	Description
GSM 06.01	ETS 300 580–1	Full-Rate Overview
GSM 06.10	ETS 300 580–2	Full-Rate Transcoding
GSM 06.10 (Phase 2+	ETS 300 961	Full-Rate Transcoding With Homing
GSM 06.11	ETS 300 580–3	Full-Rate Lost Frames and DTX
GSM 06.12	ETS 300 580–4	Full-Rate Comfort Noise
GSM 06.31	ETS 300 580–5	Full-Rate DTX
GSM 06.32	ETS 300 580–6	Full-Rate VAD

8 Vocoder Variables Available for Subsections

Table 4 shows the voice coder variables required and available for specific subsections of the voice coder/decoder.

Table 4. Vocoder Variables and Subsections

Parameter	Encode Available	Decode Required
Larc[0:7]	1	1
Nc(0)	2	1
Bc(0)	2	1
Mc(0)	2	1
Xmaxc(0)	2	1
Xm(0,0..12)	2	1
Nc(1)	3	2
Bc(1)	3	2
Mc(1)	3	2
Xmaxc(1)	3	2
Xm(1,0..12)	3	2
Nc(2)	4	3
Bc(2)	4	3
Mc(2)	4	3
Xmaxc(2)	4	3
Xm(2,0..12)	4	3
Nc(3)	5	4
Bc(3)	5	4
Mc(3)	5	4
Xmaxc(3)	5	4
Xm(3,0..12)	5	4
VAD/SP	1	
TAF/SIB/BFI		1

9 References

1. ETS 300 580 1–6 Full-Rate Speech
2. ETS 300 961 Full-Rate Speech With Homing
3. *TMS320C6x Optimizing C Compiler* (SPRU187E)
4. *TMS320C6201/6701 Peripherals Reference Guide* (SPRU190C)