Interfacing the TMS320C54xx to the TLC320AD535 and TLC320AD545 Codecs

David M. Alter — *DSP Applications – Semiconductor Group*

ABSTRACT

This application report presents a method for interfacing the TLC320AD535 and TLC320AD545 codecs from Texas Instruments (TI™) with the multichannel buffered serial port (McBSP) on TI's C54xx family of digital signal processors (DSPs). As is the case with many codecs from TI, these devices utilize a primary and secondary communication protocol to differentiate between actual signal data and codec control register data. This communication protocol is discussed, and example C-language code for the DSP is provided that initializes both the McBSP and the codec and then performs a simple codec echo function. A description of the code and code flow is given. Finally, there is a brief discussion on applying this work to other TI codecs.

Contents

List of Figures

List of Tables

1 Introduction

The TLC320AD535 and TLC320AD545 codecs from TI (hereafter, referred to as the AD535 and AD545, respectively) provide a glueless serial interface to the McBSPs on the C54xx DSP family. The AD535 offers dual channel voice/data capability, while the AD545 is a single channel data/fax device. Although functionally different, both codecs share the same serial hardware interface and communication protocols. This application report presents the hardware connections and software necessary to interface these codecs with the McBSP on the C54xx DSP. Note that older C54x DSPs without a McBSP (e.g. they may have the standard serial port or buffered serial port) can also be gluelessly interfaced, although the code provided in this application report would need to be modified.

2 Serial Communication Hardware Interface

Figures 1 and 2 show the necessary connections between each codec and the DSP. In the case of the dual channel AD535, each channel operates and is interfaced independently. In the remainder of this paper, the term codec will refer to the entire AD545 or to one of the two channels on the AD535, except where otherwise indicated by context. The codec acts as the serial bus master, sourcing both the serial clock and the frame synchronization signal. Since codec data transmission and reception always occur simultaneously, only one clock and one frame synchronization signal are present. On the single channel AD545, these signals are DT_SCLK and DT_FS, respectively. On the dual channel AD535, the signals DT_SCLK and DT_FS are dedicated for the data channel, and VC_SCLK and VC_FS handle the voice channel.

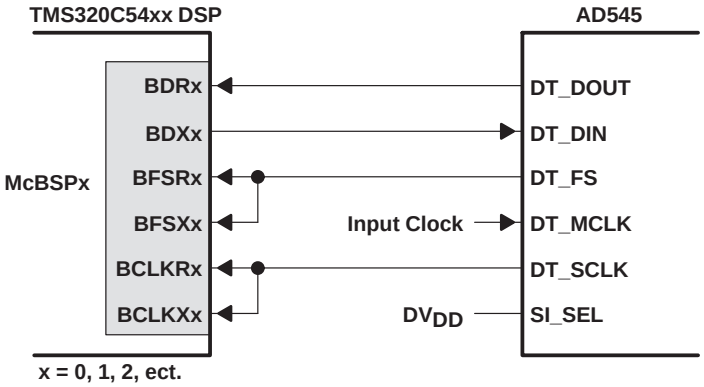

Figure 1. McBSP Interface With TLC320AD545 Codec

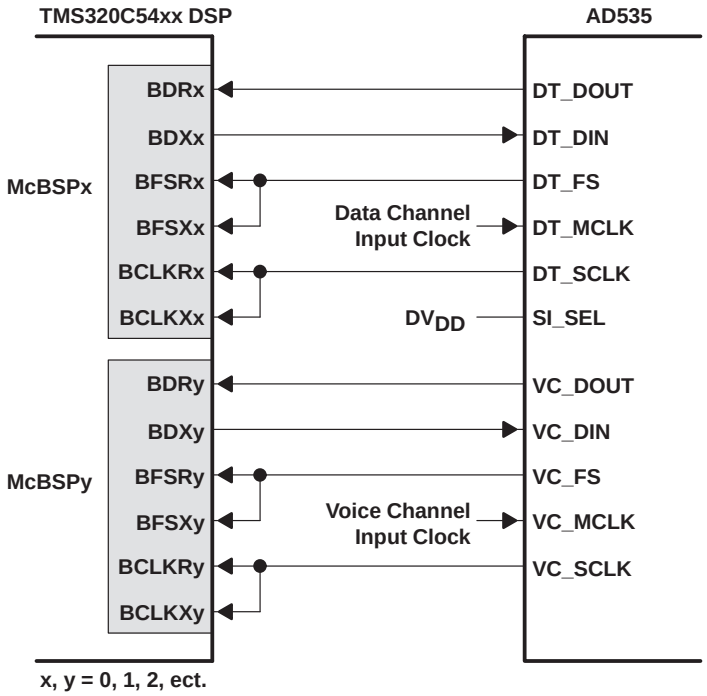

Figure 2. McBSP Interface With TLC320AD535 Codec

It is important that the I/O operating voltages of the codec and DSP be considered to avoid the need for voltage level translators between them. The glueless interfaces shown in Figures 1 and 2 assume 3.3-V interfacing between the devices. Since the TLC320AD535 and TLC320AD545 allow for operation from either 5-V or 3.3-V power supplies, the 3.3-V option should be utilized (i.e., DV_{DD} = 3.3 V). In addition, the DSP should support 3.3-V I/O. As of the time of this printing, C54xx DSPs that operate exclusively with 3.3-V I/O include the TMS320VC5402, TMS320VC5409, TMS320VC5410, TMS320VC5416, TMS320VC5420, TMS320VC5421, and TMS320VC5441. C54xx DSPs that allow for 3.3-V I/O operation given the appropriate I/O power supply include the TMS320UC5402, and TMS320UC5409. The reader should refer to the specific DSP and codec datasheets for complete information and up-to-date specifications.

The codecs support both active high and active low frame synchronization signals (DT_FS). Active low is used in this appnote to interface with the DSP, and is selected by tying the SI_SEL pin high on the codec. Note that the single SI_SEL pin on the AD535 controls both channels on the device. Figure 3 shows the timing relationships between signals on the codec serial interface.

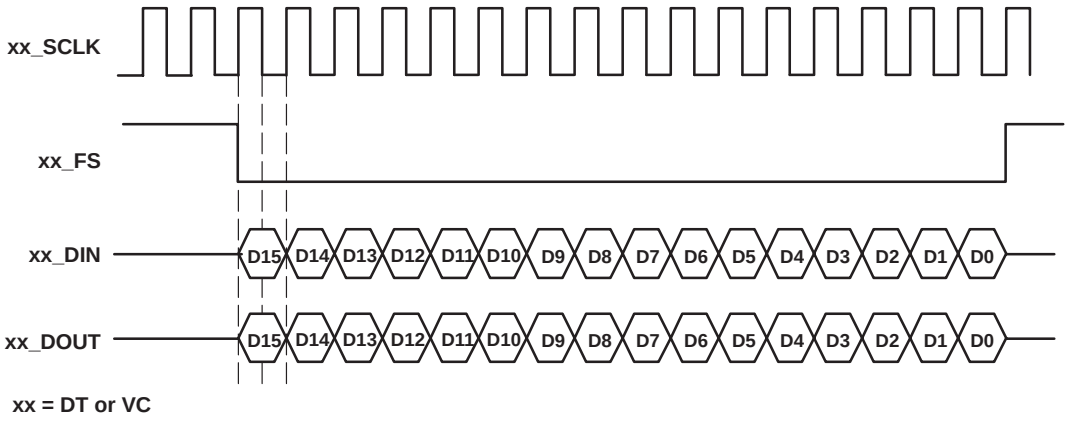

Figure 3. Serial Communication Timings for Codec in FS Low Mode

3 Configuring the McBSP Registers

The McBSP should be configured for standard operating mode (as opposed to multi-channel, SPI, or A-bis modes), with single phase communication and 16-bit word length. Since the codec provides both the serial clocks and the frame synchronization signals, the McBSP sample rate generator is not used. Taking into account the timing relationships in Figure 3, Tables 1 through 7 show suitable settings for the various McBSP control registers. A *x* value for the bit field denotes a don't care condition for register writes. Either a 0 or a 1 can be written to don't care bits. Typically, a zero is written.

Table 1. McBSP Serial Port Control Register 1 (SPCR1x) Settings

BIT	NAME	VALUE	EFFECT
15	DLB	0	Digital loopback mode disabled
14–13	RJUST	00	Right-justify and zero-fill MSBs in DRR[1,2]x
12–11	CLKSTP	00	Normal clocking for non-SPI mode
10–8	Reserved	xxx	Reserved
7	DXENA	0	No extra delay on DX pin
6	ABIS	0	A-bis mode disabled
5–4	RINTM	00	Received interrupt on end of word
3	RSYNCERR	0	Write as 0 (no rx sync error detected)
2	RFULL	x	Read-only
1	RRDY	x	Read-only
0	$\overline{\text{RRST}}$	1	Serial port receiver enabled

Table 2. McBSP Serial Port Control Register 2 (SPCR2x) Settings

BIT	NAME	VALUE	EFFECT
15–10	Reserved	xxxxxx	Reserved
9	FREE	1	Free-run (ignore emulation halt)
8	SOFT	x	Don't care when FREE=1
7	$\overline{\text{FRST}}$	0	Frame sync generator disabled
6	$\overline{\text{GRST}}$	0	Sample rate generator disabled
5–4	XINTM	00	Transmit interrupt on end of word
3	XSYNCERR	0	Write as 0 (no tx sync error detected)
2	$\overline{\text{XEMPTY}}$	0	Read-only
1	XRDY	0	Read-only

Table 3. McBSP Receive Control Register 1 (RCR1x) Settings

BIT	NAME	VALUE	EFFECT
15	Reserved	x	Reserved
14–8	RFRLEN1	0000000	1 word per frame
7–5	RWDLEN1	010	16 bits per word
4–0	Reserved	xxxxx	Reserved

Table 4. McBSP Receive Control Register 2 (RCR2x) Settings

BIT	NAME	VALUE	EFFECT
15	RPHASE	0	Single phase frame
14–8	RFRLEN2	xxxxxxx	Don't care when RPHASE=0
7–5	RWDLEN2	xxx	Don't care when RPHASE=0
4–3	RCOMPAND	00	No companding, transfer MSB first
2	RFIG	1	Ignore receive frame syncs after the first
1–0	RDATDLY	00	0 bit data delay

Table 5. McBSP Transmit Control Register 1 (XCR1x) Settings

BIT	NAME	VALUE	EFFECT
15	Reserved	x	Reserved
14–8	RFRLEN1	0000000	1 word per frame
7–5	RWDLEN1	010	16 bits per word
4–0	Reserved	xxxxx	Reserved

Table 6. McBSP Transmit Control Register 2 (XCR2x) Settings

BIT	NAME	VALUE	EFFECT
15	XPHASE	0	Sinlge phase frame
14–8	XFRLEN2	xxxxxxx	Don't care when XPHASE=0
7–5	XWDLEN2	xxx	Don't care when XPHASE=0
4–3	XCOMPAND	00	No companding, transfer MSB first
2	XFIG	1	Ignore receive frame syncs after the first
1–0	XDATDLY	00	0 bit data delay

Table 7. McBSP Pin Control Register (PCRx) Settings

BIT	NAME	VALUE	EFFECT
15–14	Reserved	00	Reserved
13	XIOEN	0	DX, FSX, CLKX pins are not GP I/O pins
12	RIOEN	0	DR, CLKS, FSR, CLKR are not GP I/O pins
11	FSXM	0	External transmit frame sync signal
10	FSRM	0	External receive frame sync signal
9	CLKXM	0	External transmit clock signal
8	CLKRM	0	External receive clock signal (DLB=0)
7	Reserved	x	Reserved
6	CLKS_STAT	x	Read-only
5	DX_STAT	x	Read-only
4	DR_STAT	x	Read-only
3	FSXP	1	Transmit frame sync active low
2	FSRP	1	Receive frame sync active low
1	CLKXP	1	Transmit data sampled on rising CLKX edge
0	CLKRP	0	Receive data sampled on falling CLKR edge

4 Configuring the Codec Registers

The DSP configures the control registers via the serial communication data link. Both codecs utilize the same primary and secondary communication protocol for register programming. Primary serial communication transfers actual data (as opposed to control register information) between the DSP and the codec ADC and DAC. Secondary serial communication is used to access the codec control registers. All serial communication is primary unless a secondary communication cycle is specifically requested by the DSP.

Secondary communication requests are made by setting the least significant bit (LSB) of the data word transmitted to the codec during primary communication. If the LSB is a 0, the next communication cycle will be primary. If the LSB is a 1, the next communication cycle will be secondary, and the data received by the codec will be diverted to program a codec control register. Figure 4 depicts the primary communication data format.

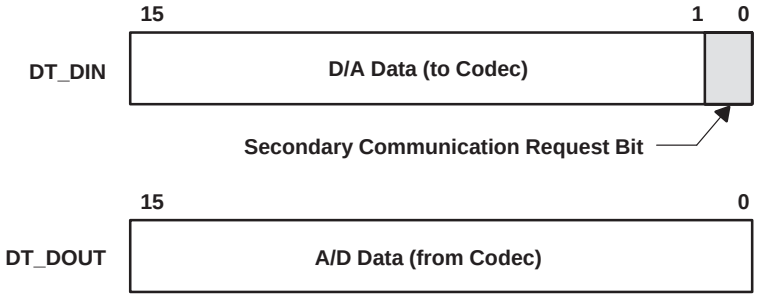

Figure 4. Primary Communication Data Format

During a secondary communication, the data word transmitted to the codec on the DT_DIN or VC_DIN line contains codec control register address information, a bit that indicates whether the register is being written to or read from, and finally the data to be written to the register in the case of a register write. Figure 5 depicts this data format for a register write. Register reads are not typically of interest during codec configuration.

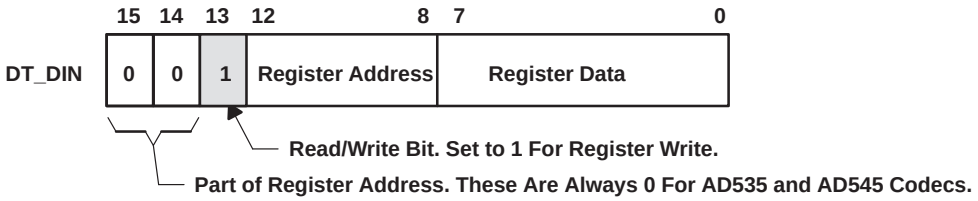

Figure 5. Secondary Communication Data Format

The AD535 codec contains 6 configurable control registers, while the AD545 codec contains only 2. The code provided with this report programs the codec registers with a generic configuration. It can be easily modified to provide any desired codec configuration. It is beyond the scope of this paper to discuss specific register settings for the codec. These will depend on the particular application in which the codec is being used. The reader is referred to the latest data manual for the codec of interest for further information[1, 2].

5 Example Code Description

Appendix A provides example C-language code for interfacing the codec with the McBSP. The code is generic in that it supports any of the 3 McBSPs (i.e., McBSP0, McBSP1, or McBSP2) currently available on C54xx platforms (e.g., C5402, C5410, C5420 DSPs), and also supports both the AD535 and AD545 codecs. The code provides a complete standalone program that initializes the McBSP and codec, and then uses the DSP CPU to perform an interrupt driven echo function (i.e., codec ADC input is copied to codec DAC output). The code is an illustrative example only. Users should modify the code as necessary in order to smoothly incorporate the various functions into their specific application software. All code has been tested using C54x Code Generation Tools v3.10. C code has been tested with no optimization and also full (-o3) optimization. The various files needed to perform a build for each codec are listed below.

AD535 Build

ad535.c	main routine
codec.c	codec interfacing functions
wait.asm	delay function
cvectors_ad535.asm	reset and interrupt vectors
ad535.inc	include file for ad535.c
codec.h	C header file for ad535.c and codec.c
ad535.cmd	example linker command file for VC5402 DSP
ad535.bat	batch file to construct executable

AD545 Build

ad545.c	main routine
codec.c	codec interfacing functions
wait.asm	delay function
cvectors_ad545.asm	reset and interrupt vectors
ad545.inc	include file for ad535.c
codec.h	C header file for ad535.c and codec.c
ad545.cmd	example linker command file for VC5402 DSP
ad545.bat	batch file to construct executable

Figures 6 and 7 show block flow charts of the code on a per function basis. Each function will now be discussed.

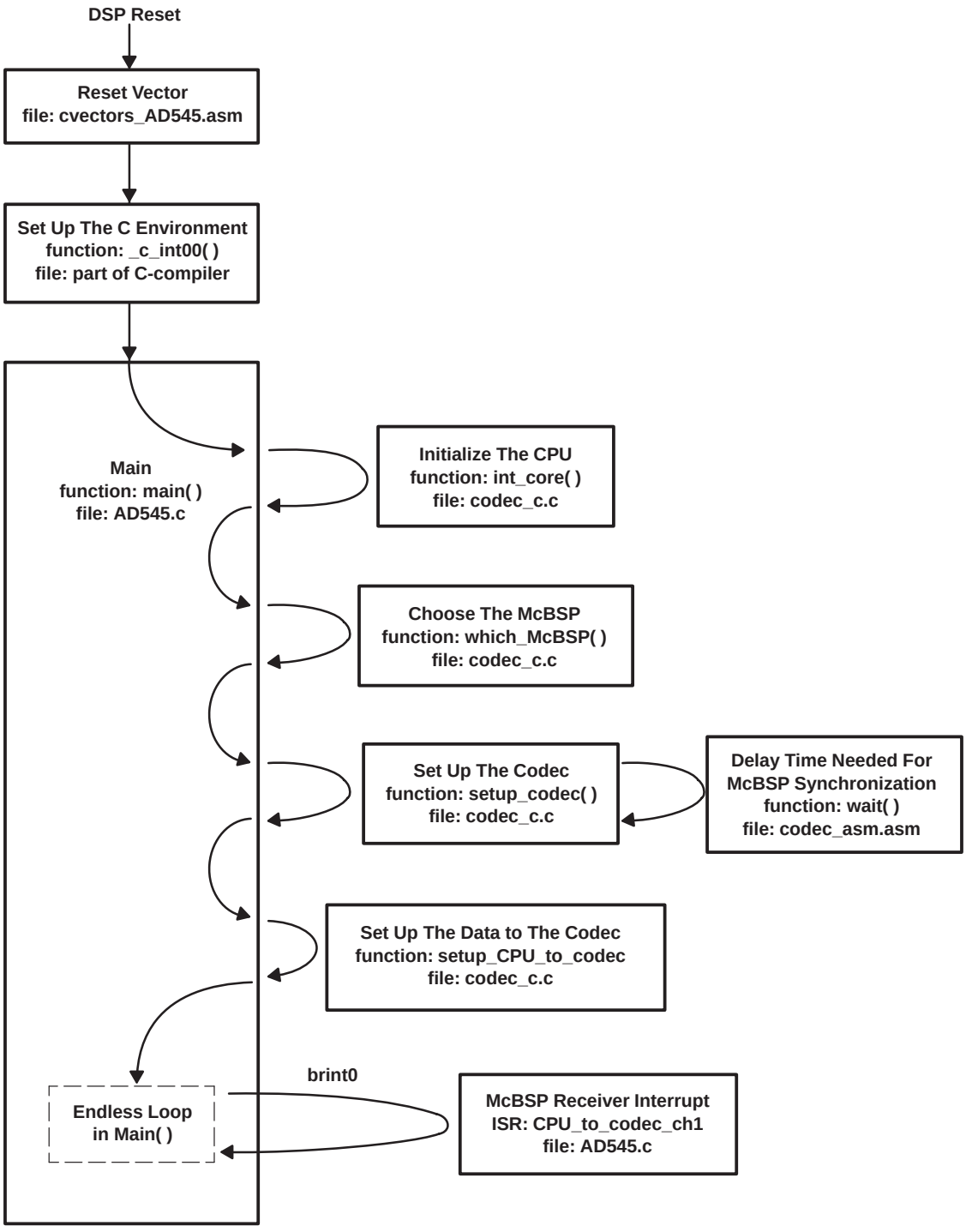

Figure 6. Function Flowchart for AD545 Example Code

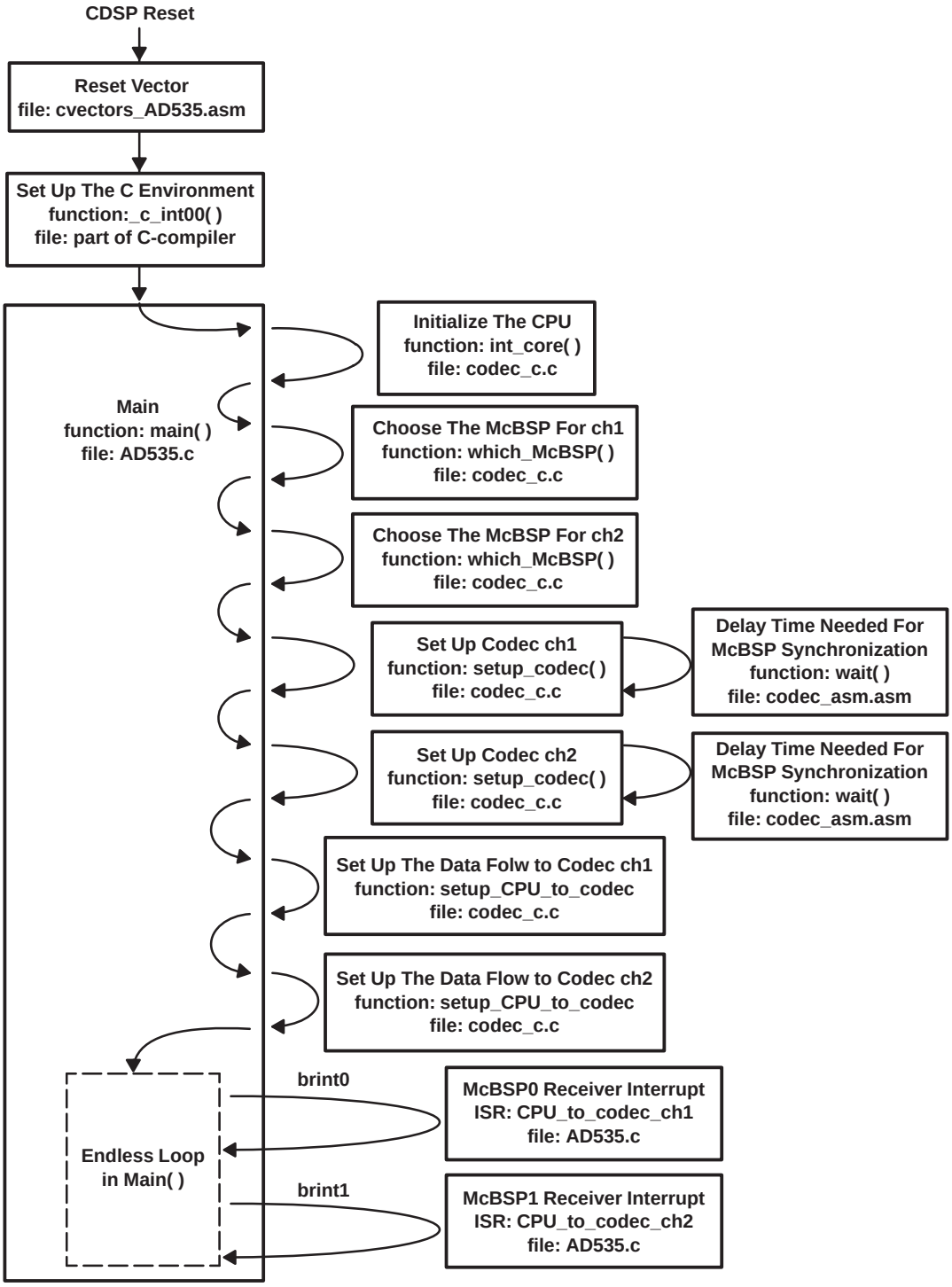

Figure 7. Function Flowchart for AD535 Example Code

5.1 Reset Vector

Complete examples of a reset and interrupt vector table are provided in the files **cvectors_ad545.asm** and **cvectors_ad535.asm**. The labels used in the vector tables are descriptive for a TMS320VC5402 DSP. However, the vector tables themselves are generic for any C54xx device. For example, the 24th vector (address 0x5C) on the VC5402 is the timer 1 interrupt, and hence the table entry is labeled *TINT1*. However, on the VC5410, this vector is external interrupt #2. The user should consult the data sheet for the device being used to determine specific vector functionality.

Two entries are of interest in the vector tables. First is the reset vector, which branches to the start of the C runtime support library at label *_c_int00*. The runtime support library is discussed further in the next section. Second are the McBSP receive interrupts. In this example program, the CPU itself handles the codec receive and transmit data. Therefore, the receive interrupt for each interfacing McBSP branches to an interrupt service routine (ISR) that handles the data. Recall that the codec simultaneously performs receive and transmit, and therefore only one McBSP interrupt, either transmit or receive, is needed. The vector table for the single channel AD545 shows the McBSP0 receive interrupt *brint0* branching to the *_CPU_to_codec_ch1*, which is the data handling ISR. For the dual channel AD535, two McBSP interrupts are used, one for each codec channel. In this example, McBSP0 and McBSP1 are used, and thus *brint0* and *brint1* have been assigned to two independent data handling ISRs. If McBSPs other than these are used (e.g., McBSP2), the vector table must be modified. In the vector table, note the use of the leading underscore for the branch addresses since these target labels were declared in C. The remaining (unused) vectors all branch to themselves, which provides a convenient way of trapping any spurious interrupts that may occur during code development and debug.

5.2 _c_int00( )

This function sets up the C environment in the DSP (e.g., stack pointer, initialized variables) and then calls `main()`. It is part of the C-compiler runtime support library and not part of the source code provided with this application report. The runtime support library (e.g., **rts.lib**) must be linked in with all C-language applications, as required by the C-compiler. This is done using the -l option with the C-compiler (in this code, the -l option was specified in the linker command files, **ad535.cmd** and **ad545.cmd**). Further, the runtime support library must be executed prior to running any C-language modules in a program. One convenient way to ensure this is to simply have the DSP reset vector branch to this function, as was done in this example code. Refer to the *TMS320C54x Optimizing C Compiler User's Guide*, literature number SPRU103C, for more information.

5.3 which_McBSP( )

This function initializes a structure (e.g., McBSP_ch1) that contains pointers to the various McBSP control registers on the DSP and also two bit masks corresponding to McBSP interrupts. The McBSP number that the codec is connected to is a passed parameter to this function. This function allows the code to work with any of the McBSPs. Once a particular McBSP has been chosen for a user application, this function can be eliminated provided the structure is initialized elsewhere. This will save some cycles as the function call and decision making used in this function will no longer be needed.

5.4 init_core( )

This function initializes various DSP core functions such as the software programmable PLL and the software wait-states for the external memory interface. This particular function is a bare bones example, written specifically for a TMS320VC5402 DSP. It is not intended for unmodified use in a real application. Users should consult the user's guide for the specific DSP they are using and build a complete initialization routine as needed for their application.

5.5 setup_codec( )

This function first sets up the specified McBSP and then uses that serial port to initialize the control registers in the codec. The control register data is passed to the function in the array `codec_ctrl_ch1[]`, and also `codec_ctrl_ch2[]` in the case of the dual channel AD535. These arrays are declared in `main()`. The first entry in each array contains the number of control registers to be programmed. The remaining entries are the actual control register values to be written. In this example code, the control data arrays are initialized using an include file, **ad545.inc** or **ad535.inc**, which is included into `main()` just before `setup_codec()` is called. Note that in the AD535 case, the control register data has been split between the voice and data channels, such that the McBSP associated with each of these channels programs the codec registers associated with its function (i.e., voice or data). However, each channel has access to all the control registers in the AD535, and the code could be modified to have one of these channels program all of the registers. Regardless of which codec is of interest, the user should modify the control data values in the include files to reflect the particular needs of their application.

5.6 wait( )

This function performs a CPU time delay. The delay length is specified in terms of CPU clock cycles, and is a passed parameter to this function. This function has been written in assembly code since this allows easy control over DSP cycle counts. In the example code, the constant `N_DELAY_CH1` (and `N_DELAY_CH2`), defined in the files **ad535.c** and **ad545.c**, serves as the delay value. Two delay intervals of 2 serial bit clocks each are needed during McBSP setup, as specified in the *TMS320C54x DSP Reference Set Volume 5: Enhanced Peripherals*, literature number SPRU302. Since the `wait()` function requires the interval to be specified in terms of CPU clock cycles, knowledge of the clock rate for the McBSP is required. On the codecs, xx_SCLK (i.e. DT_SCLK and VC_SCLK) is fixed at half the frequency of xx_MCLK (i.e., DT_MCLK and VC_MCLK). The necessary delay value can be computed as shown in equation 1.

$$N\_DELAY \geq 2 \times \frac{CPU\_clockrate}{\left(\frac{xx\_MCLK}{2}\right)} \quad (1)$$

The choice of DT_MCLK and VC_MCLK is based on sample rate requirements for the code, as specified in the codec datasheets. For example, to obtain 8 kHz data/fax channel sampling, one would select DT_MCLK to be 4.096 MHz. Assuming a 100 MHz DSP, one would obtain the minimum required delay as 98 CPU clock cycles. A similar computation can be made for the voice channel on the AD535. In this example code, this value was rounded up so that values of 100 were chosen for the delays.

5.7 setup_CPU_to_codec

This function performs the tasks necessary to enable the data receive interrupt for the specified McBSP. Once set up, the CPU will respond to incoming codec data by executing the proper ISR.

5.8 CPU_to_codec_ch1 and CPU_to_codec_ch2

These are the interrupt service routines for the McBSP receive interrupts. These functions perform a simple copy from the McBSP receive data register to the McBSP transmit data register (i.e. the routine performs an echo function). Note that to avoid requesting a secondary communication cycle from the codec, the outgoing data must have its least significant bit set to zero.

It is also possible to use the direct memory access controller (DMA) to handle the codec data, rather than the CPU. In this case, the DMA can be set up to respond directly to a McBSP receive event, and the CPU need not respond to McBSP interrupts. The DMA is typically configured to trigger a CPU interrupt after a particular number of data transfers have occurred. This can relieve significant CPU overhead. Use of the DMA will not be discussed further in this paper. The reader is referred to the *TMS320C54x DSP Reference Set Volume 5: Enhanced Peripherals*, literature number SPRU302, for further information.

6 Extension to Other Codecs

The interfacing method and code presented in this application report are easily extendable to other TI codecs besides the AD535 and AD545. One only needs to generate a new include file to initialize the array `codec_ctrl_ch1[]` (and `codec_ctrl_ch2[]` if a dual channel codec). This new file would replace AD545.INC in AD545.C (or AD535.INC in AD535.C). Two requirements must be considered when attempting to incorporate a different codec.

- The codec must be capable of utilizing the same primary and secondary communication protocol as described in this paper for the AD535 and AD545. Specifically, the secondary communication request must be under software control using the LSB of the primary communication data word as the request signal. Many codecs from TI do in fact use this protocol. Some TI codecs also have a hardware pin that can be used to make secondary communication requests, but this is usually only an option and does not preclude using software requests.
- The I/O operating voltages of the codec and DSP be considered. The glueless interface shown in Figures 1 and 2 assume 3.3 V interfacing between the devices. If the codec and DSP interfacing voltages do not match within their allowable tolerances, some type of voltage level shifting will be needed.

7 Conclusion

A method has been presented for gluelessly interfacing an AD535 or AD545 codec to the McBSP on the Texas Instruments C54xx DSP family. The glueless interface assumes 3.3 V connections between the codec and the McBSP. Modular C-code has been presented that performs McBSP and codec initialization, and then implements a simple echo function for the codec. Users should customize this code as needed for their particular application, and integrate the routines into their code.

8 References

1. TLC320AD535C/I Data Manual, literature number SLAS202A.
2. TLC320AD545C/I Data Manual, literature number SLAS206B.

Appendix A Example Code

This appendix contains all example code necessary to perform complete interfacing software builds for the AD535 and AD545 codecs.

A.1 File: AD535.BAT

```
rem *** File: AD535.BAT
rem *** Description: Windows Batch file for building AD535 executable
rem *** Author: David Alter - Texas Instruments
rem *** Last Modified: 09/20/99

c:\dsp\c54x.310\asm500 cvectors_ad535 -ls
c:\dsp\c54x.310\asm500 wait -ls
c:\dsp\c54x.310\cl500 codec -pr -al -g -k -s
c:\dsp\c54x.310\cl500 AD535 -pr -al -g -k -s -o3 -z -c -v2 codec.obj -o ad535.out
-m ad535.map ad535.cmd

rem *** End of file AD535.BAT
```

A.2 File: AD535.C

```
/*****************************************************************
* File: AD535.C                                                  *
*                                                                *
* Description: Main program for interfacing a TLC320AD535 codec to   *
* a C54xx DSP.                                                   *
*                                                                *
* Author: David M. Alter - Texas Instruments                     *
*                                                                *
* Last Modified: 06/23/99                                        *
*****************************************************************/

#include ”codec.h”

#define N_McBSP_ch1   0         /* McBSP# connected to codec ch1 */
#define N_McBSP_ch2   1         /* McBSP# connected to codec ch2 */
#define N_delay_ch1   100       /* delay value passed to wait() */
#define N_delay_ch2   100       /* delay value passed to wait() */

extern void init_core(void);
extern void setup_codec(McBSP*, int*, unsigned int);
extern void setup_CPU_to_codec(McBSP*);
extern void which_McBSP(char, McBSP*);
extern void wait(unsigned int);

/*****************************************************************/
void main(void)
{
    int   codec_ctrl_ch1[3];       /* codec control data array */
    int   codec_ctrl_ch2[5];       /* codec control data array */
    McBSP  McBSP_ch1;              /* McBSP addresses and masks */
    McBSP  McBSP_ch2;              /* McBSP addresses and masks */

/* Initialize the DSP core */
    init_core();

/* Assign addresses for the McBSP connected to the codec channel */
    which_McBSP(N_McBSP_ch1, &McBSP_ch1);
    which_McBSP(N_McBSP_ch2, &McBSP_ch2);

/* setup the codec channel */
    #include ”ad535.inc”
    setup_codec(&McBSP_ch1, codec_ctrl_ch1, N_delay_ch1);
    setup_codec(&McBSP_ch2, codec_ctrl_ch2, N_delay_ch2);
```

```
/* setup the data flow to and from the codec */
    setup_CPU_to_codec(&McBSP_ch1);
    setup_CPU_to_codec(&McBSP_ch2);

/* proceed with main routine */
    while(1) {}              /* endless loop */

}                            /* end of main() */

/*****************************************************************
* Interrupt Service Routine: CPU_to_codec_ch1()                  *
*                                                                *
* Description: This ISR performs a simple echo of data from McBSP*
* receive to McBSP transmit for codec channel 1. The LSB of the  *
* transmit data must be masked to avoid possibly making a secondary *
* communication request to the codec.                            *
*                                                                *
* Prototype: interrupt void CPU_to_codec_ch1(void)               *
*****************************************************************/
interrupt void CPU_to_codec_ch1(void)
{

#if N_McBSP_ch1 == 0
    *DXR10 = *DRR10 & (0xFFFE);

#elif N_McBSP_ch1 == 1
    *DXR11 = *DRR11 & (0xFFFE);

#else
    *DXR12 = *DRR12 & (0xFFFE);

#endif

}
/*****************************************************************
* Interrupt Service Routine: CPU_to_codec_ch2()                  *
*                                                                *
* Description: This ISR performs a simple echo of data from McBSP *
* receive to McBSP transmit for codec channel 2. The LSB of the  *
* transmit data must be masked to avoid possibly making a        *
* secondary communication request to the codec.                  *
*                                                                *
* Prototype: interrupt void CPU_to_codec_ch1(void)               *
*****************************************************************/
interrupt void CPU_to_codec_ch2(void)
{

#if N_McBSP_ch2 == 0
    *DXR10 = *DRR10 & (0xFFFE);

#elif N_McBSP_ch2 == 1
    *DXR11 = *DRR11 & (0xFFFE);

#else
    *DXR12 = *DRR12 & (0xFFFE);

#endif

}
/*** End of file AD535.C ***/
```

A.3 File: AD535.CMD

```
/*****************************************************************
* File: AD535.CMD                                                *
*                                                                *
* Description: Example linker command file for TMS320VC5402 DSP. *
*                                                                *
* Author: David M. Alter - Texas Instruments                     *
*                                                                *
* Last Modified: 06/23/99                                        *
*****************************************************************/

cvectors_ad535.obj
codec.obj
wait.obj

-l rts.lib
-stack 200h

/* C5402 Configuration: MP/MC'=1, OVLY=1, DROM=1 */

MEMORY
{
  PAGE 0:  /* Program Memory */
   VECS:     org=00080h, len=00080h    /* part of DARAM_P */
   DARAM_P:   org=00100h, len=01F00h   /* 1st 8K of 16K DARAM */
   EXT_P:    org=04000h, len=0BF80h    /* external */

  PAGE 1:  /* Data Memory */
   B2:      org=00060h, len=00020h     /* scratch-pad */
   DARAM_D:   org=02000h, len=02000h   /* 2nd 8K of 16K DARAM */
}

SECTIONS
{
    .text    >   DARAM_P   PAGE 0
    .cinit   >   DARAM_P   PAGE 0
    .switch  >   DARAM_P   PAGE 0
    .const   >   DARAM_D   PAGE 1
    .bss     >   DARAM_D   PAGE 1
    .stack   >   DARAM_D   PAGE 1
    .sysmem  >   DARAM_D   PAGE 1
    vectors  >   VECS      PAGE 0

}
/*** End of file AD535.CMD ***/
```

A.4 File: AD535.INC

```
/*********************************************************************
* File: AD535.inc                                                    *
*                                                                    *
* Description: This is a C-language include file that defines        *
* control register values for the TLC320AD535 codec.                 *
*                                                                    *
* Author: David M. Alter - Texas Instruments, Inc.                   *
* Last Modified: 06/23/99                                            *
*********************************************************************/

/*
  Control 1 Register: Data Channel Control
   0000000100000000b = 0x0100
   |||||||||||||+++-- 000 = monitor amp PGA gain = mute
   ||||||||||||+----- 0 = DAC output sent to monitor amp input
   |||||||||||+------ 0 = reserved
   ||||||||||+------- 0 = data channel digital loopback not asserted
   |||||||||+-------- 0 = data channel software power down disabled
   ||||||||+--------- 0 = data channel software reset not asserted
   |||++++----------- 00001 = control 1 reg address
   ||+---------------- 0 = write
   ++----------------- 00 always zero
  Control 2 Register: Data Channel Control
   0000001000000000b = 0x0200
   ||||||||||||||+-- 0 = data channel ADC overflow (read-only)
   ||||||||||||||+--- 0 = monitor speaker driver gain = mute
   ||||||||||+++---- 000 = DAC Data out PGA gain = 0 dB
   ||||||||+++------- 000 = Data in (DTRX) PGA gain = 0 dB
   |||++++----------- 00010 = control 2 reg address
   ||+---------------- 0 = write
   ++----------------- 00 always zero
  Control 3 Register: Voice Channel Control
   0000001100000000b = 0x0300
   ||||||||||||+++-- 000 = TAPI output buffer gain = 0 dB
   |||||||||||||+--- 0 = handset preamp selected for ADC input
   ||||||||||||+---- 0 = microphone preamp not selected for ADC input
   |||||||||||+----- 0 = TAPI preamp not selected for ADC input
   ||||||||||+------ 0 = voice channel analog loopback not asserted
   |||||||||+------- 0 = voice channel digital loopback not asserted
   ||||||||+-------- 0 = voice channel software power down disabled
   |||||||+--------- 0 = voice channel software reset not asserted
   |||++++----------- 00011 = control 3 reg address
   ||+---------------- 0 = write
   ++----------------- 00 always zero
  Control 4 Register: Voice Channel Control
   0000010000000000b = 0x0400
   ||||||||||++++++-- 000000 = voice ADC input PGA gain = 0 dB
   |||||||||+-------- 0 = microphone preamp gain = 0 dB
   ||||||||+--------- 0 = TAPI preamp gain = 0 dB
   |||++++----------- 00100 = control 4 reg address
   ||+---------------- 0 = write
   ++----------------- 00 always zero
  Control 5 Register: Voice Channel Control
   0000010100000000b = 0x0500
   ||||||||||||||+-- 0 = voice channel ADC overflow (read-only)
   ||||||||||||||+--- 0 = 60 ohm speaker L/R buffer gain = mute
   ||||||||++++++---- 000000 = voice DAC output PGA gain = 0 dB
   |||++++----------- 00101 = control 5 reg address
   ||+---------------- 0 = write
   ++----------------- 00 always zero
```

```
  Control 6 Register: Voice Channel Control
   0000011000000000b = 0x0600
   |||||||||+++++++-- 0000000 = reserved
   ||||||||+--------- 0 = handset out buffer gain = 0 dB
   |||++++-----------  00110 = control 6 reg address
   ||+---------------  0 = write
   ++----------------  00 always zero
*/

/* Data channel control register data  */
   codec_ctrl_ch1[0]=2;    /* # of control registers      */
   codec_ctrl_ch1[1]=0x0100;  /* AD535 control 1 reg data */
   codec_ctrl_ch1[2]=0x0200;  /* AD535 control 2 reg data */

/* Voice channel control register data */
   codec_ctrl_ch2[0]=4;    /* # of control registers      */
   codec_ctrl_ch2[1]=0x0300;  /* AD535 control 3 reg data */
   codec_ctrl_ch2[2]=0x0400;  /* AD535 control 4 reg data */
   codec_ctrl_ch2[3]=0x0500;  /* AD535 control 5 reg data */
   codec_ctrl_ch2[4]=0x0600;  /* AD535 control 6 reg data */

/*** End of file AD535.INC ***/
```

A.5 File: AD545.BAT

```
rem *** File: AD545.BAT
rem *** Description: Windows Batch file for building AD545 executable
rem *** Author: David Alter - Texas Instruments
rem *** Last Modified: 09/20/99

c:\dsp\c54x.310\asm500 cvectors_ad545 -ls
c:\dsp\c54x.310\asm500 wait -ls
c:\dsp\c54x.310\cl500 codec -pr -al -g -k -s
c:\dsp\c54x.310\cl500 AD545 -pr -al -g -k -s -o3 -z -c -v2 codec.obj -o ad545.out
-m ad545.map ad545.cmd

rem *** End of file AD545.BAT
```

A.6 File: AD545.C

```
/************************************************************************
* File: AD545.C                                                         *
*                                                                       *
*Description: Main program for interfacing a TLC320AD545 codec to a     *
* C54xx DSP.                                                            *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/23/99                                               *
************************************************************************/
#include ”codec.h”
#define N_McBSP_ch1   0         /* McBSP# connected to the codec */
#define N_delay_ch1   100       /* delay value passed to wait()  */
extern void init_core(void);
extern void setup_codec(McBSP*, int*, unsigned int);
extern void setup_CPU_to_codec(McBSP*);
extern void which_McBSP(char, McBSP*);
extern void wait(unsigned int);
/************************************************************************/
void main(void)
{
    int   codec_ctrl_ch1[3];     /* codec control data array  */
    McBSP  McBSP_ch1;            /* McBSP addresses and masks */
/* Initialize the DSP core   */
    init_core();
/* Assign addresses for the McBSP connected to the codec channel */
    which_McBSP(N_McBSP_ch1, &McBSP_ch1);
/* Setup the codec channel   */
    #include ”ad545.inc”
    setup_codec(&McBSP_ch1, codec_ctrl_ch1, N_delay_ch1);
/* Setup the data flow to and from the codec */
    setup_CPU_to_codec(&McBSP_ch1);
/* Proceed with main routine */
    while(1) {}              /* endless loop  */
}                            /* end of main() */
/*****************************************************************
* Interrupt Service Routine: CPU_to_codec_ch1()                  *
*                                                                *
* Description: This ISR performs a simple echo of data from McBSP*
* receive to McBSP transmit for codec channel 1. The LSB of the  *
* transmit data must be masked to avoid possibly making a secondary*
* communication request to the codec.                            *
*                                                                *
* Prototype: interrupt void CPU_to_codec_ch1(void)               *
*****************************************************************/
interrupt void CPU_to_codec_ch1(void)
{
#if N_McBSP_ch1 == 0
    *DXR10 = *DRR10 & (0xFFFE);
#elif N_McBSP_ch1 == 1
    *DXR11 = *DRR11 & (0xFFFE);
#else
    *DXR12 = *DRR12 & (0xFFFE);
#endif
}
/*** End of file AD545.C ***/
```

A.7 File: AD545.CMD

```
/*****************************************************************
* File: AD545.CMD                                                *
*                                                                *
* Description: Example linker command file for TMS320VC5402 DSP. *
*                                                                *
* Author: David M. Alter - Texas Instruments                     *
*                                                                *
* Last Modified: 06/23/99                                        *
*****************************************************************/

cvectors_ad545.obj
codec.obj
wait.obj

-l rts.lib
-stack 200h

/* C5402 Configuration: MP/MC'=1, OVLY=1, DROM=1 */

MEMORY
{
  PAGE 0:  /* Program Memory */
   VECS:      org=00080h, len=00080h   /* part of DARAM_P */
   DARAM_P:   org=00100h, len=01F00h   /* 1st 8K of 16K DARAM */
   EXT_P:     org=04000h, len=0BF80h   /* external */

  PAGE 1:  /* Data Memory */
   B2:        org=00060h, len=00020h   /* scratch-pad */
   DARAM_D:   org=02000h, len=02000h   /* 2nd 8K of 16K DARAM */
}

SECTIONS
{
    .text    >   DARAM_P   PAGE 0
    .cinit   >   DARAM_P   PAGE 0
    .switch  >   DARAM_P   PAGE 0
    .const   >   DARAM_D   PAGE 1
    .bss     >   DARAM_D   PAGE 1
    .stack   >   DARAM_D   PAGE 1
    .sysmem  >   DARAM_D   PAGE 1
    vectors  >   VECS      PAGE 0

}
/*** End of file AD545.CMD ***/
```

A.8 File: AD545.INC

```
/*********************************************************************
* File: AD545.inc                                                    *
*                                                                    *
* Description: This is a C-language include file that defines        *
* control register values for the TLC320AD545 codec.                 *
*                                                                    *
* Author: David M. Alter - Texas Instruments, Inc.                   *
* Last Modified: 06/23/99                                            *
*********************************************************************/

/*

  Control 1 Register:
   0000000100000000b = 0x0100
   |||||||||||||+++-- 000 = monitor amp PGA gain = mute
   ||||||||||||+----- 0 = DAC output sent to monitor amp input
   |||||||||||+------ 0 = analog loopback not asserted
   ||||||||||+------- 0 = digital loopback not asserted
   |||||||||+-------- 0 = software power down disabled
   ||||||||+--------- 0 = software reset not asserted
   |||++++----------- 00001 = control reg 1 address
   ||+---------------- 0 = write
   ++----------------- 00 always zero

  Control 2 Register:
   0000001000000000b = 0x0200
   |||||||||||||||+-- 0 = ADC overflow (read-only)
   ||||||||||||||+--- 0 = monitor speaker driver gain = mute
   |||||||||||+++---- 000 = DAC Data out PGA gain = 0 dB
   ||||||||+++------- 000 = Data in (DTRX) gain = 0 dB
   |||++++----------- 00010 = control reg 2 address
   ||+---------------- 0 = write
   ++----------------- 00 = always zero
*/

   codec_ctrl_ch1[0]=2;       /* # of control registers */
   codec_ctrl_ch1[1]=0x0100;  /* AD545 control 1 reg data */
   codec_ctrl_ch1[2]=0x0200;  /* AD545 control 2 reg data */

/*** End of file AD545.INC ***/
```

A.9 File: CODEC.C

```
/************************************************************************
* File: CODEC.C                                                         *
*                                                                       *
* Description: C functions for interfacing the codec to a C54xx DSP.    *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/23/99                                               *
************************************************************************/

#include ”codec.h”

/************************************************************************
* Function: init_core                                                   *
*                                                                       *
* Description: This function sets up various items in the CPU core.     *
*                                                                       *
* Prototype: void init_core(void)                                       *
*                                                                       *
* Parameters: none                                                      *
************************************************************************/
void init_core(void) {

/* Setup the PLL. Since CLKMD pin settings and device type are
  unknown, be safe by disabling the PLL first, and then re-configure.*/

    *CLKMD = 0x0000;        /* disable PLL, run in /2 mode */
    *CLKMD = 0x90b2;        /* PLL x10 for 10MHz input */

/* Other setup */

    *SWWSR = 0x2492;        /* wait-states: 2 p, 2 d, 2 i/o */
    *SWCR = 0x0000;         /* wait-state multiplier = 1 */
    *PMST = *PMST | 0x0020; /* OVLY=1 (RAM in prog space) */
}                           /* end of init_core() */

/************************************************************************
* Function: setup_codec                                                 *
*                                                                       *
* Description: This function performs the serial communication          *
* necessary to setup the codec connected to one of the McBSP’s.         *
* It leaves the codec running and the McBSP handling tx and rx          *
* data, but it does not setup the CPU or DMA’s to feed the data         *
* to/from the McBSP.                                                    *
*                                                                       *
* Prototype: void setup_codec(McBSP McBSPx,                             *
*               int *codec_ctrl_data);                                  *
*               unsigned int N_delay                                    *
* Parameters:                                                           *
*  McBSPx      = structure of McBSP information                         *
*  *codec_ctrl_data = pointer to start of control data array            *
*  N_delay     = the number of DSP clock cycles that occur in           *
*           2 serial bit clocks of the codec during codec               *
*           initialization.                                             *
************************************************************************/
void setup_codec(McBSP *McBSPx, int *codec_ctrl_data, unsigned int N_delay)
{
    unsigned int DMPREC_SAVE;     /* place to save DMPREC reg */
    int i;                        /* general purpose integer  */

/* All interrupts must be disabled for this routine since codec
  initialization is poll based, and a serial communication period
  cannot be missed. */

    asm(” SSBX INTM”);           /* disable global interrupts */
```

```
/* Configure the specified McBSP */
    *McBSPx->SPSAx = SPCR1x_SUBADDR;  /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0000;         /* disable rx */
    *McBSPx->SPSAx = SPCR2x_SUBADDR;  /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0200;         /* disable tx, FREE=1  */
    *McBSPx->SPSAx = PCRx_SUBADDR;    /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x000c;         /* external FS and CLK, tx on
                                         rising edge, rx on falling */
    *McBSPx->SPSAx = RCR1x_SUBADDR;   /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0040;         /* 1 word/frame, 16-bit/word  */
    *McBSPx->SPSAx = RCR2x_SUBADDR;   /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0004;         /* 1 rx phase, no compand */
    *McBSPx->SPSAx = XCR1x_SUBADDR;   /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0040;         /* 1 word/frame, 16-bit/word */
    *McBSPx->SPSAx = XCR2x_SUBADDR;   /* set McBSP sub-address */
    *McBSPx->SPSADx = 0x0004;         /* 1 tx phase, no compand */
    wait(N_delay);                    /* must wait 2 bit clock cycles
                                         for proper McBSP init */
/* Configure the McBSP interrupt for polling operation. Must set
  INTOSEL[1:0]=00b in DMPREC register to mux McBSP BXINTx interrupt
  to the core for McBSP1 and McBSP2 on some C54xx devices */
    DMPREC_SAVE = *DMPREC;            /* save the DMPREC register */
    *DMPREC = *DMPREC & 0xff3f;       /* set INTOSEL[1:0]=00b */
/* Configure the codec */
    *McBSPx->SPSAx = SPCR1x_SUBADDR;  /* set McBSPx sub-address */
    *McBSPx->SPSADx = 0x0001;         /* enable rx, rx int on e/o word */
    *McBSPx->SPSAx = SPCR2x_SUBADDR;  /* set McBSPx sub-address */
    *McBSPx->SPSADx = 0x0201;         /* enable tx, tx interrupt on
                                         end of word, FREE=1 */
    wait(N_delay);                    /* must wait 2 bit clock cycles
                                         for proper McBSP init */
    *IFR = McBSPx->BXINTx_MASK;       /* clear BXINTx flag */
    for(i=1; i <= codec_ctrl_data[0]; i++) {
        *McBSPx->DXR1x = 0x0001;
            /* primary comm data to request a secondary comm */
        while( !(*IFR & McBSPx->BXINTx_MASK) ) {}
           /* primary data in DXR1x reg, wait for BXINT flag */
        *IFR = McBSPx->BXINTx_MASK;   /* clear BXINT flag */
        *McBSPx->DXR1x = codec_ctrl_data[i];
                                      /* secondary comm data */
        while( !(*IFR & McBSPx->BXINTx_MASK) ) {}
           /* secondary data in DXR1 reg, wait for BXINT flag */
        *IFR = McBSPx->BXINTx_MASK;   /* clear BXINT flag */
    }
/* Post-setup cleanup */
    *McBSPx->DXR1x = 0x0000;          /* flush last word tx'd from DXR1
                                         register in case its LSB was 1 */
    *DMPREC = DMPREC_SAVE;            /* restore the DMPREC register */
}                   /* end of setup_codec() */
```

```
/*****************************************************************
* Function: setup_CPU_to_codec                                   *
*                                                                *
* Description: This function sets up the interrupts needed for the   *
* CPU to directly handle tx and rx data to the codec. Simultaneous   *
* transmit and receive are assumed, such that only the McBSP receive *
* interrupt is setup. The transmit interrupt is disabled.            *
*                                                                *
* Prototype: void setup_CPU_to_codec(McBSP McBSPx)               *
*                                                                *
* Parameters:                                                    *
*  McBSPx      = structure of McBSP information                  *
*****************************************************************/
void setup_CPU_to_codec(McBSP *McBSPx)
{
    volatile int  temp;               /* general purpose int */

/* Must set INTOSEL[1:0]=00b in DMPREC register to mux McBSP BRINTx
   interrupt to the core for McBSP1 and McBSP2 on some C54xx devices.*/

    if( (McBSPx->McBSPnum==1) || (McBSPx->McBSPnum==2) ) {
        *DMPREC = *DMPREC & 0xff3f;    /* set INTOSEL[1:0]=00b */
    }

    *IFR = McBSPx->BRINTx_MASK;        /* clear BRINT flag */
    temp = *DRR10;                     /* read DRR10 to clear RRDY bit */
    *IMR = *IMR | McBSPx->BRINTx_MASK; /* enable BRINT interrupt */
    asm(" RSBX INTM");                 /* enable global interrupts */

}                                      /* end of setup_CPU_to_codec() */
/*****************************************************************
* Function: which_McBSP                                          *
*                                                                *
* Description: This function sets up control register addresses and  *
* interrupt mask contstants for the selected McBSP.              *
*                                                                *
* Prototype: void which_McBSP(char N, McBSP *McBSPx)             *
*                                                                *
* Parameters:                                                    *
*  N    = the McBSP number that the codec is connected to        *
*  *McBSPx = structure of McBSP register pointers                *
*****************************************************************/
void which_McBSP(char N, McBSP *McBSPx)
{
/* Assign pointers for the specified McBSP */

            McBSPx->McBSPnum = N;

    if(N == 1) {                  /* McBSP1 selected */
            McBSPx->SPSAx = SPSA1;
            McBSPx->SPSADx = SPSAD1;
            McBSPx->DRR1x = DRR11;
            McBSPx->DXR1x = DXR11;
            McBSPx->BRINTx_MASK = BRINT1_MASK;
            McBSPx->BXINTx_MASK = BXINT1_MASK;
    }
    else if(N == 2) {             /* McBSP2 selected */
            McBSPx->SPSAx = SPSA2;
            McBSPx->SPSADx = SPSAD2;
            McBSPx->DRR1x = DRR12;
            McBSPx->DXR1x = DXR12;
            McBSPx->BRINTx_MASK = BRINT2_MASK;
            McBSPx->BXINTx_MASK = BXINT2_MASK;
    }
    else {                       /* McBSP0 is default */
```

```
            McBSPx->SPSAx = SPSA0;
            McBSPx->SPSADx = SPSAD0;
            McBSPx->DRR1x = DRR10;
            McBSPx->DXR1x = DXR10;
            McBSPx->BRINTx_MASK = BRINT0_MASK;
            McBSPx->BXINTx_MASK = BXINT0_MASK;
    }
}                       /* end of which_McBSP() */
/*** End of file CODEC.C ***/
```

A.10 File: CODEC.H

```
/************************************************************************
* File: codec.h                                                         *
*                                                                       *
* Description: C header file containing variable type definitions and   *
* C54xx register address definitions.                                   *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/23/99                                               *
************************************************************************/

/*** Type definitions ***/

typedef struct McBSP{
    unsigned char McBSPnum;        /* McBSP # codec is connected to */
    volatile unsigned int *SPSAx;  /* McBSPx sub-addr reg */
    volatile unsigned int *SPSADx; /* McBSPx sub-addr data reg */
    volatile unsigned int *DRR1x;  /* McBSPx data rx reg 1 */
    volatile unsigned int *DXR1x;  /* McBSPx data tx reg 1 */
    unsigned int BRINTx_MASK;      /* IFR/IMR mask for BRINTx */
    unsigned int BXINTx_MASK;      /* IFR/IMR mask for BXINTx */
} McBSP;

/*** C54x core memory mapped register definitions ***/
/*  This is an incomplete set. Only those registers
   needed by the example code have been included. */

#define IMR    (volatile unsigned int *)0x0000 /* interrupt mask reg */
#define IFR    (volatile unsigned int *)0x0001 /* interrupt flag reg */
#define PMST   (volatile unsigned int *)0x001d /* processor mode status */
#define SWWSR  (volatile unsigned int *)0x0028 /* sw wait-state reg */
#define CLKMD  (volatile unsigned int *)0x0058 /* PLL clock mode reg */

/*** VC54xx McBSP Memory-Mapped Register Address Definitions ***/

#define DRR20  (volatile unsigned int *)0x0020 /* McBSP0 data rx reg 2 */
#define DRR10  (volatile unsigned int *)0x0021 /* McBSP0 data rx reg 1 */
#define DRR21  (volatile unsigned int *)0x0040 /* McBSP1 data rx reg 2 */
#define DRR11  (volatile unsigned int *)0x0041 /* McBSP1 data rx reg 1 */
#define DRR22  (volatile unsigned int *)0x0030 /* McBSP2 data rx reg 2 */
#define DRR12  (volatile unsigned int *)0x0031 /* McBSP2 data rx reg 1 */

#define DXR20  (volatile unsigned int *)0x0022 /* McBSP0 data tx reg 2 */
#define DXR10  (volatile unsigned int *)0x0023 /* McBSP0 data tx reg 1 */
#define DXR21  (volatile unsigned int *)0x0042 /* McBSP1 data tx reg 2 */
#define DXR11  (volatile unsigned int *)0x0043 /* McBSP1 data tx reg 1 */
#define DXR22  (volatile unsigned int *)0x0032 /* McBSP2 data tx reg 2 */
#define DXR12  (volatile unsigned int *)0x0033 /* McBSP2 data tx reg 1 */

#define SPSA0  (volatile unsigned int *)0x0038 /* McBSP0 sub-addr reg */
#define SPSA1  (volatile unsigned int *)0x0048 /* McBSP1 sub-addr reg */
#define SPSA2  (volatile unsigned int *)0x0034 /* McBSP2 sub-addr reg */
```

```
#define SPSAD0 (volatile unsigned int *)0x0039 /* McBSP0 sub-addr data reg*/
#define SPSAD1 (volatile unsigned int *)0x0049 /* McBSP1 sub-addr data reg*/
#define SPSAD2 (volatile unsigned int *)0x0035 /* McBSP2 sub-addr data reg*/

#define SPCR1x_SUBADDR 0x0000 /* McBSPx control reg 1 sub-addr */
#define SPCR2x_SUBADDR 0x0001 /* McBSPx control reg 2 sub-addr */
#define RCR1x_SUBADDR  0x0002 /* McBSPx rx control reg 1 sub-addr */
#define RCR2x_SUBADDR  0x0003 /* McBSPx rx control reg 2 sub-addr */
#define XCR1x_SUBADDR  0x0004 /* McBSPx tx control reg 1 sub-addr */
#define XCR2x_SUBADDR  0x0005 /* McBSPx tx control reg 2 sub-addr */
#define SRGR1x_SUBADDR 0x0006 /* McBSPx sample rate gen reg 1 sub-addr */
#define SRGR2x_SUBADDR 0x0007 /* McBSPx sample rate gen reg 2 sub-addr */
#define MCR1x_SUBADDR  0x0008 /* McBSPx multichannel reg 1 sub-addr */
#define MCR2x_SUBADDR  0x0009 /* McBSPx multichannel reg 2 sub-addr */
#define RCERAx_SUBADDR 0x000a /* McBSPx rx ch enable reg A sub-addr */
#define RCERBx_SUBADDR 0x000b /* McBSPx rx ch enable reg B sub-addr */
#define XCERAx_SUBADDR 0x000c /* McBSPx tx ch enable reg A sub-addr */
#define XCERBx_SUBADDR 0x000d /* McBSPx tx ch enable reg B sub-addr */
#define PCRx_SUBADDR   0x000e /* McBSPx pin control reg sub-addr */

/*** other register definitions ***/

#define DMPREC (volatile unsigned int *)0x0054
                /* DMA priority & enable control reg */
#define SWCR  (volatile unsigned int *)0x002B
                /* software wait-state control reg */

/*** Interrupt related definitions ***/

#define BRINT0_MASK   0x0010     /* mask for BRINT0 flag */
#define BRINT1_MASK   0x0400     /* mask for BRINT1 flag */
#define BRINT2_MASK   0x0040     /* mask for BRINT2 flag */
#define BXINT0_MASK   0x0020     /* mask for BXINT0 flag */
#define BXINT1_MASK   0x0800     /* mask for BXINT1 flag */
#define BXINT2_MASK   0x0080     /* mask for BXINT2 flag */

/*** End of file CODEC.H ***/
```

A.11 File: CVECTORS_AD535.ASM

```
*************************************************************************
* File: CVECTORS_AD535.ASM                                              *
*                                                                       *
* Description: reset and interrupt vector table.                        *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/22/99                                               *
*************************************************************************
    .ref  _c_int00, _CPU_to_codec_ch1, _CPU_to_codec_ch2

    .sect ”vectors”
rs:     BD   _c_int00       ;reset, SINTR
        NOP
        NOP
nmi:    BD   nmi            ;NMI, SINT16
        NOP
        NOP
sint17: BD   sint17         ;SINT17
        NOP
        NOP
sint18: BD   sint18         ;SINT18
        NOP
        NOP
```

```
sint19: BD   sint19               ;SINT19
        NOP
        NOP
sint20: BD   sint20               ;SINT20
        NOP
        NOP
sint21: BD   sint21               ;SINT21
        NOP
        NOP
sint22: BD   sint22               ;SINT22
        NOP
        NOP
sint23: BD   sint23               ;SINT23
        NOP
        NOP
sint24: BD   sint24               ;SINT24
        NOP
        NOP
sint25: BD   sint25               ;SINT25
        NOP
        NOP
sint26: BD   sint26               ;SINT26
        NOP
        NOP
sint27: BD   sint27               ;SINT27
        NOP
        NOP
sint28: BD   sint28               ;SINT28
        NOP
        NOP
sint29: BD   sint29               ;SINT29
        NOP
        NOP
sint30: BD   sint30               ;SINT30
        NOP
        NOP
int0:   BD   int0                 ;INT0, SINT0
        NOP
        NOP
int1:   BD   int1                 ;INT1, SINT1
        NOP
        NOP
int2:   BD   int2                 ;INT2, SINT2
        NOP
        NOP
tint0:  BD   tint0                ;TINT0, SINT3
        NOP
        NOP
brint0: BD   _CPU_to_codec_ch1    ;BRINT0, SINT4
        NOP
        NOP
bxint0: BD   bxint0               ;BXINT0, SINT5
        NOP
        NOP
dmac0:  BD   dmac0                ;DMAC0, SINT6
        NOP
        NOP
tint1:  BD   tint1                ;TINT1 or DMAC1, SINT7
        NOP
        NOP
int3:   BD   int3                 ;INT3, SINT8
        NOP
        NOP
```

```
hpint:  BD   hpint                ;HPINT, SINT9
        NOP
        NOP
brint1: BD   _CPU_to_codec_ch2    ;BRINT1 or DMAC2, SINT10
        NOP
        NOP
bxint1: BD   bxint1               ;BXINT1 or DMAC3, SINT11
        NOP
        NOP
dmac4:  BD   dmac4                ;DMAC4, SINT12
        NOP
        NOP
dmac5:  BD   dmac5                ;DMAC5, SINT13
        NOP
        NOP
rsvd1:  BD   rsvd1                ;reserved
        NOP
        NOP
rsvd2:  BD   rsvd2                ;reserved
        NOP
        NOP
;End of file CVECTORS_AD535.ASM
```

A.12 File: CVECTORS_AD545.ASM

```
*************************************************************************
* File: CVECTORS_AD545.ASM                                              *
*                                                                       *
* Description: reset and interrupt vector table.                        *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/22/99                                               *
*************************************************************************
    .ref  _c_int00, _CPU_to_codec_ch1
    .sect ”vectors”
rs:     BD   _c_int00             ;reset, SINTR
        NOP
        NOP
nmi:    BD   nmi                  ;NMI, SINT16
        NOP
        NOP
sint17: BD   sint17               ;SINT17
        NOP
        NOP
sint18: BD   sint18               ;SINT18
        NOP
        NOP
sint19: BD   sint19               ;SINT19
        NOP
        NOP
sint20: BD   sint20               ;SINT20
        NOP
        NOP
sint21: BD   sint21               ;SINT21
        NOP
        NOP
sint22: BD   sint22               ;SINT22
        NOP
        NOP
sint23: BD   sint23               ;SINT23
```

```
        NOP
        NOP
sint24: BD   sint24              ;SINT24
        NOP
        NOP
sint25: BD   sint25              ;SINT25
        NOP
        NOP
sint26: BD   sint26              ;SINT26
        NOP
        NOP
sint27: BD   sint27              ;SINT27
        NOP
        NOP
sint28: BD   sint28              ;SINT28
        NOP
        NOP
sint29: BD   sint29              ;SINT29
        NOP
        NOP
sint30: BD   sint30              ;SINT30
        NOP
        NOP
int0:   BD   int0                ;INT0, SINT0
        NOP
        NOP
int1:   BD   int1                ;INT1, SINT1
        NOP
        NOP
int2:   BD   int2                ;INT2, SINT2
        NOP
        NOP
tint0:  BD   tint0               ;TINT0, SINT3
        NOP
        NOP
brint0: BD   _CPU_to_codec_ch1   ;BRINT0, SINT4
        NOP
        NOP
bxint0: BD   bxint0              ;BXINT0, SINT5
        NOP
        NOP
dmac0:  BD   dmac0               ;DMAC0, SINT6
        NOP
        NOP
tint1:  BD   tint1               ;TINT1 or DMAC1, SINT7
        NOP
        NOP
int3:   BD   int3                ;INT3, SINT8
        NOP
        NOP
hpint:  BD   hpint               ;HPINT, SINT9
        NOP
        NOP
brint1: BD   brint1              ;BRINT1 or DMAC2, SINT10
        NOP
        NOP
bxint1: BD   bxint1              ;BXINT1 or DMAC3, SINT11
        NOP
        NOP
dmac4:  BD   dmac4               ;DMAC4, SINT12
        NOP
        NOP
dmac5:  BD   dmac5               ;DMAC5, SINT13
```

```
        NOP
        NOP
rsvd1:  BD   rsvd1              ;reserved
        NOP
        NOP
rsvd2:  BD   rsvd2              ;reserved
        NOP
        NOP
;End of file CVECTORS_AD545.ASM
```

A.13 File: WAIT.ASM

```
*************************************************************************
* File: WAIT.ASM                                                        *
*                                                                       *
* Function: wait()                                                      *
*                                                                       *
* Description: This C-callable function executes a NOP delay loop      *
* for specified number of CPU clock cycles. Maximum delay is 65,535     *
* cycles, which equates to 655 us on a 100MHz device.                   *
*                                                                       *
* Prototype: void wait(unsigned int N)                                  *
*                                                                       *
* Parameters:                                                           *
*  N = length of the delay in CPU clock cycles                          *
*                                                                       *
* Author: David M. Alter - Texas Instruments                            *
*                                                                       *
* Last Modified: 06/24/99                                               *
*************************************************************************

   .def  _wait

   .text
_wait:
   STM   #0008h, AR0 ;AR0 points to ACCL
   RPT   *AR0     ;repeat the # of times specified in ACCL
   NOP            ;do nothing in the delay loop
   RET            ;return

;End of file WAIT.ASM
```

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.