Interfacing TMS320C5x and Parallel Stereo CODEC with other TMS320 DSP Considerations

APPLICATION REPORT: SPRA097

Mathew George, Jr

Digital Signal Processing Solutions
December 1997

TEXAS INSTRUMENTS

IMPORTANT NOTICE

TRADEMARKS

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Tables

Interfacing TMS320C5x and Parallel Stereo CODEC with other TMS320 DSP Considerations

Abstract

The CS4231 is a 16 bit stereo codec chip with various features, such as multiple analog inputs and outputs, mixing, and gain capabilities, that make them ideal for computer sound applications [1][2]. The digital interface consists of a parallel 8-bit bus and is meant to directly connect to a PC (ISA/EISA) interface. Since it is a glueless interface to a PC bus, it looks a bit different from a TMS320C5x or other TMS320 DSPs with asynchronous memory interface. This Application Note describes an optimum interface to a TMS320C5x DSP. Note that other TMS320 DSP asynchronous buses are very similar and most of the interface described here will also apply to any TMS320 DSP system. In this implementation the TMS320C5x and codec share the same bus with SRAM. This tradeoff keeps pin count and thus cost down, at the cost of performance. A separate codec bus mapped into the DSP's memory or I/O space would make the system more efficient and is briefly described at the end of this Application Note along with other system considerations.

Product Support

World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Email

For technical issues or clarification on switching products, please send a detailed email to dsph@ti.com. Questions receive prompt attention and are usually answered within one business day.

INTRODUCTION

The CS4231 is a 16 bit stereo codec chip with various features, such as multiple analog inputs and outputs, mixing, and gain capabilities, that make them ideal for computer sound applications [1][2]. The digital interface consists of a parallel 8-bit bus and is meant to directly connect to a PC (ISA/EISA) interface. Since it is a glueless interface to a PC bus, it looks a bit different from a TMS320C5x or other TMS320 DSPs with asynchronous memory interface. This Application Note describes an optimum interface to a TMS320C5x DSP. Note that other TMS320 DSP asynchronous buses are very similar and most of the interface described here will also apply to any TMS320 DSP system. In this implementation the TMS320C5x and codec share the same bus with SRAM. This tradeoff keeps pin count and thus cost down, at the cost of performance. A separate codec bus mapped into the DSP’s memory or I/O space would make the system more efficient and is briefly described at the end of this Application Note along with other system considerations.

SYSTEM CONFIGURATION

This TMS320C5x implementation was done with a TPC1280 FPGA acting as the glue logic. It was meant to be a prototype for an ASIC/cDSP type system with minimum hardware and concurrent playback and capture. Thus the system was designed within these parameters, assuming a roadmap to ASIC/cDSP. There are a variety of other ways that this interface could be accomplished if discrete logic or an FPGA with FIFOs will be in the final system. Thus some aspects could be more efficient using a faster ASIC or even discrete logic or PLDs. The needs of the system seem to differ based on the DSP software/OS to be used. As much flexibility was put into this system as possible (such as polled and interrupt modes) to allow determination by the programmer based on the specific system.

System Modes

Thus the following three progressive levels of interface were implemented:

1) **Programmed I/O** - Necessary for initialization of the codec, but slow in actual operation due to the use of bit polling for capture and playback. Uses the least amount of pins and the DSP does all the work. This is not an efficient configuration.

2) **Interrupt driven** - Various possible configurations, but the general idea is that the DSP is somehow interrupted by the codec for capture and playback. It is faster than mode 1, but uses more pins and logic. Again the DSP does all the work. In this implementation, the number of interrupt pins used was minimized as a tradeoff for more software protocol. This mode is *probably* more efficient if classic assembly-level, low-overhead DSP code is used.

3) **Stereo Codec DMA** - Again various configurations, but the general idea is that the DSP is put in -HOLD (i.e. taken off the bus) based on an interrupt. Then external logic (i.e. a DMA controller implemented in the FPGA and modeled after the Buffered Serial Port seen in the TMS320C56/57 and TMS320C54x DSPs [7]) will copy the data for capture and playback between the DSP and memory. Mode 3 needs more pins and is faster than mode 1. (Again, as in mode 2, the number of interrupt pins was minimized). The DSP is oblivious to this task until an interrupt tells it that input or output buffers in memory are full and need to be processed. But there can be some overhead if the DSP needs to access external memory [3]. Mode 3 uses the same number of pins as mode 2 and (regarding speed) is *probably* more efficient, if a less-classic, C-based multi-tasking operating system, higher-overhead DSP code is used.

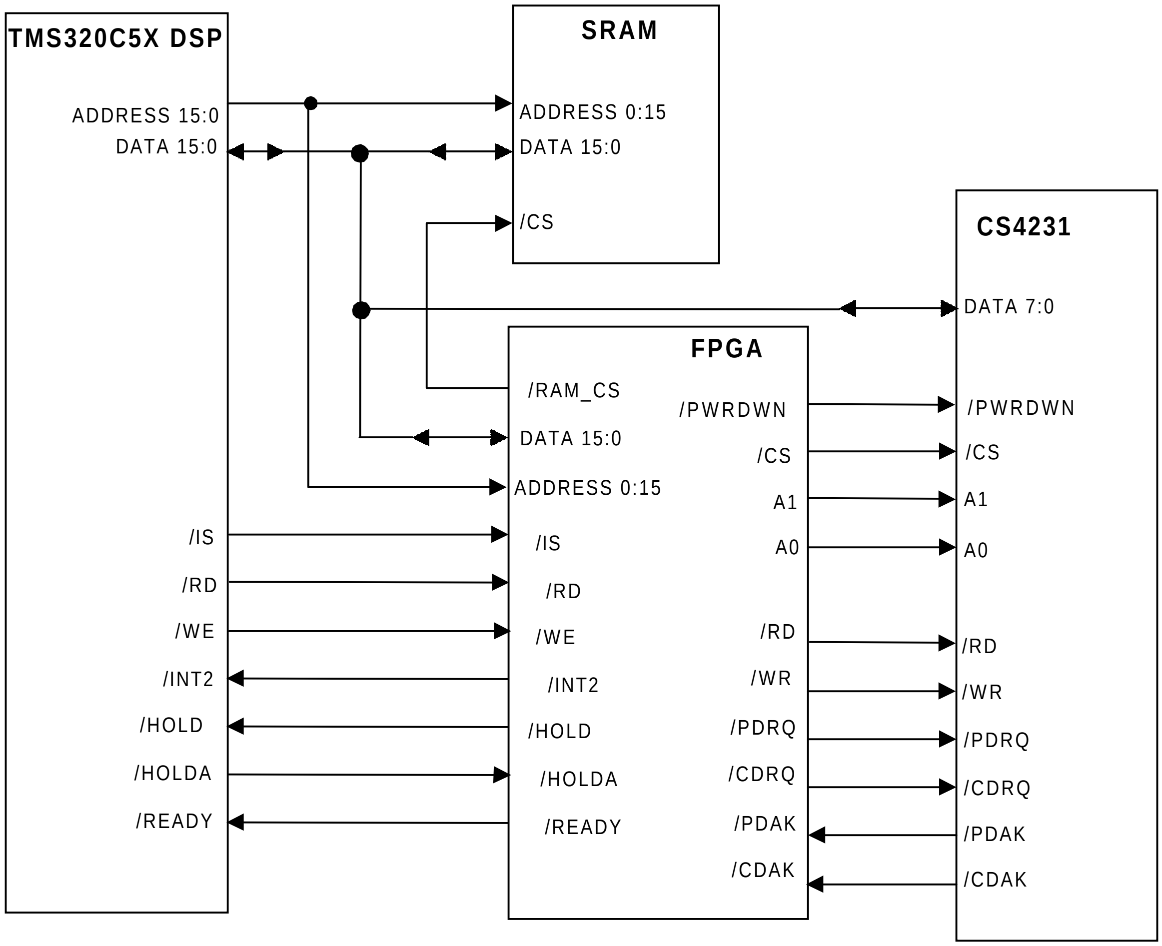
NOTE:

Note that in mode 1, the codec is placed in the PIO mode and uses the command interface. The command interface is there by default in a polled fashion and no additional pins are required. But while in modes 2 and 3, the DSP is in DMA mode and uses DMA pins (PDRQ/CDRQ and -PDAK/-CDAK). Note that these modes may be mixed, using one for playback and the other for capture if so desired.

System Block Diagram

The following block diagram shows how the digital aspects of how the DSP/codec system appears (Figure 1). All three modes are supported here. Some pins may be eliminated if just a particular mode is needed. These are described in more detail in the System Operation section.

Figure 1. Connection of TMS320C5x DSP to Parallel Stereo Codec



Three devices are connected to the TMS320C5x external bus: the SRAM, the codec and an FPGA. The SRAM is usually the most used, since the program is running from there. It is mapped into the program and/or data space (dual mapped in this case with a zero decode logic [4]). The four codec parallel Interface Registers R0-R3 are mapped into the DSP I/O space from 58h-5Bh. Various FPGA registers that control the codec (and other things) are mapped into DSP I/O space 50h-54h and 5Ch-5Fh.

System Registers and Bits

Table 1 describes the registers used in this implementation. Much of this comes from the original implementation and can be ignored. Registers with any codec relevant bits are in bold (for a full description of these registers please see reference [5]):

Table 1. Smart Registers

Register	Memory Address		Default	Access	
	PCMCIA Common Memory	'C5x I/O Space	Value	PC	DSP
SIGR	000400h		xxxx	R/W	
Reserved	000000h				
DSPCR	000002h		0108h	R/W	
DSPSR	000004h		0xxxh	R	
DSPTXD	000006h	0050h	0000h	R	W
DSPRXD	000008h	0051h	0000h	W	R
Reserved	00000Ah-00000Fh				
PCSR		**0052h**	**000xh**		**R**
BIOR		**0053h**	**000xh**		**R/W || R**
SYSCFG		**0054h**	**0080h**		**R/W**
Reserved		0055h-0057h			
CODEC R0		**0058h**	**xx80h**		**R/W**
CODEC R1		**0059h**	**xx80h**		**R/W**
CODEC R2		**005Ah**	**xx80h**		**R/W**
CODEC R3		**005Bh**	**xx80h**		**R/W**
ARR		**005Ch**	**0000h**		**R/W**
BKR		**005Dh**	**0000h**		**R/W**
AXR		**005Eh**	**0000h**		**R/W**
BKX		**005Fh**	**xx80h**		**R/W**

The codec relevant bits are described in the following few Tables 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11. Notice that they are in bold; please ignore the other bits. (Again for a full description of these all these registers and bits please see [5]). Note that some of the bits used for Mode 3 were placed in PC accessible registers DSPCR/DSPSR so that capture/play data was in shared memory that the PC could directly access. But this has not been stressed in this application note. See [5] for more detail.

Table 2. PC Status Register Definition

15-8	7	6	5	4	3	2	1	0
Reserved	**XH**	**RH**	**XINT**	**RINT**	**PDQR**	**CDQR**	RXFULL	TXEMPTY

Table 3. PCSR Explanation

Bit	Name	Description
2	**CDRQ**	This bit reflects the value of the stereo codec's CDRQ pin
3	**PDRQ**	This bit reflects the value of the stereo codec's PDRQ pin.
4	**RINT**	Receive (Capture) Interrupt: This bit becomes active when the DSP needs to service a capture interrupt. In Mode 2 (BRE=0) this means that the DSP must move data from the codec to memory, and the bit is cleared by doing enough of these moves to make CDRQ inactive. In Mode 3 (BRE=1) it means that the capture buffer from the codec is full and needs servicing by the DSP, and the bit is cleared when PCSR is read (So just do it once). The default value of this bit is 0.
5	**XINT**	Transmit (Playback) Interrupt: This bit becomes active when the DSP needs to service a playback interrupt. In Mode 2 (BXE=0) this means that the DSP must move data from memory to the codec, and the bit is cleared by doing enough of these moves to make PDRQ inactive. In Mode 3 (BXE=1) it means that the playback buffer from the codec is full and needs servicing by the DSP, and the bit is cleared when PCSR is read (So just do it once). The default value of this bit is 0.
6	**RH**	(Receive Half) This bit indicates which half of the buffer has been processed. It should be considered a full flag so if RH=0 then the first half of the buffer (defined by ARR and BKR) is full and needs to be processed. The default value is 0.
7	**XH**	(Transmit Half) This bit indicates which half of the buffer has been processed. It should be considered an empty flag so if RH=0 then the first half of the buffer (defined by AXR and BKX) is empty and needs to be filled. The default value is 0.
8	Reserved	These bits are reserved and may reflect any value that has no relevance to the operation of the card.

Table 4. Bit I/O Register Definition

15	14	13	12	11	10-8	7-4	3	2	1-0
-PDAK	**-CDAK**	**CODECA1**	**CODECA0**	**-PWRDWN**	O2-O0	Reserved	**PDRQ**	**CDRQ**	I1-I0

Table 5. Bit I/O Explanation

Bit	Name	Description
2	**CDRQ**	This bit reflects the value of the stereo codec's CDRQ pin.
3	**PDRQ**	This bit reflects the value of the stereo codec's PDRQ pin.
4-7	Reserved	These bits are reserved and may reflect any value that has no relevance to the operation of the card.
11	**-PWRDWN**	This bit output is directly connected to the -PWRDN pin of the codec for reset of the codec. The default is 0.
12-13	**CODEC A0** **CODEC A1**	These are the address bits to the codec that are latched by the FPGA to make the t_{ADHD} timing. Their corresponding pins will latch the last value of the address bits following a codec access. *They totally bypass the BIOR protocol (presently) their bit value has no meaning. There is no way to make them bit outputs.* The default is 0.
14	**-CDAK**	This bit reflects that value of the pin to codec -CDAK input when in Mode 2 (BRE=0 in SYSCFG Register). Thus this pin is used to manually toggle -CDAK in Mode 2. If in Mode 3 (BRE=1), then this bit will not reflect the -CDAK pin since the FSM will handle the toggling. The default is 0. Be sure to change to one in Mode 2 before -PWRDWN=1.
15	**-PDAK**	This bit reflects that value of the pin to codec -PDAK input when in Mode 2 (BXE=0 in SYSCFG Register). Thus this pin is used to manually toggle -PDAK in Mode 2. If in Mode 3 (BXE=1), then this bit will not reflect the -PDAK pin since the FPGA FSM will handle the toggling. The default is 0. Be sure to change to one in Mode 2 before -PWRDWN=1.

Table 6. DSP Control Register Bit Definitions

15-12	11-8	7	6	5-4	3	2	1-0
EXCP7-4 ❘❘ EXDP3-0	EXCP3-0 ❘❘ EXPP3-0	CPD/ -SPD	**BXE**	GLODAP 1-0	PHD	**BRE**	CLK DIVISOR 1-0

Table 7. SYSCFG Explanation

Bit	Name	Description
2	**BRE**	Receive Autobuffering Enable Bit. This bit enables the DMA for receive (capture) operation if BRE=1, i.e. Mode 3. This bit must be 0 for capture in mode 1 or 2 to occur. *Note that BRE is mutually exclusive to BXE.* The default is 0.
6	**BXE**	Transmit Autobuffering Enable Bit. This bit enables the DMA for transmit (playback) operation if BXE=1, i.e. Mode 3. This bit must be 0 for playback in mode 1 or 2 to occur. *Note that BXE is mutually exclusive to BRE.* The default is 0.

Stereo Codec Registers R0-R3:

- See [1] or [2] for definition of:
- Index Address Register
- Indexed Data Register
- Status Register
- PIO Data Register

Table 8. Address Receive Register (ARR) Bit Definitions

15-0
Receive (Capture) Buffer 16-Bit Beginning Address

The ARR 16 bit register (Table 8) gives the beginning address of the receive buffer for Stereo Codec external DMA.

Table 9. Receive Buffer (BXR) Size Register Bit Definitions

15-0
Receive (Capture) Buffer 16-Bit Ending Address

This 16 bit register (Table 9) gives the ending address of the receive buffer for Stereo Codec external DMA. (Note that this is slightly different than the BSP spec because of limited combinatorial gates in the FPGA).

Table 10. Address Transmit Register (AXR) Bit Definitions:

15-0
Transmit (Playback) Buffer 16-Bit Beginning Address

This 16 bit register (Table 10) gives the beginning address of the transmit buffer for Stereo Codec external DMA.

Table 11. Transmit Buffer (BXK) Size Register:

15-0
Transmit (Playback) Buffer 16-Bit Ending Address

This 16 bit register (Table 11) gives the ending address of the transmit buffer for Stereo Codec external DMA. (Note that this is slightly different than the BSP spec because of limited combinatorial gates in the FPGA).

SYSTEM OPERATION

The architecture described in this Application Note allows for the use of all three modes. DSP specific system level considerations can be found in various literature items available from TI, especially [6].

From the codec standpoint, it needs to be programmed after coming out of reset (unfortunately misnamed as -PDWN). This operation is done by addressing the R0-R3 registers in decoded logic and can be observed in the code section for all modes. Basically this programming involves setting of codec I/O channels, data format, gains, etc. since the huge flexibility in these parts rarely allow the system to be up and running, as is seen with more classic DSP codecs such as the TLC3204x family. Use of IN/OUT instructions was essential in the design because of the relatively slow FPGA accesses when compared to that of the DSP, especially turn-off time. Memory-mapped I/O accesses using the DSP are NOT advised.

Another system aspect that should be appreciated is how slow the codec cycles are (several hundred nanoseconds) compared to a 40 or 50 MIP DSP. This aspect is due to the analog noise margins that increased digital speeds would incur. This fact drove the decision to ignore certain functions that are provided in the codec (such as capture and play ready bits in modes 2 and 3, thus ignoring the codec INT pin) and to move these functions to the FPGA. The assumption is that when the interface logic is made discrete, or more likely into an ASIC or cDSP, the system will run operate faster.

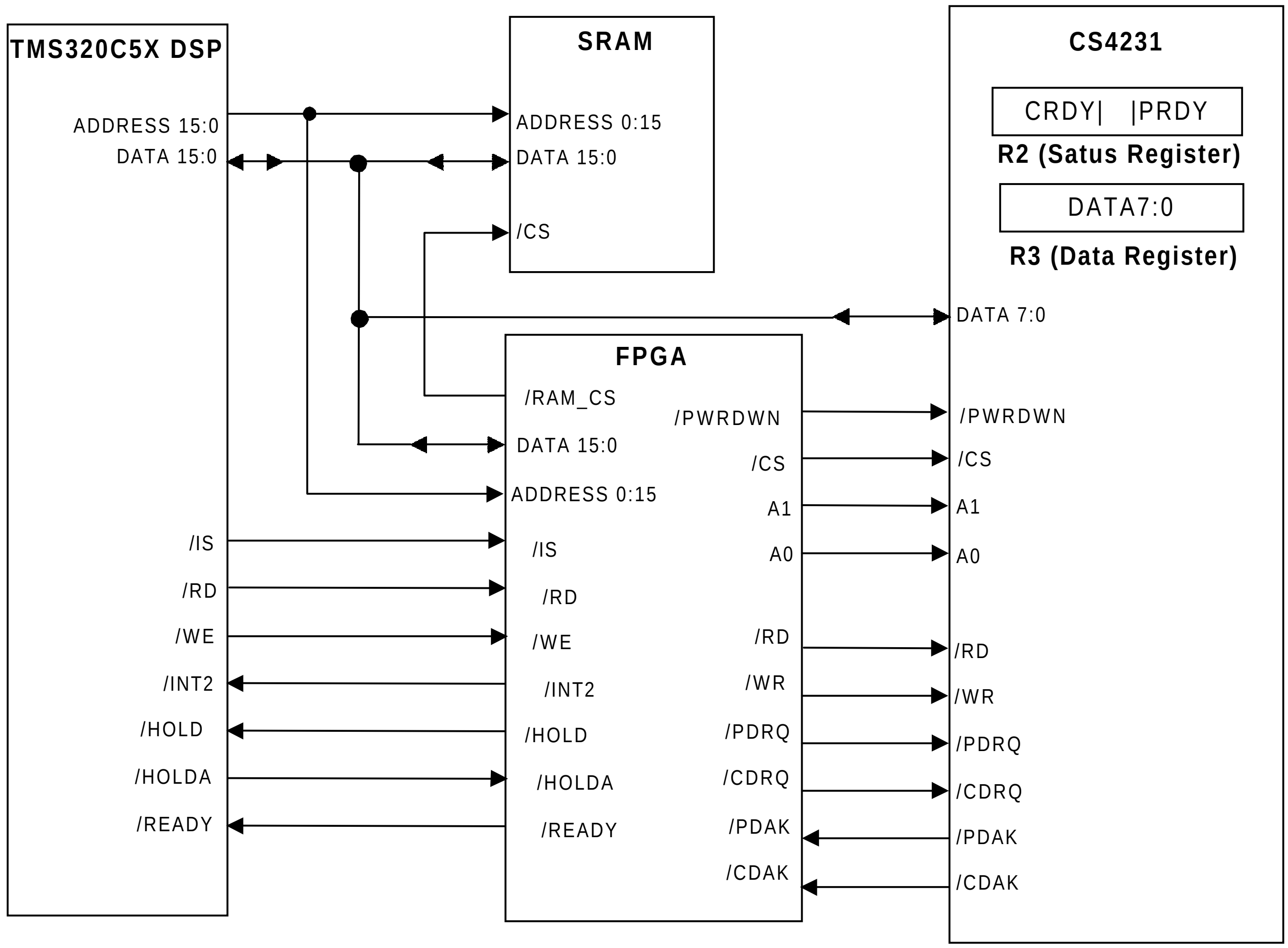
Mode 1 Programmed I/O

From a hardware standpoint, the decode necessary for the programming makes this almost a default mode. The playback function is readily available since the DSP is writing to the codec. But in the case of capture, the t_{ADHD} address hold timing from the DSP is too fast for the codec which, as mentioned earlier, was made for a slower PC bus. Thus latching of the address lines for the codec is essential for reliable capture.

NOTE:

Figure 2 shows mode 1 PIO, whose only difference from the main block diagram in Figure 1 is the stress on stereo codec Direct Registers.

Figure 2. Mode 1 - PIO Connection of TMS320C5x Parallel Stereo Codec



From the software standpoint, the codec must first be programmed into the PIO mode and various bits polled. Then PIO Capture and Playback bits (CRDY/PRDY) in the codec Status Register R2 (called Direct 2 Status on AD65) is polled by the DSP in a tight loop before receiving or sending data. Obviously this mode ends up becoming slow since it is poll driven. Also note that each of these codec accesses are in the hundreds of nanoseconds, slow to a DSP. Code for PIO that does a simple digital loop back can be seen in Appendix A.

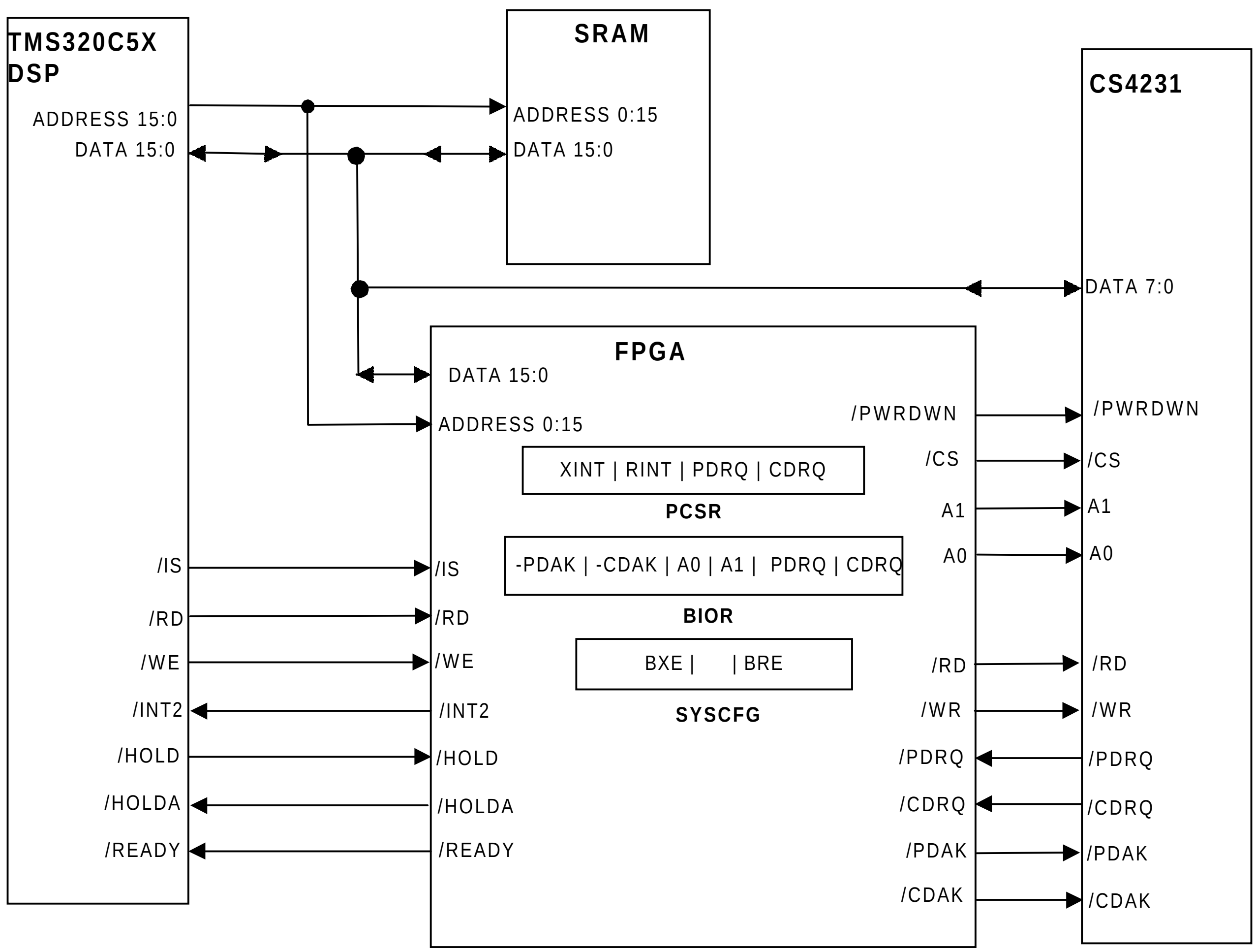
```
Loop: Test codec Status Register R2 for CRDY/PRDY
      Read/write data from/to codec Data Register R3
      Goto Loop
```

Thus the other two modes place the codec in the DMA mode using the PDRQ and CDRQ pins. As mentioned earlier, the INT pin is not used because it does not differentiate between capture and play. This function was moved to the FPGA. This is not much faster, but was done based on the assumption that an ASIC/cDSP implementation was the final goal. Note that though the next two modes use the same pins, they handle data movement quite differently.

Mode 2 Interrupt Driven

In this mode after putting the codec into DMA mode the DSP handles not only the data movement interrupt-based, but also the handshake with DMA acknowledge pins -CDAK and -PDAK. The following Figure 3 shows the relevant configuration.

Figure 3. Mode 2 - Interrupt Connection of TMS320C5x Parallel Stereo Codec



The PDRQ/CDRQ requests have been multiplexed to the same DSP interrupt pin -INT2. The CDRQ and PDRQ bits in the DSPSR register must then be read and the appropriate action taken. Whether capture or play, the DSP must write to the CDAK or PDAK bits in the SYSCFG register to manually exercise the DMA mode. Thus the following sequence must occur in the ISR, assuming simultaneous capture and play.

1) Read PCSR and test if CDRQ, PDRQ or both are high.
2) Write 0 to -CDAK/-PDAK bit in SYSCFG (based on whichever is higher priority) to acknowledge the DMA cycle.
3) Read/write data value from/to codec before/after writing/reading to memory.
4) Write 1 to -CDAK/-PDAK bit in SYSCFG to end DMA cycle acknowledgement.
5) Repeat for number of bytes.

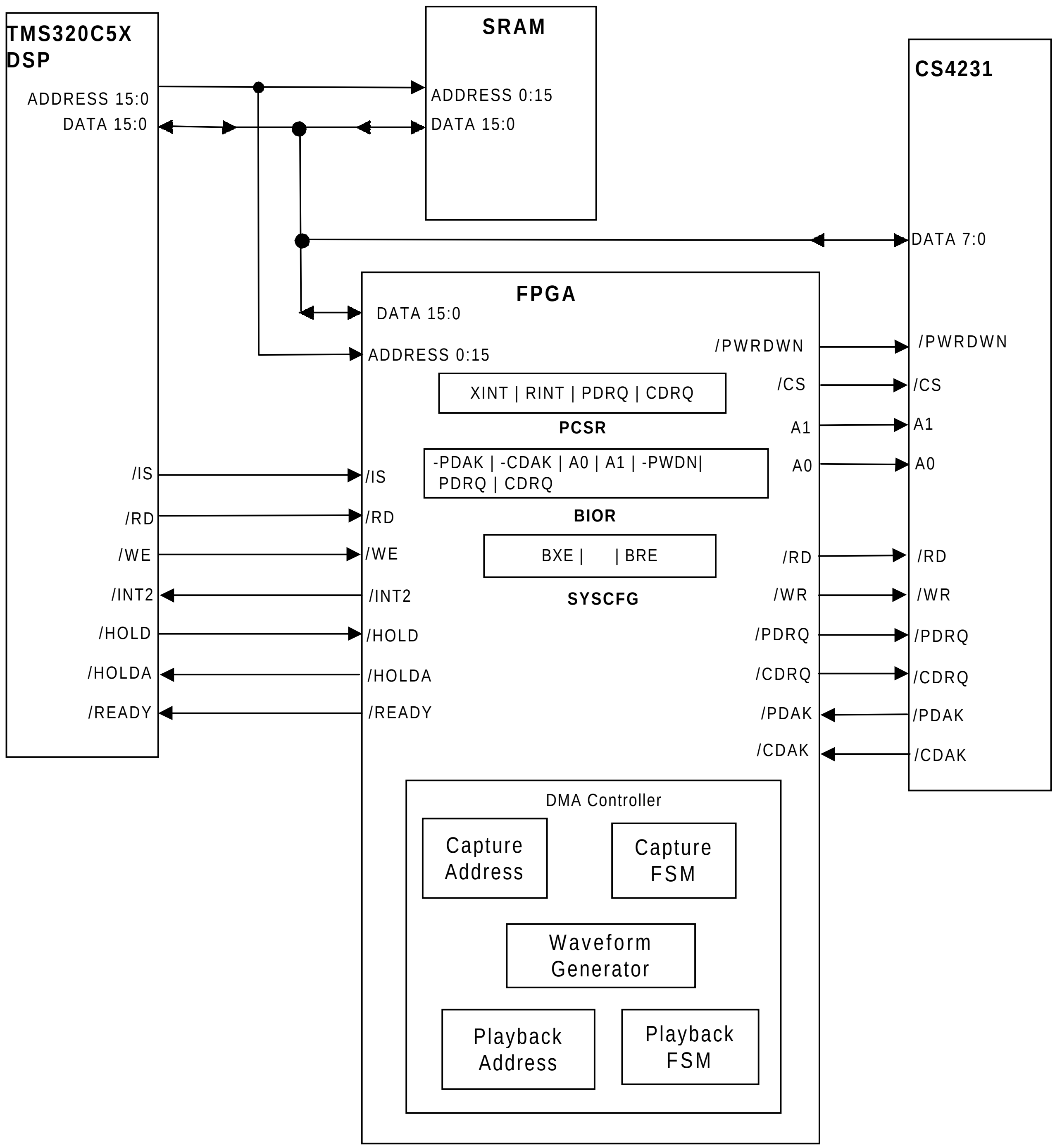
The last step is required since FPGA will pend and interrupt the DSP if PDRQ or CDRQ is high. If the DSP chooses to mask this interrupt, then data will be lost. Code that exercises some simple digital loopback in this mode may be found in Appendix A.

Mode 3 Stereo Codec DMA

In this mode the DSP is taken off the bus (using the -HOLD/-HOLDA protocol) and the data is copied back and forth between the DSP and memory. A DMA controller is designed into the FPGA that consists of three parts each for receive and transmit. These parts are: a finite state machine to act as the intelligent entity, address generation blocks, and wave form generation module. Parts of the wave form generation module can be shared between capture and playback parts since the single bus prevents them from occurring at the exact same time (even though they may run concurrently). If there is any conflict for resources, priority has been given to the capture module.

The following figure shows how these modules are constructed in the FPGA.

Figure 4. Mode 3 - DMA Controller Connection of TMS320C5x Parallel Stereo Codec



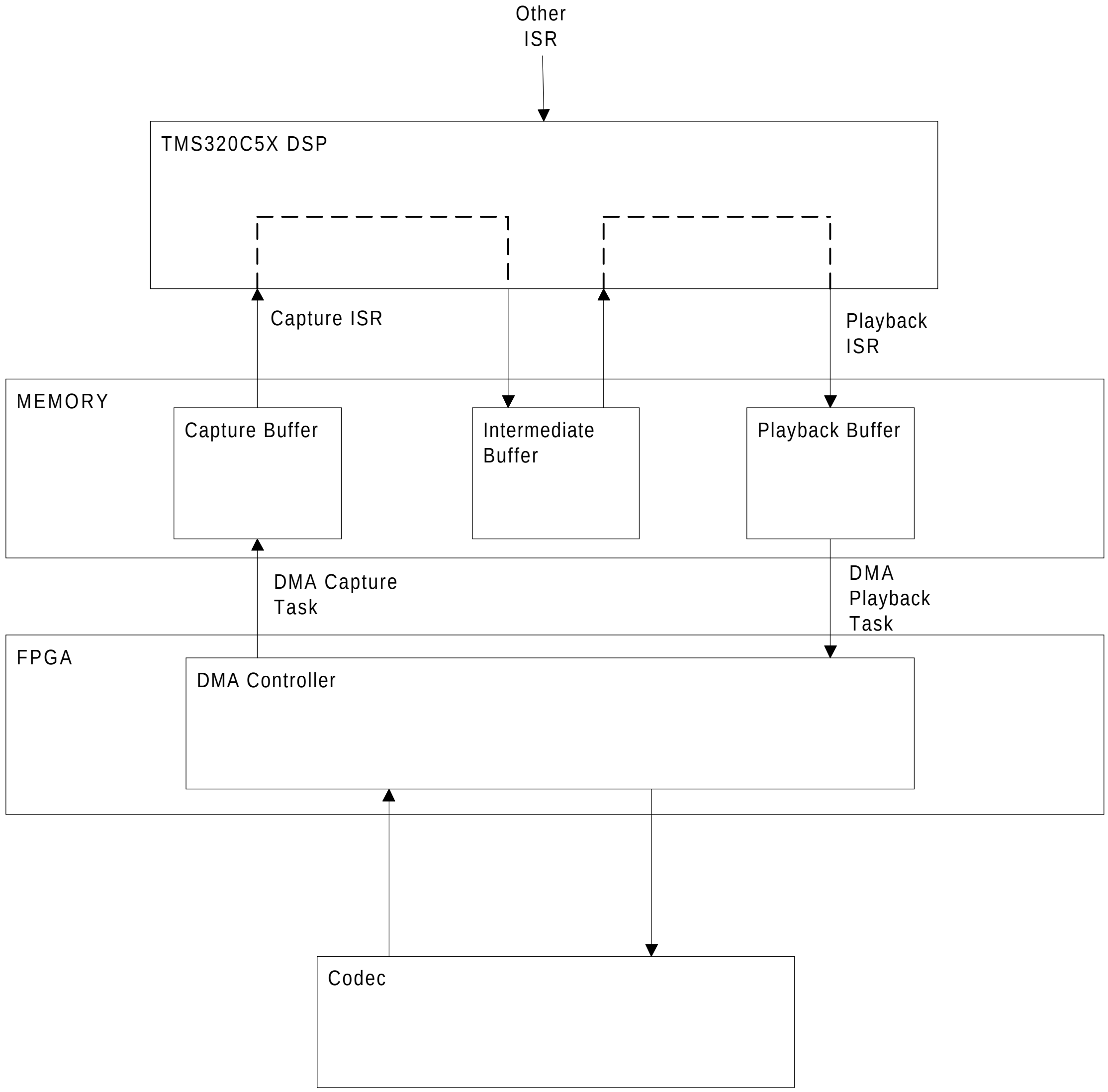
As mentioned earlier, this logic was designed to look similar to [7]. The capture and playback buffers may be placed anywhere in external memory. The DSP will be interrupted when the DMA controller reaches either the halfway point or end of the buffer. Then the RH/XH flag must be read to determine which half of the buffer needs servicing. The DMA controller for capture or play is enabled by the BRE and BXE flags respectively. The schematics for the FSM's and Address Generators can be seen in Appendix B. A full description of this would be found in [9].

From a software standpoint, after the DMA is setup the DSP needs to deal with the data only when a buffer is full or empty. The code looks similar to the interrupt driven except that the acknowledges are handled by the FSM and large buffers of data are used. Thus the following sequence must occur in the ISR, assuming simultaneous capture and play.

1) Read PCSR and test if RINT, XINT or both are high.
2) Test RH or XH to see which buffer may now be manipulated.
3) Change buffer pointers for main code to deal with these buffers.

In Appendix A, some code may be found that does some simple digital loopback. Appendix A also has a bit more interesting loopback code (Figure 5).

Figure 5. Mode 3 - Test Code in Appendix A (Digital Loopback with DMA)

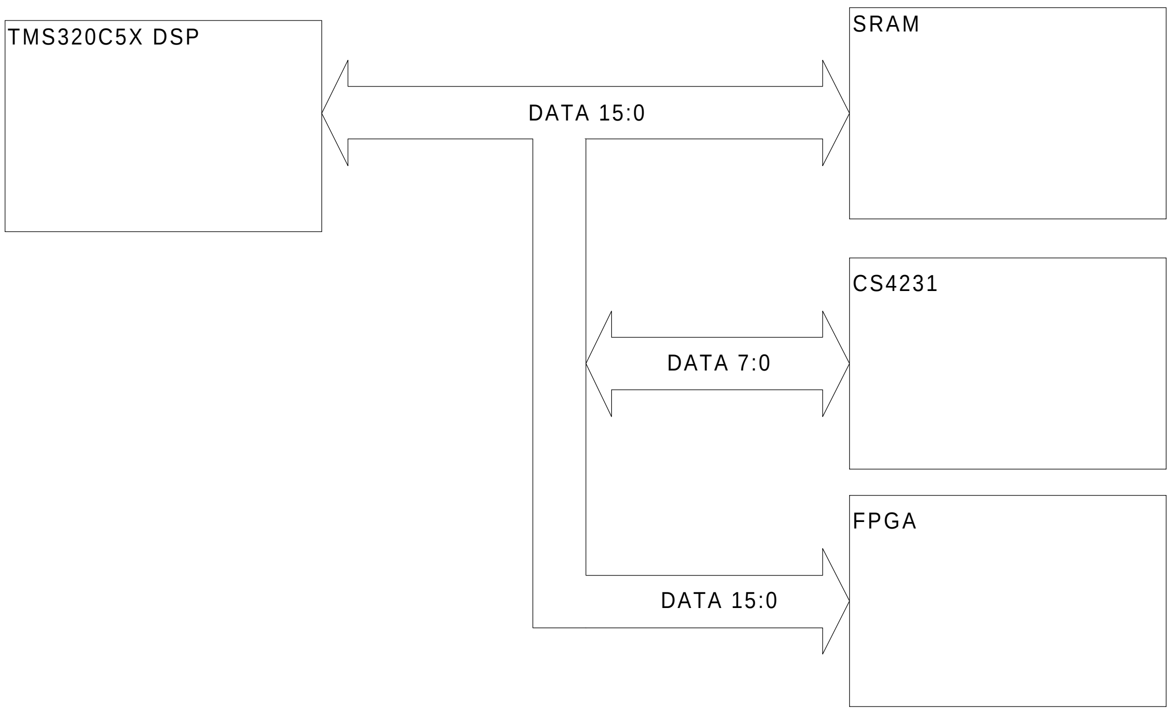


That sets up an intermediate buffer so that the capture ISR copies data from the capture buffer and puts it into the intermediary buffer. Meanwhile the playback ISR copies data from the intermediary buffer to the playback buffer. While all this is happening, the FSM's are copying data to and from the codec from the capture and playback buffers. Also a second interrupt is occurring while all this is happening.

SYSTEM CONSIDERATIONS

The basic system described previously in this Application Note looks as follows in Figure 6.

Figure 6. Basic DSP/SRAM/Codec System



The problem with this architecture, which should minimize the time that the DSP is not accessing the SRAM, is compounded by the relatively slow access time of the stereo codec in relation to the codec. Three possible methods of speeding up this interface are described below, with different costs:

1) FIFO
2) External DMA
3) Serial Port (with internal DSP DMA)

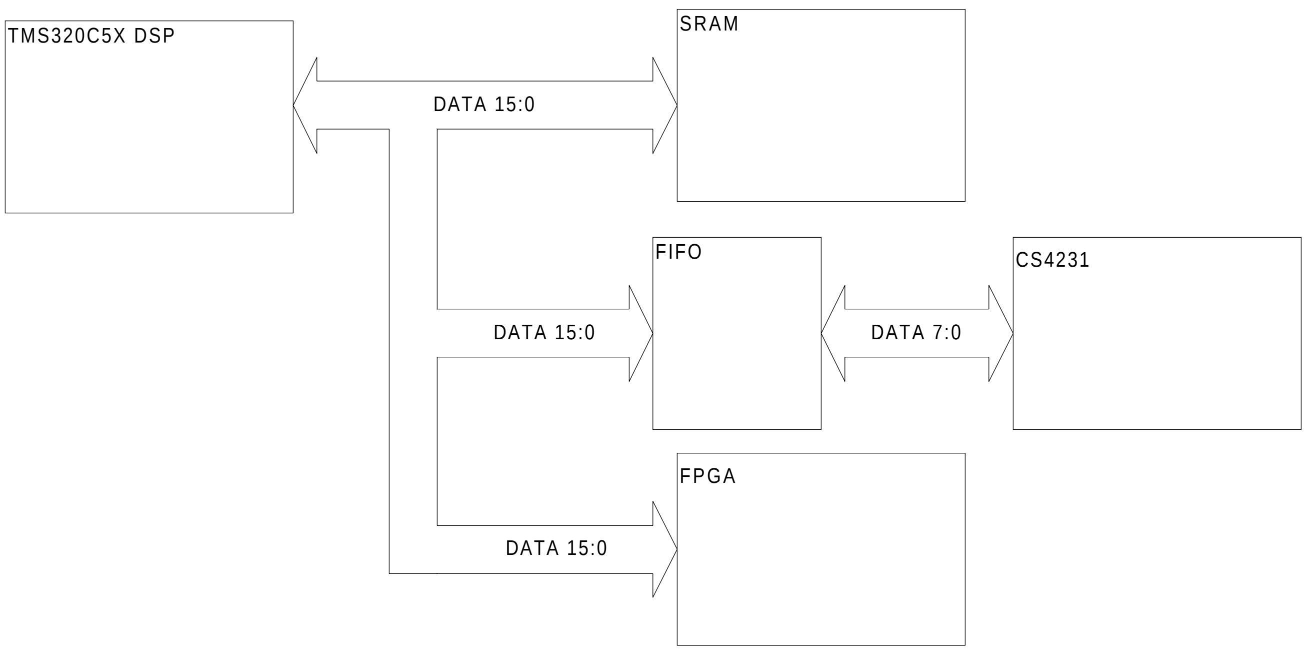
NOTE:

Note that they all ignore the speed of the FPGA/ASIC/logic (which will be addressed later) and that only two were implemented in this paper.

FIFO

In this implementation a FIFO is added as a classical mismatched speed bus interface solution. If the FIFO is fast enough, then the DSP can access data at zero wait states even though the codec interface is in the hundreds of nanoseconds. This method is probably the medium simplicity though higher cost since an additional part (and also interface glue logic) will need to be added. See Figure 7.

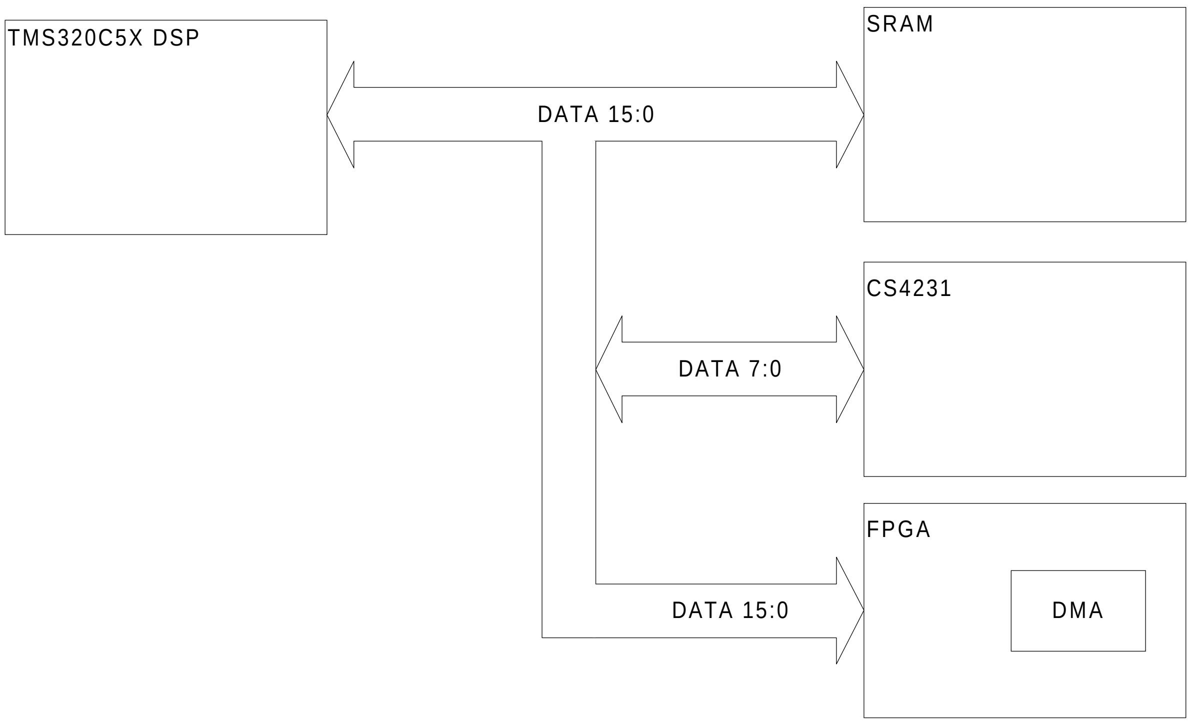
Figure 7. FIFO DSP/SRAM/Codec System



External DMA

This implementation was discussed in detail in this paper. It takes the burden off the DSP to do the data moving work, but is quite complicated. It is not that expensive, since existing logic may be used. This method is probably the most complex, and is medium-to-higher cost since much design and added logic are needed. See Figure 8. Note that TMS320 DSPs with their own DMA controllers exist, but much data is still going over the same bus, hence limiting the bandwidth.

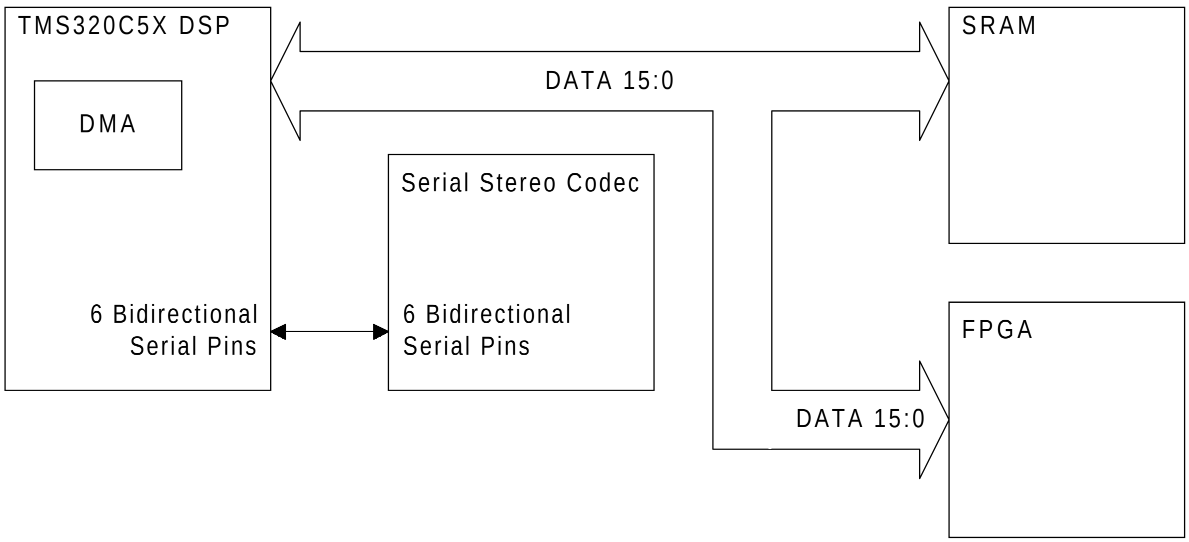
Figure 8. External DMA DSP/SRAM/Codec System



Serial Codec with DSP “DMA”

In this implementation an alternative serial bus is given to the codec so that it does not conflict with the SRAM. It is important here for the DSP to have some sort of bi-directional DMA capability, which may be seen in TMS320C5x/C54x parts with the bufferred serial port or TMS320C32/TMS320C6xxx with multiple DMA channels (see Figure 9). This system probably makes the most sense for the mass market customer who is not doing a cDSP/ASIC.

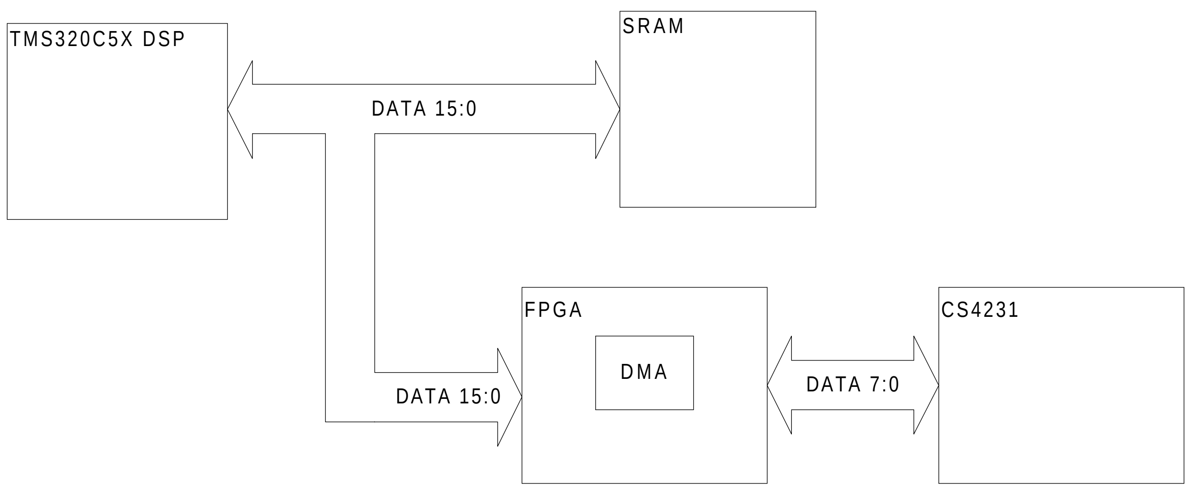
Figure 9. DSP with “DMA”/SRAM/Serial Codec System



Parallel Codec with Separate Bus and External DMA Optional

This implementation should be mentioned as the ideal for performance. A separate data bus is created through logic to feed the DSP. Thus the DSP is protected from the slowness of the codec in a FIFO like style if a fast ASIC is used for the logic. The use of the DMA is optional based on the system. Of course this implementation will cost the most pins. This implementation probably makes the most sense for the cDSP customer.

Figure 10. DSP/SRAM/Parallel Codec with Separate Bus and External DMA System



CONCLUSION

There are MANY ways to hook up these two parts. The important thing is to understand the parts and find the optimum configuration from both hardware AND software aspects. The limiting tradeoffs are those between design complexity and bus bandwidth.

REFERENCES

1) Crystal Data Book
2) TLC320AD65 Data Sheet
3) TMS320C5x User's Guide (p. HOLD)
4) Designer Notebook Page #4x.
5) PCMCIA DSP/Memory Card for the TMS320C5x Specification
6) PCMCIA DSP/Memory Card for the TMS320C5x Application Note
7) TMS320C57 Buffered Serial Port Spec
8) TLC320AD65 Data Sheet - P_PORT Addendum
9) FPGA Spec

Appendix A - Code

The following section has all sorts of code that was written for a system based on [5]. Note in all cases that the codec is first initialized in either PIO or DMA mode and then put into its function. Note that there is a good amount of redundancy in the code, but all of it is fully functional.

Mode 1 - PIO Digital Loopback

This code initializes the codec in PIO mode with appropriate inputs and gains and then uses mode 1 - PIO to do the digital loopback one sample at a time.

```
                .mmregs
                .include "cs4231.inc"

LEFT_ADC_GAIN   .set  02h
CFS             .set  CS4231_48KHZ

                .sect "vectors"

                b         init      ;00; Reset.
                .space    2 * 16    ;02; Vector for external interrupt
                                          ; INT1.
                .space    2 * 16    ;04; Vector for external interrupt
                                          ; INT2.
                .space    2 * 16    ;06; Vector for external interrupt
                                          ; INT3.
                .space    2 * 16    ;08; timer interrupt vector - TINT
                .space    2 * 16    ;0A; Serial port receive interrupt
                                          ; RINT
                .space    2 * 16    ;0C; Serial port transmit
                                          ; interrupt XINT
                .space    4 * 16    ;0E; vectors for TDM port TRNT,
                                          ; TXNT
                .space    2 * 16    ;12; vector for external interrupt
                                          ; INT4
                .space    14 * 16   ;14; Reserved space. Should not be
                                             ; used.
                .space    2 * 16    ;22; Trap instruction vector
                .space    2 * 16    ;24; Non Maskable Interrupt vector
                .space    4 * 16    ;26; Reserved space. Should not be
                                          ; used.

                .text
init:
                ldp       #0
                splk      #0000000110b, IMR
                clrc      OVM       ; Disable overflow saturation mode
                splk      #001Fh, CWSR
```

```
                    splk    #0000h, PDWSR
                    splk    #0FFFAh, IOWSR       ; 3 wait (I/O)

                    call    _InitAIC

                    lacc    #08000h  ; set up circular buffer pointers
                    sacl    CBSR1    ; buffer1 = 8000 -> ffff
                    lacc    #0FFFFh  ;
                    sacl    CBER1    ;
                    lar     AR7,#08000h ; set receive buffer
                    lacc    #11101111b  ;
                    sacl    CBCR

                    lacc    #0
                    mar     *, AR7
                    rpt     #8000h
                    sacl    *+

                    SET_CS4231  I9, CS4231_ENABLE
                                     ; Capture and Playback enabled.

                    Clrc    INTM     ; Globally enable interrupts

Loop:
                    in      TEMP0, CS4231_R2
                    nop
                    lacl    TEMP0
                    and     #00100000B
                    bcnd    Capture, NEQ

                    lacl    TEMP0
                    and     #00000010B
                    bcnd    Playback, NEQ
                    b       Loop

Capture:
                    in      *, CS4231_R3
                    lacl    *
                    and     #0FFh
                    sacb

                    in      *, CS4231_R3
                    lacc    *, 8
                    sacl    TEMP0
                    lacb
                    or      TEMP0
                    sacl    *

                    b       Loop


Playback:
```

```
                 lacc     *+
                 sacl     TEMP0
                 out      TEMP0, CS4231_R3
                 rpt      #07h
                 sfr
                 sacl     TEMP0
                 out      TEMP0, CS4231_R3
                 b        Loop


;*****************************************************************
;
; _InitAIC() - AIC Initialization Routine
;
;*****************************************************************
_InitAIC:
                 zap
                 sacl     TEMP1       ; TEMP1 <- clears MCE

                 lacc     #8F00h      ; Take out of reset
                 sacl     TEMP0
                 out      TEMP0, CS4231_PDWN
                 rpt      #0F000h     ; Wait a while longer (Dag)
                 nop

Wait_80:
                 in       TEMP0, CS4231_R1
                 nop
                 cpl      #80h, TEMP0
                 bcnd     Wait_80, TC
;
; ====== Clear previous states ================
;
                 IS_CS4231_RDY TEMP0 ; Wait for PIO ready

                 SET_CS4231    I9, 0 ; Disable playback and capture
                 out           TEMP0, CS4231_R2
                                     ; Clear interrupts

; ====== Configure CODEC ======================

                 IS_CS4231_RDY TEMP0

                 SET_CS4231 SETMCE | I12, CS4231_SETMODE2
                                           ; Set the MODE2

                 SET_CS4231 SETMCE | I9, CS4231_P_PIO | CS4231_C_PIO


                 SET_CS4231 SETMCE | I28, CS4231_MONO | CS4231_16LINEAR
                                       ; Set The Capture mode to
                                       ; 16-bit linear
```

```
                  SET_CS4231  SETMCE | I8, CFS | CS4231_MONO |
                                                   CS4231_16LINEAR
                                        ; Set 16-bit, mono.

                  rpt      #0FFFFh
                  nop

                  IS_CS4231_RDY  TEMP0    ; Clocks must resynch.

                  SET_CS4231  SETMCE | I0, CS4231_MIC | LEFT_ADC_GAIN
                        ; Left ADC input source select : Microphone.

                  SET_CS4231  SETMCE | I6, CS4231_SETGAIN
                           ; Left DAC attenuator.
;
; ====== Recalibrate CS4231 =====================
;
                  lacc     #SETMCE | I9   ; Set up R0 by setting Index
                                          ;   to I9
                  sacl     TEMP0          ; and setting MCE.
                  out      TEMP0, CS4231_R0
                                          ; Set the ACAL bits in

                  in    TEMP0, CS4231_R1
                  nop
                  lacl  TEMP0    ; Set ACI bit in I9 without clearing
                  or    #CS4231_SETACAL  ; any other bits. This will
                  sacl  TEMP0         ; cause device to recalibrate.
                  out   TEMP0, CS4231_R1
                  nop
                  out   TEMP1, CS4231_R0
                           ; Clear MCE to initiate calibration
                  nop

Wait_cal:
                  in    TEMP0, CS4231_R1
                  nop
                  cpl   #80h, TEMP0
                  bcnd  Wait_cal, TC

                  lacl  #I11
                  sacl  TEMP0
                  out   TEMP0, CS4231_R0
                                    ; Wait until the device is
Notdone:                            ; finished calibrating.
                  in    TEMP0, CS4231_R1
                  nop
                  bit   TEMP0, BIT5       ; This bit indicates the
                                          ;  status of calibration.
                  bcnd  Notdone, TC
                  ret
```

Mode 2 - Interrupt Driven Digital Loopback

This code initializes the codec in DMA simultaneous capture/playback mode with appropriate inputs and gains and then uses mode 2 - Interrupt Driven to do the digital loopback one sample at a time. Again the DSP is doing the moving based on interrupts from the codec based on PDRQ/CDRQ. Also the DSP must manually toggle -CDAK/-PDAK lines.

* This uses DMA mode   - Interrupt PDRQ

```
                  .mmregs
                  .sect "vectors"

                 B  start      ;00; RESET
INT1             B  pc_isr     ;02; vectors for interrupts int1-int3
INT2             B  codec_isr
INT3             B  INT3
                 .space  2 * 16  ;08; timer interrupt vector - TINT
receive B        receive       ;0A; Serial port receive interrupt
                                          ; RINT
transmit B       transmit      ;0C; Serial port transmit interrupt
                                          ; XINT
                 .space  4 * 16  ;0E; vectors for TDM port TRNT,
                                          ; TXNT
                 .space  2 * 16  ;12; vector for external interrupt
                                          ; INT4
                 .space  14 * 16     ;14; Reserved space. Should
                                          ; not be used.
                 .space  2 * 16      ;22; Trap instruction vector
                 .space  2 * 16      ;24; Non Maskable Interrupt
                                          ; vector
                 .space  4 * 16      ;26; Reserved space. Should
                                          ; not be used.

                 .text
start:           LDP  #0
                 LACC #0000h        ; Reset codec
                 SACL 61h
                 OUT  61h, 53h
                 RPT  #0f000h       ; Wait
                 NOP
                 LACC #0c800h  ;Take out of reset & -CDAK/-PDAK high
                 SACL 61h
                 OUT  61h, 53h
                 RPT  #0f000h       ; Wait a while longer (Dag)
                 NOP
stuck:           IN   61h, 58h      ; Checking for 80h
                 LACL 61h
                 AND  #0080h
                 BCND stuck, NEQ
```

```
                  LACC #0009h   ; Flip MCE bit=1 and Register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0008h   ; Switch to DMA Dual Channel Mode
                  SACL 61h
                  OUT  61h, 59h

                  LACC #000bh   ; Flip MCE bit=0 and Register 11
                  SACL 61h
                  OUT  61h, 58h

stuck1:           IN   61h, 58h    ; Checking for 80h
                  LACL 61h
                  AND  #0080h
                  BCND stuck1, NEQ

stuck2:           IN   61h, 58h    ; Checking for 20h on register 11
                  LACL 61h
                  AND  #0020h
                  BCND stuck2, NEQ

                  LACC #0009h   ; Set to register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #000bh   ; Enable any playback/capture
                  SACL 61h
                  OUT  61h, 59h

                  LACC #0000h   ; Set to Left Input Register
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0000h   ; Write to Left Input Register
;Making this 7h sounds better, but codec cuts out!
                  SACL 61h
                  OUT  61h, 59h
                  LACC #0006h   ; Set to Left Output Register
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0000h   ; Write to Left Output Register
                  SACL 61h
                  OUT  61h, 59h

                  lacc  #08000h ; set up circular buffer pointers
                  sacl  CBSR1   ; buffer1 = 8000 -> ffff
                  lacc  #08000h
        sacl    CBSR2
                  lacc  #0800fh ; buffer2 = ffff -> 8000
                  sacl  CBER1
                  lacc  #0800fh ; buffer2 = ffff -> 8000
        sacl    CBER2
                  lar   AR6,#08000h      ; set receive buffer
                  lar   AR7,#08000h      ; set transmit buffer
```

```
                 lacc  #0000000011111110b  ; buf1=AR7, buf2=AR6,
                                              ; enable both
                 sacl  CBCR                ;
; Set up block registers
                 LACC  #8000h              ; ARR Value
                 SACL  61h
                 OUT   61h, 5Ch            ; Write
                 LACC  #08010h             ; BKR Value
                 SACL  61h
                 OUT   61h, 5Dh            ; Write
                 LACC  #0a000h             ; AXR Value
                 SACL  61h
                 OUT   61h, 5Eh            ; Write
                 LACC  #0a010h             ; BKX Value
                 SACL  61h
                 OUT   61h, 5Fh            ; Write

;Set up DMA and Interrupts
                 LACC  #0ffffh
                 SACL  IFR
                 LACC  #02h        ; mask on PC and codec ISR
                 SACL  IMR
                 IN    61h, 51h    ; bogus read from DSPRXD
                 LACC  #080h       ; disable BXE and BRE
                 SACL  61h
                 OUT   61h, 54h
                 CLRC  INTM

loop:            NOP
                 IN   61h, 51h     ; bogus read from DSPRXD
                 OUT  61h, 50h     ; bogus write to DSPTXD
                 NOP
                 B  loop

                 MAR  *, AR6
loop1:           IN   61h, 5ah
                 LACL 61h
                 AND  #20h
                 BCND loop1, EQ
                 IN   *, 5bh        ; Read data
loop2:           IN   61h, 5ah
                 LACL 61h
                 AND  #02h
                 BCND loop2, EQ
                 OUT  *+, 5bh      ; Write data
;                RPT  #5000        ; Wait
;                NOP
                 B loop1
playback:
                 RPT  #5000        ; Wait
                 NOP
                 OUT  *+, 5bh      ; Write data
```

```
                B     playback

pc_isr:         NOP
                SETC XF
                RPT  #100
                NOP
                CLRC XF
                RETE

; This one is for Level2
codec_isr:
                SETC  XF
                NOP
                IN    60h, 52h       ; Read PCSR
                LACL  60h
; Need to change these bits, eh!
                AND   #0004h         ; Test CDRQ
                BCND  cap_serve, NEQ
ptest:          LACL  60h            ; Test PDRQ
                AND   #0008h
                BCND  play_serve, NEQ
isr_done NOP
                CLRC  XF
                RETE

cap_serve:
                MAR   *,AR6
                LACC  #08800h           ; -CDAK low
                SACL  61h
                OUT   61h, 53h
                IN    61h, 5bh          ; Read Value
                LACL  61h
                SACL  *+
                LACC  #0c800h           ; -CDAK high
                SACL  61h
                OUT   61h, 53h
                B     ptest

play_serve:
                MAR   *,AR7
                LACC  #04800h           ; -PDAK low
                SACL  61h
                OUT   61h, 53h
                LACL  *+
                SACL  61h
                OUT   61h, 5bh          ; Play Value
                LACC  #0c800h           ; -PDAK high
                SACL  61h
                OUT   61h, 53h
                B     isr_done
```

Mode 3 - Stereo Codec DMA Digital Loopback

This code initializes the codec in DMA simultaneous capture/playback mode with appropriate inputs and gains and then uses mode 3 - Stereo Codec to do the digital loopback one sample at a time. But here the DMA controller is doing the moving based on CDRQ/PDRQ, not the DSP. Capture and playback buffers are set as one and the same (8000h-B000h in DSP data memory). The DSP processes the data based on interrupts from the DMA controller. Also the FSM's in the DMA Controller must toggle the -CDAK/-PDAK lines.

* This uses DMA mode

```
                 .mmregs
                 .sect "vectors"

                 B  start      ;00; RESET
INT1             B  pc_isr     ;02; vectors for interrupts int1-int3
INT2             B  codec_isr
INT3             B  INT3
                 .space   2 * 16   ;08; timer interrupt vector - TINT
receive B        receive    ;0A; Serial port receive interrupt RINT
transmit B       transmit   ;0C; Serial port transmit interrupt XINT
                 .space 4 * 16 ;0E; vectors for TDM port TRNT, TXNT
                 .space 2 * 16 ;12; vector for external interrupt
                                      ; INT4
                 .space   14 * 16  ;14; Reserved space. Should not
                                      ; be used.
                 .space   2 * 16   ;22; Trap instruction vector
                 .space   2 * 16   ;24; Non Maskable Interrupt vector
                 .space   4 * 16   ;26; Reserved space. Should not be
                                      ; used.

                 .text
start:           LDP  #0
                 LACC #0c000h         ; Reset codec
                 SACL 61h
                 OUT  61h, 53h
                 RPT  #0f000h         ; Wait
                 NOP
                 LACC #0c800h         ; Take out of reset
                 SACL 61h
                 OUT  61h, 53h
                 RPT  #0f000h         ; Wait a while longer (Dag)
                 NOP
stuck:           IN   61h, 58h        ; Checking for 80h
                 LACL 61h
                 AND  #0080h
                 BCND stuck, NEQ
```

```
                  LACC #0049h   ; Flip MCE bit=1 and Register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0008h   ; Switch to DMA Dual Channel Mode
                  SACL 61h
                  OUT  61h, 59h

                  LACC #0048h   ; Set to register 8
                  SACL 61h
                  OUT  61h, 58h
;                 LACC #0040h   ; 16 `bit mono
                  LACC #0000h   ;  8 `bit mono
                  SACL 61h
                  OUT  61h, 59h

                  LACC #000bh   ; Flip MCE bit=0 and Register 11
                  SACL 61h
                  OUT  61h, 58h

stuck1:           IN   61h, 58h ; Checking for 80h
                  LACL 61h
                  AND  #0080h
                  BCND stuck1, NEQ

stuck2:           IN   61h, 58h ; Checking for 20h on register 11
                  LACL 61h
                  AND  #0020h
                  BCND stuck2, NEQ

                  LACC #0009h   ; Set to register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #000bh   ; Enable any playback/capture
                  SACL 61h
                  OUT  61h, 59h

                  LACC #0000h   ; Set to Left Input Register
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0000h   ; Write to Left Input Register
;Making this 7h sounds better, but codec cuts out!
                  SACL 61h
                  OUT  61h, 59h
                  LACC #0006h   ; Set to Left Output Register
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0000h   ; Write to Left Output Register
                  SACL 61h
                  OUT  61h, 59h

                  lacc  #09000h    ; set up circular buffer pointers
                  sacl  CBSR1      ; buffer1 = 8000 -> ffff
```

```
        sacl    CBSR2                       ;
                 lacc  #0900fh              ; buffer2 = ffff -> 8000
                 sacl  CBER1                ;
        sacl    CBER2                       ;
                 lar   AR6,#09000h          ; set receive buffer
                 lar   AR7,#09000h          ; set transmit buffer
                 lacc  #0000000011101111b  ; buf1=AR7, buf2=AR6,
                                               ; enable both
                 sacl  CBC
; Set up block registers
                 LACC  #8000h      ; ARR Value
                 SACL  61h
                 OUT   61h, 5Ch    ; Write
                 LACC  #0b000h     ; BKR Value
                 SACL  61h
                 OUT   61h, 5Dh    ; Write
                 LACC  #8000h      ; AXR Value
                 SACL  61h
                 OUT   61h, 5Eh    ; Write
                 LACC  #0b000h     ; BKX Value
                 SACL  61h
                 OUT   61h, 5Fh    ; Write

;Set up DMA and Interrupts
                 LACC  #0ffffh
                 SACL  IFR
                 LACC  #03h        ; mask on PC and codec ISR
                 SACL  IMR
                 IN    61h, 51h    ; bogus read from DSPRXD
                 LACC  #0c4h       ; Enable BXE and BRE
                 SACL  61h
                 OUT   61h, 54h
                 CLRC  INTM
;                SETC  INTM

loop:            NOP
                 IN   61h, 51h     ; bogus read from DSPRXD
                 OUT  61h, 50h     ; bogus write to DSPTXD
                 NOP
                 B     loop

                 MAR  *, AR6
loop1:           IN   61h, 5ah
                 LACL 61h
                 AND  #20h
                 BCND loop1, EQ
                 IN   *, 5bh        ; Read data
loop2:           IN   61h, 5ah
                 LACL 61h
                 AND  #02h
                 BCND loop2, EQ
                 OUT  *+, 5bh       ; Write data
```

```
;                   RPT  #5000            ; Wait
;                   NOP
                    B     loop1
playback:
                    RPT  #5000            ; Wait
                    NOP
                    OUT  *+, 5bh          ; Write data
                    B     playback

pc_isr: NOP
                    SETC XF
                    RPT  #100
                    NOP
                    CLRC XF
                    RETE

codec_isr:
                    SETC  XF
                    NOP
                    IN 60h, 52h ; Read PCSR - Will clear all interrupts!
                    LACL  60h
                    AND   #0010h        ; Test RHFULL
                    BCND  cap_serve, NEQ
ptest:              LACL  60h           ; Test XHFULL
                    AND   #0020h
                    BCND  play_serve, NEQ
isr_done NOP
                    CLRC  XF
                    RETE

cap_serve:
                    LACL  60h           ; Test which half DMA is cap
                    AND   #0040h
                    BCND  dmachigh, NEQ
dmaclow             NOP                 ; Process high
                    MAR   *,AR6
                    RPT   #0eh
                    BLDD  #8000h, *+
                    B     isr_done
dmachigh            NOP                 ;Process low
                    MAR   *,AR6
                    RPT   #0eh
                    BLDD  #8000h, *+
                    B     isr_done

play_serve:
                    LACL  60h           ; Test which half DMA is play
                    AND   #0080h
                    BCND  dmaphigh, NEQ
dmaplow             NOP                 ; Process high
                    MAR   *,AR7
                    RPT   #0eh
```

```
                BLDD  *+, #8010h
                B     isr_done
dmaphigh        NOP                   ;Process low
                MAR   *,AR7
                RPT   #0eh
                BLDD  *+, #8010h
                B     isr_done
```

Mode 3 - Stereo Codec DMA Digital Loopback with Intermediate Buffer and Interrupts

This code is graphically represented in Figure 5. It initializes the codec in DMA simultaneous capture/playback mode with appropriate inputs and gains and then uses mode 3 - Stereo Codec to do the digital loopback one sample at a time. Again the DMA controller is doing the moving based on CDRQ/PDRQ, not the DSP. The DSP processes the data based on interrupts from the DMA controller. But as an added twist, the capture/play buffers are different now (8000h-8010h and A000h-A0010h, respectively) and there is an intermediate buffer (9000h-9010h) and the DSP ISR's move the data to and from it. Also the FSM's in the DMA Controller must toggle -CDAK/-PDAK lines.

DMA Cap Play

* This uses DMA mode

```
                 .mmregs
                 .sect "vectors"

                B  start      ;00; RESET
INT1            B  pc_isr     ;02; vectors for interrupts int1-int3
INT2            B  codec_isr
INT3            B  INT3
                .space   2 * 16   ;08; timer interrupt vector - TINT
receive B       receive     ;0A; Serial port receive interrupt RINT
transmit B      transmit   ;0C; Serial port transmit interrupt XINT
                .space   4 * 16       ;0E; vectors for TDM port
                                          ; TRNT, TXNT
                .space   2 * 16       ;12; vector for external
                                          ; interrupt INT4
                .space   14 * 16      ;14; Reserved space.  Should
                                          ; not be used.
                .space   2 * 16       ;22; Trap instruction vector
                .space   2 * 16       ;24; Non Maskable Interrupt
                                          ; vector
                .space   4 * 16       ;26; Reserved space.  Should
                                          ; not be used.

                .text
start:          LDP  #0
                LACC #0C000h     ; Reset codec
```

```
                  SACL 61h
                  OUT  61h, 53h
                  RPT  #0f000h     ; Wait
                  NOP
                  LACC #0C800h     ; Take out of reset
                  SACL 61h
                  OUT  61h, 53h
                  RPT  #0f000h     ; Wait a while longer (Dag)
                  NOP
stuck:            IN   61h, 58h    ; Checking for 80h
                  LACL 61h
                  AND  #0080h
                  BCND stuck, NEQ

                  LACC #0049h      ; Flip MCE bit=1 and Register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0008h      ; Switch to DMA Dual Channel Mode
                  SACL 61h
                  OUT  61h, 59h

                  LACC #0048h      ; Set to register 8
                  SACL 61h
                  OUT  61h, 58h    ; works better with 8
;                 LACC #0040h      ; 16 `bit mono
                  LACC #0000h      ;  8 `bit mono
                  SACL 61h
                  OUT  61h, 59h

                  LACC #000bh      ; Flip MCE bit=0 and Register 11
                  SACL 61h
                  OUT  61h, 58h

stuck1:           IN   61h, 58h    ; Checking for 80h
                  LACL 61h
                  AND  #0080h
                  BCND stuck1, NEQ

stuck2:           IN   61h, 58h    ; Checking for 20h on register 11
                  LACL 61h
                  AND  #0020h
                  BCND stuck2, NEQ

                  LACC #0009h      ; Set to register 9
                  SACL 61h
                  OUT  61h, 58h
                  LACC #000bh      ; Enable any playback/capture
                  SACL 61h
                  OUT  61h, 59h

                  LACC #0000h      ; Set to Left Input Register
                  SACL 61h
```

```
                  OUT  61h, 58h
                  LACC #0000h      ; Write to Left Input Register
;Making this 7h sounds better, but codec cuts out!
                  SACL 61h
                  OUT  61h, 59h
                  LACC #0006h      ; Set to Left Output Register
                  SACL 61h
                  OUT  61h, 58h
                  LACC #0000h      ; Write to Left Output Register
                  SACL 61h
                  OUT  61h, 59h

                  lacc  #09000h    ; set up circular buffer pointers
                  sacl  CBSR1         ; buffer1 = 8000 -> ffff
                  lacc  #0a000h
        sacl    CBSR2                 ;
                  lacc  #0900fh       ; buffer2 = ffff -> 8000
                  sacl  CBER1         ;
                  lacc  #0a00fh       ; buffer2 = ffff -> 8000
        sacl    CBER2                 ;
                  lar   AR6,#09000h   ; set receive buffer
                  lar   AR7,#0a000h   ; set transmit buffer
                  lacc  #0000000011111110b
                               ; buf1=AR7, buf2=AR6, enable both
                  sacl  CBCR        ;
; Set up block registers
                  LACC  #8000h     ; ARR Value
                  SACL  61h
                  OUT   61h, 5Ch   ; Write
                  LACC  #08010h    ; BKR Value
                  SACL  61h
                  OUT   61h, 5Dh   ; Write
                  LACC  #0a000h    ; AXR Value
                  SACL  61h
                  OUT   61h, 5Eh   ; Write
                  LACC  #0a010h    ; BKX Value
                  SACL  61h
                  OUT   61h, 5Fh   ; Write

;Set up DMA and Interrupts
                  LACC  #0ffffh
                  SACL  IFR
                  LACC  #02h       ; mask on PC and codec ISR
                  SACL  IMR
                  IN     61h, 51h  ; bogus read from DSPRXD
                  LACC  #0c4h      ; Enable BXE and BRE
                  SACL  61h
                  OUT   61h, 54h
                  CLRC  INTM

loop:             NOP
                  IN   61h, 51h    ; bogus read from DSPRXD
```

```
                  OUT  61h, 50h     ; bogus write to DSPTXD
                  IN   61h, 5ch     ; bogus read from DSPRXD
                  IN   61h, 5dh     ; bogus read from DSPRXD
                  IN   61h, 5eh     ; bogus read from DSPRXD
                  IN   61h, 5fh     ; bogus read from DSPRXD
                  NOP
                  B    loop

                  MAR  *, AR6
loop1:            IN   61h, 5ah
                  LACL 61h
                  AND  #20h
                  BCND loop1, EQ
                  IN   *, 5bh       ; Read data
loop2:            IN   61h, 5ah
                  LACL 61h
                  AND  #02h
                  BCND loop2, EQ
                  OUT  *+, 5bh      ; Write data
;                 RPT  #5000        ; Wait
;                 NOP
                  B    loop1
playback:
                  RPT  #5000        ; Wait
                  NOP
                  OUT  *+, 5bh      ; Write data
                  B    playback

pc_isr: NOP
                  SETC XF
                  RPT  #100
                  NOP
                  CLRC XF
                  RETE

codec_isr:
                  SETC  XF
                  NOP
                  IN 60h, 52h ; Read PCSR - Will clear all interrupts!
                  LACL  60h
                  AND   #0010h      ; Test RHFULL
                  BCND  cap_serve, NEQ
ptest:            LACL  60h         ; Test XHFULL
                  AND   #0020h
                  BCND  play_serve, NEQ
isr_done NOP
                  CLRC  XF
                  RETE


cap_serve:
                  LACL  60h            ; Test which half DMA is cap
```

```
                  AND   #0040h
                  BCND  dmachigh, NEQ
dmaclow           NOP                  ; Process high
                  MAR   *,AR6
                  RPT   #07h
                  BLDD  #8000h, *+
                  IN    61h, 5ch       ; Bogus Read
                  B     ptest
dmachigh          NOP                  ;Process low
                  MAR   *,AR6
                  RPT   #07h
                  BLDD  #8008h, *+
                  IN    61h, 5ch       ; Bogus Read
                  B     ptest

play_serve:
                  LACL  60h            ; Test which half DMA is play
                  AND   #0080h
;Why?
                  BCND  dmaphigh,  NEQ
dmaplow           NOP                  ; Process high
                  MAR   *,AR7
                  RPT   #07h
                  BLDD  #9000h, *+
                  IN    61h, 5eh       ; Bogus Read
                  B     isr_done
dmaphigh          NOP                  ;Process low
                  MAR   *,AR7
                  RPT   #07h
                  BLDD  #9008h, *+
                  IN    61h, 5eh       ; Bogus Read
                  B     isr_done
```

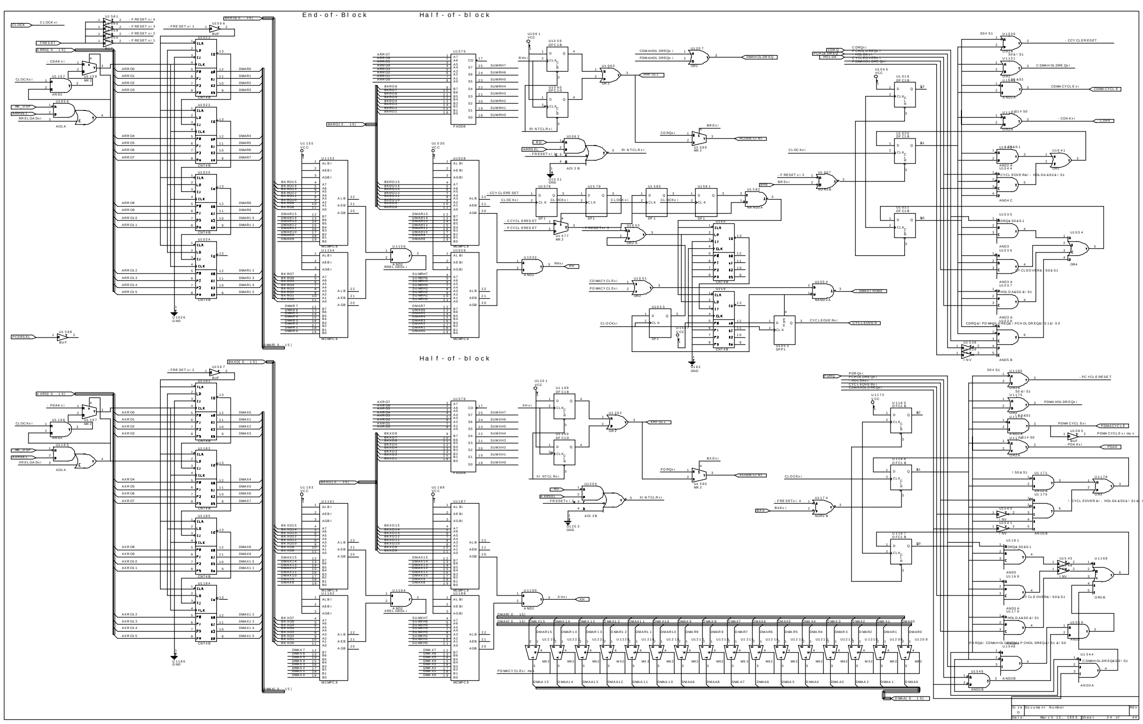
Appendix B - Stereo Codec DMA Structure

This Appendix addresses some specifics about the external DMA Controller that copies data between codec and external memory.

Stereo Codec DMA Schematic

Figure 11 displays the main schematic for the DMA controller. The top half is the capture section while the bottom is playback. All the way in the top left corner are the address generator counters. Farther to the right are the RH adders and comparators along with interrupt generators. Then all the way in the right top corner (with the 3 flip-flops) is the capture FSM. Right in the middle is the beginning of the waveform generators (continued in another sheet not shown here) that is a counter that executes the codec read and write cycles. Finally, tucked into the bottom left, are multiplexors going to the ‘C51 address bus. For much more detail see [9].

Figure 11. DMA Controller for Parallel Codec



Stereo Codec DMA FSM's

The following two ABEL files describe the FSM's used for capture and play respectively. Note that capture was given higher priority.

```
module  FSM_maker

    U5      device      'P22V10';

    "inputs
    CLK          Pin 1;
    CDRQ         Pin 2;
    PCHOLDREQ    Pin 3;
    HOLDA_       Pin 4;
    CYCLEOVER    Pin 5;
    PDMAHOLDREQ  Pin 6;

    "outputs
    CDMAHOLDREQ  Pin 23;
    CDMACYCLE    Pin 22;
    CDAK_        Pin 21;
    CCYCLERESET_ Pin 20;
    S0           Pin 19;
    S1           Pin 18;

    FSM          = [S1, S0];

    "states

    idle         = ^b000;
    reqbus       = ^b001;
    cycle        = ^b010;
    dmafork      = ^b011;

state_diagram FSM

state idle:
    CDMAHOLDREQ  =0;
    CDMACYCLE    =0;
    CDAK_        =1;
    CCYCLERESET_ =1;
    if [(PCHOLDREQ==0) & (CDRQ==1) & (PDMAHOLDREQ==0)] then reqbus
    else                 idle;

state reqbus:
    CDMAHOLDREQ  =1;
    CDMACYCLE    =0;
    CDAK_        =1;
    CCYCLERESET_ =0;
    if [(CYCLEOVER==0) & (HOLDA_==0)] then cycle
    else                 reqbus;

state cycle:
```

```
    CDMAHOLDREQ  =1;
    CDMACYCLE    =1;
    CDAK_        =0;
    CCYCLERESET_ =1;
    if (CYCLEOVER==1) then dmafork
    else                cycle;

state dmafork:
    CDMAHOLDREQ=1;
    CDMACYCLE =0;
    CDAK_     =1;
    CCYCLERESET_ =1;
    if (CDRQ==1) then reqbus
    else               idle;

end FSM_maker




module  FSM_maker

    U5      device      'P22V10';

    "inputs
    CLK          Pin 1;
    PDRQ         Pin 2;
    PCHOLDREQ    Pin 3;
    HOLDA_       Pin 4;
    CYCLEOVER    Pin 5;
    CDMAHOLDREQ  Pin 6;

    "outputs
    PDMAHOLDREQ  Pin 23;
    PDMACYCLE    Pin 22;
    PDAK_        Pin 21;
    PCYCLERESET_ Pin 20;
    S0           Pin 19;
    S1           Pin 18;

    FSM          = [S1, S0];

    "states

    idle         = ^b000;
    reqbus       = ^b001;
    cycle        = ^b010;
    dmafork      = ^b011;

state_diagram FSM

state idle:
```

```
    PDMAHOLDREQ  =0;
    PDMACYCLE    =0;
    PDAK_        =1;
    PCYCLERESET_ =1;
    if [(PCHOLDREQ==0) & (PDRQ==1) & (CDMAHOLDREQ==0)] then
    reqbus
    else                idle;

state reqbus:
    PDMAHOLDREQ  =1;
    PDMACYCLE    =0;
    PDAK_        =1;
    PCYCLERESET_ =0;
    if [(CYCLEOVER==0) & (HOLDA_==0) & (CDMAHOLDREQ==0)] then cycle
    else                reqbus;

state cycle:
    PDMAHOLDREQ  =1;
    PDMACYCLE    =1;
    PDAK_        =0;
    PCYCLERESET_ =1;
    if (CYCLEOVER==1) then dmafork
    else                cycle;

state dmafork:
    PDMAHOLDREQ  =1;
    PDMACYCLE    =0;
    PDAK_        =1;
    PCYCLERESET_ =1;
    if (PDRQ==1) then reqbus
    else               idle;

end FSM_maker
```