AN1790
Rev. 1, 8/2005

Programming the CS4218 CODEC for Use With DSP56300 Devices

By Thomas Lay

Through mathematical algorithms implemented on the DSP, members of the DSP56300 family can accomplish various of tasks and different kinds of digital signal processing. However, to obtain useful information, it is often necessary to interact with the outside world. To satisfy this requirement, the CS4218 16-bit Audio CODEC CMOS device is integrated with the current DSP563xx evaluation modules. The CODEC performs analog-to-digital (A/D) and digital-to-analog (D/A) conversion, filtering, and level setting.

A sample program is included in this document to demonstrate the use of the CS4218 CODEC with a Freescale DSP. The program explains the steps for interfacing the Freescale DSP with the CS4218 CODEC. The sample program shows in detail the use of the enhanced synchronous serial interface ports (ESSI) and how the DSP ESSI ports interface, initialize, and transport data between the DSP and CS4218 CODEC.

The following source code files, which assist in programming the CODEC, are available at the web site listed on the back cover of this document:

- `Ioequ.asm`: Important I/O equates for the DSP5630xEVM modules
- `Intequ.asm`: Interrupt equates for the DSP563xx EVM modules
- `Ada_equ.asm`: Equates for initializing the CODEC.
- `Ada_Init.asm`: Initialization code for the ESSI and CODEC

CONTENTS

freescale™ semiconductor

- `Vectors.asm`: Vector table for the DSP563xx EVM modules
- `Echo.asm`: Example of CODEC programming

Throughout this document, the sample code references equates in `Ada_equ.asm`, `Ioequ.asm`, and `Intequ.asm`.

1 CODEC Overview

The CS4218 stereo audio CODEC comprises many devices that perform A/D and D/A conversion. The CODEC consists of two delta-sigma A/D converters, two delta-sigma D/A converters, input anti-aliasing filters, output smoothing filters, programmable input gain, and programmable output attenuators. These separate CODEC components allow the DSP to receive data from the CODEC, process the data, and transmit processed data back to the CODEC. The data travels through special serial ports using the DSP ESSI ports and the CODEC's specialized pins.

The CODEC modes of operation are configured by setting certain pins on the CODEC high or low, specifically SMODE1, SMODE2, and SMODE3 pins. The mode to which the DSP563xx evaluation modules are physically set is Serial Mode 4 (SM4). In SM4 mode, the control information for the CODEC is separated from the data information, reducing the bandwidth needed by the data serial ports and simplifying the programming procedures.

Within SM4 mode are four sub modes. These secondary modes specify two things: whether the CODEC functions in the master mode or the slave mode and the number of bits per frame. For the DSP evaluation boards discussed in this document, the secondary modes are physically configured to sub mode 0 so that the CODEC functions in the master mode and sets the frame size to 32 bits. Operating in the master mode, the CODEC sends the serial bit clock and frame synchronization pulses to indicate the start and stop of a data frame. In addition, sub mode zero specifies that each frame consists of two 16-bit words: a left-channel 16 bit word and a right-channel 16 bit word. The left and right channels are sent to and from the CODEC with the most significant bits (MSBs) first. The properties defined by the sub modes apply to both the input data going into the CODEC (SDIN) and the output data coming from the CODEC (SDOUT). See **Figure 1**.

Control information is sent to the CODEC on a different serial interface than the data information. The control information consists of a list of attributes that specify properties such as level settings. Although 31 bits must be set in the control information, only 23 bits are useful. The other 8 bits are set to zero. For details on the CS4218 CODEC, refer to the *Crystal CS4218 CODEC Datasheet*.

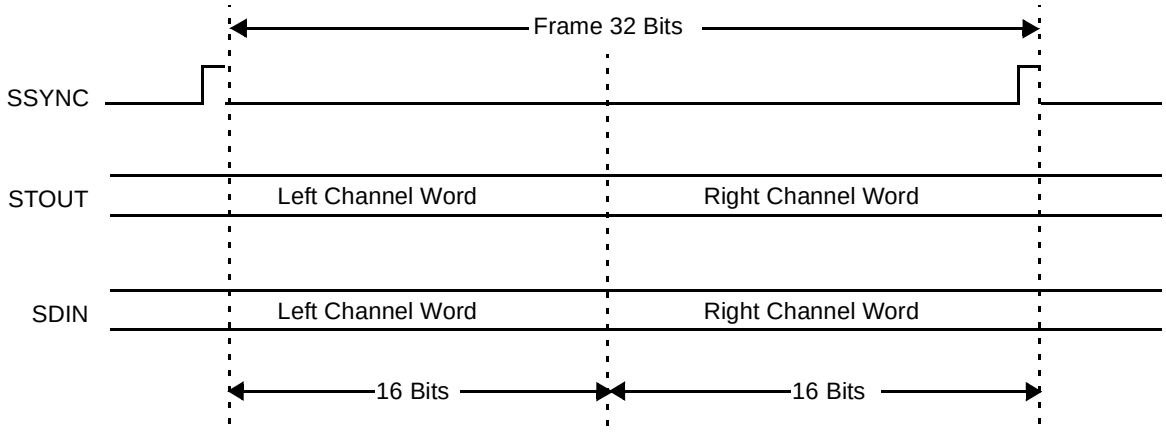

Figure 1. CODEC Data Format

2 ESSI Port Overview

The Freescale DSP563xx evaluation modules discussed in this document have two ESSI ports, ESSI0 and ESSI1, that form one of the major serial interfaces to external peripherals. Each port consists of six unique pins that allow performance of various functions, depending on how they are configured. Each port can function as either an ESSI or a General-Purpose Input/Output port (GPIO).

Operate the ESSI port in ESSI mode to synchronize your tasks with a master clock. In addition, certain control actions and direction flow are set automatically. Operate the ESSI port in GPIO mode to specify exactly how data is transferred and the direction of data flow. The drawback to GPIO mode is that you must thoroughly understand GPIO port usage in order to program for GPIO. The example in this document illustrates both the ESSI and GPIO modes of operation.

2.1 ESSI/GPIO Pins

The ESSI port uses six pins to transfer information. You can configure each pin to function in the ESSI mode or the GPIO mode by modifying the port control registers (see **Table 1**).

Table 1. ESSI Pins

Pin Name	Pin Function
Serial Control 0 (SC0/PC0)	Has a multitude of functions depending on how control registers are set
Serial Control 1 (SC1/PC1)	Has a multitude of functions depending on how control registers are set
Serial Control 2 (SC2/PC2)	Has a multitude of functions depending on how control registers are set
Serial Clock (SCK/PC3)	Serves as a provider or a receiver of the serial bit rate clock
Serial Receive Data (SRD/PC4)	Receives serial data
Serial Transmit Data (STD/PC5)	Transmits serial data

2.2 ESSI Port Registers

In either ESSI or GPIO mode, certain registers apply specifically to each mode, with the exception of two registers that determine how the ESSI ports are used: port control register C (PCRC) and port control register D (PCRD). Port control register C configures the ESSI0's functional mode; port control register D configures the ESSI1's functional mode. Setting the corresponding bit/pin on the port control register to 1 configures the pin to operate in the ESSI mode. Setting the corresponding bit/pin to 0 configures the pin to function in the GPIO mode. Notice that each pin is individually configured to be in the ESSI mode or the GPIO mode.

2.2.1 ESSI/GPIO Shared Registers

Table 2 lists the functions of the ESSI/GPIO shared registers.

Table 2. ESSI/GPIO Shared Registers

Register	Function
Port Control Register C (PCRC)	Controls whether to use the ESSI0 port in ESSI mode or GPIO mode
Port Control Register D (PCRD)	Controls whether to use the ESSI1 port in ESSI mode or GPIO mode.

The ESSI consists of 12 registers specific to the ESSI mode. There are two sets of ESSI registers; one for ESSI0 and the other for ESSI1. **Table 3** lists the ESSI registers.

Table 3. ESSI Registers

Register Name	Function
Control Register A (CRA)	Controls ESSI Mode operations.
Control Register B (CRB)	Controls ESSI Mode operations.
Status Register (SSISR)	Describes status and serial flags.
Transmit Slot Mask Register A (TSMA)	Determines when to transmit during a given time slot.
Transmit Slot Mask Register B (TSMB)	Determines when to transmit during a given time slot.
Receive Slot Mask Register A (RSMA)	Determines when to receive during a given time slot.
Receive Slot Mask Register B (RSMB)	Determines when to receive during a given time slot.
Time Slot Register (TSR)	Prevents data transmission during a time slot.
Receive Data Register (RX)	Read-only register that receives data.
Transmit Data Register 0 (TX0)	Transfer data for transmitter 1
Transmit Data Register 1 (TX1)	Transfer data for transmitter 2
Transmit Data Register 2 (TX2)	Transfer data for transmitter 3

In the GPIO mode, the ESSI port accesses four registers specific to GPIO mode (see **Table 4**).

Table 4. GPIO Registers

Register Name	Function
Port Direction Register C (PRRC)	Controls the direction of data flow for ESSI0 port in GPIO mode
Port Direction Register D (PRRD)	Controls the direction of data flow for ESSI1 port in GPIO Mode.
Port Data Register C (PDRC)	Stores data received or transmitted for ESSI0 port in GPIO mode.
Port Data Register D (PDRD)	Stores data received or transmitted for ESSI1 port in GPIO mode.

After a pin is set to function in the GPIO mode, the direction of data flow must be configured to specify for the ESSI port whether the pin receives data or transmits data. Setting the pin/bit to 0 on Port Direction Register C (PRRC) configures the GPIO pin as an input; setting the pin/bit to 1 configures the GPIO pin as an output. To receive or transmit data in GPIO mode, use the port data registers (PDRs). If the pin/bit functions as an input, the value in that pin/bit reflects the value on that pin. If the pin/bit functions as an output, the value on the pin/bit is the value being transmitted.

For details on the ESSI ports, refer to the DSP563xxEVM user's manual and the application note, *DSP56300 Enhanced Synchronous Serial Interface (ESSI) Programming*, (AN1764) at the web site listed on the back cover of this document.

2.3 Digital Interface (ESSI – CODEC)

For the DSP563xx evaluation modules discussed in this document, the CODEC is configured to function in SM4 mode. SM4 mode separates the data information from the CODEC control information so that two serial ports are required to transfer data and CODEC control information. Both the ESSI0 and ESSI1 ports control and transfer data between the DSP and the CODEC. Typically, ESSI0 controls data transfers while ESSI1 controls CODEC control information transfers.

ESSI0 performs three functions with the CODEC:

- Transfers data to and from the CODEC
- Receives synchronization pulses
- Performs the reset function on the CODEC

In contrast, ESSI1 controls and transfers CODEC control information. **Table 5** shows the definitions of the ESSI pins.

Table 5. ESSI Pins

ESSI0/ESSI1 Pin	CS4218 Codec Pin	Description
STD0 (ESSI0)	SDIN	Data transfer from ESSI0 to CODEC
SRD0 (ESSI0)	SDOUT	Data transfer from CODEC to ESSI0
SCK0 (ESSI0)	SCLK	Clock sent by CODEC (Master)
SC00 (ESSI0)	~RESET	Reset CODEC from ESSI0
SC02 (ESSI0)	SSYNC	Frame Synchronization pulse from CODEC
SC10 (ESSI1)	~CCS	Control Information gate
SC11 (ESSI1)	CCLK	Clock sent by ESSI1 to set control information
SC12 (ESSI1)	CDIN	Control data transfer from ESSI1

Physically, the ESSI port pins connect to the serial pins on the CODEC though jumper connections. To ensure correct operation using the example code referenced in this document, refer to **Table 6** and **Table 7** for the correct jumper settings for the DSP5630xEVM boards. **Figure 2** on page 2-6 shows the pin set-up between the DSP ESSI ports and the CODEC. For details on the pin layouts and jumper settings between the CODEC and DSP, consult the DSP user's manual for the respective evaluation modules.

Table 6. JP5 Jumper Block (ESSI0)

JP5	ESSI Pin	Codec Pin
1-2	SCK0	SCLK
3-4	SC00	~RESET
5-6	STD0	SDIN
7-8	SRD0	SDOUT
9-10	SC01	—
11-12	SC02	SSYNC

Table 7. JP4 Jumper Block (ESSI1)

JP4	ESSI Pin	Codec Pin
1-2	SCK1	-
3-4	SC10	~CCS
5-6	STD1	-
7-8	SRD1	-
9-10	SC11	CDIN
11-12	SC12	CCLK

3 Programming the CS4218 CODEC

For proper operation of the CS4218 CODEC device with Freescale DSPs requires a three-phase procedure. Each phase plays an essential role in properly setting up constants, interfacing and initializing, and correctly using the CS4218 CODEC with the Freescale DSP:

- *Phase 1: Setting up global constants.* Includes such activities as setting up buffer spaces and pointers, setting CODEC control information constants, and defining interface constants and pins.
- *Phase 2: Interfacing and initializing the ESSI and the CODEC.* Comprises the bulk of the work needed to obtain a working interface between the DSP563xx and the CODEC. The procedures include setting up and initializing the CODEC ports, setting up and initializing the ESSI ports, and interfacing the CODEC and ESSI ports.
- *Phase 3: Data transferring mechanisms.* Includes information on the types of data transfer methods. Although three types of data transfer methods are available—polling, DMA, and interrupts— this document discusses only interrupts.

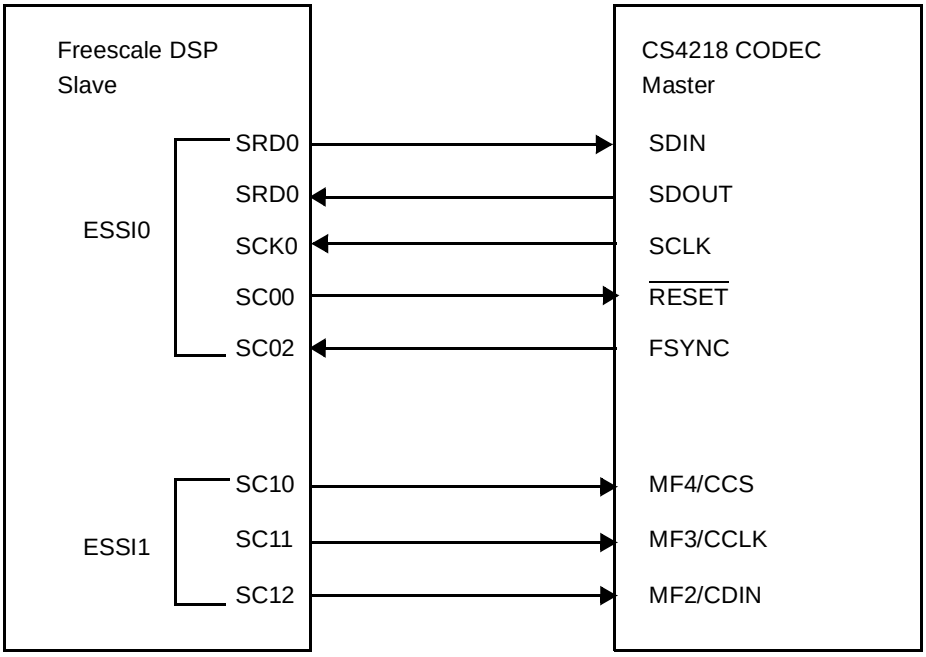

Figure 2. ESSI/CODEC Pin Set-Up

3.1 Phase 1: Set Up Global Constants

To set up the global constants, you must set up the buffer space and pointers and then define the CODEC control parameters.

3.1.1 Set Up Buffer Space and Pointers

The buffer space and pointers temporarily store the incoming and outgoing data. These variables are receive and transmit buffers and pointers. In addition to temporary storage, the pointers offer a method to access the memory location of the stored data. **Example 1** demonstrates the task of setting up transmit and receive buffers and pointers.

Example 1. Set Up Transmit and Receive Buffers and Pointers

```
;Receive buffer and pointer

RX_BUFF_BASE          equ     *
RX_data_1_2           ds      1             ; Left receive channel audio
RX_data_3_4           ds      1             ; Right receive channel audio
RX_PTR                ds      1             ; Receive pointer

;Transmit buffer and pointer

TX_BUFF_BASE          equ     *
TX_data_1_2           ds      1             ; Left transmit channel audio
TX_data_3_4           ds      1             ; Right transmit channel audio
TX_PTR                ds      1             ; Transmit pointer
```

3.1.2 Define CODEC Control Parameters

To specify parameters of the A/D and D/A conversion and other audio parameters, the control information must be declared. Parameters such as left and right attenuation, left and right gain, line input selects, and mask interrupts, are configured in the control information. The control information consists of 32 bits of information. Although only 23 bits contain useful information, a minimum of 31 bits must be set. **Table 8** lists the bit definitions.

Table 8. CS4218 CODEC Control Information (MSB)

Descriptions	Bit	Values
Not Applicable	31	0
Mask Interrupt	30	0 = no mask on MF5:\INT 1 = mask on MF5:\INT
D01	29	N/A
Left output D/A Attenuation (1.5 dB steps)	28 – 24	00000 = No attenuation 11111 = Max attenuation (-46.5 dB)
Right output D/A Attenuation (1.5 dB steps)	23 – 19	00000 = No attenuation 11111 = Max attenuation (-46.5 dB)
Mute D/A output	18	0 = output not muted 1 = output muted
Left Input Select	17	0 = LIN1 1 = LIN2
Right Input Select	16	0 = RIN1 1 = RIN2
Left input D/A Gain (1.5 dB steps)	15 – 12	00000 = no gain 11111 = max gain (22.5 dB)
Right input D/A Gain (1.5 dB steps)	11 – 8	00000 = no gain 11111 = max gain (22.5 dB)
Not Applicable	7 – 0	0000000

Suppose that the following requirements are needed for an application:

1. No mask for the interrupt pin.
2. No left or right D/A attenuation.

3. Muting turned off.
4. LIN2 and RIN2 selected. (On the EVM boards input 2 is used for both left and right channels.)
5. No left and right D/A gains.

Example 2 illustrates the procedure of setting the CODEC control information using these specified control parameters.

Example 2. Setting Codec Control Information

```
NO_MASK_INT             equ             $000000
NO_LEFT_ATTN            equ             $000000                 ; 0 dB
NO_RIGHT_ATTN           equ             $000000                 ; 0 dB
LIN2                    equ             $000200         ; use LIN2 on EVM
RIN2                    equ             $000100         ; use RIN2 on EVM
NO_LEFT_GAIN            equ             $000000         ; 0 dB
NO_RIGHT_GAIN           equ             $000000         ; 0 dB
NO_MUTING               equ             $000000

CTRL_WD_12      equ     NO_MASK_INT+NO_LEFT_ATTN+NO_RIGHT_ATTN+LIN2+RIN2+
                        NO_MUTING

CTRL_WD_34      equ     NO_LEFT_GAIN+NO_RIGHT_GAIN
```

The CS4218 CODEC data sheet reverses the bit-order of the control information. For instance, bit 1 should be the mask interrupt instead of bit 30. However, since most of the work with the ESSI ports and CODEC is done using MSB first, **Table 8** is modified to reverse the bit order from the CODEC data sheet to simplify control information programming.

Note: The evaluation modules used in this document select line 2 of right and left inputs. Therefore, bits 17 (Left Input Select) and bits 16 (Right Input Select) should be configured to select LIN 2 (1) when the DSP563xx EVM evaluation modules are used.

3.2 Phase II: Initialize and Interface the ESSI and CODEC Ports

After certain constants for the CODEC and the ESSI are defined, the next step is to initialize the ESSI and CODEC interface. Initialization starts with the ESSI ports, which includes resetting the ESSI ports, modifying ESSI control registers, and configuring ESSI/GPIO functionality. Second, the CODEC must also be initialized, which entails resetting the CODEC and sending in CODEC control information, as follows:

1. Reset the ESSI ports.
2. Modify the ESSI control registers.
3. Configure ESSI or GPIO functionality.
4. Reset the CODEC.
5. Modify CODEC control information.
6. Deassert ESSI reset and enable interrupts.

3.2.1 Initialize ESSI Ports

The first step in initializing the ESSI port is to reset the ESSI ports by sending a value of zero into Port Control Registers C and D. (Although ESSI1 is to be used as a GPIO, it is recommended that you also perform the reset on ESSI1.) **Example 3** illustrates the reset procedure of the ESSI ports.

Example 3. ESSI Port Reset Procedure

```
movep          #$0000,x:M_PCRC              ; reset ESSI0 C control register port
movep          #$0000,x:M_PCRD              ; reset ESSI1 D control register port
```

The next step is to set the control parameters for the ESSI port by adjusting the bits on the ESSI Control Register A (CRA0) and ESSI Control Register B (CRB0). Describing the meaning of each bit on the registers is beyond the scope of this document. You can find definitions of each bit in the respective DSP563xx user's manuals. However, this document does cover certain typical settings that must be made for the CODEC to work properly with the ESSI ports. **Table 9** displays the settings to be made with Control Register A.

Table 9. Settings for Control Register A

Bit Name	Description	Bit Position	Value (Binary)
Reserved	Reserved	23	0
SSC1	SC1 pin = serial I/O flag	22	0 (SC1 flag set)
WL[2:0]	Word Length control	21-19	010 (16 bit control word)
ALC	Alignment Control	18	0 (Align to bit 23)
Reserved	Reserved	17	0
DC[4:0]	Frame Rate Divider Control	16-12	00001 (2 time slots per frame)
PSR	Prescaler Range	11	1 (ESSI clock is divided by one)
Reserved	Reserved	10-8	000
PM[7:0]	Prescale Modulus Select	7-0	00000111 (ESSI clock divided by 8)

Table 10 lists the typical required settings for Control Register B to ensure functionality between the ESSI ports and the CODEC.

Table 10. Settings for Control Register B

Bit Name	Description	Bit Position	Value (Binary)
REIE	Receive exception interrupt	23	1 (enabled)
TEIE	Transmit exception interrupt	22	1 (enabled)
RLIE	Receive last slot interrupt	21	1 (enabled)
TLIE	Transmit last slot interrupt	20	1 (enabled)
RIE	Receive interrupt	19	1 (enabled)
TIE	Transmit interrupt	18	1 (enabled)
RE	Receive register	17	1 (enabled)
TE0	Transmit register 0	16	1 (enabled)
TE1	Transmit register 1	15	0 (disabled)
TE2	Transmit register 2	14	0 (disabled)
MOD	Mode	13	1 (Network Mode)
SYN	Synchronization mode	12	1 (Synchronous mode)
CKP	Clock polarity	11	0 (Data and frame sync clocked on rising edge)

Table 10. Settings for Control Register B (Continued)

Bit Name	Description	Bit Position	Value (Binary)
FSP	Frame Sync. Polarity	10	0 (positive polarity)
FSR	Frame Synch Relative Timing	9	1 (Frame synch begins one bit before first bit of data word)
FSL	Frame Sync. Length	8-7	10 (Rx-bit length: TX-bit length)
SHFD	Shift direction	6	0 (shift MSB first)
SCKD	Clock source direction	5	0 (SCK is input clock)
SCD2	SC2 pin direction	4	0 (SC2 is input)
SCD1	SC1 pin direction	3	1 (SC1 is output)
SCD0	SC0 pin direction	2	1 (SC0 is output)
OF[1:0]	Output flags	1-0	N/A

Only the ESSI0 control parameters are configured. Since ESSI1 functions in GPIO mode, the control parameters do not need to be set. **Example 4** illustrates the task of setting up the control registers for the ESSI0 port according to the specifications in **Table 9** and **Table 10**.

Example 4. Setting Control Registers for the ESSI0 Port

```
;Setting ESSI0 Control Parameters

; Control Register A
movep   #$101807,x:M_CRA0               ; 12.288MHz/16 = 768KHz SCLK
                                        ; prescale modulus = 8
                                        ; frame rate divider = 2
                                        ; 16-bits per word
                                        ; 32-bits per frame
                                        ; 16-bit data aligned to bit 23

; Control Register B
movep   #$ff330c,x:M_CRB0               ; Enable REIE,TEIE,RLIE,TLIE,
                                        ; RIE,TIE,RE,TE0
                                        ; network mode, synchronous,
                                        ; out on rising/in on falling
                                        ; shift MSB first
                                        ; external clock source drives SCK
                                        ; (codec is master)
                                        ; RX frame sync pulses active for
                                        ; 1 bit clock immediately before
                                        ; transfer period
                                        ; positive frame sync polarity
                                        ; frame sync length is 1-bit
```

3.2.2 Configuring GPIO Pins

Recall that the ESSI0 pins function in ESSI mode, while the ESSI1 pins operate in GPIO mode. As **Figure 2** shows, some pins affect only the control information of the CODEC, while other pins deal with the transfer of data. Because the CODEC on the DSP563xx EVM boards is configured to operate in SM4 mode, the control information runs on a serial line separate from the data lines. Additionally, SM4 specifies that the control information be configured only once unless a change is needed.

The full ESSI port mode is not necessary for controlling the CODEC control information. Instead, GPIO mode transfers the control information. Any pins that control the CODEC control information are configured for GPIO mode; otherwise ESSI mode is used. The following pins are used as GPIO pins to control the transfer of CODEC control information.

- SC00 (CODEC_RESET pin)
- SC10 (CCS pin)
- SC11 (CCLK pin)
- SC12 (CDIN pin)

These pins correspond to specific bits on the port data registers. For instance, the CODEC_RESET pin on the CODEC connects to the SC00 pin on ESSI0. This pin corresponds to bit 0 on Port Data Register C. Refer to **Table 11** and **Table 12** for details on the correspondence between physical pins and port data registers.

Table 11. Port Data Register C Pin/Bit Correspondence

Bit Name (ESSI0)	Bit Name (Codec)	Bit Position Register C	Functionality Mode
Reserve for future use	N/A	6–23	N/A
STD	SDIN	5	ESSI
SRD	SDOUT	4	ESSI
SCK	SCLK	3	ESSI
SC02	FSYNC	2	ESSI
SC01	N/A	1	N/A
SC00	CODEC_RESET	0	GPIO

Table 12. Port Data Register D Pin/Bit Correspondence

Bit Name (ESSI1)	Bit Name (Codec)	Bit Position Register D	Functionality Mode
Reserve for future use	N/A	6–23	N/A
STD	N/A	5	N/A
SRD	N/A	4	N/A
SCK	N/A	3	N/A
SC12	CDIN	2	GPIO
SC11	CCLK	1	GPIO
SC10	CCS	0	GPIO

Using the information in **Table 11** and **Table 12**, you can define global constants to simplify programming. **Example 5** illustrates the task of defining the pin/bit correspondence for the GPIO pins.

Example 5. Defining GPIO Pin/Bit Correspondence

```
; ESSI0 - audio data port control register C
; DSP                    CODEC
; ---------------------------
CODEC_RESET                  equ     0       ; bit0  SC00 ---> CODEC_RESET~

; ESSI1 - control data port control register D
; DSP                    CODEC
;----------------------------
CCS                          equ     0       ; bit0  SC10 ---> CCS~
CCLK                         equ     1       ; bit1  SC11 ---> CCLK
CDIN                         equ     2       ; bit2  SC12 ---> CDIN
```

After constants are set up for the GPIO pins, the port control registers must be configured. First, the CODEC_RESET pin (pin 0) is configured to function as a GPIO pin. However, other pins on ESSI0, are configured to work in ESSI mode. A 0 value is sent into bit 0 in Port Control Register C, while a value of 1 is sent to the other five pertinent bits.

Additionally, the CCS pin, the CCLK pin, and CDIN pin must function as GPIO pins on the ESSI1 port. Bit 0 (CCS), bit 1 (CCLK), and bit 2 (CDIN) in Port Control Register D are set to 0 to allow those pins to operate in GPIO mode. Since the other pins in Port Control Register D are not used, they should be set to a value of 0 for future compatibility.

At this point, the ESSI functionality should be disabled prior to initializing the CODEC. Therefore the pins on ESSI0 are not configured to function in ESSI mode until the CODEC is initialized. However, the GPIO pins are configured as shown in **Example 6**.

Example 6. GPIO Pin Configuration

```
; Port Control Register C
movep          #$0000,x:M_PCRC              ; Setting pin 0 for GPIO

; Port Control Register D
movep          #$0000,x:M_PCRD              ; Setting pin 0, pin 1, and pin 2
                                            ; to GPIO mode
```

Since ESSI0 pin 0 and ESSI1are used in GPIO mode, the direction of data flow must be declared. The direction of flow determines which device transmits data and which device receives data. To set the direction of data flow, we set Port Direction Registers C and D, (register C refers to ESSI0 and register D refers to ESSI1). Setting the pin/bit on the Port Direction Register to 1 configures the pin/bit as an output; setting the pin/bit to 0 configures the pin/bit as an input. Therefore, in order to configure the pins using the Data Direction Registers to mimic the direction flow information in **Figure 2**, the following bits must be set. **Table 13** and **Table 14** show the bit settings for the Data Direction Registers.

Table 13. Data Direction Register C

Bit Name	Bit Position	Value (Binary)
Other bits	6-23	X* (0)
STD0	5	X (0)
SRD0	4	X (0)
SCK0	3	X (0)
SC02	2	X (0)

Table 13. Data Direction Register C (Continued)

Bit Name	Bit Position	Value (Binary)
SC01	1	X (0)
SC00	0	1 (CODEC_RESET is output)
Notes: The X value is a “don’t care” value, but for future compatibility, a value of 0 is assigned in place of don’t cares.		

Table 14. Data Direction Register D

Bit Name	Bit Position	Value (Binary)
Other bits	6-23	X (0)*
STD1	5	X (0)
SRD1	4	X (0)
SCK1	3	X (0)
SC12	2	1 (CDIN is output)
SC11	1	1 (CCLK is output)
SC10	0	1 (CCS is output)
Notes: The X value is a “don’t care” value, but for future compatibility, a value of 0 is assigned in place of don’t cares.		

Example 7 illustrates the bit settings in the Data Direction registers.

Example 7. Code Form Settings in Data Direction Registers

```
; Data Direction Register C
movep   #$0001,x:M_PRRC                 ; set SC00=CODEC_RESET~ as output
; Data Direction Register D
movep   #$0007,x:M_PRRD                 ; set SC10=CCS~ as output
                                        ; set SC11=CCLK as output
                                        ; set SC12=CDIN as output
```

3.2.3 Initializing the CODEC ports

The next step is initializing the CODEC, as follows:

- Reset the CODEC.
- Wait for the CODEC to reset.
- Send the control information for the CODEC. Note that control information needs to be sent only when a change is made to the control parameters.

To reset the CODEC, send a 0 value into the CODEC_RESET pin. Recall that we have defined a global variable called CODEC_RESET. Thus, to reset the CODEC, you clear the CODEC_RESET bit in Port Data Register C on the ESSI port. In addition, set the CSS pin to 0 to notify the CODEC that control information is to be modified. Since the CODEC requires a minimum of 50 ms to reset, you typically program a delay into the DSP to allow for the CODEC to reset. **Example 8** summarizes these procedures.

Example 8. Code Format Procedures

```
bclr    #CODEC_RESET,x:M_PDRC       ; assert CODEC_RESET~ (bit 0 on ESSI0)
bclr    #CCS,x:M_PDRD               ; assert CCS~ (bit 0 on ESSI1)
```

```
;----reset delay for codec----
do      #1000,_delay_loop
rep     #1000                       ; A delay greater than 50 ms
nop
_delay_loop
```

When the CODEC is reset, the CODEC control information must be sent from the DSP to the CODEC ports. However, the CODEC_RESET pin must first be turned off (set to 1). Refer to **Table 15** for information on the options of each bit/pin. **Example 9** demonstrates the task of deasserting the CODEC reset.

Example 9. Deasserting Code Reset

```
bset    #CODEC_RESET,x:M_PDRC       ; deassert CODEC_RESET~ (pin 0 on ESSI0)
```

Table 15. CODEC Pins

Pin Name	Description	Values
~CODEC_RESET	Resets the CODEC	0 = Reset CODEC 1 = Disable Reset
FSYNC	Indicates a start of a frame	Rising edge = New Frame
SCLK	Serial clock	Rising Edge = data is received Falling edge = data is transmitted
SDOUT	Serial data output line	N/A
SDIN	Serial data input line	N/A
~CCS	Enables setting of CODEC control parameters	0 = enabled 1 = disabled
CDIN	Serial control information input line	N/A
CCLK	Clock for control parameters	Rising edge = control parameters sent

The CODEC is now ready to receive the control information. It ignores the first set of control information sent after a reset, so a dummy set of control information is sent ahead of the correct control information. Two global variables are defined to simplify programming:

- CTRL_WD_HI: The high word in the control information.
- CTRL_WD_LO: The low word in the control information.

To send the control information from the ESSI to the CODEC, perform the following steps:

1. Set up the registers to send dummy control information.
2. Send the control words.
3. Set up the registers to send correct control information.
4. Send the control words.

Example 10 illustrates these procedures.

Example 10. Sending Code Information

```
CTRL_WD_HI          ds      1           ; Upper Control word
CTRL_WD_LO          ds      1           ; Lower Control word

dummy_control
        move    #0,x0
        move    x0,x:CTRL_WD_HI         ; send dummy control data
```

```
        move    x0,x:CTRL_WD_LO
        jsr     codec_control

set_control
        move    #CTRL_WD_12,x0          ; recall constant set previously
                                        ; for upper control info
        move    x0,x:CTRL_WD_HI         ; set hi control word to hi constant
        move    #CTRL_WD_34,x0          ; recall constant set previously
                                        ; for lower control info
                                        ; set low control word to low constant
        move    x0,x:CTRL_WD_LO         ; 16 bit data aligned to bit 23
        jsr     codec_control
```

The control words are sent serially to the CDIN pin of the CODEC. The `codec_control` subroutine performs this action. Following is one method of sending in the control words:

1. Clear the CCS bit to allow the CODEC to accept control information.
2. Set the CCLK bit on the CODEC high (control bits are sent on the rising edge of the clock).
3. Determine whether the MSB of the control information is 1 or 0.
4. Send the MSB value to the CDIN pin.
5. Set CCLK to low on the CODEC to start the next cycle.
6. Shift-left the control word.
7. Repeat 16 times.

This procedure must be performed once for the upper 16-bit control word and then once for the lower 16-bit control word. **Example 11** illustrates these procedures.

Example 11. Sending in Control Words

```
;------------------------------------------------------------
; codec_control routine
;       Input:  CTRL_WD_LO and CTRL_WD_HI
;       Output: CDIN
;       Description: Used to send control information to CODEC
;       NOTE: does not preserve the 'a' register.
;------------------------------------------------------------
codec_control
        clr     a
        bclr    #CCS,x:M_PDRD                   ; assert CCS
        move    x:CTRL_WD_HI,a1                 ; upper 16 bits of control data
        jsr     send_codec                      ; shift out upper control word
        move    x:CTRL_WD_LO,a1                 ; lower 16 bits of control data
        jsr     send_codec                      ; shift out lower control word
        bset    #CCS,x:M_PDRD                   ; dissert CCS
        rts

;------------------------------------------------------------
; send_codec routine
;       Input:  a1 contains control information
;       Ouput: sends bits to CDIN
;       Description: Determines bits to send to CDIN
;------------------------------------------------------------
send_codec
                do      #16,end_send_codec      ; 16 bits per word
```

```
                bset    #CCLK,x:M_PDRD          ; toggle CCLK clock high
                jclr    #23,a1,bit_low          ; test msb
                bset    #CDIN,x:M_PDRD          ; send high into CDIN
                jmp     continue
bit_low
                bclr    #CDIN,x:M_PDRD          ; send low into CDIN
continue
                rep     #2                      ; delay
                nop
                bclr    #CCLK,x:M_PDRD          ; restart cycle
                lsl     a                       ; shift control word to 1 bit
                                                ; to left
end_send_codec
            rts
```

The `codec_control` subroutine performs most of the work of sending the information to the CODEC ports. First, the CSS bit is cleared to permit the modification of the control registers on the CODEC. Then the control words are loaded into registers and sent out to another subroutine that sends the data serial out to the CODEC ports. After both the upper and lower control words are sent, the CCS bit is reset to 1 to disallow changing of the control information on the CODEC.

The `send_codec` subroutine serves as the workhorse for the `codec_control` routine. This routine pushes the individual bits of the control words into the CODEC. First, it sets the clock (CCLK) high to allow the bit to be sent. Then it determines what the most significant bit (MSB) is and either sends in a 0 or 1 to the CDIN pin, depending on the MSB. A delay is incorporated into the routine to allow the information to be sent. Afterwards, the clock (CCLK) is set low to allow the cycle to begin again. The control word is shifted to serve the next MSB bit. These procedures are performed 16 times to serve all the bits in the control word.

3.2.4 Enabling Interrupts/ESSI Ports

Once the ESSI port and CODEC ports are configured and initialized, there are just three more steps to complete the interface between the ESSI and the CODEC:

1. Set the priority level of the interrupts. This parameter is determined by the application.
2. Enable interrupts on the DSP.
3. Enable the ESSI port.

Recall that in order to set the functionality of ESSI pin, the port control registers must be configured. Setting the corresponding pin/bit to 1 enables ESSI mode, and setting the pin/bit to 0 disables ESSI mode and enables the GPIO mode. As stated in **Section 3.2.2**, "Configuring GPIO Pins" the following pins/bits must be configured as GPIO pins:

- CODEC_RESET pin (bit 0 on ESSI0)
- CCS pin (bit 0 on ESSI1)
- CCLK pin (bit 1 on ESSI1)
- CDIN pin (bit 2 on ESSI1)

Therefore, on Port Control Register C, bit 0 is set to 0. Other pertinent pins should be set to 1 in order to configure the other pins as ESSI pins. On Port Control Register D, bits 0, 1, and 2 should all be set to the value of 0 to allow GPIO functionality on those pins. Because the other pins are not connected to the CODEC, the other bits do not have an effect.

Example 12 demonstrates setting the interrupt priority level, enabling the priority, and setting the ESSI/GPIO functionality of the ESSI ports.

Example 12. ESSI Port Priority and Functionality Setting

```
movep          #$000c,x:M_IPRP       ; set interrupt priority level for ESSI0
                                     ; to 3
andi           #$fc,mr               ; enable interrupts
movep          #$003e,x:M_PCRC       ; enable ESSI mode for
                                     ; bit 5,bit 4,bit 3,bit 2,bit 1.
                                     ; enable GPIO mode for
                                     ; bit 0


movep          #$0000,x:M_PCRD       ; enable GPIO mode for
                                     ; bit 2, bit 1, bit 0.
                                     ; Other bits are don't care.
```

3.3 Phase III: Data Transferring Mechanism

The three methods for transferring data from the CODEC to and from the ESSI port are polling, DMA, and interrupts. This document demonstrates only the use of interrupts.

3.3.1 Interrupts and Interrupt Service Routines

The ESSI device has six interrupts:

- ESSI receive data with exception status
- ESSI receive data
- ESSI receive last slot
- ESSI transmit data with exception status
- ESSI transmit last slot
- ESSI transmit data

Each interrupt is triggered on certain status bits and is cleared by performing an interrupt service routine. The following sections explain what specific status bits trigger the interrupts and what must be done to clear the interrupts. For details on the properties and functionality of each type of interrupt and how to set up the interrupt service routines, refer to the DSP563xx EVM user's manual.

3.3.2 ESSI Receive Data with Exception Status Interrupt

The interrupt occurs when the following properties are true:

- The receive exception interrupt is turned on (CRB[23]).
- The receive data register is full.
- A receiver overrun error occurs.

The interrupt is triggered when the receiver overrun bit is set. When the interrupt is serviced, you must first clear the receiver overrun bit (SSISR0[5]) and then receive the Receive BUFFER. Perform the following steps:

1. Clear the receive overrun bit.
2. Save the necessary context.
3. Load the receive buffer pointer.
4. Move the received data to the receive buffer.
5. Update the receive buffer pointer.
6. Restore context.

Example 13 illustrates the procedures for servicing the ESSI Exception Status interrupt.

Example 13. Servicing the ESSI Exception Status Interrupt

```
;ESSI Receive Data with Exception Interrupt Service Routine
;-------------------------------------------------------------------

ssi_rxe_isr
                                    ; Clear receives overrun bit
bclr          #5,x:M_SSISR0         ; (M_SSISR0 refers to status register)
                                    ; explicitly clears overrun flag

                                    ; Save Context
move          r0,x:(r7)+            ; Save r0 to the stack.
move          m0,x:(r7)+            ; Save m0 to the stack.
move          #1,m0                 ; Modulus 2 buffer.


move          x:RX_PTR,r0           ; Load the pointer to the rx buffer.

nop                                 ; Delay

movep         x:M_RX0,x:(r0)+       ; Move received data to receive buffer

move          r0,x:RX_PTR           ; Update rx buffer pointer.
                                    ; Restore Context
move          x:-(r7),m0            ; Restore m0.
move          x:-(r7),r0            ; Restore r0.
rti
```

3.3.3 ESSI Receive Data Interrupt

The interrupt occurs when the following properties are true:

- The receive interrupt is turned on (CRB[19])
- The receive data register is full

To service the interrupt, you must receive the data in the following steps:

1. Save the necessary context.
2. Load the receive buffer pointer.
3. Move the received data to the receive buffer.

4. Update the receive buffer pointer.
5. Restore context.

Example 14 illustrates the procedures for servicing the ESSI Receive Data interrupt.

Example 14. Servicing the ESSI Receive Data Interrupt

```
;ESSI Receive Data Interrupt Service Routine
;---------------------------------------------------------------------
ssi_rx_isr

                                        ; Save Context
move            r0,x:(r7)+              ; Save r0 to the stack.
move            m0,x:(r7)+              ; Save m0 to the stack.
move            #1,m0                   ; Modulus 2 buffer.


move            x:RX_PTR,r0             ; Load the pointer to the rx buffer.

nop                                     ; Delay

movep           x:M_RX0,x:(r0)+         ; Move received data to receive buffer

move            r0,x:RX_PTR             ; Update rx buffer pointer.
                                        ; Restore Context
move            x:-(r7),m0              ; Restore m0.
move            x:-(r7),r0              ; Restore r0.

rti
```

3.3.4 ESSI Receive Last Slot Interrupt

The interrupt occurs when the following properties are true:

- The receive last slot interrupt is turned on (CRB[21]).
- The last time slot ends.

The receive last slot interrupt guarantees that the previous frame has been serviced and the next frame is ready to be serviced. The interrupt allows the programmer to redefine pointers to the buffer to be reset so that a new frame can be serviced. To prepare for the next frame, follow these steps:

1. Save Context.
2. Reset the receive buffer.
3. Restore context.

Example 15 demonstrates the steps required for servicing the ESSI Receive Last Slot interrupt.

Example 15. Servicing the ESSI Receive Last Slot Interrupt

```
; receive last slot interrupt service routine

ssi_rxls_isr
                                        ; Save context
move            r0,x:(r7)+              ; Save r0 to the stack.

move            #RX_BUFF_BASE,r0        ; Reset rx buffer pointer just in
```

```
                                        ; case it was corrupted.
move          r0,x:RX_PTR               ; Update rx buffer pointer.

move          x:-(r7),r0                ; Restore r0.
rti
```

3.3.5 ESSI Transmit Data with Exception Status Interrupt

The interrupt occurs when the following properties are true:

- The transmit exception interrupt is turned on (CRB[22]).
- The transmit data register is empty.
- A transmit underrun error occurs.

The interrupt is triggered when the transmit underrun bit is set. When the interrupt is serviced, you must first clear the transmit underrun bit (SSISR0[4]) and then transmit the transmit BUFFER, as follows:

1. Clear the transmit underrun bit.
2. Save the necessary context.
3. Load the transmit buffer pointer.
4. Move the transmit buffer data to the transmit register.
5. Update the transmit buffer pointer.
6. Restore context.

Example 16 illustrates the procedures for servicing the ESSI Transmit Data With Exception Status interrupt.

Example 16. Servicing the ESSI Transmit Data with Exception Status Interrupt

```
; transmit data with exception status interrupt service routine

ssi_txe_isr
                                  ; Clear underrun bit
bclr          #4,x:M_SSISR0       ; (M_SSISR0 pointers to status register)

                                  ; Save Context
move          r0,x:(r7)+          ; Save r0 to the stack.
move          m0,x:(r7)+          ; Save m0 to the stack.
move          #1,m0               ; Modulus 2 buffer.

                                  ; Load transmit pointer to transmit ;
                                  ; buffer
move          x:TX_PTR,r0         ; Load the pointer to the tx buffer.

nop                               ; delay

                                  ; Move Transmit buffer data to transmit
                                  ; register
movep         x:(r0)+,x:M_TX00    ; SSI transfer data register.

                                  ; Update transmit buffer pointer
move          r0,x:TX_PTR         ; Update tx buffer pointer.

                                  ; Restore Context
```

```
move            x:-(r7),m0              ; Restore m0.
move            x:-(r7),r0              ; Restore r0.
rti
```

3.3.6 ESSI Transmit Last Slot Interrupt

The interrupt occurs when the following properties are true:

- The transmit last slot interrupt is turned on (CRB[20]).
- The last time slot begins.

The use of the Transmit Last Slot interrupt guarantees that the previous frame has been serviced and the next frame is ready to be serviced. The interrupt allows the programmer to redefine pointers to the buffer to be reset so that a new frame can be serviced. To prepare for the next frame the following steps are followed:

1. Save context.
2. Reset the transmit buffer.
3. Restore context.

Example 17 illustrates the procedures for servicing the ESSI Transmit Last Slot interrupt.

Example 17. ESSI Transmit Last Slot Interrupt Service

```
; transmit last slot interrupt service routine
ssi_txls_isr
                                        ; Save Context
move            r0,x:(r7)+              ; Save r0 to the stack.

                                        ; Reset Transmit buffer pointer
move            #TX_BUFF_BASE,r0        ; Reset pointer.
move            r0,x:TX_PTR             ; Reset tx buffer pointer just in
                                        ; case it was corrupted.

                                        ; Restore Context
move            x:-(r7),r0              ; Restore r0.
rti
```

3.3.7 ESSI Transmit Data Interrupt

The interrupt occurs when the following properties are true:

- The receive interrupt is turned on (CRB[18]).
- The transmit data register is empty.

To service the interrupt, the you must transmit the data, as follows:

1. Save the necessary context.
2. Load the transmit buffer pointer.
3. Move the transmit buffer data to the transmit register.
4. Update the transmit buffer pointer.
5. Restore context.

Example 18 illustrates the procedures for servicing the ESSI Transmit Data interrupt.

Example 18. Servicing the ESSI Transmit Data Interrupt

```
; transmit data interrupt service routine
ssi_tx_isr

; Save Context
move          r0,x:(r7)+                ; Save r0 to the stack.
move          m0,x:(r7)+                ; Save m0 to the stack.
move          #1,m0                     ; Modulus 2 buffer.


move          x:TX_PTR,r0               ; Load the pointer to the tx
                                        ; buffer.

nop                                     ; delay

movep         x:(r0)+,x:M_TX00          ; SSI transfer data register.

move          r0,x:TX_PTR               ; Update tx buffer pointer.

                                        ;Restore Context

move          x:-(r7),m0                ; Restore m0.
move          x:-(r7),r0                ; Restore r0.

rti
```

4 Example Application

An example program is provided to illustrate the use of the CODEC. The following files are included in a package to be distributed with this document:

- `Ioequ.asm`Important I/O equates
- `Intequ.asm` Interrupt equates for the DSP EVM modules
- `Ada_equ.asm`Equates to initialize the CODEC
- `Ada_Init.asm` Initialization code for the ESSI and CODEC
- `Vectors.asm`Vector table for the DSP EVM modules
- `Echo.asm`Sample code that illustrates DSP processing

These files include all the procedures discussed in **Section 2.3**, "Digital Interface (ESSI – CODEC)"and **Section 3**, "Programming the CS4218 CODEC". They assist you to quickly generate an application using the CS4218 CODEC. If a desired property in the control information is needed, you can make simple modifications to these files.

4.1 Echo Program

The echo program example simulates an echo of an input signal using a number of time-delayed samples. To implement a time-delayed echo on the DSP, a sample is fed into the DSP from the CODEC. The new sample is divided by two to maintain stability and then added to a time-delayed sample. The sum of the signals is again divided by two and then sent out to the CODEC. **Figure 3** displays the block diagram of this process.

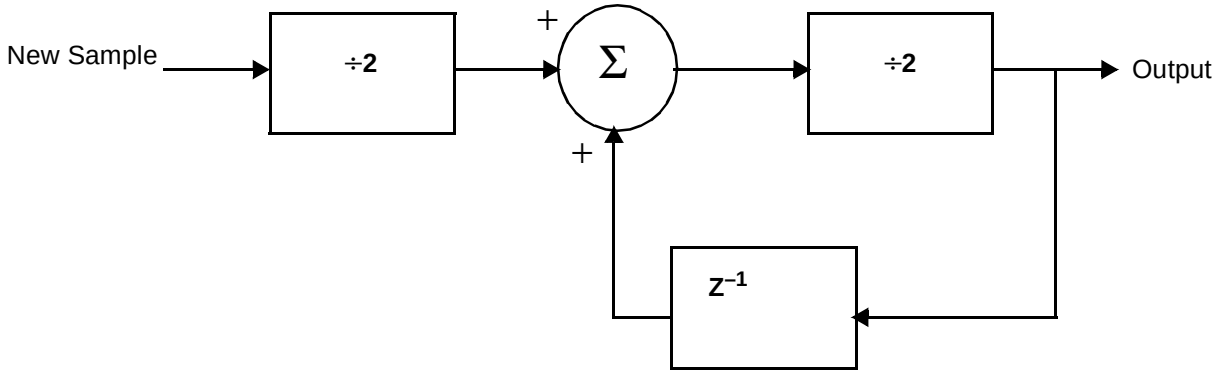

Figure 3. Block Diagram of a Delayed Sample (Echo)

4.2 Echo Code

The control information constants for the CODEC must be defined, as follows:

1. Include CODEC and I/O files.
2. Define transmit and receive buffer and pointers.
3. Define CODEC control constants.

Example 19 illustrates the tasks of including initialization and interface files, defining transmit and receive buffers and pointers, and setting up control word constants.

Example 19. Include, Define, and Set-Up Tasks

```
;*********************************************************************

      nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
       list

;*********************************************************************
;---Buffer for talking to the CS4218

       org    x:$0
RX_BUFF_BASE          equ           *
RX_data_1_2           ds     1      ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4           ds     1      ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE          equ           *
TX_data_1_2           ds     1      ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4           ds     1      ; data time slot 3/4 for TX ISR (right audio)

RX_PTR          ds      1    ; Pointer for rx buffer
TX_PTR          ds      1    ; Pointer for tx buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN
```

After the constants are set up for the CODEC, the DSP is set up and initialized, as follows:

1. Notify the DSP of the speed at which the PLL is running. For this application the PLL is set to 86.016MHz.
2. Mask the interrupts with the correct values.
3. Initialize the hardware stack pointer.
4. Operate DSP on Mode 0.
5. Initialize the data interrupt stack pointer, which is the stack used in the ISR for the CODEC.
6. Assert linear addressing for the stack pointer used by the data interrupts.

Example 20 illustrates the initialization of the DSP.

Example 20. DSP Initialization Procedure

```
org     p:$100
START
main
        movep           #$040006,x:M_PCTL           ; PLL 7 X 12.288 = 86.016MHz
        ori             #3,mr                       ; mask interrupts
        movec           #0,sp                       ; clear hardware stack pointer
        move            #0,omr                      ; operating mode 0
        move            #$40,r7                     ; initialize stack pointer for ISR
        move            #-1,m7                      ; linear addressing
```

After the DSP is initialized, the CODEC must be initialized. Using the supplied code, you can make a jump statement to start the CODEC/ESSI initialization routine. **Example 21** demonstrates this procedure.

Example 21. Initializing CODEC/ESSI

```
jsr     ada_init                ;initialize CODEC/ESSI
```

The DSP and CODEC are ready to receive, process, and transmit data. The echo implementation requires that a buffer be set up and initialized. The code in **Example 22** performs these steps.

Example 22. Setting Up and Initializing Buffer

```
move            #$0400,r4               ; start echo buffer at $400
move            #$03FF,m4               ; make echo buffer 1024 deep

clr             a                       ; clear a
rep             #$03FF                  ; clear the echo buffer
move            a,l:(r4)+
```

To receive data from the ESSI port, data is received at the beginning of the frame. Check the status bits to ensure that data receive starts at the beginning of the frame and not in the middle. Once a receive frame synchronization is detected, data can move through the receive memory location, RX_BUFF_BASE. Then, the data can be processed and moved into the transmit pointer. All of these procedures can be implemented in an infinite loop to receive, process, and transmit the data continuously. **Example 23** illustrates the implementation of the echo program.

Example 23. Implementation of Echo Program

```
echo_loop

        jset            #3,x:M_SSISR0,*             ; wait for rx frame sync
        jclr            #3,x:M_SSISR0,*             ; wait for rx frame sync
        clr a
```

```
        clr b
        move            x:RX_BUFF_BASE,a            ; receive left
        move            x:RX_BUFF_BASE+1,b          ; receive right
        asr             a               x:(r4),x0   ; divide them by 2 and get oldest
        asr             b               y:(r4),y0   ; samples from buffer
        add             x0,a                        ; add the new samples and the old
        add             y0,b
        asr             a                           ; reduce magnitude of new data
                                                    ; (to ensure stability)
        asr             b
        move            a,x:(r4)                    ; save the altered samples
        move                    b,y:(r4)+           ; and bump the pointer
        move            a,x:TX_BUFF_BASE            ; transmit left
        move            b,x:TX_BUFF_BASE+1          ;transmit right

        jmp             echo_loop
echo
```

The data is quickly divided by two after it is received from the left and right channels. Then the left and right channels are added to the time-delayed samples, which are stored on the echo buffer. The magnitude is reduced by two, and the echo buffer is updated with the newest output sample. The left-and-right processed channels are then sent to the transmit buffers and on to the ESSI port and finally to the CODEC. The procedures loop infinitely until manually stopped. **Example 24** combines all the separate pieces of the echo code into an application that performs the time-delayed echo.

Example 24. Application of Echo Code

```
;*********************************************************************

      nolist
      include 'ioequ.asm'
      include 'intequ.asm'
      include 'ada_equ.asm'
      include 'vectors.asm'
       list

;*********************************************************************

;---Buffer for talking to the CS4218

       org    x:$0
RX_BUFF_BASE          equ           *
RX_data_1_2           ds     1      ; data time slot 1/2 for RX ISR (left audio)
RX_data_3_4           ds     1      ; data time slot 3/4 for RX ISR (right audio)

TX_BUFF_BASE          equ    *
TX_data_1_2           ds     1      ; data time slot 1/2 for TX ISR (left audio)
TX_data_3_4           ds     1      ; data time slot 3/4 for TX ISR (right audio)

RX_PTR                       ds     1       ; Pointer for rx buffer
TX_PTR                       ds     1       ; Pointer for tx buffer

CTRL_WD_12      equ     MIN_LEFT_ATTN+MIN_RIGHT_ATTN+LIN2+RIN2
CTRL_WD_34      equ     MIN_LEFT_GAIN+MIN_RIGHT_GAIN
```

```
        org     p:$100
START
main
        movep           #$040006,x:M_PCTL           ; PLL 7 X 12.288 = 86.016MHz
        ori             #3,mr                       ; mask interrupts
        movec           #0,sp                       ; clear hardware stack pointer
        move            #0,omr                      ; operating mode 0
        move            #$40,r7                     ; initialize stack pointer for isr
        move            #-1,m7                      ; linear addressing
        jsr             ada_init                    ; initialize codec

        move            #$0400,r4                   ; start echo buffer at $400
        move            #$03FF,m4                   ; make echo buffer 1024 deep

        clr             a                           ; clear a
        rep             #$03FF                      ; clear the echo buffer
        move            a,l:(r4)+

echo_loop

        jset            #3,x:M_SSISR0,*                 ; wait for rx frame sync
        jclr            #3,x:M_SSISR0,*                 ; wait for rx frame sync
        clr a
        clr b
        move            x:RX_BUFF_BASE,a                ; receive left
        move            x:RX_BUFF_BASE+1,b              ; receive right
        asr             a       x:(r4),x0               ; divide them by 2 and get oldest
        asr             b       y:(r4),y0               ; samples from buffer
        add             x0,a                            ; add the new samples and the old
        add             y0,b
        asr             a                               ; reduce magnitude of new data
                                                    ; (to ensure stability)
        asr             b
        move            a,x:(r4)                        ; save the altered samples
        move            b,y:(r4)+                       ; and bump the pointer
        move            a,x:TX_BUFF_BASE                ; transmit left
        move            b,x:TX_BUFF_BASE+1              ; transmit right

        jmp             echo_loop

        include 'ada_init.asm'                      ; used to include codec
                                                    ; initialization routines
echo
         end
```

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN1790
Rev. 1
8/2005