APR39
Rev. 1, 8/2005

Programming the DSP56300 Enhanced Filter Coprocessor (EFCOP)

Tina M. Redheendran

The enhanced filter coprocessor (EFCOP) is a is a fully programmable complex filter that functions as a general-purpose peripheral module of the DSP56307, DSP56311, DSP56321, and DSP56L307 devices. The EFCOP optimized modes of operation perform complex finite impulse response (FIR) filtering, infinite impulse response (IIR) filtering, adaptive FIR filtering, and multichannel FIR filtering. The EFCOP filter operations are completed concurrently with the DSP56300 core operations with minimal CPU intervention.

The EFCOP has dedicated modes of operation optimized for cellular basestation applications. In a transceiver basestation, the EFCOP can perform complex matched filtering to maximize the signal-to-noise ratio (SNR) within an equalization process. In a transcoder basestation or a mobile switching center, the EFCOP can perform all types of FIR and IIR filtering within a vocoder, as well as LMS-type echo cancellation.

This document describes the EFCOP programming model and presents two application examples:

- A complete IIR filter
- An LMS echo canceller

It is assumed that you have access to the available documentation for your DSP56300 device, which is located on the website listed on the back cover of this document. You can download the example EFCOP code presented in this application report from the same web site.

CONTENTS

1 EFCOP Programming Model

This section describes the registers for configuring and operating the EFCOP. The *DSP56307 User's Manual* discusses EFCOP programming in detail, including the basic types of filter algorithms that can be processed.

1.1 EFCOP Description

As **Figure 1** shows, the EFCOP comprises the following main functional blocks:

- Peripheral module bus (PMB) interface, including:
  - Data input buffer
  - Constant input buffer
  - Output buffer
  - Filter counter
- Filter data memory (FDM) bank
- Filter coefficient memory (FCM) bank
- Filter multiplier-accumulator (FMAC) machine
- Address generator
- Control logic

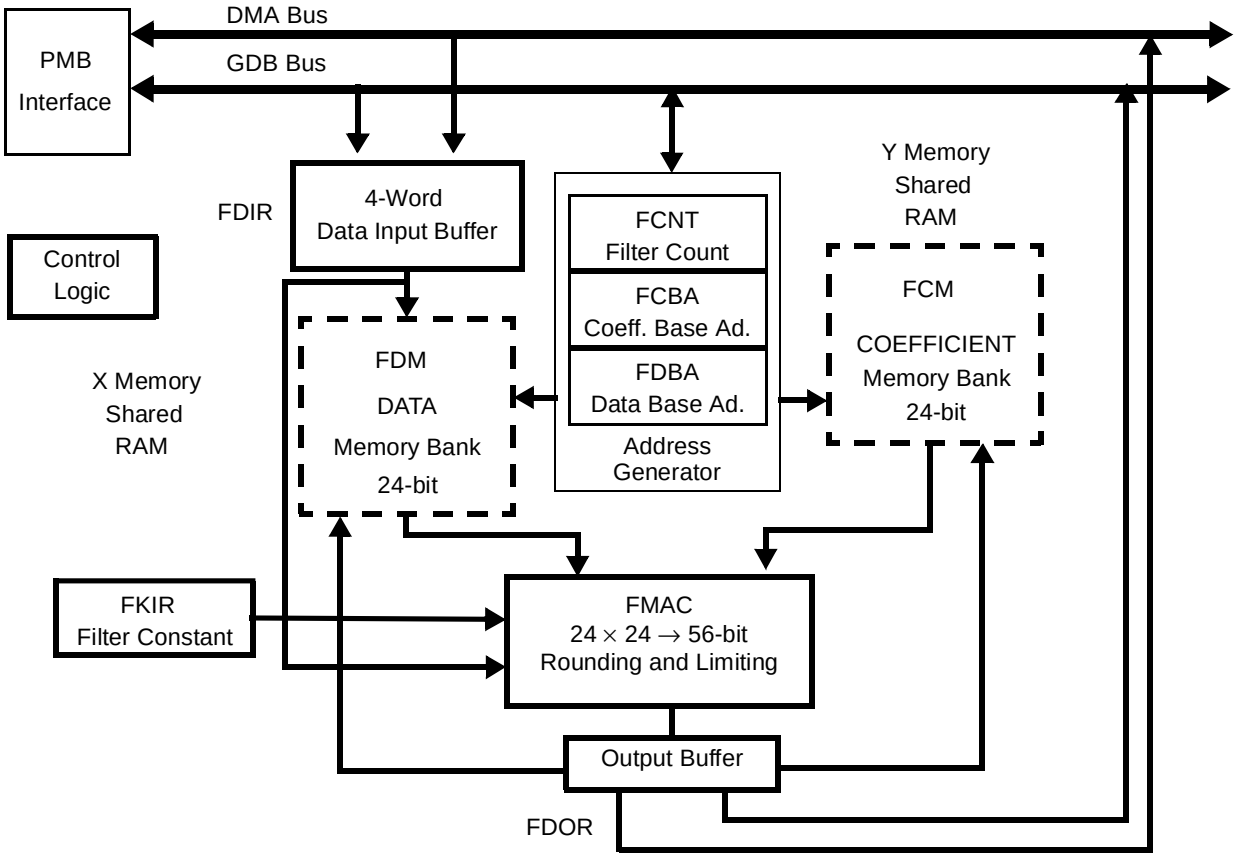

Figure 1. EFCOP Block Diagram

1.2 EFCOP Registers

Table 1 lists the EFCOP registers available to the digital signal processing programmer. The next subsections describe these registers in detail.

Table 1. EFCOP Registers and Base Addresses

Address	EFCOP Register Name
$FFFFB0	Filter data input register (FDIR)
$FFFFB1	Filter data output register (FDOR)
$FFFFB2	Filter K-constant register (FKIR)
$FFFFB3	Filter count register (FCNT)
$FFFFB4	Filter control status register (FCSR)
$FFFFB5	Filter ALU control register (FACR)
$FFFFB6	Filter data buffer base address (FDBA)
$FFFFB7	Filter coefficient base address (FCBA)
$FFFFB8	Filter decimation/channel register (FDCH)

1.2.1 Filter Data Input Register (FDIR)

The FDIR is a 24-bit 4-word-deep FIFO for DSP-to-EFCOP data transfers. Up to four data samples can be written into the FDIR using the same address. Data from the FDIR is transferred to the FDM for filter processing. For proper operation, write data to the FDIR only if the FDIBE status bit is set, indicating that the FIFO is empty. Writing to the FDIR clears the FDIBE bit. Data transfers can be triggered by an interrupt request (for core transfers) or a DMA request (for DMA transfers). Both the DSP56300 core and the DMA controller can access the FDIR for writes.

1.2.2 Filter Data Output Register (FDOR)

The FDOR is a 24-bit read-only register for EFCOP-to-DSP data transfers. The result of the filter processing is transferred from the FMAC to the FDOR. For proper operation, read data from the FDOR only if the FDOBF status bit is set, indicating that the FDOR contains data. Reading from the FDOR clears the FDOBF bit. Data transfers can be triggered by an interrupt request (for core transfers) or a DMA request (for DMA transfers). The FDOR is accessible for reads by the DSP56300 core and the DMA controller.

1.2.3 Filter K-Constant Input Register (FKIR)

The FKIR is a 24-bit write-only register for DSP-to-EFCOP constant transfers. The filter constants are written to the FKIR before echo cancellation processing and transferred to the FMAC adder. The FKIR is accessible for reads or writes only by the DSP56300 core.

1.2.4 Filter Count Register (FCNT)

The FCNT register is a 24-bit read/write register for selecting the filter length (number of filter taps). Always write the initial count into the FCNT register before enabling the EFCOP—that is, setting the FEN bit (bit 0 of the FCSR). Do not change the contents of the FCNT register unless the EFCOP is in the individual reset state (FEN = 0). In the individual reset state, the EFCOP module is inactive, but the contents of the FCNT register are preserved. **Table 2** describes the FCNT register bits.

Table 2. FCNT Register Bits

Bit Number	Mnemonic	Value	Function
23–12	—		These bits are reserved and should be written with 0
11–0	FCNT		**Filter Count**
		—	These bits should be written with the number of coefficient values minus one

1.2.5 EFCOP Control Status Register (FCSR)

The FCSR is a 24-bit read/write register by which the DSP56300 core controls the main operation modes of the EFCOP and monitors the EFCOP status. All FCSR bits are cleared after hardware and software reset. To ensure proper operation, do not change the FCSR bits unless the EFCOP is in individual reset state (i.e., FEN = 0) except FEN, FDIOE, FDIIE, FUPD, and FADP. **Table 3** describes the FCSR bits.

Table 3. FCSR Bits

Bit Number	Mnemonic	Value	Function
23–16	—		These bits are reserved and should be written with 0
15	FDOBF		**Filter Data Output Buffer Full** - status bit
		0	FDOR is empty
		1	FDOR is full and ready to be read by the Core or DMA
14	FDIBE		**Filter Data Input Buffer Empty** - status bit
		0	FDIR is full
		1	FDIR is empty and ready to be written to by the Core or DMA
13	FCONT		**Filter Contention** - sticky status bit
		0	Memory contention has not occurred
		1	Memory contention occurred between the Core and the EFCOP
12	FSAT		**Filter Saturation** - sticky status bit
		0	Overflow or underflow has not occurred
		1	Overflow or underflow occurred
11	FDOIE		**Filter Data Output Interrupt Enable**
		0	Interrupt disabled
		1	Interrupt enabled
10	FDIIE		**Filter Data Input Interrupt Enable**
		0	Interrupt disabled
		1	Interrupt enabled
9	—		This bit is reserved and should be written with 0
8	FSCO		**Filter Shared Coefficients mode** -valid only in multichannel mode (FMLC bit in FCSR = 1)
		0	Sequential coefficients
		1	Shared coefficients
7	FPRC		**Filter Processing State Initialization mode** - valid only with FIR filter type (FLT bit in FCSR = 0)
		0	Initialization enabled
		1	Initialization disabled

Table 3. FCSR Bits (Continued)

Bit Number	Mnemonic	Value	Function
6	FMLC		Filter Multichannel mode
		0	Multichannel mode disabled
		1	Multichannel mode enabled
5–4	FOM		**Filter Operation mode** - valid only with FIR filter type (FLT bit in FCSR = 0)
		00	Mode 0: Real FIR filter
		01	Mode 1: Full complex FIR filter
		10	Mode 2: Complex FIR filter with alternate real and imaginary outputs
		11	Mode 3: Magnitude
3	FUPD		**Filter Update** - valid only with FIR filter type (FLT bit in FCSR = 0), automatically cleared by the EFCOP and automatically set in adaptive mode (FADP bit in FCSR = 1)
		0	Coefficient update is complete
		1	Begin coefficient update
2	FADP		**Filter Adaptive mode** - valid only with FIR filter type (FLT bit in FCSR = 0)
		0	Adaptive mode disabled
		1	Adaptive mode enabled
1	FLT		**Filter Type**
		0	FIR filter
		1	IIR filter
0	FEN		**Filter Enable**
		0	EFCOP disabled and in the individual reset state
		1	EFCOP enabled

1.2.6 EFCOP ALU Control Register (FACR)

The FACR is a 24-bit read/write register by which the DSP56300 core controls the main operation modes of the EFCOP arithmetic logic unit (ALU). All FACR bits are cleared after hardware and software reset. **Table 4** describes the FACR bits.

Table 4. FACR Bits

Bit Number	Abbrev.	Value	Function
23–7	—		These bits are reserved and should be written with 0
6	FISL		**Filter Input Scale** - scaling in each case is determined by the FSCL[1:0] bits in the FCSR
		0	Scale both the IIR feedback terms and the IIR input
		1	Scale only the IIR feedback terms
5	FSA		**Filter Sixteen-bit Arithmetic mode**
		0	Disables sixteen-bit arithmetic mode
		1	Enables sixteen-bit arithmetic mode
4	FSM		**Filter Saturation mode**
		0	Disables saturation mode

Table 4. FACR Bits (Continued)

Bit Number	Abbrev.	Value	Function
		1	Enables saturation mode
3–2	FRM		**Filter Rounding mode**
		00	Convergent rounding
		01	Twos complement rounding
		10	Truncation (no rounding)
		11	Reserved
1–0	FSCL		**Filter Scaling**
		00	Scaling factor = 1 (no shift)
		01	Scaling factor = 8 (3-bit arithmetic left shift)
		10	Scaling factor = 16 (4-bit arithmetic left shift)
		11	Reserved

1.2.7 EFCOP Data Base Address (FDBA)

The FDBA is a 16-bit read/write counter register used as an address pointer to the EFCOP FDM bank. The FDBA points to the location to write the next data sample. The FDBA points to a modulo delay buffer of size M, defined by the filter length (M = FCNT[11:0] + 1). The address range of this modulo delay buffer is defined by lower and upper address boundaries. The lower address boundary is the FDBA value with 0s in the k LSBs, where $2^k \geq M \geq 2^{k-1}$, and therefore must be a multiple of 2^k. The upper boundary is equal to the lower boundary plus (M – 1). Since $M \leq 2^k$ once M is chosen (FCNT is assigned), the sequential series of data memory blocks (each of length 2^k) is created where multiple circular buffers for multichannel filtering can be located. If $M < 2^k$, there is a space between sequential circular buffers of $2^k - M$. The address pointer is not required to start at the lower address boundary or to end on the upper address boundary. It can point anywhere within the defined modulo address range. If the data address pointer (FDBA) increments and reaches the upper boundary of the modulo buffer, it wraps around to the lower boundary.

1.2.8 EFCOP Coefficient Base Address (FCBA)

The FCBA is a 16-bit read/write counter register used as an address pointer to the EFCOP FCM bank. The FCBA points to the first location of the coefficient table. The FCBA points to a modulo buffer of size M, defined by the filter length (M = FCNT[11:0] + 1). The address range of this modulo buffer is defined by lower and upper address boundaries. The lower address boundary is the FCBA value with 0s in the k LSBs, where $2^k \geq M \geq 2^{k-1}$, and therefore must be a multiple of 2^k. The upper boundary is equal to the lower boundary plus (M – 1). Since $M \leq 2^k$ once M is chosen (FCNT is assigned), the sequential series of coefficient memory blocks (each of length 2^k) is created where multiple circular buffers for multichannel filtering can be located. If $M < 2^k$, there is a space between sequential circular buffers of $2^k - M$. The FCBA address pointer must be assigned to the lower address boundary (must have k 0s in its LSBs). In a compute session, the coefficient address pointer always starts at the lower boundary and ends at the upper address boundary. Therefore, reading FCBA always gives the value of the lower address boundary.

1.2.9 Decimation/Channel Count Register (FDCH)

The FDCH is a 24-bit read/write register for setting the number of channels used in multichannel mode and setting the decimation ratio in FIR filter mode. FDCH should be written before the EFCOP is enabled—that is, setting the FEN bit (bit 0 of the FCSR). FDCH should be changed only when the EFCOP is in the individual reset state (FEN = 0). Otherwise, improper operation may result. In the individual reset state, the EFCOP module is inactive, but the contents of the FDCH register are preserved. **Table 5** describes the FDCH bits.

Table 5. FDCH Register Bits

Bit Number	Abbrev.	Value	Function
23–12	—		These bits are reserved and should be written with 0
11–8	FDCM		**Filter Decimation**
			These bits should be written with the decimation factor minus one
7–6	—		These bits are reserved and should be written with 0
5–0	FCHL		**Filter Channels** - valid only in multichannel mode (FMLC bit of FCSR = 1)
			These bits should be written with the number of channels minus one

2 IIR Filter Example

This section describes how to implement a complete infinite impulse response (IIR) filter using the EFCOP. It gives the theoretical background, the filter design, the example code, and the results of the example filter.

2.1 IIR Filter Theory

The difference equation for an IIR filter is:

Equation 1

$$y(n) = \sum_{i=0}^{N} B_i x(n-i) + \sum_{j=1}^{M} A_j y(n-j)$$

where $x(n)$ is the filter input at time n, $y(n)$ is the filter output at time n, N is the number of feed-forward filter coefficients minus one, B_i are the feed-forward filter coefficients, M is the number of feed-back filter coefficients, and A_j are the feed-back filter coefficients.

Equation 1 can be rewritten as:

Equation 2

$$w(n) = \sum_{i=0}^{N} B_i x(n-i)$$

and

Equation 3

$$y(n) = S\left(w(n) + \sum_{j=1}^{M} A_j y(n-j)\right)$$

where all the coefficients are scaled down by *S*. The block diagram of Equation 2 and Equation 3 is shown in **Figure 2**.

The EFCOP implements an IIR filter using the logic of **Figure 2**. First, an FIR mode session calculates *w(n)* using Equation 2 and *x(n)* as the input. Then, an IIR mode session calculates *y(n)* using Equation 3 and *w(n)* as the input.

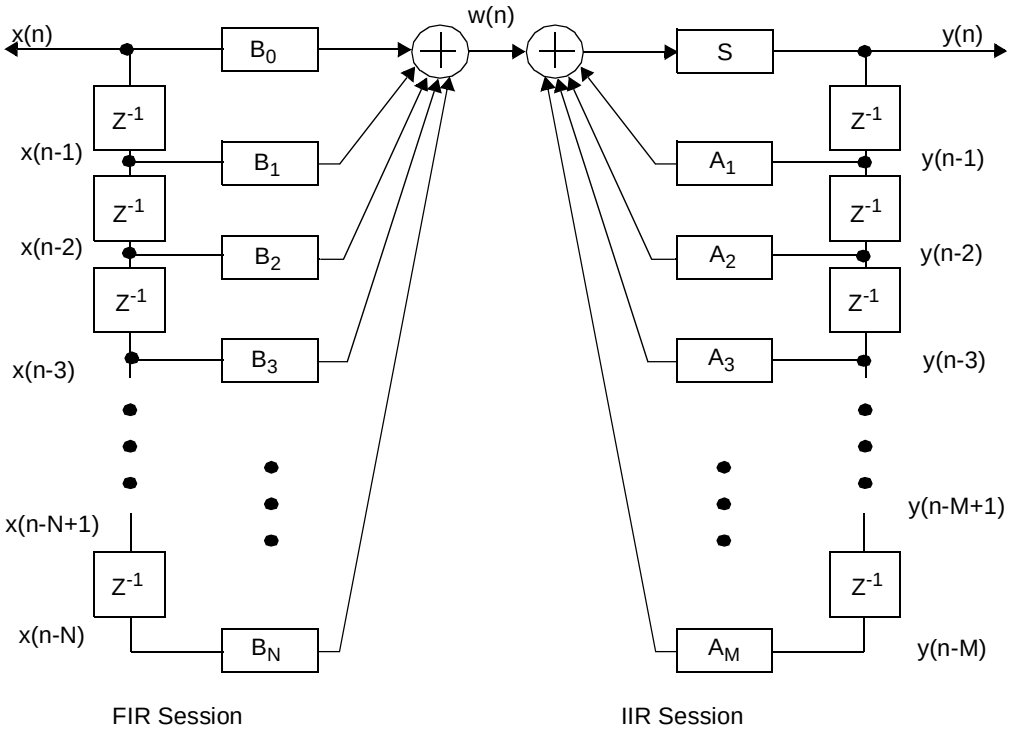

Figure 2. General IIR Block Diagram

2.2 IIR Filter Design

This example implements a butterworth lowpass filter with M = N = 3 and a cut-off frequency of $0.8W_n$, where W_n is half the sampling rate. The filter coefficients for these design parameters (determined using Matlab) are shown in **Table 6**.

Table 6. Example Filter Coefficients

$B_0 = 0.5276$	—
$B_1 = 1.5829$	$A_1 = -1.7600$
$B_2 = 1.5829$	$A_2 = -1.1829$
$B_3 = 0.5276$	$A_3 = -0.2781$

Many of these coefficients have magnitudes greater than 1, which cannot be expressed in the DSP's fixed point numerical representation. Thus, the coefficients are scaled down by eight before they are used with the EFCOP and the EFCOP scaling factor bits are set to scale up the output of the IIR filter by eight. **Table 7** shows the scaled coefficients.

Table 7. Scaled Example Coefficients

B_0 = 0.0660	—
B_1 = 0.1979	A_1 = -0.2200
B_2 = 0.1979	A_2 = -0.1479
B_3 = 0.0660	A_3 = -0.0348

Figure 3 shows the block diagram for this example.

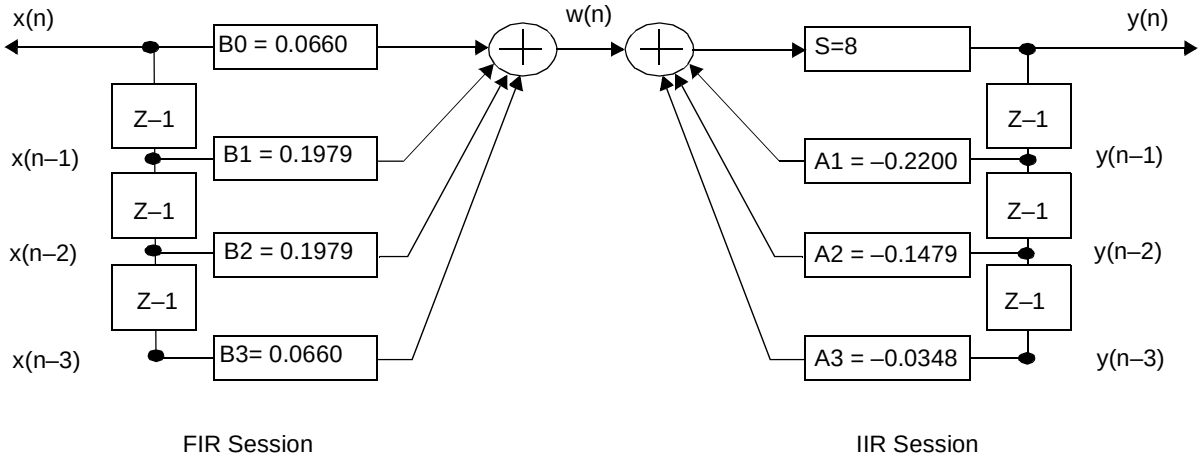

Figure 3. IIR Block Diagram

2.3 IIR Filter Example Code

The IIR filter example code is divided into four sections:

- Initialize the constants
- Implement the FIR filter session
- Implement the IIR filter session
- Initialize the filter input, coefficients, and taps

2.3.1 Initialization of Constants

The first section of the code, shown in **Example 1**, initializes the filter constants and defines the constants to control the EFCOP and DMA data transfers. The input/output equate and interrupt equate files are included. The following memory address locations are initialized:

`START` Start of the program.

`INPUT` Input data *x(n)*.

`FIR_OUT` Output of the FIR session and input of the IIR session *w(n)*.

OUTPUT	Output of the IIR session *y(n)*.
FIR_FDBA IIR_FDBA FIR_FCBA IIR_FCBA	Memory address pointers for the FIR and IIR filter data and coefficient buffers. These constants are written to the EFCOP data buffer base address (FDBA) and the EFCOP coefficient buffer base address (FCBA). The EFCOP shares the lowest 4K memory locations of X and Y memory with the DSP core for the data and coefficient buffers, respectively.

The constant initialization section defines the following constants to control the EFCOP:

FIR_FCSR	Written to the EFCOP control status register (FCSR) to control the main operation modes of the EFCOP. This constant configures the EFCOP in real FIR filter mode with processing initialization disabled, and it sets the EFCOP enable bit for the FIR filter session.
IIR_FCSR	Written to the EFCOP control status register (FCSR) to control the main operation modes of the EFCOP. This constant configures the EFCOP in IIR filter mode, and it sets the EFCOP enable bit for the IIR filter session.
IIR_FACR	Written to the EFCOP ALU control register (FACR) to control the main operation of the EFCOP ALU for the IIR filter session. The IIR_FACR constant sets the scaling factor of the IIR filter output to eight.
FIR_LEN	Defines the filter length. FIR_LEN is set to four because there are four FIR (feed-forward) filter coefficients, B_i, $i=0...3$ for this example. FIR_LEN – 1 is written to the EFCOP filter count register (FCNT) for the FIR filter session.
IIR_LEN	Defines the filter length. IIR_LEN is set to three because there are three IIR (feed-back) filter coefficients, A_j, $j=1...3$ for this example. IIR_LEN – 1 is written to the FCNT register for the IIR filter session.

The constant initialization section also defines constants to control the DMA transfers. The code uses two DMA channels, channel 0 to transfer the input data to EFCOP data input register (FDIR) and channel 1 to transfer the output data from the EFCOP data output register (FDOR).

FIR_NUMIN	Written to DMA counter register 0 (DCO0) to set the number of DMA transfers to FDIR for the FIR session.
IIR_NUMIN	Written to DMA counter register 0 (DCO0) to set the number of DMA transfers to FDIR for the IIR session.

Because FDIR is a 4-word-deep register, mode B of the DMA transfers four input words at a time to FDIR. With mode B, DOC0 is separated into two sections: DCOL (bits 0–11) and DCOH (bits 12–23). DCOH is set to the number of transfers minus one. DCOL is set to the number of words in each transfer minus one. The input file for this example has 1024 points. Thus, DOCH is set to 255 (or $0FF) and DCOL is set to 3. The total number of words transferred is equal to $(255 + 1) * (3 + 1) = 1024$.

FIR_NUMOUT	Written to DCO1 to set the number of DMA transfers from FDOR for the FIR session.
IIR_NUMOUT	Written to DCO1 to set the number of DMA transfers from FDOR for the IIR filter session.

Because FDOR is one word deep, mode A of the DMA transfers one output word at a time from FDOR. With mode A, DCO1 is set to the number of DMA transfers minus one. Thus, FIR_NUMOUT and IIR_NUMOUT are set to the number of output values minus one, or 1023 (or $3FF).

Example 1. IIR Filter Constant Initialization

```
;*****************************************************************
       nolist
       INCLUDE "ioequ.asm"
       INCLUDE "intequ.asm"
       list
;*****************************************************************
;      CONSTANTS
;*****************************************************************
START          equ $100       ; Main program starting address
INPUT          equ $2000      ; FIR session source address
FIR_OUT        equ $1000      ; FIR session destination address
OUTPUT         equ $3000      ; IIR session destination address
FIR_FDBA       equ 0          ; FIR Data Start Address x:$0
IIR_FDBA       equ 100        ; IIR Data Start Address x:$100
FIR_FCBA       equ 0          ; FIR Coeff Start Address y:$0
IIR_FCBA       equ 100        ; IIR Coeff Start Address y:$100
FIR_FCSR       equ $081       ; Enable EFCOP FIR Mode 0
IIR_FCSR       equ $003       ; Enable EFCOP IIR Mode 0
IIR_FACR       equ $001       ; Enable EFCOP IIR Scale by 8 Mode
FIR_LEN        equ 4          ; EFCOP FIR length
IIR_LEN        equ 3          ; EFCOP IIR length
FIR_NUMIN      equ $0FF003    ; DMA0 Count (256*4=1024 word xfers) FIR inputs
FIR_NUMOUT     equ $3FF       ; DMA1 Count (1024 word xfers) FIR outputs
IIR_NUMIN      equ $0FF003    ; DMA0 Count (256*4=1024 word xfers) IIR inputs
IIR_NUMOUT     equ $3FF       ; DMA1 Count (1024 word xfers) IIR outputs
```

2.3.2 FIR Filter Session

The second part of the code, shown in **Example 2**, implements the FIR filter session and calculates *w(n)* from Equation 2. The reset vector is set to the beginning of the program. The `FIR_LEN, FIR_FDBA`, and `FIR_FCBA` constants are written to the appropriate EFCOP registers, as described in Section 2.3.1, “Initialization of Constants.” `FIR_FCSR` is written to the FCSR to enable the EFCOP.

Channel 0 of the DMA transfers the input data from memory to the FDIR four words at a time. **Figure 4** shows how the DMA transfer is completed. The DMA is initialized to complete this transfer as follows:

- *Identify the source of the data transfer*. The memory address location of the input data, `INPUT`, is written to the DMA source address register for channel 0 (DSR0).
- *Identify the destination of the data transfers*. The memory-mapped address location of the FDIR is written to the DMA destination address register for channel 0 (DDR0).
- *Specify the number of data transfers*. `FIR_NUMIN`, which is described in Section 2.3.1, “Initialization of Constants.” is written to DCO0.
- *Designate the offset increment.* The DMA offset register 0 (DOR0) is used with mode B to increment the DMA source address register after each transfer. For this example, the input data is stored sequentially in memory. Therefore, DOR0 is written with the number 1 to increment the DMA source address register by one after each transfer.
- *Specify the transfer properties.* The DMA control register for channel 0 (DCR0) controls the DMA channel 0 operation. The value written to DCR0 sets the transfer to trigger from the EFCOP input buffer empty request. This value also sets the source transfer to mode to B using the offset register

DOR0. The destination transfer mode is set to A with no updating of the destination register because the input data should always be transferred to the FDIR. The source memory space is set to X memory because the input data is stored in X memory, as discussed in Section 2.3.4, “Coefficients, Taps, and Input.” The destination memory space is set to Y memory because all EFCOP registers including FDIR are mapped to internal Y I/O memory. Finally, DMA channel 0 is enabled.

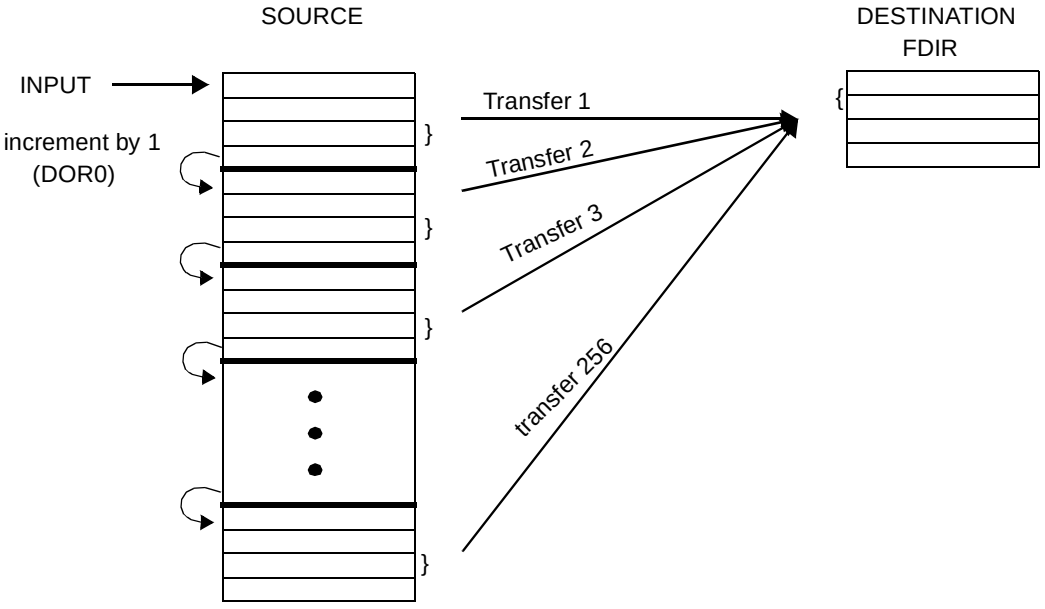

Figure 4. DMA Channel 0 Transfer

Channel 1 of the DMA transfers the output data from the FDOR to memory. **Figure 5** shows how the DMA transfer is completed. The DMA is initialized to complete this transfer as follows:

- *Identify the source of the data transfer.* The memory-mapped address location of the FDOR is written to the DMA source address register for channel 1 (DSR1).
- *Identify the destination of the data transfer*. The memory address location of the FIR output data, `FIR_OUT`, is written to the DMA destination address register for channel 1 (DDR1).
- *Specify the number of data transfers.* `FIR_NUMOUT`, which is described in Section 2.3.1, “Initialization of Constants.” is written to DCO1.
- *Specify the transfer properties.* The DMA control register for channel 1 (DCR1) controls the DMA channel 1 operation. The value written to DCR1 sets the transfer to trigger from the EFCOP output buffer full request. This value also sets the source transfer to mode to A with no updating of the source register because the output data should always be transferred from FDOR. The destination transfer mode is set to A with post increment by one because the output data is stored sequentially to memory. The source memory space is set to Y memory because all EFCOP registers including FDOR are mapped to internal Y I/O memory. The destination memory space is set to X memory because the FIR output data is stored in X memory. Finally, DMA channel 1 is enabled.

Bits 0 and 1 of the DMA status register (DSTR) are set when the last word is stored in the destination and channel operation completes for channels 0 and 1, respectively. The program polls these bits and waits until the DMA transfers complete before continuing. Finally, the EFCOP is put into personal reset mode by clearing FCSR so that the EFCOP can be programmed for the IIR filter session.

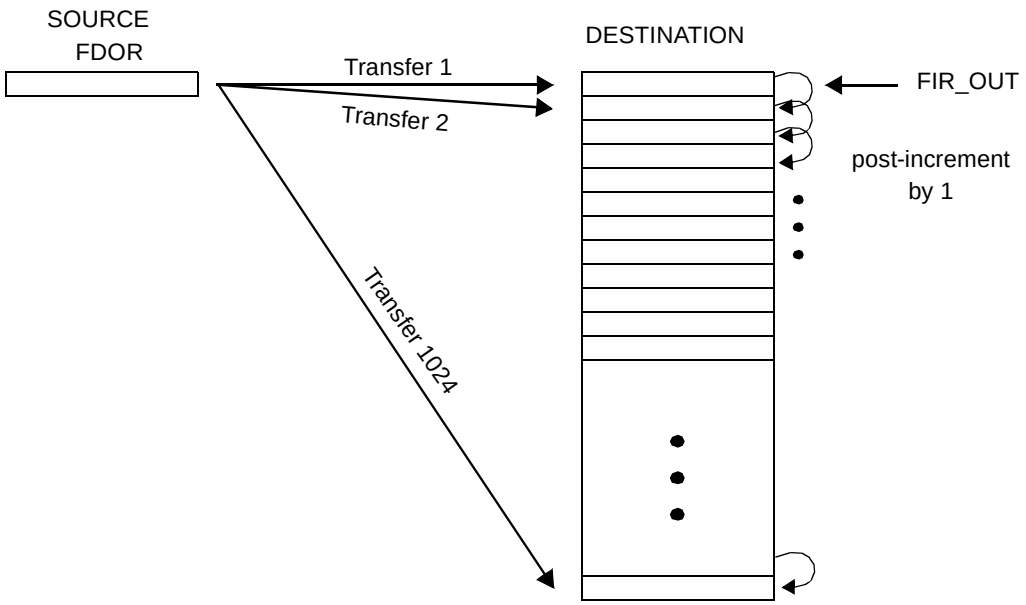

Figure 5. DMA Channel 1 Transfer

Example 2. FIR Filter Session Code

```
;*********************************************************************
;*             FIR Filter Section
;*********************************************************************
        org     P:0
        jmp     START

        org     P:START
        movep   #FIR_LEN-1,y:M_FCNT          ; FIR length
        movep   #FIR_FDBA,y:M_FDBA           ; FIR Data Start Address
        movep   #FIR_FCBA,y:M_FCBA           ; FIR Coeff Start Address
        movep   #FIR_FCSR,y:M_FCSR           ; Enable EFCOP

; DMA 0 init to input DATA to EFCOP

        movep   #INPUT,x:M_DSR0              ; DMA source is the INPUT data buffer
        movep   #M_FDIR,x:M_DDR0             ; DMA destination is the EFCOP input register
        movep   #FIR_NUMIN,x:M_DCO0          ; DMA count in mode B
        movep   #$1,x:M_DOR0                 ; DMA offset is 1
        movep   #$94AA04,x:M_DCR0            ; DMA control reg with line mode FDIBE request

; DMA 1 init to output DATA from EFCOP

        movep   #M_FDOR,x:M_DSR1             ; DMA source is the EFCOP output register
        movep   #FIR_OUT,x:M_DDR1            ; DMA destination is the FIR_OUT data buffer
        movep   #FIR_NUMOUT,x:M_DCO1         ; DMA count
        movep   #$8EB2C1,x:M_DCR1            ; DMA control register with FDOBF request

        jclr    #0,x:M_DSTR,*                ; Wait till DMA 0 ends
        jclr    #1,x:M_DSTR,*                ; Wait till DMA 1 ends
        movep   #$000,y:M_FCSR               ; Reset EFCOP
```

2.3.3 IIR Filter Session

The third part of the code, shown in **Example 3**, implements the IIR filter session and calculates *y(n)* from Equation 3. The `IIR_LEN, IIR_FDBA, IIR_FCBA,` and `IIR_FACR` constants are written to the appropriate EFCOP registers, as described in Section 2.3.1, “Initialization of Constants.” `IIR_FCSR` is written to the FCSR to enable the EFCOP.

Channel 0 of the DMA transfers the input data from memory to the FDIR, four words at a time. **Figure 4** shows how the DMA transfer is completed except that the source data is located at `FIR_OUT` instead of `INPUT`. The DMA is initialized to complete this transfer as follows:

- *Identify the source of the data transfer*. The memory address location of the input data, in this case `FIR_OUT`, is written to DSR0.
- *Identify the destination of the data transfer*. The memory-mapped address location of the FDIR is written to DDR0.
- *Specify the number of data transfers.* `IIR_NUMIN,` which is described in Section 2.3.1, “Initialization of Constants.” is written to DCO0.
- *Designate the offset increment.* DOR0 is used with mode B to increment the DMA source address register after each transfer. For this example, the input data is stored sequentially in memory. Therefore, DOR0 is written with the number 1 to increment the DMA source address register by one after each transfer.
- *Specify the transfer properties.* DCR0 controls the DMA channel 0 operation. The value written to DCR0 sets the transfer to trigger from the EFCOP input buffer empty request. This value also sets the source transfer to mode B using the offset register DOR0. The destination transfer mode is set to A with no updating of the destination register because the input data should always be transferred to FDIR. The source memory space is set to X memory because the input data is stored in X memory. The destination memory space is set to Y memory because all EFCOP registers including the FDIR are mapped to internal Y I/O memory. Finally, DMA channel 0 is enabled.

Channel 1 of the DMA transfers the output data from FDOR to memory. **Figure 5** shows how the DMA transfer is completed except that the destination data is located at `OUTPUT` instead of `FIR_OUT`. The DMA is initialized to complete this transfer as follows:

- *Identify the source of the data transfer*. The memory-mapped address location of FDOR is written to the DMA source address register for channel 1 (DSR1).
- *Identify the destination of the data transfer*. The memory address location of the IIR output data, `OUTPUT`, is written to the DMA destination address register for channel 1 (DDR1).
- *Specify the number of data transfers.* `IIR_NUMOUT`, which is described in Section 2.3.1, “Initialization of Constants.” is written to DCO1.
- *Specify the transfer properties.* The DMA control register for channel 1 (DCR1) controls the DMA channel 1 operation. The value written to DCR1 sets the transfer to trigger from the EFCOP output buffer full request. This value also sets the source transfer to mode to A with no updating of the source register because the output data should always be transferred from FDOR. The destination transfer mode is set to A with post increment by one because the output data is stored sequentially to memory. The source memory space is set to Y memory because all EFCOP registers including FDOR are mapped to internal Y I/O memory. The destination memory space is set to X memory because the IIR output data is stored in X memory. Finally, DMA channel 1 is enabled.

Bits 0 and 1 of the DMA status register (DSTR) are set when the last word is stored in the destination and channel operation completes for channels 0 and 1, respectively. The program polls these bits, and when the DMA transfers complete the program is finished.

Example 3. IIR Filter Session Code

```
;*********************************************************************
;*             IIR Filter Section
;*********************************************************************
       movep   #IIR_LEN-1,y:M_FCNT          ; IIR length
       movep   #IIR_FDBA,y:M_FDBA           ; IIR Data Start Address
       movep   #IIR_FCBA,y:M_FCBA           ; IIR Coeff. Start Address
       movep   #IIR_FACR,y:M_FACR           ; IIR Control Register
       movep   #IIR_FCSR,y:M_FCSR           ; Enable EFCOP

; DMA 0 init to input DATA to EFCOP

       movep   #FIR_OUT,x:M_DSR0            ; DMA source is the FIR_OUT data buffer
       movep   #M_FDIR,x:M_DDR0             ; DMA destination is the EFCOP input buffer
       movep   #IIR_NUMIN,x:M_DCO0          ; DMA count in mode B
       movep   #$1,x:M_DOR0                 ; DMA offset is 1
       movep   #$94AA04,x:M_DCR0            ; DMA control reg with line mode FDIBE request

; DMA 1 init to output DATA from EFCOP

       movep   #M_FDOR,x:M_DSR1             ; DMA source is the EFCOP out register
       movep   #OUTPUT,x:M_DDR1             ; DMA destination is the OUTPUT data buffer
       movep   #IIR_NUMOUT,x:M_DCO1         ; DMA count
       movep   #$8EB2C1,x:M_DCR1            ; DMA control reg with FDOBF request

       jclr    #0,x:M_DSTR,*                ; Wait till DMA 0 ends
       jclr    #1,x:M_DSTR,*                ; Wait till DMA 1 ends
stop_label
       stop
```

2.3.4 Coefficients, Taps, and Input

The final part of the code, shown in Example 3-4, initializes the coefficients, taps, and input for the filter. The coefficient values are described in Section 2.2, “IIR Filter Design.” The memory address pointers for the coefficients, `FIR_FCBA` and `IIR_FCBA`, are defined in Section 2.3.1, “Initialization of Constants.” The EFCOP shares the lowest 4K memory locations of Y memory with the DSP core for the coefficient buffers. Thus, the coefficients are stored in Y memory. *Notice that the coefficients are stored in reverse order* such that the coefficient with the largest index is stored first and the coefficient with the smallest index is stored last.

The FIR filter taps must be initialized because processing state initialization mode is disabled for the FIR filter in the `FIR_FCSR` constant. Also, the IIR filter taps must be initialized because the EFCOP assumes that the data taps are initialized before the EFCOP is enabled and therefore does not initialize the taps for IIR filter mode. The filter taps are all initialized to zero. This tells the EFCOP that the values of the FIR input *x(n)* and the IIR output *y(n)* are zero for $n < 0$. The number of taps needed for each filter is equal to the number of filter coefficients. The memory address pointers for the taps, `FIR_FDBA` and `IIR_FDBA`, are defined in Section 2.3.1, “Initialization of Constants.” The EFCOP shares the lowest 4K memory locations of X memory with the DSP core for the filter tap buffers. Thus, the filter taps are stored in X memory.

The last lines of the code specify the input data. The memory address pointer for the input data, INPUT, is defined in Section 2.3.1, “Initialization of Constants.” The file input.dat, which contains the input data, is included at this memory location. For more information on the input.dat file, consult the next section.

Example 4. Coefficients, Inputs, and Taps Code

```
;*********************************************************************
;*            COEFFICIENTS, INPUTS, & TAPS
;*********************************************************************
       org           y:FIR_FCBA
       dc            0.06595304781274      ; b(3)/8
       dc            0.19785914343823      ; b(2)/8
       dc            0.19785914343823      ; b(1)/8
       dc            0.06595304781274      ; b(0)/8

       org           y:IIR_FCBA
       dc            -0.03475748970432     ; a(3)/8
       dc            -0.14786165775473     ; a(2)/8
       dc            -0.22000523504290     ; a(1)/8

       org           x:FIR_FDBA
       dc            $000000
       dc            $000000
       dc            $000000
       dc            $000000

       org           x:IIR_FDBA
       dc            $000000
       dc            $000000
       dc            $000000

       org           x:INPUT

       INCLUDE       "input.dat"
```

2.4 Filter Results

This section describes the results for this filter example by presenting the input and the output data. The filter input data (calculated using Matlab) is gaussian random noise with a mean of 0.0 and a variance of 1.0. The data is then scaled so that the magnitudes of all of the values are less than 1. The filter output data is stored in X memory beginning at the memory address pointer, OUTPUT, that is defined in Section 2.3.1, “Initialization of Constants.”

To show the effect of the filter, the frequency spectrum of the input and output is plotted (using Matlab) in **Figure 6**. As **Figure 6** shows, the frequency spectrum of the output is the same as the frequency spectrum of the input for all frequency values less than $0.8W_n$, where W_n is half the sampling rate. However, since the output is processed through the lowpass IIR filter, the frequency spectrum of the output is greatly attenuated for frequency values greater than $0.8W_n$. Thus, the IIR filter is working properly and filtering the input signal as expected.

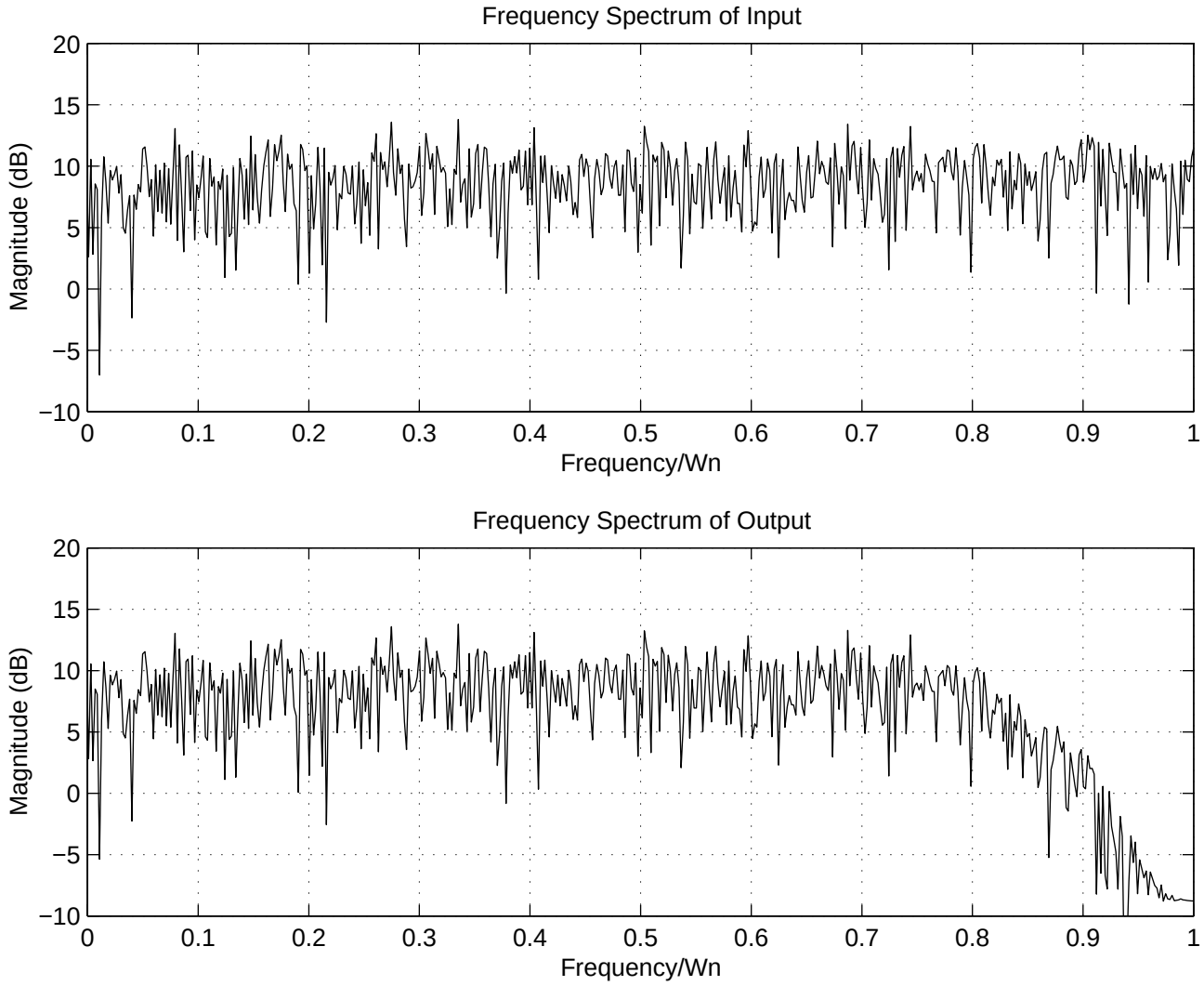

Figure 6. Frequency Spectrum of Input and Output

3 Echo Canceller Example

This section describes how to implement a complete LMS electrical echo canceller using the EFCOP. It gives the theoretical background, the filter design, the example code, and the results of the echo canceller.

3.1 Echo Canceller Theory

Figure 7 shows the block diagram for the echo canceller in this example. This figure depicts a near-end electrical echo canceller. At the near-end is a four-wire system with separate signal paths for the transmit signal and receive signal. The transmit and receive signals are combined via a hybrid into a single two-wire signal for connection to the public phone network at the far-end. The hybrid also introduces an unwanted echo of the near-end signal *x(n)* into the receive path. The adaptive filter determines the delay and attenuation of the echo introduced by the hybrid and generates an estimate of the echo, *y(n)*, that can be subtracted from the received signal + the echo, *s(n)*. The result is a cancellation of most of the echo, leaving only the desired received signal *e(n)*. The adaptive filter also uses the received signal *e(n)* to help track the delay and attenuation of the echo.

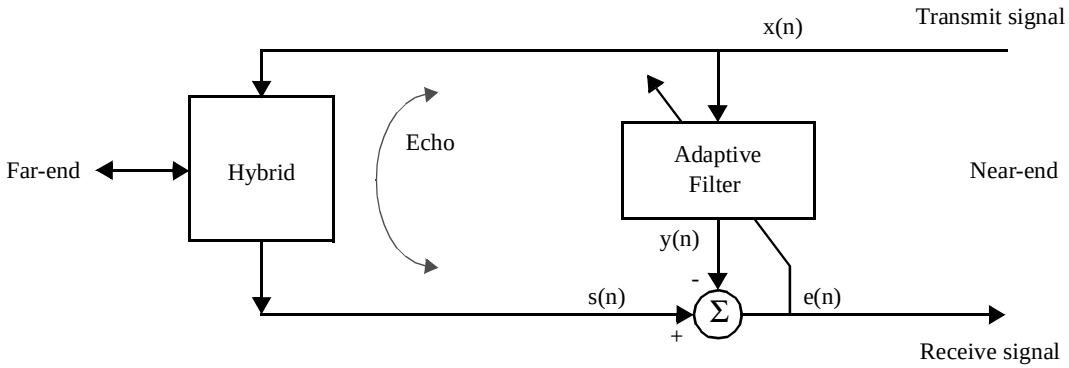

Figure 7. Echo Canceller Block Diagram

The output of the adaptive filter is calculated as follows:

Equation 4

$$y(n) = \sum_{i=0}^{L-1} h_n(i)x(n-i)$$

where $y(n)$ is the estimated echo of the near-end signal at time n, $x(n)$ is the near-end signal at time n, L is the number of filter coefficients, and $h_n(i)$ are the filter coefficients for time n. After the output signal is calculated, the filter coefficients are updated. First, the error signal is calculated by

Equation 5

$$e(n) = s(n) - y(n)$$

where $e(n)$ is the error signal or the far-end signal plus the residual echo of the near-end signal at time n, and $s(n)$ is the far-end signal plus the echo of the near-end signal at time n.

Next, the coefficient update step is calculated as follows:

Equation 6

$$K_e(n) = K(n)e(n)$$

where $K_e(n)$ is the coefficient update step at time n and $K(n)$ is the step size at time n. Finally, the filter coefficients are updated for the next time period using the following equation:

Equation 7

$$h_{n+1}(i) = h_n(i) + K_e(n)x(n-i)$$

The EFCOP implements Equation 4 using a FIR filter session. The EFCOP also implements a coefficient update session to calculate the new filter coefficients using Equation 7.

3.2 Echo Canceller Design

The example discussed in this section implements an echo canceller using an adaptive filter as previously described. Recall that the purpose of the adaptive filter is to generate an estimate of the near-end echo that can be subtracted from the far-end signal + the echo, $s(n)$. However, the adaptive filter interprets the far-end signal as noise. Thus, it is difficult for the adaptive filter to estimate the echo when the far-end signal is present. It is much

easier for the adaptive filter to estimate the echo when *s(n)* contains only the near-end echo. Therefore, some echo cancellers are designed to detect when the far-end signal is absent and update the filter coefficients only during these times. When the far-end signal is present, these echo cancellers set the step size, *K*, to zero so that the filter coefficients are not changed. In other applications, these echo cancellers might reduce the step size significantly so that the filter coefficients are changed only a small amount when the far-end signal is present.

The echo canceller in this example reduces the step size when the far-end signal is present. For an LMS echo canceller, the step size $K(n) = K$, a constant that does not vary with time. This example uses K = 0.4 when the far-end signal is absent and $K = 0.004$ when the far-end signal is present.

Detecting the presence of a far-end signal must be done by DSP core and not the EFCOP. Therefore, this example does not address how to detect the presence of the far-end signal. Instead, the input file containing the far-end signal + the echo, *s(n)*, is created so that the far-end signal is absent for 300 samples and then present for 100 samples, and the example code automatically reduces the step size after 300 samples This input file is named `far.dat`.

Both the far-end and the near-end signals are gaussian random noise (generated with Matlab) with a mean of 0.0 and a variance of 1.0. The near-end signal input file is 400 samples long and the near-end signal is uncorrelated with the far-end signal. The near-end signal input file is named `near.dat`. The near-end signal is delayed by three samples and attenuated by $\sqrt{0.1} = 0.316$ to create the near-end echo. Then the near-end echo is added to the far-end signal to create the `far.dat` input file. Both input signals are scaled so that the magnitudes of all of the values are less than 1.

For example, the filter coefficients are set to zero before the processing starts. With no *a priori* knowledge about the echo, this is as good a starting point as any. The filter coefficients change and become non-zero when the processing begins and the coefficients are updated.

3.3 Example Code

A real-life LMS electrical echo canceller requires 48 coefficients to cancel 6 ms of echo with a sampling rate of 8KHz (8000 samples/sec $*$ 0.006 seconds = 48). A real echo canceller also requires thousands of samples to converge. This example is scaled down to simplify the explanations and shorten the running time and input files.

The echo canceller example code is divided into four sections:

- Declare the necessary constants.
- Initialize the EFCOP.
- Implement the coefficient update with an interrupt service routine.
- Initialize the filter inputs and coefficients.

3.3.1 Declaration of Constants

The first part of the code, shown in **Example 5**, defines the constants for the echo canceller and defines a constant to control the EFCOP. The input/output equate and interrupt equate files are included. The following memory address locations are initialized:

`START`	Start of the program.
`NEAR_SIG`	Near-end signal data *x(n)*.
`FAR_SIG`	Far-end signal plus the echo of the near-end signal data *s(n)*.

ECHO	Estimated echo of the near-end signal data *y(n)*.
FDBA_ADDRS FCBA_ADDRS	Memory address pointers for the filter data and coefficient buffers. These constants are written to FDBA and FCBA. The EFCOP shares the lowest 4K memory locations of X and Y memory with the DSP core for the data and coefficient buffers, respectively.

The constant initialization section also defines the following constants to control the EFCOP:

FCSR	Written to FCSR to control the main operation modes of the EFCOP. It configures the EFCOP in real FIR filter mode with adaptive filter mode enabled. FCSR also enables the data output buffer full interrupt. Finally, FCSR sets the EFCOP enable bit.
FIR_LEN	Defines the filter length. FIR_LEN is set to ten because there are ten filter coefficients, $h_n(i)$, *i=0...9* for this example. FIR_LEN – 1 is written to FCNT.
K1 and K2	Set to the step sizes that update the filer coefficients. K1 is used when the far-end signal is absent, and K2 is used when the far-end signal is present.
COUNT and COUNTK	Defines the number of data samples to process. For this example, there are 400 input data samples. FIR_LEN – 1, or 9, of these samples initialize the filter. Thus, 400 – 9 = 391 data samples are processed. The constant determines when the program is to change the step size. When there are COUNTK - 1, or 100 samples left to process, the program changes *K* from K1 to K2.

Example 5. Constant Definition Code

```
;*****************************************************************
       nolist
       INCLUDE "ioequ.asm"
       INCLUDE "intequ.asm"
       list
;*****************************************************************
;      CONSTANTS
;*****************************************************************
START          equ $100       ; Main program starting address
NEAR_SIG       equ $3000      ; Points to the Near-end data, x(n)
FAR_SIG        equ $2000      ; Points to the Far_end data, s(n)
ECHO           equ $1000      ; Points to the Echo data, y(n)
FDBA_ADDRS     equ 0          ; Data Start Address x:$0
FCBA_ADDRS     equ 0          ; Coeff Start Address y:$0
FCSR           equ $805       ; Enable EFCOP ADP FIR Mode 0 with DOBF interrupt
FIR_LEN        equ 10         ; Filter Length
K1             equ 0.4        ; Step size-Coef Update Constant-No Noise
K2             equ 0.004      ; Step size-Coef Update Constant-Noise
COUNT          equ 391        ; Data Count-390 total data samples
COUNTK         equ 101        ; Data Count to change K after 300 samples
```

3.3.2 EFCOP Initialization

The second part of the code, shown in **Example 6**, initializes the EFCOP for the echo canceller. The reset vector is set to the beginning of the program. The command to jump to the interrupt code is placed at the EFCOP output buffer full interrupt starting address. EFCOP interrupts are enabled at an interrupt priority level of 2 by setting the appropriate bits in the interrupt priority register peripherals (IPRP). The interrupt mask bits 0 and 1, bits 8 and 9 in the status register (SR), are cleared to permit interrupts at all priority levels. The following registers are initialized:

- Register b0 is initialized with COUNT to control the number of data samples to process, as described in Section 3.3.1, “Declaration of Constants.”
- Address registers r2 and r0 are initialized to the beginning of the near-end signal data, *x(n)* (NEAR_SIG), and the echo signal data, *y(n)* (ECHO).
- Address register r3 is initialized for the far-end signal plus the echo of the near-end signal data buffer, *s(n)* (FAR_SIG). This buffer is incremented by FIR_LEN - 1 because the first FIR_LEN - 1 data samples of *x(n)* are used to initialize the filter and the *x(n)* and *s(n)* data buffers should be aligned after the filter initialization.
- The y0 register is initialized with the first value for the step size, *K*.

The FIR_LEN, FDBA_ADDRS, and FCBA_ADDRS constants are written to the appropriate EFCOP registers, as described in Section 3.3.1, “Declaration of Constants.” FCSR is written to FCSR to enable the EFCOP. The first FIR_LEN samples of the near-end signal are written to the EFCOP data input register, FDIR: FIR_LEN - 1 samples to initialize the filter and one more sample to begin the first filter session. In the adaptive filter mode the EFCOP filters one sample of data and then waits until a value for $K_e(n)$ is written to the FKIR. Once a value is written to FKIR, the EFCOP performs a coefficient update session. When the output buffer is full, the EFCOP requests interrupt service from the core, and the interrupt code updates the filter coefficients. At this point, the program waits until the EFCOP data output interrupt enable bit is cleared. The interrupt code clears this bit when all data samples are processed. The program waits until the final filter update session is finished, and then the program is complete.

Example 6. EFCOP Initialization Code

```
;*****************************************************************
;*            Initialization
;*****************************************************************
       org     P:0
       jmp     START

       org     p:(I_FDOBF)                 ; EFCOP Output Buffer Full Interrupt
                                           ; Starting Address
       jsr     >kdo                        ; Jump to Interrupt Code

       org     p:START
       movep   #$c00,x:M_IPRP              ; Enable interrupts in IPR
       bclr    #8,SR                       ; Enable interrupts in SR
       bclr    #9,SR

       move    #0,b                        ; Init Counter
       move    #COUNT,b0
       move    #NEAR_SIG,r2                ; Init Pointer to Near-end Data, x(n)
       move    #ECHO,r0                    ; Init Pointer to Echo Data, y(n)
       move    #FAR_SIG+FIR_LEN-1,r3       ; Init Pointer to Far-end Data, s(n)
       move    #K1,y0                      ; Init K

       movep   #FIR_LEN-1,y:M_FCNT         ; Filter Length
       movep   #FDBA_ADDRS,y:M_FDBA        ; Data Start Address
       movep   #FCBA_ADDRS,y:M_FCBA        ; Coeff Start Address
       movep   #FCSR,y:M_FCSR              ; Enable EFCOP

       rep     #FIR_LEN                    ; Init Filter
       movep   x:(r2)+,y:M_FDIR
```

```
        jset    #11,y:M_FCSR,*              ; Wait till FDOIE is cleared
        jset    #3,y:M_FCSR,*               ; Wait until last update is complete
stop_label
        stop
```

3.3.3 Interrupt Code to Implement the Coefficient Update

The third part of the code, shown in **Example 7**, calculates Equation 5 and Equation 6 and then starts the filter coefficient update session by writing the step parameter to FKIR. When the program reaches this point, the EFCOP has just completed a FIR filter session and placed the output into FDOR, causing a EFCOP output buffer full interrupt request. Updating of the coefficients proceeds in the following steps:

1. The interrupt code moves the filter output, *y(n)*, from FDOR to the `ECHO` data buffer and increments the `ECHO` data buffer pointer.
2. The `ECHO` data and the current `FAR_SIG` data, *s(n)*, are moved to data registers, incrementing the `FAR_SIG` data buffer pointer.
3. The current error signal, *e(n)* is calculated as in Equation 5.
4. The step size, located in register `y0`, is multiplied by the error signal to calculate the coefficient update step parameter as in Equation 6.
5. The step parameter is loaded into FKIR.
6. The EFCOP performs the coefficient update session, as in Equation 7, and replaces the filter coefficients with the updated coefficients.
7. The next input sample is written from the `NEAR_SIG` data buffer to the input register, FDIR, incrementing the `NEAR_SIG` data buffer pointer.
8. The program determines if step size needs to be changed by comparing the counter in register `b0` to the value from `COUNTK`. If these values are equal, the step size is changed by writing `K2` to the `y0` register. Otherwise the step size is not changed.
9. The counter is decremented and as long as the counter is not equal to zero the interrupt exits.

The process repeats when the EFCOP places the next output into FDOR. When the counter is equal to zero, the EFCOP output buffer full interrupt is disabled and the processing stops.

Example 7. Interrupt Code

```
;*********************************************************************
;*            Interrupt Code
;*********************************************************************
kdo
        movep   y:M_FDOR,x:(r0)+            ; Move y(n) to memory buffer
        move    y:M_FDOR,x1                 ; Move y(n) to x1
        move    x:(r3)+,a                   ; Move s(n) to a
        sub     x1,a                        ; a = e(n) = s(n) - y(n)
        move    a,x0                        ; x0 = e(n)
        mpy     x0,y0,a                     ; a = Ke = K*e(n)
        movep   a,y:M_FKIR                  ; Move Ke to FKIR
        movep   x:(r2)+,y:M_FDIR            ; Move x(n) to FDIR
        clr     a                           ; Check if K needs to be changed
        move    #COUNTK,a0
        cmp     a,b
        jne     samek                       ; Change K to K2 if
```

```
        move    #K2,y0                  ; there are 100 samples left
samek   dec     b                       ; Decrement the counter
        jne     cont                    ; Jump to cont if counter is not zero
        nop
        bclr    #11,y:M_FCSR            ; Disable interrupt
cont
        rti
```

3.3.4 Initialization of Coefficients and Input

The final part of the code, shown in **Example 8**, initializes the coefficients and inputs for the echo canceller. The coefficient values are initialized to zero as described in Section 3.2, “Echo Canceller Design.” The memory address pointer for the coefficients, `FCBA_ADDRS`, is defined in Section 3.3.1, “Declaration of Constants.” The EFCOP shares the lowest 4K memory locations of Y memory with the DSP core for the coefficient buffers. Thus, the coefficients are stored in Y memory. The filter taps do not need to be initialized for this example because processing state initialization mode is enabled in the `FCSR` constant.

The last lines of the code specify the input data. The input data includes the near-end signal data *x(n)* (`NEAR_SIG`) and the far-end signal plus the echo of the near-end signal data *s(n)* (`FAR_SIG`). The input files that contain these signals, `near.dat` and `far.dat`, are described in Section 3.2, “Echo Canceller Design.” The memory address pointers for the input data, `FAR_SIG` and `NEAR_SIG`, are defined in Section 3.3.1, “Declaration of Constants.” The `far.dat` and `near.dat` files are included at these memory locations.

Example 8. Coefficient and Input Code

```
        org             y:FCBA_ADDRS
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000
        dc              $000000

        org             x:FAR_SIG

        INCLUDE         "far.dat"

        org             x:NEAR_SIG

        INCLUDE         "near.dat"
```

3.4 Echo Canceller Results

This section describes the results for the echo canceller example, presenting the filter coefficients and the received signal, *e(n)*. If the filter is working properly, the filter coefficients show the delay and the attenuation of the echo. **Table 8** shows the filter coefficients after 100, 200, 300, and 400 samples. *Notice that the coefficients are stored in reverse order* so that the coefficient with the largest index is stored first and the coefficient with the smallest index is stored last as they are stored in the DSP memory.

Recall that the far-end signal is absent for the first 300 samples. During this time the filter is adapting only to the near-end echo. The third coefficient from the bottom becomes more dominant as the number of samples increases. This signifies that the near-end echo is delayed three samples as described in Section 3.2, “Echo Canceller Design.” The magnitude of the third coefficient approaches the attenuation factor of the near-end echo $\sqrt{0.1} = 0.316$ as the number of samples increases. Thus, the adaptive filter coefficients show the delay and attenuation of the echo properly and the filter is working as expected.

The filter coefficients for $n = 400$ show the effect of the far-end signal on the adaptive filter. Recall that the far-end signal is present for the last 100 samples and that the adaptive filter interprets the far-end signal as noise. Thus, the filter coefficients degrade when $n = 400$. The third coefficient is not as dominant as it is when $n = 300$. However, the step size is reduced for the last 100 samples. Thus, the coefficients are not significantly affected and the adaptive filter still does an acceptable job of cancelling the near-end echo, as indicated by the received signal, *e(n)*.

Table 8. Filter Coefficients

	n = 100	n = 200	n = 300	n = 400
h(9) =	–0.0052	–0.0003	–0.0000	–0.0047
h(8) =	–0.0062	–0.0010	0.0000	–0.0008
h(7) =	0.0014	–0.0005	0.0000	0.0027
h(6) =	0.0020	0.0005	–0.0000	0.0015
h(5) =	0.0045	0.0008	0.0000	0.0029
h(4) =	0.0049	0.0004	0.0000	0.0002
h(3) =	–0.0068	–0.0007	0.0000	0.0035
h(2) =	0.2999	0.3150	0.3162	0.3197
h(1) =	–0.0034	–0.0003	0.0000	0.0046
h(0) =	0.0026	0.0008	0.0000	–0.0049

Table 9 shows the received signal, *e(n)*, the far-end signal, and the error between these two signals for the last 20 samples. The received signal is calculated in the interrupt code. The far-end signal is obtained from Matlab before the near-end echo is added to create the far-end plus the echo of the near-end signal, *s(n)*. The error is the far-end signal minus the received signal. The table shows that the error between the two signals is very small. Thus, the adaptive filter works properly and generates an acceptable estimate of the echo, even when the far-end signal is present.

Table 9. Received/Far-End Signal Error

n	e(n)	Far-End (n)	Error
381	–0.2092	–0.2063	0.0029
382	–0.3894	–0.3895	–0.0001
383	0.3776	0.3781	0.0005
384	0.1931	0.1964	0.0033
385	0.3200	0.3245	0.0045
386	0.1265	0.1301	0.0036
387	–0.2400	–0.2394	0.0006
388	0.1939	0.1953	0.0014
389	–0.0094	–0.0078	0.0016
390	–0.5286	–0.5289	–0.0003

Table 9. Received/Far-End Signal Error (Continued)

n	e(n)	Far-End (n)	Error
391	–0.1407	–0.1391	0.0016
392	0.4876	0.4880	0.004
393	–0.2476	–0.2458	0.0018
394	–0.3945	–0.3958	–0.0013
395	0.0678	0.0714	0.0036
396	0.2297	0.2308	0.0011
397	–0.1418	–0.1407	0.0011
398	0.1386	0.1342	–0.0044
399	–0.5797	–0.5796	0.0001
400	0.0654	0.0665	0.0011

4 Correlation Notes

This section gives a few notes on how to implement correlations using the EFCOP. The general correlation equation for real valued signals is:

Equation 8

$$r_{ab}(k) = \sum_{n} a(n)b(k+n)$$

where $r_{ab}(k)$ is the cross-correlation between signals $a(n)$ and $b(n)$. If $a(n) = b(n)$, then Equation 8 is the auto-correlation.

Equation 8 is similar to the general convolution equation implemented by the EFCOP:

Equation 9

$$y(k) = \sum_{n} h(n)x(k-n) = \sum_{n} x(n)h(k-n)$$

where $y(n)$ is the result of filtering the signal $x(n)$ with the filter coefficients $h(n)$.

Equation 8 converts into the second part of**Equation 9** if the filter input signal, $x(n)$, is replaced with the $a(n)$ signal and the filter coefficients are replaced with the $b(n)$ signal values in reverse order. However, the EFCOP filter coefficients are stored in memory in reverse order. Thus, implementing a cross-correlation using the EFCOP is as simple as using the first signal as the input signal and the second signal as the filter coefficients, making sure that the second signal is stored in memory in the proper non-reversed order.

5 Programmer's Reference

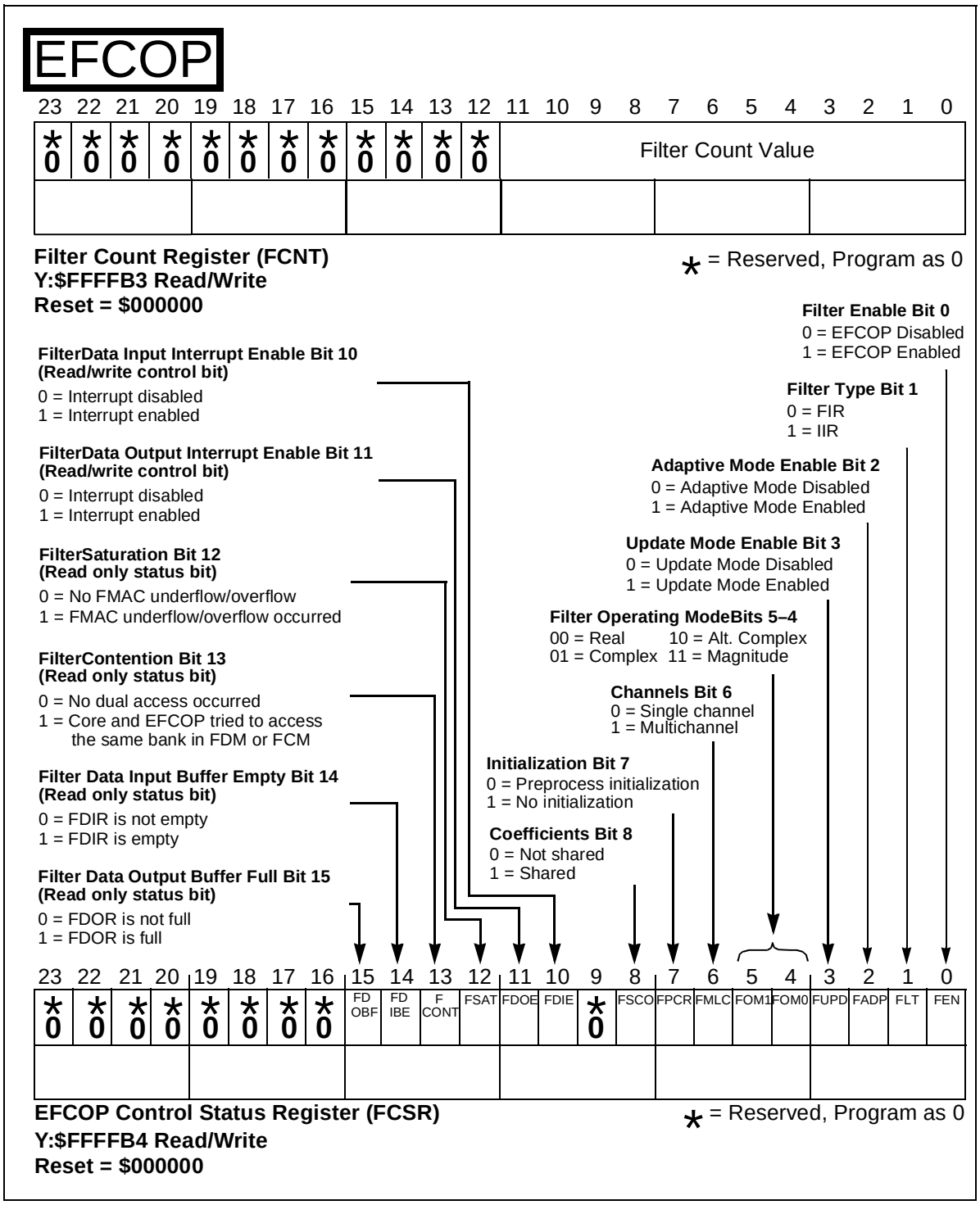

Figure 8. EFCOP Counter and Control Status Registers (FCNT and FCSR)

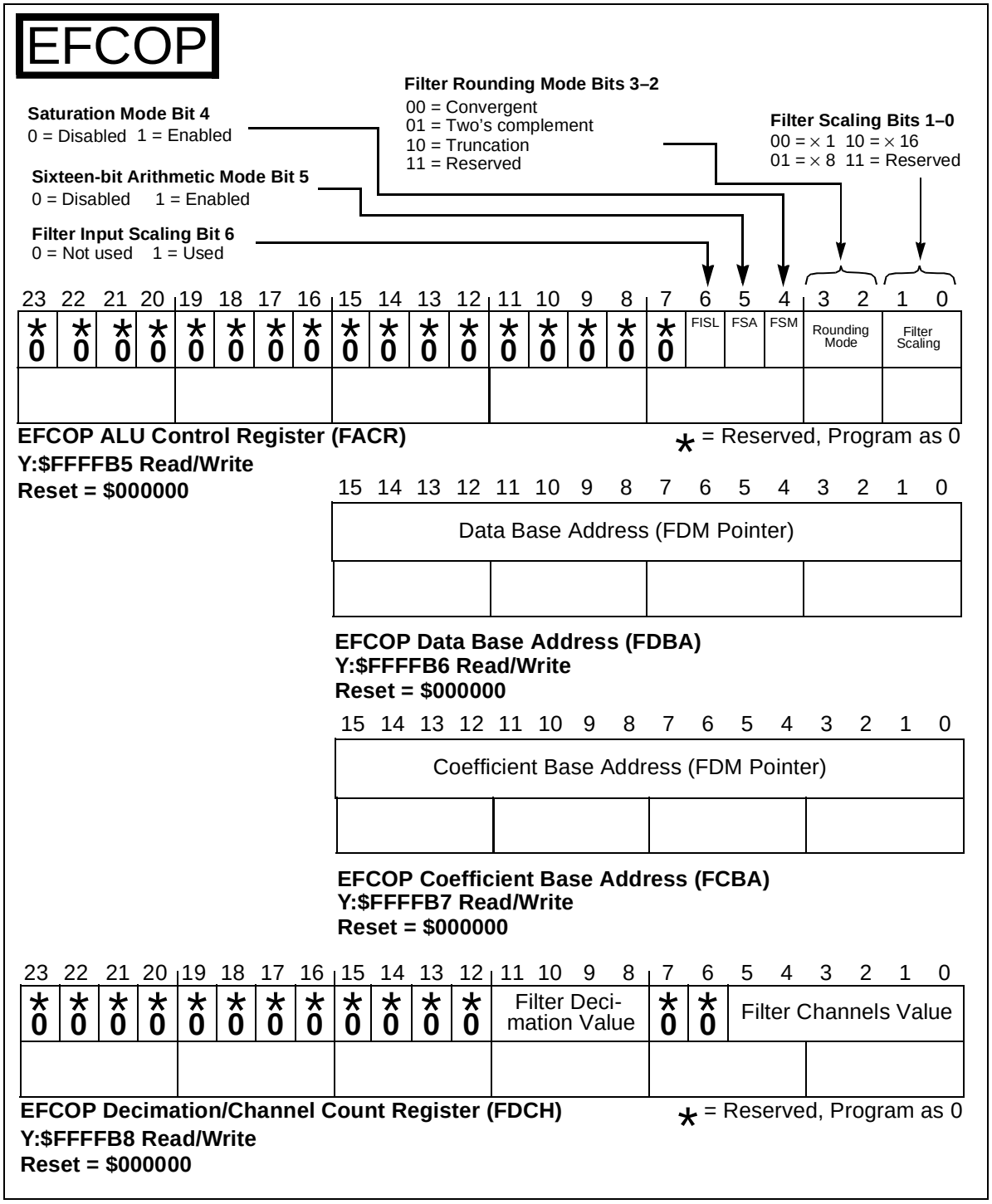

Figure 9. EFCOP FACR, FDBA, FCBA, and FDCH Registers

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.: APR39
Rev. 1
8/2005