Chapter 9
Enhanced Filter Coprocessor (EFCOP)

The MSC8101 EFCOP module is a general-purpose, fully programmable filter with 32-bit resolution. It has optimized modes of operation to perform real and complex finite impulse response (FIR) filtering, infinite impulse response (IIR) filtering, adaptive FIR filtering, and multichannel FIR filtering. EFCOP filter operations complete concurrently with SC140 core operations, with minimal CPU intervention. For optimal performance, the EFCOP has one dedicated filter multiplier accumulator (FMAC) unit. As a result, the SC140 core/EFCOP combination offers multiple multiply-accumulate (MAC) filtering capabilities.

Its dedicated modes make the EFCOP a very flexible filtering coprocessor with operations optimized for cellular base station applications. In a transceiver base station, the EFCOP performs complex matched filtering to maximize the signal-to-noise ratio (SNR) in an equalizer. The coefficients of the matched filter can be determined by a cross-correlation filtering process between a received training sequence and a known reference sequence. In a transcoder base station 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 chapter discusses how to program the EFCOP to operate in different modes and how to use the different methods for transferring data into and out of the EFCOP. Code examples demonstrate how the EFCOP can be programmed to complete a variety of tasks. The focus is on programming the EFCOP internally from the SC140 core. The EFCOP can also be programmed from an external device on the PowerPC system bus, and all the programming issues discussed here still apply.

9.1 Programming the Control Registers

The EFCOP is programmed by writing the desired settings to the memory-mapped control registers. **Table 9-1** summarizes the EFCOP control registers.

Table 9-1. EFCOP Control Registers

Register Name	Description
Filter Count Register (FCNT)	A 16-bit read/write register that specifies the number of filter taps. The count stored in the FCNT register is used by the EFCOP address generation logic to generate correct addressing to the filter data memory (FDM) and filter coefficient memory (FCM).
EFCOP Control Register (FCTL)	A 16-bit read/write register used by the SC140 core to program the EFCOP.
EFCOP ALU Control Register (FACR)	A 16-bit read/write register used by the SC140 core to program the EFCOP data ALU operating modes.
EFCOP Data Base Address Register (FDBA)	A 16-bit read/write register used by the SC140 core to indicate the EFCOP data buffer base start address pointer in FDM RAM.
EFCOP Coefficient Base Address Register (FCBA)	A 16-bit read/write register by which the SC140 core indicates the EFCOP coefficient buffer base start address pointer in FCM RAM.
EFCOP Decimation/Channel Count Register (FDCH)	A 16-bit read/write register that sets the number of channels in multichannel mode and the filter decimation ratio. The EFCOP address generation logic uses this information to supply the correct addressing to the FDM and FCM.
EFCOP Status Register (FSTR)	A 16-bit read-only register used by the SC140 core to examine the status of the EFCOP module.
Filter Data Input Register (FDIR)	An 8-word deep 32-bit wide FIFO used for core-to-EFCOP and DMA data transfers. Data from the FDIR is transferred to the FDM for filter processing.
Filter Data Output Register (FDOR)	An 8-word deep 32-bit wide FIFO used for EFCOP-to-core and DMA data transfers. Data is transferred to FDOR after processing of all filter taps is completed for a specific set of input samples.
Filter K-Constant Input Register (FKIR)	A 32-bit read/write register for core-to-EFCOP constant transfers.

The EFCOP operates in many different modes based on the settings of these control registers. However, the EFCOP performs only two basic modes of processing, FIR filter type and IIR filter type processing. Various operating modes are available for each filter type. The following sections discuss the two basic operating modes.

9.2 Specifying the Operating Modes for the FIR Filter Type

This section discusses the various operating modes that are available for the FIR filter type. The FIR filter type is selected by clearing the FCTL[14]:FLT bit, and it performs the processing shown in **Figure 9-1** using the following equation:

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

For each sample to be filtered, the EFCOP completes the following steps:

1. Take an input, *x(n)*, from the FDIR.
2. Save the input while shifting the previous inputs down in the FDM. The shifting down of the previous inputs is accomplished by incrementing the value in the FDBA register by one.
3. Multiply each input in the FDM by the corresponding coefficient, B_i, stored in the FCM.
4. Accumulate the multiplication results.
5. Place accumulation result, *w(n)*, into the FDOR.

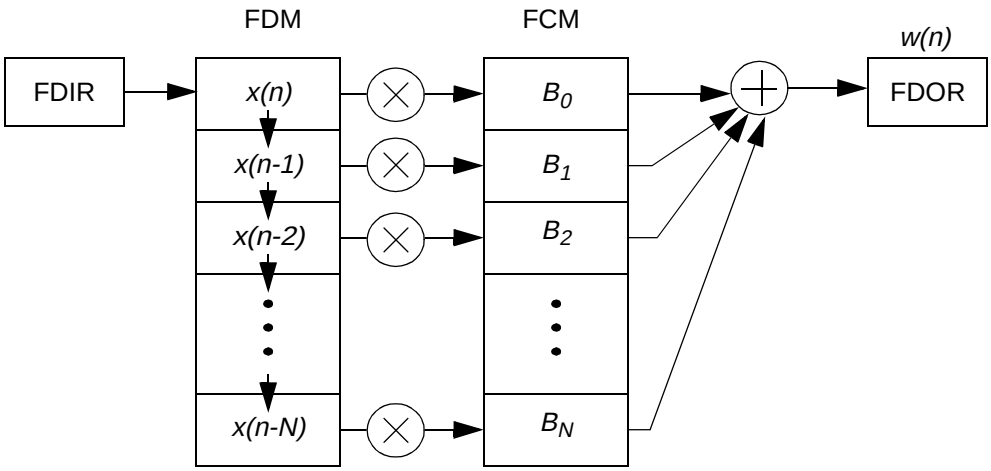

Figure 9-1. FIR Filter Block Diagram

Four operating modes are available for the FIR filter type: real, complex, alternating complex, and magnitude mode.

9.2.1 Real Mode

Real mode performs FIR type filtering with real data and is selected by clearing both FCTL[10–11]:FOM bits. For each sample (the real input) written to the FDIR, one sample (the real output) is read from the FDOR. In Real mode, the number written to the FCNT register should be one minus the number of filter coefficients.

Two other options are available with the real FIR filter type: Adaptive and Multichannel modes. These modes can be used singly or together.

9.2.1.1 Adaptive Mode

The Adaptive mode provides a way to update the coefficients based on filter input, *x(n)*, using the following equation:

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

where $h_n(i)$ is the *i*th coefficient at time *n*. The coefficients are updated when the FCTL[12]:FUPD bit is set. When this bit is set, the EFCOP checks to see if a value has been written to the FKIR. If no value is written, the EFCOP halts processing until a value is written to the FKIR. When a value is written to the FKIR, the EFCOP updates all the coefficients based on the preceding equation using the value in the FKIR for $K_e(n)$. The EFCOP automatically clears the FCTL[12]:FUPD bit when the coefficient update completes.

If the coefficients are to be updated after every input sample, Adaptive mode is enabled by setting the FCTL[13]:FADP bit. In Adaptive mode, the EFCOP automatically sets the FCTL[12]:FUPD bit after each input sample is processed. This allows for continuous processing using interrupts that includes a filter session and a coefficient update session with minimal core intervention.

Additionally, the EFCOP can generate an interrupt request to the SC140 core when the coefficient update is complete. This interrupt is controlled by the FCTL[6]:FUDIE bit. If this bit is clear, the interrupt is disabled. If this bit is set, the interrupt is enabled. When the interrupt is enabled, it is triggered by the FSTR[13]:FUDN bit. The EFCOP sets FSTR[13]:FUDN when the coefficient update session completes. Thus, when both FSTR[13]:FUDN and FCTL[6]:FUDIE are set, the EFCOP requests the coefficient update complete interrupt from the SC140 core.

9.2.1.2 Multichannel Mode

In Multichannel mode, several channels of data are processed concurrently. This mode is selected by setting the FCTL[9]:FMLC bit. The number of channels to process is one plus the number in the FDCH[10–15]:FCHL bits. The number of channels can be programmed

to be from 1 to 64. For each time period, the EFCOP expects to receive the samples for each channel sequentially. This process repeats for consecutive time periods.

Filtering is performed with the same filter or different filters for each channel using the FACR[8]:FSCO bit. If this bit is set, the same set of coefficients is used for all channels. If FACR[8]:FSCO is clear, the coefficients for each channel are stored sequentially in memory with the beginning address of each coefficient buffer at the next 2^k address (where $2^{k-1} \leq$ filter length $\leq 2^k$). If the filter length is less than 2^k there is a space between the sequential buffers of 2^k minus the filter length.

9.2.2 Complex Mode

Complex mode performs FIR type filtering with complex data based on the following equations, in which *Re* is the real part and *Im* is the imaginary part:

$$Re(F(n)) = \sum_{i=0}^{N-1} Re(H(i)) \cdot Re(D(n-i)) - Im(H(i)) \cdot Im(D(n-i))$$

$$Im(F(n)) = \sum_{i=0}^{N-1} Re(H(i)) \cdot Im(D(n-i)) + Im(H(i)) \cdot Re(D(n-i))$$

where *H(n)* is the coefficients, *D(n)* is the input data, and *F(n)* is the output data at time *n*. For every two samples written to the FDIR (the real part followed by the imaginary part of the input), two samples (the real part followed by the imaginary part of the output) are read from the FDOR.

Complex mode is selected by writing 01 to the FCTL[10–11]:FOM bits. When Complex mode is used, the number written to the FCNT register should be twice the number of filter coefficients minus one, (2*filter length) –1. Also, the coefficients should be stored in the FCM with the real part of the coefficient in the memory location preceding the location holding the imaginary part.

9.2.3 Alternating Complex Mode

Alternating Complex mode performs FIR type filtering with complex data providing alternating real and complex results based on the following equations:

$$Re(F(n|_{even})) = \sum_{i=0}^{N-1} Re(H(i)) \cdot Re(D(n-i)) - Im(H(i)) \cdot Im(D(n-i))$$

$$Im(F(n|_{odd})) = \sum_{i=0}^{N-1} Re(H(i)) \cdot Im(D(n-i)) + Im(H(i)) \cdot Re(D(n-i))$$

where *H(n)* is the coefficients, *D(n)* is the input data, and *F(n)* is the output data at time *n*. For every two samples (the real part followed by the imaginary part of the input) written to the FDIR, one sample (alternating between the real part and the imaginary part of the output) is read from the FDOR.

Alternating Complex mode is selected by writing 10 to FCTL[10–11]:FOM bits. When Alternating Complex mode is used, the number written to the FCNT register should be twice the number of filter coefficients minus one, (2*filter length) –1. Also, the coefficients should be stored in the FCM with the real part of the coefficient in the memory location preceding the location holding the imaginary part.

9.2.4 Magnitude Mode

Magnitude mode calculates the magnitude of an input signal using the following equation:

$$F(n) = \sum_{i=0}^{N-1} D(n-i)^2$$

where *D(n)* is the input data and *F(n)* is the output data at time *n*. For each sample (the real input) written to the FDIR, one sample (the real magnitude of the input signal) is read from the FDOR. Magnitude mode is selected by setting both FCTL[10–11]:FOM bits. In Magnitude mode, the number written to the FCNT register should be the number of data samples to compute the magnitude of minus one, and the value in the FCBA register is ignored.

9.2.5 Data and Coefficient Initialization

Before the first sample can be processed, the filter must be initialized, meaning that the input samples for times before $n = 0$ (assuming that time starts at 0) must be loaded into the FDM. The number of samples needed to initialize the filter is the number of filter coefficients.

The Data Initialization mode is selected via the FCTL[8]:FPRC bit:

- If FCTL[8]:FPRC is set, initialization is disabled and the EFCOP assumes that the SC140 core wrote the initial input values to the FDM before the EFCOP was enabled. Thus, the first value written to FDIR is the first sample to be filtered.
- If FCTL[8]:FPRC is clear, initialization mode is enabled and the EFCOP initializes the FDM by receiving the number of coefficient data samples through the FDIR. These samples are loaded into the FDM buffer and after the last value is loaded the EFCOP begins processing the first result.

The EFCOP also allows the coefficient buffer to be initialized before processing begins. The Coefficient Initialization mode is selected via FCTL[7]:FCIM:

- If FCTL[7]:FCIM is clear, Coefficient Initialization mode is disabled and processing completes as described earlier, depending on how the FCTL[8]:FPRC bit is set.
- If FCTL[7]:FCIM is set, the coefficients are initialized by a coefficient update session with the original coefficients equal to zero, as in the following equation:

$$h(i) = Kx(i)$$

  The data buffer, $x(i)$, should be initialized first, either by the SC140 core or the EFCOP with data initialization. Then, K should be written to FKIR, and the EFCOP initializes the coefficient buffer accordingly.

  If data and coefficient initialization are both enabled, data initialization completes first but the EFCOP does not begin processing the first result. After FKIR is written, the EFCOP initializes the coefficients and then another input sample must be written to FDIR to begin processing.

Coefficient initialization is usually used with Adaptive mode or can be used to clear the coefficients by writing zero to FKIR.

9.2.6 Decimation

Decimation is another option that can be used with any of the four FIR filter type modes. However, decimation cannot be used in conjunction with the Adaptive or Multichannel modes. Decimation, also known as “downsampling,” decreases the sampling rate. The decimation ratio defines the number of input samples per output sample. The decimation ratio is one plus the number in the FDCH[4–7]:FDCM bits. The decimation ratio can be programmed from 1 to 16.

For Real and Magnitude modes, the decimation ratio number of the sample must be written to the FDIR before an output sample can be read from the FDOR. For Complex

mode, two times the decimation ratio number of samples must be written to the FDIR (one for the real part and one for the imaginary part of the input) before two output samples (one for the real part and one for the imaginary part of the output) can be read from the FDOR. For Alternating Complex mode, two times the decimation ratio number of samples must be written to the FDIR (one for the real part and one for the imaginary part of the input) before one output sample (alternating between the real part and the imaginary part of the output) can be read from the FDOR.

9.3 Specifying the Operating Modes for the IIR Filter Type

This section discusses the various operating modes that are available for the IIR filter type. To process a complete IIR filter, a FIR filter type session followed by an IIR filter type session is needed. The IIR filter type is selected by setting the FCTL[14]:FLT bit, and it performs the processing shown in **Figure 9-2** using the following equation:.

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

For each sample input to the FDIR, the EFCOP completes the following steps:

1. Multiply each previous output value in the FDM by the corresponding coefficient, *A*, stored in the FCM.
2. Accumulate the multiplication results.
3. Add the input, *w(n)*, from the FDIR.
4. Place the accumulation result, *y(n)*, in the FDOR.
5. Save the output while shifting the previous outputs down in the FDM. The shifting down of the previous outputs is accomplished by incrementing the value in the FDBA register by one.

Only the Real and the Multichannel Operation modes are available for the IIR filter type. Thus, the FCTL[10–11]:FOM bits are ignored when the IIR filter type is used. The Real Operation mode performs IIR type filtering with real data. For each sample (the real input) written to the FDIR, one sample (the real output) is read from the FDOR. In Real mode, the number written to the FCNT register should be the number of filter coefficients minus one.

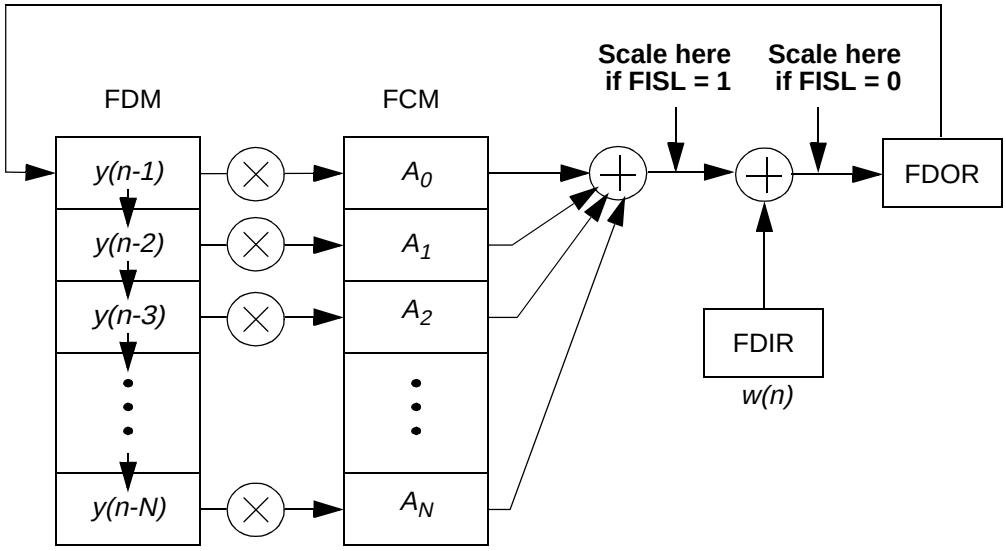

Figure 9-2. IIR Filter Block Diagram

Multichannel mode for the IIR filter type works exactly the same way as for FIR filter type as explained in **Section 9.2.1.2**, *Multichannel Mode*, on page 9-4. Decimation and Adaptive modes are not available with the IIR filter type.

Initialization is always disabled with the IIR filter type, and the FCTL[8]:FPRC bit is ignored. Thus, the SC140 core must write the initial input values to the FDM before the EFCOP is enabled. The first value written to the FDIR is always the first sample to be filtered.

9.4 Specifying the ALU Modes

Two modes that affect the arithmetic operation of the EFCOP are rounding and input scaling. These ALU modes are independent of the filter type.

9.4.1 Rounding

Rounding mode is selected via the FACR[12–13]:FRM bits. These bits select the type of rounding performed by the EFCOP data ALU (DALU) during arithmetic operations. The EFCOP DALU performs the following types of rounding:

- Convergent rounding (FACR[12–13]:FRM = 00)
- Twos complement rounding (FACR[12–13]:FRM = 01)
- No rounding, that is, truncation (FACR[12–13]:FRM = 10)

9.4.2 Input Scaling

The Input Scaling mode affects only IIR filtering and the coefficient update session of adaptive FIR filtering. The FACR[9]:FISL and FACR[14–15]:FSCL bits, determine how the outputs are scaled. The result can be scaled up by the following values:

- One, that is, no scaling (FACR[14–15]:FSCL = 00)
- Eight (FACR[14–15]:FSCL = 01)
- Sixteen (FACR[14–15]:FSCL = 10)

For IIR type filtering, FACR[9]:FISL determines whether the IIR input is scaled. When FACR[9]:FISL is set, the IIR feedback terms are scaled, but the IIR input, $w(n)$ is not scaled. This case is represented by the following equation:

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

where S is the scaling factor. When FACR[9]:FISL is clear, the EFCOP ALU scales both the IIR feedback terms and the IIR input. This case is represented by the following equation:

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

Figure 9-2 also shows where the scaling occurs, depending on the value of FISL.

For coefficient update sessions, FACR[9]:FISL determines whether the original coefficients are scaled. When FACR[9]:FISL is set, the EFCOP ALU scales only the input/constant term and not the original coefficients. This is represented by the following equation:

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

When FACR[9]:FISL is clear, both the input/constant term and the original coefficients are scaled. This is represented by the following equation:.

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

9.5 Transferring Data In and Out of the EFCOP

When the EFCOP is programmed and enabled, it waits until input data is written to the Filter Data Input Register (FDIR). The FDIR is an 8-element deep FIFO, so up to eight 32-bit wide data samples can be written into FDIR at the same address. When the EFCOP finishes processing the input data from the FDIR, it sends the results to the FDOR. The FDOR is an 8-element deep read-only FIFO, so up to eight 32-bit wide data samples can be read from FDOR at the same address.

There are three methods for transferring data to or from the EFCOP data registers:

- *Polling*. The easiest method, but it demands a large amount of the core processing power. The SC140 core cannot be involved in other processing activities while it is polling the input and output buffer bits.
- *Interrupts.* This method requires more code, but the core can process other routines while the EFCOP is computing.
- *DMA.* This method requires even less core intervention and the set-up code is minimal, but the DMA channels must be available.

The following sections describe each transfer method.

9.5.1 Polling

The EFCOP Status Register (FSTR) contains bits that notify when data is ready for transfer to or from the EFCOP. These bits determine when to interact with the EFCOP. For proper operation, the SC140 core must write to the FDIR only when it is empty or not full and read from the FDOR only when it is full or not empty.

FSTR[11]:FIBNF and FSTR[12]:FDIBE determine when to write to the FDIR:

- FSTR[11]:FIBNF is set when the FDIR is not full (that is, at least one of the locations is empty). Thus, when FSTR[11]:FIBNF is set, the SC140 core can write only one sample of data to the FDIR.
- FSTR[12]:FDIBE is set when the FDIR is empty (that is, all eight of the locations are empty). Thus, when FSTR[12]:FDIBE is set, the core can write up to eight samples of data to FDIR. This bit is set immediately after the EFCOP is enabled by setting FCTL[15]:FEN.

FSTR[9]:FOBNE and FSTR[10]:FDOBF determine when to read from the FDOR:

- FSTR[9]:FOBNE is set when the FDOR is not empty (that is, at least one of the locations is full). Thus, when this bit is set, the SC140 core can read only one sample of data from the FDOR.

- FSTR[10]:FDOBF is set when the FDOR is full (that is, all eight of the locations are full). Thus, when this bit is set, the SC140 core can read up to eight samples of data from the FDOR.

For an example of EFCOP programming with polling, see **Section 9.6.1**, *Complex FIR Filter with Polling*, on page 9-14.

9.5.2 Interrupts

The EFCOP provides five interrupts. **Table 9-2** describes these interrupts, including how they are enabled and triggered and the location of the vector address.

Table 9-2. EFCOP Interrupts

Signal	Description	Enabled by Setting	Triggered When	Vector Address (Offset from VBA)
$\overline{\text{IRQ0}}$	Data Input FIFO Not Full	FCTL[4]:FINFIE	FSTR[11]:FIBNF and FCTL[4]:FINFIE are set simultaneously	0x800
$\overline{\text{IRQ1}}$	Data Input FIFO Empty	FCTL[5]:FIEIE	FSTR[12]:FDIBE and FCTL[5]:FIEIE are set simultaneously	0x840
$\overline{\text{IRQ2}}$	Data Output FIFO Full	FCTL[3]:FOFIE	FSTR[10]:FDOBF and FCTL[3]:FOFIE are set simultaneously	0x880
$\overline{\text{IRQ3}}$	Data Output FIFO Not Empty	FCTL[2]:FONEIE	FSTR[9]:FOBNE and FCTL[2]:FONEIE are set simultaneously	0x8C0
$\overline{\text{IRQ4}}$	Coefficient Update Done	FCTL[6]:FUDIE	FSTR[13]:FUDN and FCTL[6]:FUDIE are set simultaneously	0x900

In general, configuring interrupts requires two steps:

1. Set up the interrupt routine by placing the code to be run during the interrupt at the appropriate interrupt vector address.

   The location of the vector address is shown in **Table 9-2** and depends on the setting of the Vector Base Address (VBA) Register. The memory allocation for each interrupt is 64 bytes. To extend the interrupt code size further, service routines can be used.

2. Enable the interrupts by setting bits in various control registers:

   a. To enable the desired interrupts, set the appropriate bits in the FCTL.

   b. To determine the interrupt priority level of the core, set the interrupt mask bits (I0–2) bits of the Status Register (SR).

   c. To determine the priority level for each enabled interrupt, set the PIC Edge/Level-Triggered Interrupt Priority Registers A and B (ELIRA and ELIRB) Interrupt Priority Level (PILxx) bits.

d. Clear the Interrupt Trigger Mode (PEDxx) bits of the ELIRx registers because all peripheral interrupts are level triggered.

e. Enable the interrupts by issuing an **ei** (enable interrupts) instruction.

For an example of EFCOP programming with interrupts, see **Section 9.6.2**, *Adaptive Filter With Interrupts*, on page 9-16.

9.5.3 DMA

The Direct Memory Access (DMA) controller is an on-chip device that permits data transfers to and from the EFCOP without intervention of the SC140 core. The DMA can move data to the EFCOP input register and from the EFCOP output register. The DMA allows dual access and flyby transactions to the EFCOP. Flyby transactions occur directly between the EFCOP and internal SRAM, and they do not require access to the DMA FIFO.

The DMA request source is controlled by the requestor number bits (RQNUM, bits 19-23) in the DMA Channel Configuration Register (DCHCRx). If these bits are equal to 00010, the DMA is triggered by an EFCOP read request (the FDOR needs to be read because it is full or not empty). If these bits are equal to 00011, the DMA is triggered by an EFCOP write request (when the FDIR needs to be written because it is empty or not full).

On the EFCOP side, DMA transfers are controlled by the FCTL[1]:FDIM and FCTL[0]:FDOM bits.

- FCTL[1]:FDIM controls the data input mode:
  - When FCTL[1]:FDIM is clear, the EFCOP issues a transfer request from the DMA when the input buffer is not full (when FSTR[FIBNF] is set). Use this mode when the DMA is programmed to transfer single 32-bit long samples to the FDIR (DMA BD_ATTR field TSZ bits equal to 011). Burst or single transfers of more than 32-bits cause an error when the FCTL[1]:FDIM is clear.
  - When FCTL[1]:FDIM is set, the EFCOP issues a transfer request from the DMA when the input buffer is empty (when FSTR[FDIBE] is set). Use this mode when the DMA is programmed to transfer more than one 32-bit long sample to the FDIR. For example, to use the PowerPC local bus most efficiently, 64-bits can be transferred to the FDIR with this mode and the DMA TSZ bits equal to 000. Also, the complete FDIR buffer can be written using DMA burst mode (TSZ equal to 100) with FCTL[1]:FDIM set.
- FCTL[0]:FDOM controls the data output mode:
  - When FCTL[0]:FDOM is clear, the EFCOP issues a transfer request from the DMA when the output buffer is not empty (when FSTR[9]:FOBNE is set). Use

this mode when the DMA is programmed to transfer single 32-bit long samples from the FDOR (DMA BD_ATTR field TSZ bits equal to 011). Burst or single transfers of more than 32-bits cause an error when FCTL[0]:FDOM is clear.

— When FCTL[0]:FDOM is set, the EFCOP issues transfer request from the DMA when the output buffer is full (when FSTR[9]:FOBNE is set). Use this mode when the DMA is programmed to transfer more than one 32-bit long sample from the FDOR. For example, to use the PowerPC local bus most efficiently, 64-bits can be transferred from the FDOR with this mode and the DMA TSZ bits equal to 000. Also, the complete FDOR buffer can be read using DMA burst mode (TSZ equal to 100) and FCTL[0]:FDOM set.

For an example of EFCOP programming with DMA, see **Section 9.6.3**, *Real IIR Filter with DMA*, on page 9-19.

9.6 Programming Examples

The code examples in this section illustrate how to program the EFCOP to complete three basic EFCOP operations: FIR filtering, IIR filtering, and adaptive filtering. These examples also illustrate the three EFCOP data transfer methods. The examples use equate labels for the location of the EFCOP registers and assume that these equates are declared prior to the example code. These labels include the register name preceded with “M_”.

9.6.1 Complex FIR Filter with Polling

The code in **Example 9-1** exhibits a simple way to program the EFCOP using polling to transfer complex data in and out of the EFCOP data registers. In Complex mode, each filter operation begins with a write of two data samples to the FDIR: one for the real part followed by one for the imaginary part of the input data. Two 32-bit data samples are written to the FDIR register using two **move.l** instructions. More efficiently, two 32-bit data samples (the real and the imaginary part) can be written to the FDIR with one **move.2l** instruction. This code programs the EFCOP to implement a complex FIR filter based on the equation from **Section 9.2.2**, as follows:

1. The address register pointers are initialized for the filter input and output data (`INPUT` and `OUTPUT`) and for the EFCOP input and output registers.
2. EFCOP control parameters are written to the appropriate memory-mapped control registers as follows:
   a. The FDBA and FCBA registers are written with 0x400. To determine where the FDM and FCM are located in memory based on these register settings, recall that these registers contain the offset of the FDM and FCM from their base addresses (0x70000 for the FDM and 0x78000 for the FCM) in four-byte

resolution and that memory is addressed in one byte resolution. Therefore, the FDM and FCM are located at 0x400 multiplied by four plus the base address, which is memory location 0x71000 for the FDM and 0x79000 for the FCM.

b. The FCNT constant defines the filter length and is equal to twice the number of complex filter coefficients (that is, if there are five complex filter coefficients, FCNT should be 10). FCNT –1 is written to the filter count register.

c. The value 0x0011 is written to FCTL to enable the EFCOP in complex FIR filter mode with data initialization enabled.

3. Before the first filter operation, the FDM buffer must be initialized because the FPRC bit is clear. To initialize the FDM buffer for complex data, FCNT/2 complex samples must be written to the FDM through the FDIR. The code uses a short loop to write the first FCNT/2 complex samples of the input data to the FDIR using **move.2l** instructions.

4. The code waits until an output is available in the FDOR register by polling the FSTR[9]:FOBNE bit. The code tests the FSTR[9]:FOBNE bit and jumps back to the test if the bit is not set. When this bit is set, the output of the filter operation is ready and the code continues.

5. The FDOR is read using a **move.2l** instruction, and the complex value is placed into the output data buffer.

6. The next complex input data sample is written to the FDIR, and the process continues until all the input data values are written to the FDIR.

7. The last output value is read, and the filter is complete.

Example 9-1. Complex FIR Filter Code

```
move.w #INPUT,r0        move.w #OUTPUT,r1      ;Init data pointers
move.l #M_FDIR,r2
move.l #M_FDOR,r3

move.w #$400,d0                                ;Init FDBA
move.w d0,M_FDBA
move.w #$400,d0                                ;Init FCBA
move.w d0,M_FCBA
move.w #FCNT-1,d0                              ;Init FCNT
move.w d0,M_FCNT
move.w #$0011,d0                               ;Init FCTL
move.w d0,M_FCTL

doensh0 #FCNT/2                                ;Init data taps
move.2l (r0)+,d0:d1
loopstart0
move.2l d0:d1,(r2)
move.2l (r0)+,d0:d1
loopend0
```

```
            dosetup0 empty                          doen0 #NSAMP
            loopstart0
empty       move.w M_FSTR,d4
            bmtstc #$0040,d4.l
            jt empty                                ;Wait until out not empty

            move.2l (r3),d2:d3
            move.2l d2:d3,(r1)+                     ;Read output
            move.2l d0:d1,(r2)
            move.2l (r0)+,d0:d1                     ;Write input
            loopend0

endempty
            move.w M_FSTR,d4
            bmtstc #$0040,d4.l
            jt endempty                             ;Wait until out not empty
            move.2l (r3),d2:d3
            move.2l d2:d3,(r1)                      ;Read last output
```

Before this code can run, the SC140 core must initialize the coefficient buffer because EFCOP coefficient initialization is disabled for this example. **Example 9-2** shows an easy way to do this. Here the coefficients are positioned, with **org** and **dcl** directives, at the location of the FCM (which is memory location 0x79000 as described earlier). For each complex coefficient, the real and imaginary parts are stored in memory separately with the real part first. *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.

Example 9-2. Coefficient Initialization

```
            org P:$00079000                         ;Init coefficients
            dcl [Re H(FCNT-1)]
            dcl [Im H(FCNT-1)]
                                 .
                                 .
                                 .
            dcl [Im H(0)]
            dcl [Im H(0)]
```

Code such as shown in **Example 9-1** and **Example 9-2** may not appear in an actual application, but this code shows how the EFCOP works in a very simple way. The next example shows a more sophisticated way to use the EFCOP.

9.6.2 Adaptive Filter With Interrupts

The code in **Example 9-3** shows how to program the EFCOP to implement a real FIR filter using interrupts as the transfer method. In Real mode, each filter operation begins by writing one 32-bit data sample to the FDIR register using a **move.l** instruction. This code

uses Adaptive mode and the data output not empty interrupt ($\overline{\text{IRQ3}}$) to update the coefficients as shown in **Section 9.2.1.1**, as follows:

1. Address register pointers are initialized for the filter input and output data (`INPUT` and `OUTPUT`).
2. The EFCOP control parameters are written to the appropriate memory mapped control registers as follows:
   a. FDM and FCM are located at the beginning of the shared memory, so the FDBA and FCBA registers are written with zero.
   b. The `FCNT` constant defines the filter length and is equal to the number of real filter coefficients. `FCNT - 1` is written to the filter count register.
   c. The value 0x2105 is written to the control register to enable the EFCOP in real FIR filter mode with Adaptive mode enabled and data and coefficient initialization enabled. This value also enables the Data Output Not Empty ($\overline{\text{IRQ3}}$) interrupt.
3. The code enables interrupts as follows:
   a. The appropriate bits in the core control registers are set. The interrupt mask bits (I0–2) of the SR are cleared to permit all interrupt priority levels.
   b. The value 0x7000 is written to the ELIRA. This value sets the PIL[30–32] bits to assign $\overline{\text{IRQ3}}$ with a priority level a six. This value also clears the PED3 bit to assign $\overline{\text{IRQ3}}$ to be level triggered.
   c. Interrupts are enabled by issuing an **ei** instruction.
4. The FDM buffer must be initialized because the FPRC bit is clear. To initialize the FDM buffer, `FCNT` samples are written to the FDM through the FDIR. The code uses a short loop to write the first `FCNT` input samples to the FDIR using a **move.l** instruction.
5. The FCM buffer must be initialized because the FCTL[7]:FCIM bit is set. After the `FKIR` value is written to the K-constant input register, the EFCOP initializes the FCM buffer using a coefficient update session with the original coefficients equal to zero (so it does not matter what the values are in the FCM buffer memory locations before the code begins).
6. Processing begins with the write of the first input data sample to the FDIR.

Example 9-3. Adaptive Filter Code

```
move.w #INPUT,r0        move.w #OUTPUT,r1      ;Init data pointers

move.w #0,d0                                   ;Init FDBA
move.w d0,M_FDBA
move.w #0,d0                                   ;Init FCBA
move.w d0,M_FCBA
move.w #FCNT-1,d0                              ;Init FCNT
move.w d0,M_FCNT
move.w #$2105,d0                               ;Init FCTL
move.w d0,M_FCTL

bmclr #$00E0,sr.h                              ;Enable all IPL
move.w #$7000,d0                               ;Out not empty IPL 6
move.w d0,M_ELIRA
ei                                             ;Enable interrupts

doensh0 #FCNT
loopstart0                                     ;Init data taps
move.l (r0)+,d0
move.l d0,M_FDIR
loopend0

move.l #FKIR,d0                                ;Init coeffs
move.l d0,M_FKIR

                       move.l (r0)+,d0         ;Write first input
move.l d0,M_FDIR
```

The main code is now complete and the SC140 core can run other application code while the EFCOP completes the filter operation. When the EFCOP completes the filter operation, it places the result in the FDOR, which triggers the data output not empty interrupt. The processor jumps to the appropriate interrupt vector address.

The interrupt vector code, shown in **Example 9-4**, uses an equate label, I_IRQ3, for the location of the EFCOP data output not empty interrupt vector address. The example assumes that this equate is declared prior to the example code. The interrupt vector code includes the command to jump to the interrupt service routine.

Example 9-4. Interrupt Vector Code

```
org P:I_IRQ3                                   ;Output not empty vector
jsr OBNE_ISR
rte
```

The interrupt service routine code, shown in **Example 9-5**, completes the processing as follows:

1. The code moves the filter output from FDOR to the `output` data buffer.
2. The step parameter is loaded into FKIR. Once FKIR is loaded, the EFCOP performs the coefficient update session, as discussed in **Section 9.2.1.1**, and replaces the filter coefficients with the updated coefficients.
3. The next input sample is written from the `input` data buffer to the FDIR to begin the next filter operation, and the process begins again.

Example 9-5. Interrupt Service Routine Code

```
OBNE_ISR                                        ;Output not empty ISR
            move.l M_FDOR,d0                    ;Read output
            move.l d0,(r1)+
            move.l #FKIR,d0                     ;Write coef update param
            move.l d0,M_FKIR
            move.l (r0)+,d0                     ;Write input
            move.l d0,M_FDIR
            rts
```

Example 9-5 shows the basics of adaptive filtering. The transfer method of this example is more complex than in the polling example, but the SC140 core can process other routines while the EFCOP is computing. Code similar to this example may be seen in an actual application. This example is not complete, because it does not contain code to determine how many samples to process. This example also uses a simple constant for the coefficient update parameter. Other applications may calculate a value to update the coefficients.

9.6.3 Real IIR Filter with DMA

The code in **Example 9-6** and **Example 9-7** shows how to program the EFCOP to implement a real IIR filter using DMA to transfer data into and out of the EFCOP data registers. Processing a complete IIR filter requires two sessions: a FIR filter session (see **Section 9.2**) followed by an IIR filter session (see **Section 9.3**). This example uses dual-access DMA transactions for the FIR session and flyby DMA transactions for the IIR session. Each DMA transaction transfers `NSAMP` bytes, so the EFCOP processes NSAMP/4 32-bit samples.

Example 9-6 and **Example 9-7** use equate labels for the location of the DMA channel configuration registers and DMA Channel Parameter RAM fields. The code assumes that these equates are declared prior to the example code. These labels include the register or field name preceded with “M_”. This code also assumes that banks 10 and 11 of the memory controller are configured to allow the DMA to access the internal DSP SRAM

through the UPMC and to access the EFCOP registers through the GPCM, respectively. The memory buffers to which the IN_ADDR, TMP_ADDR, and OUT_ADDR equates point must be within the memory range of bank 10. The FDIR_ADDR and FDOR_ADDR equates must point to the EFCOP input and output register locations in bank 11.

The code in **Example 9-6** shows the code for the FIR session using dual-access DMA transactions. Four DMA channels are used for the FIR session: two channels (0 and 1) to transfer the data to the FDIR and two channels (2 and 3) to transfer the data from the FDOR. The DMA transfers occur in single transfer mode, that is, the DMA transfers one 32-bit sample to the FDIR whenever the FDIR is not full, and the DMA transfers one 32-bit sample from the FDOR whenever the FDOR is not empty. The FIR session proceeds as follows:

1. The following control parameters are written to the EFCOP control registers:
   a. The FDM and FCM are located at an offset from the beginning of the shared memory defined by the FIR_FDBA and FIR_FCBA constants.
   b. The FIR_FCNT constant defines the FIR filter length and is equal to the number of real filter coefficients. FIR_FCNT – 1 is written to the Filter Count Register.
   c. The value 0x0081 is written to FCTL to enable the EFCOP in real FIR filter mode with data initialization disabled. This value also sets the input and output data modes to single-transfer.
2. DMA Channel 0 transfers the data from memory to the DMA FIFO. The channel 0 DMA control registers are programmed as follows:
   a. The address location of the input data, IN_ADDR, is written to the DMA buffer address pointer field (BD_ADDR0).
   b. The total number of bytes to transfer, NSAMP, is written to the DMA buffer size field (BD_SIZE0).
   c. To configure channel 0 for 32-bit read transactions, the value 0x00000190 is written to the DMA attribute field (BD_ATTR0).
   d. To enable channel 0 as dual access transaction initiated by the DMA, the value 0x80000045 is written to the DMA Channel Configuration Register (DCHCR0).
3. DMA Channel 1 transfers the data from the DMA FIFO to the FDIR. The channel 1 DMA control registers are programmed as follows:
   a. The address location of the FDIR in bank 11, FDIR_ADDR, is written to the DMA buffer address pointer field (BD_ADDR1).
   b. The total number of bytes to transfer, NSAMP, is written to the DMA buffer size field (BD_SIZE1).

c. To configure channel 1 for 32-bit write transactions without incrementing the buffer address (always transfers to the FDIR), the value 0x08000180 is written to the DMA attribute field (BD_ATTR1).

d. To enable channel 1 in dual access mode triggered by an EFCOP write request, the value 0x80010305 is written to the DMA Channel Configuration Register (DCHCR1).

4. DMA Channel 2 transfers the data from the FDOR to the DMA FIFO. The channel 2 DMA control registers are programmed as follows:

a. The address location of the FDOR in bank 11, `FDOR_ADDR`, is written to the DMA buffer address pointer field (BD_ADDR2).

b. The total number of bytes to transfer, `NSAMP`, is written to the DMA buffer size field (BD_SIZE2).

c. The value 0x08000190 is written to the DMA attribute field (BD_ATTR2). This value configures channel 2 for 32-bit read transactions without incrementing the buffer address (always transfers from the FDOR).

d. The value 0x80020205 is written to the DMA Channel Configuration Register (DCHCR2). This value enables channel 2 in dual access mode triggered by an EFCOP read request.

5. DMA Channel 3 transfers the data from the DMA FIFO to memory. The channel 3 DMA control registers are programmed as follows:

a. The address location of the FIR output data, `TMP_ADDR`, is written to the DMA buffer address pointer field (BD_ADDR3).

b. The total number of bytes to transfer, `NSAMP`, is written to the DMA buffer size field (BD_SIZE3).

c. To configure channel 3 for a 32-bit write transaction, the value 0x00000180 is written to the DMA attribute field (BD_ATTR3).

d. To enable channel 3 as dual access transaction initiated by the DMA, the value 0x80030045 is written to the DMA Channel Configuration Register (DCHCR3).

Once the DMAs and EFCOP are programmed, they work together requesting and sending data without intervention of the SC140 core. After all `NSAMP/4` data samples are processed, the IIR session begins.

Example 9-6. FIR Filter Session

```
            move.w #FIR_FDBA,d0                     ;Init FDBA
            move.w d0,M_FDBA
            move.w #FIR_FCBA,d0                     ;Init FCBA
            move.w d0,M_FCBA
            move.w #FIR_FCNT-1,d0                   ;Init FCNT
            move.w d0,M_FCNT
            move.w #$0081,d0                        ;Init FCTL
            move.w d0,M_FCTL

;DMA0 init to transfer Memory to DMA FIFO
            move.l #IN_ADDR,d0                      ;Init source address
            move.l d0,M_BDADDR0
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE0
            move.l #$00000190,d0                    ;Init channel 0 attrib
            move.l d0,M_BDATTR0
            move.l #$80000045,d0                    ;Init channel 0 config
            move.l d0,M_DCHCR0

;DMA1 init to transfer DMA FIFO to FDIR
            move.l #FDIR_ADDR,d0                    ;Init destination address
            move.l d0,M_BDADDR1
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE1
            move.l #$08000180,d0                    ;Init channel 1 attrib
            move.l d0,M_BDATTR1
            move.l #$80010305,d0                    ;Init channel 1 config
            move.l d0,M_DCHCR1
;DMA2 init to transfer FDOR to DMA FIFO
            move.l #FDOR_ADDR,d0                    ;Init source address
            move.l d0,M_BDADDR2
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE2
            move.l #$08000190,d0                    ;Init channel 2 attrib
            move.l d0,M_BDATTR2
            move.l #$80020205,d0                    ;Init channel 2 config
            move.l d0,M_DCHCR2
;DMA3 init to transfer DMA FIFO to Memory
            move.l #TMP_ADDR,d0                     ;Init destination address
            move.l d0,M_BDADDR3
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE3
            move.l #$00000180,d0                    ;Init channel 3 attrib
            move.l d0,M_BDATTR3
            move.l #$80030045,d0                    ;Init channel 3 config
            move.l d0,M_DCHCR3
```

Example 9-7 shows the code for the IIR session using flyby DMA transactions. Two DMA channels are used for the IIR session: channel 0 to transfer the data to the FDIR and channel 1 to transfer the data from the FDOR. The DMA transfers occur in burst transfer mode, that is, channel 0 transfers eight 32-bit samples to the FDIR whenever the FDIR is

empty and channel 1 transfers eight 32-bit samples from the FDOR whenever the FDOR is full. The IIR session proceeds as follows:

1. The EFCOP is disabled by clearing the FCTL before the IIR session parameters are programmed into the FCTL.
2. The following control parameters are written to the EFCOP control registers:
   a. The FDM and FCM are located at an offset from the beginning of the shared memory defined by the `IIR_FDBA` and `IIR_FCBA` constants.
   b. The `IIR_FCNT` constant defines the IIR filter length and is equal to the number of real filter coefficients. `IIR_FCNT` – 1 is written to the Filter Count Register.
   c. The `IIR_FACR` constant is written to the Filter ALU Control Register. This constant can be used to enable scaling for the IIR output if necessary.
   d. The value 0xC083 is written to FCTL to enable the EFCOP in IIR filter mode. This value also sets the input and output data modes to burst transfer.
3. DMA Channel 0 of the DMA is used in flyby mode to transfer the input data from memory to the FDIR in burst mode; that is, the DMA transfers eight 32-bit samples to the FDIR whenever the FDIR is empty. The DMA control registers are programmed as follows:
   a. The address location of the input data (the FIR session output), `TMP_ADDR`, is written to the DMA buffer address pointer field (BD_ADDR0).
   b. The total number of bytes to transfer, `NSAMP`, is written to the DMA buffer size field (BD_SIZE0).
   c. To configure the DMA for a burst read transaction, the value 0x00000210 is written to the DMA attribute field (BD_ATTR0).
   d. To enable DMA channel 0 in flyby mode triggered by an EFCOP write request, the value 0x80004305 is written to the DMA Channel Configuration Register (DCHCR0).
4. DMA Channel 1 of the DMA is used in flyby mode to transfer the output data from the FDOR to memory in burst mode; that is, the DMA transfers eight 32-bit samples from the FDOR whenever the FDOR is full. The DMA control registers are programmed as follows:
   a. The address location of the output data, `OUT_ADDR`, is written to the DMA buffer address pointer field (BD_ADDR1).
   b. The total number of bytes to transfer, `NSAMP`, is written to the DMA buffer size field (BD_SIZE1).
   c. The value 0x00000200 is written to the DMA attribute field (BD_ATTR1). This value configures the DMA for a burst write transaction.

d. The value 0x80014204 is written to the DMA Channel Configuration Register (DCHCR1). This value enables DMA channel 1 in flyby mode triggered by an EFCOP read request.

Example 9-7. IIR Filter Session

```
            move.w #0,d0                            ;Disable EFCOP
            move.w d0,M_FCTL

            move.w #IIR_FDBA,d0                     ;Init FDBA
            move.w d0,M_FDBA
            move.w #IIR_FCBA,d0                     ;Init FCBA
            move.w d0,M_FCBA
            move.w #IIR_FCNT-1,d0                   ;Init FCNT
            move.w d0,M_FCNT
            move.w #IIR_FACR,d0                     ;Init FACR
            move.w d0,M_FACR
            move.w #$C083,d0                        ;Init FCTL
            move.w d0,M_FCTL

;DMA0 init to input DATA to EFCOP
            move.l #TMP_ADDR,d0                     ;Init source address
            move.l d0,M_BDADDR0
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE0
            move.l #$00000210,d0                    ;Init channel 0 attrib
            move.l d0,M_BDATTR0
            move.l #$80004305,d0                    ;Init channel 0 config
            move.l d0,M_DCHCR0

;DMA1 init to output DATA from EFCOP
            move.l #OUT_ADDR,d0                     ;Init destination address
            move.l d0,M_BDADDR1
            move.l #NSAMP,d0                        ;Init transfer size
            move.l d0,M_BDSIZE1
            move.l #$00000200,d0                    ;Init channel 1 attrib
            move.l d0,M_BDATTR1
            move.l #$80014204,d0                    ;Init channel 1 config
            move.l d0,M_DCHCR1
```

Once the DMAs and EFCOP are programmed, they work together requesting and sending data without intervention of the SC140 core. When all `NSAMP/4` data samples are processed, the filter is complete.

Example 9-6 and **Example 9-7** show the basics of IIR filtering and DMA transactions with the EFCOP. The transfer method of this example requires the least core intervention once the EFCOP and DMAs are programmed and requires no address registers. Code similar to this may appear in an actual application. However, this example is not complete because it does not contain code to initialize the FDM and FCM. This example also does not contain code to determine when the DMA transfers of each session are complete, which can be done with DMA interrupts or polling.

9.7 Related Reading

MSC8101 User's Guide (This manual)

Chapter 1, *MSC8101 Overview* **Section 1.3.7.3**, *Buffer Descriptors*, on page 1-13

Chapter 6, *DMA Channels*

MSC8101 Reference Manual

Chapter 15, *Direct Memory Access (DMA)*

Chapter 18, *Enhanced Filter Coprocessor (EFCOP)*