Addendum

MPC8xxRMAD
Rev. 0.1, 10/2003

MPC8xx Digital Signal Processing Addendum

Manuals that this addendum is applicable to are:

MPC885RM, Rev 0.1
MPC866UM, Rev 1.1
MPC862UM, Rev 2.0
MPC860UM, Rev 2.0
MPC855TUM, Rev 1
MPC850UM, Rev 1
MPC823UM, Rev 1

The CPM on members of the MPC8xx PowerQUICC family (which includes the MPC850, MPC823, MPC855T, MPC860, MPC857DSL, MPC857T, MPC862, MPC852T, MPC859DSL, MPC859, MPC866, MPC885, MPC880, MPC875, and MPC870) provides specialized hardware and library functions to support DSP applications. The communication processor's multiply-and-accumulate (MAC) handles real or complex numbers, and the address generator calculates modulo addressing for circular buffer structures. Library functions include finite impulse response (FIR) filtering done with or without adaptive equalization, data compression, and scrambling. The following topics are addressed in this addendum:

1 Features

The following outlines the user interface to the DSP functionality:

- ROM microcode library provides basic DSP routines for such applications as V.32bis and V.34
- Data and function parameters are formatted by the core using function descriptors (FDs)
- DSP routines are core-initiated using CP commands (INIT DSP and START DSP)
- Maskable interrupts are issued to the core upon completion of DSP routines

The following summarizes the DSP features of the CPM:

- 16-bit × 16-bit multiply-and-accumulate (MAC) engine
  - Two 40-bit accumulators with overflow saturation logic
  - Two 32-bit input registers
  - One MAC operation per clock (2-clock latency, 1-clock blockage)
  - A single instruction triggers a sequence of one, two or four MACs
  - Concurrent operation with other instructions
  - Complex (16-bit real, 16-bit imaginary) FIR loop: 4 clocks per 4 multiplies
- Load/store instructions with automatic post increment/decrement
  - Post increment/decrement by 0, 1, 2, or 4
  - Modulo addressing and modifier for circular buffer support

2 DSP Functionality

DSP functionality can be divided into three layers—hardware, firmware, and software. Figure 1 shows the DSP functionality implementation.

Core Software	Function descriptor chain in external memory defines the sequence and data flow of the DSP functions.
CPM Firmware	Generic DSP microcode routine library stored in the internal ROM.
CPM Hardware	MAC and address generator modules in the CP architecture.

Figure 1. DSP Functionality Implementation

The user defines the software layer to build an application. A software interface is defined that enables parameters (pointer to filter coefficients, and pointers to input and output buffers) to be passed between the core and the CPM. Several functions can be chained together to reduce core intervention and interrupt rates, assuming that all data structures are in the dual-port RAM. Two special DSP host commands signal the CPM to initialize or execute the DSP function descriptor (FD) chain. A maskable interrupt signals the core to resume control once the CPM executes the chain.

Table 1 lists the available DSP functions with opcodes.

Table 1. DSP Library Functions

Function	Opcode	Input	Coefficient	Output	Application
FIR1	00001	Real	Real	Real	Decimation, Rx interpolation
FIR2	00010	Complex	Real	Complex	Tx filter, Rx filter
FIR3	00011	Complex	Complex	Real/Complex	EC computation, equalizer
FIR5	00101	Complex	Complex	Real/Complex	Fractionally spaced equalizer
FIR6	00110	Real	Complex	Complex	—
IIR	00111	Real	Real	Real	Biquad filter
MOD	01000	Complex	Complex	Real/Complex	Tx modulation
DEMOD	01001	Real	Complex	Complex	Rx demodulation
LMS1	01010	—	—	—	EC update, equalizer update (T/2, T/3)
LMS2	01011	—	—	—	Equalizer update (2T/3)
WADD	01100	Real	—	Real	Interpolation

3 DSP Function Descriptors (FDs)

Similar in structure to SCC buffer descriptors, a function descriptor (FD) specifies the DSP function and contains function-specific parameters. Prepared in external memory, a group of FDs can be chained together to form a circular queue of programmable length. There are two such FD chains—one for the transmitter and one for the receiver. (FD chains are logically equivalent to BD tables for SCCs.) Figure 2 shows the FD chain structure.

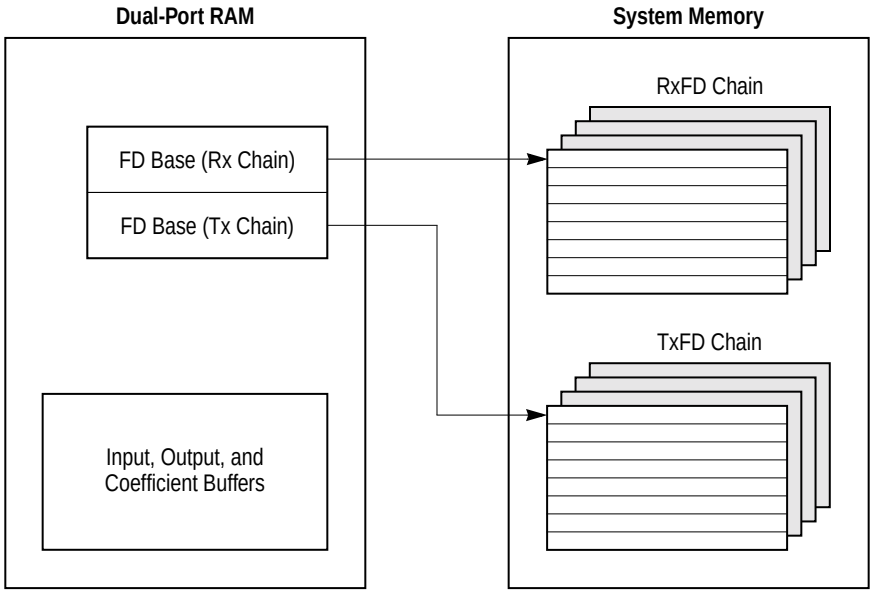


Figure 2. DSP Function Descriptor (FD) Chain Structure

A FD consists of eight 16-bit entries. The first entry contains status and control bits including the function opcode. The remaining seven entries contain the function's parameter packet. Figure 3 shows the general structure of a FD.

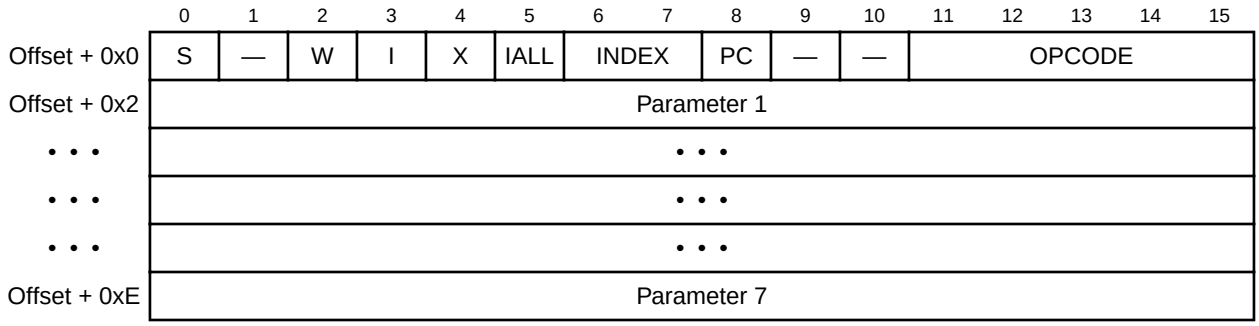


Figure 3. Function Descriptor (FD) Structure

Table 2 describes the status and control bits. All of the library functions use the stop, wrap, and interrupt bits. The use of the remaining control bits, apart from the opcode, depends on the particular function. The parameter packets are described with the individual functions.

Table 2. FD Status and Control Bits

Bits	Name	Description
0	S	Stop processing. 0 Do not stop after processing this FD. 1 Stop after processing this FD.
1	—	Reserved
2	W	Wrap to the beginning of the chain. Determines the length of the FD chain. 0 Not the last FD in the chain. 1 The last FD in the chain. After this FD has been processed, the CP returns to the top of the chain pointed to by FDBASE.
3	I	Interrupt the core. 0 No interrupt is generated after this function is processed. 1 A maskable interrupt is generated after this function is processed.
4	X	Complex number option. Used to specify a real/complex output, or a real/complex scalar for LMS. 0 Use only the real component. 1 Use both the real and imaginary components.
5	IALL	Auto-increment X for all iterations. 0 X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in FD[INDEX] after the last iteration. 1 X (input) data pointer is incremented (Modulo M+1) by the number of samples specified in FD[INDEX] after each iteration.
6–7	INDEX	Auto-increment index. 00 X (input) pointer is not incremented. 01 X (input) pointer is incremented by one sample. 10 X (input) pointer is incremented by two samples. 11 X (input) pointer is incremented by three samples.

Table 2. FD Status and Control Bits (continued)

Bits	Name	Description
8	PC	Preset coefficients pointer. 0 Coefficients pointer is not preset after each iteration. 1 Coefficients pointer is preset to CBASE after each iteration.
11–15	OPCODE	Function operation code. Specifies the function to be executed. See Table 1 above.

4 Data Representation

The inputs, coefficients, and outputs are represented by 16-bit, fixed-point, two's-complement numbers. Figure 4 shows the real number representation.

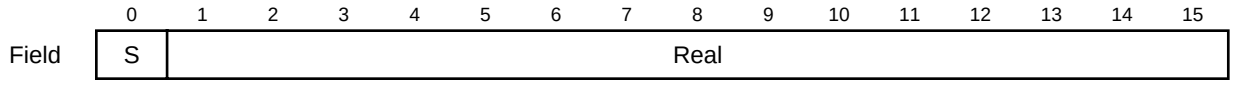


Figure 4. Real Number Representation

A complex number is represented by a pair of 16-bit components—16 bits for the imaginary component and 16 bits for the real component. Figure 5 shows the complex number representation.

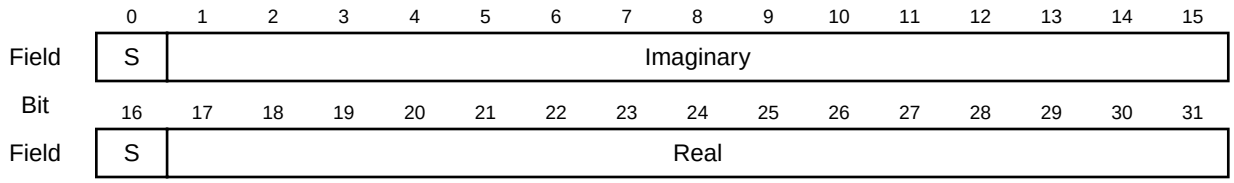


Figure 5. Complex Number Representation

5 Input and Output Buffers

The input and output buffers are circular in implementation, with their sizes programmed in the FD parameter packets. The input and output buffer lengths are (M + 1) and (N + 1) bytes, respectively, where (M + 1) and (N + 1) are both multiples of four.

The input and output buffers must each be aligned on natural boundaries in the dual-port RAM. A natural boundary is an address evenly divisible by 2^z, where 2^z is greater than the size of the buffer. For example, an input buffer with a size of 24 bytes must reside in dual-port RAM at an address evenly divisible by 32 (2^5). Figure 6 illustrates a circular buffer.

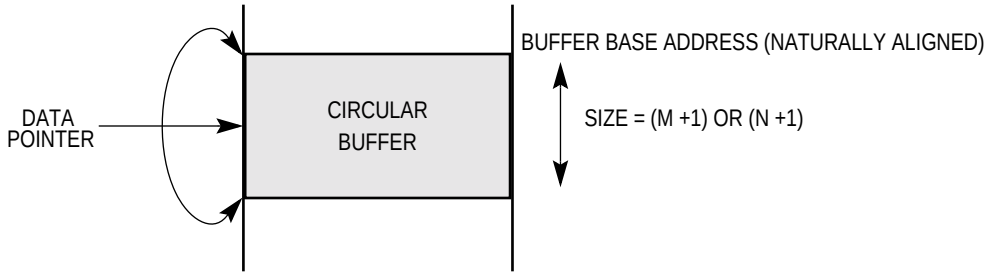


Figure 6. Circular Buffer

6 Buffer and Coefficient Base Pointers (CBASE, XPTR, XYPTR)

The input buffer, output buffer, and coefficient buffer pointers are 16-bit offsets from the base of the dual-port RAM. These include CBASE and the buffer pointers in the structures pointed to by XPTR and XYPTR. The structures pointed to by XPTR and XYPTR consist of a halfword-aligned array of the 16-bit pointers as defined by the specific DSP library function.

7 DSP Parameter RAM

Two areas of the dual-port RAM hold the DSP parameters and scratchpad. The Rx chain (DSP1) parameter area begins at the dual-port RAM offset 0x1EC0, and the Tx chain (DSP2) parameter area begins at 0x1FC0.

The FDBASE parameter defines the starting address for the FD chain in system memory. FDBASE should be 16-byte aligned and initialized before issuing INIT_DSP. Table 3 shows the DSP*x* parameter RAM memory map.

Table 3. DSP*x* Parameter RAM Memory Map

Offset [1]	Name	Width	Description
0x00	**FDBASE**	Word	Function descriptor chain base address.
0x04	FD_PTR	Word	Current FD pointer
0x08	DSTATE	Word	DSP state
0x0C	—	Word	Reserved
0x10	DSTATUS	Hword	Current FD status
0x12	I	Hword	Current FD number_of_iterations
0x14	TAP	Hword	Current FD number_of_taps
0x16	CBASE	Hword	Current FD coefficient buffer base
0x18	—	Hword	Current FD sample buffer_size – 1
0x1A	XPTR	Hword	Current FD pointer to input buffer
0x1C	—	Hword	Current FD output buffer_size – 1
0x1E	YPTR	Hword	Current FD pointer to output buffer
0x20	M	Hword	Current FD input buffer_size – 1
0x22	—	Hword	Current FD input buffer data pointer
0x24	N	Hword	Current FD output buffer_size – 1
0x26	—	Hword	Current FD output buffer data pointer
0x28	K	Hword	Current FD coefficient buffer_size – 1
0x2A	—	Hword	Current FD coefficient buffer data pointer

[1] Offset from DSP base. DSP1 base is 0x1EC0; DSP2 base is 0x1FC0.

8 DSP CP Commands

Once the DSP parameters are in place, use INIT DSP and START DSP to begin processing the FD chain. Table 4 provides descriptions of these commands.

Table 4. DSP Opcodes

OPCODE	Command	Description
1100	INIT DSP	Initializes the DSP chain. Deactivates the current chain and initializes the current FD pointer (FD_PTR) to the FD chain base address (FDBASE).
1101	START DSP	Starts the DSP chain. Activates the current chain.

9 DSP Function Priority within the CPM

The execution of DSP functions has a priority level within the CPM that tracks the priority level programmed for IDMA; see the chapter, “Communications Processor,” in the specific microprocessor user’s manual for details. The IDMA priority (and thus the DSP priority) is programmed in the RCCR. IDMA and DSP effectively share the same priority slot; however, within that slot, DSP has priority. See the section, “RISC Controller Configuration Register (RCCR)” for details.

10 DSP Event/Mask Registers (SDSR/SDMR)

Since there is no dedicated DSP event register, the CP uses the SDMA status register (SDSR) to report the maskable DSP interrupts to the core. Figure 7 shows the register format. Note that writing a 1 to the corresponding bits clears the events. SDSR is cleared by reset and can be read at any time.

	0	1	2	3	4	5	6	7
Field	SBER	—					DSP2	DSP1
Reset	0							
R/W	R/W							
Addr	0x908 (SDSR); 0x90C (SDMR)							

Figure 7. DSP Event/Mask Registers (SDSR/SDMR)

Table 5 describes the SDSR/SDMR fields.

Table 5. SDSR/SDMR Field Descriptions

Bits	Name	Description
0	SBER	SDMA channel bus error. Indicates that an error caused the SDMA channel to be terminated during a read or write cycle. The SDMA bus error address can be retrieved from the SDMA address register (SDAR) at internal address (IMMR offset) 0x904.
1–5	—	Reserved. Must be cleared.
6	DSP2	DSP chain2 (Tx) interrupt. Set when the current FD in the transmitter chain has been completed. However, DSP2 only reports if the descriptor’s I bit is set.
7	DSP1	DSP chain1 (Rx) interrupt. Set when the current FD in the receiver chain has been completed. However, DSP1 only reports if the descriptor’s I bit is set.

The SDMA mask register (SDMR) is used to mask the DSP interrupts. The SDMR mirrors the bit format of the SDSR. Setting an SDMR bit enables the corresponding interrupt in SDSR; clearing a bit masks the interrupt. Reset clears the SDMR, disabling all interrupts.

11 FIR Library Functions

The DSP library provides five basic finite-impulse response filters, each specializing in a different combination of real or complex coefficients, input samples, and output. The following sections describe each variety of FIR filter. Table 6 shows the parameter packet common to all FIR filters.

Table 6. FIR Parameter Packet

Offset [1]	Name	Description
0x2	**I**	Number of iterations
0x4	**K**	Number_of_taps – 1. The number of taps should be a multiple of four.
0x6	**CBASE**	Filter coefficient vector base address
0x8	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (4 real or 2 complex samples).
0xA	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
0xC	**N**	Output buffer_size – 1. When FD[X]=1, the minimum output buffer size is 8 (2 complex outputs); when FD[X]=0, it is 4 (2 real outputs).
0xE	—	Reserved

[1] Offset from base of the FD.

11.1 FIR1–Real C, Real X, and Real Y

Using the values provided in the parameter packet, FIR1 implements a basic finite-impulse filter, shown in Figure 8, with K real coefficients, real input samples, and real output. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).

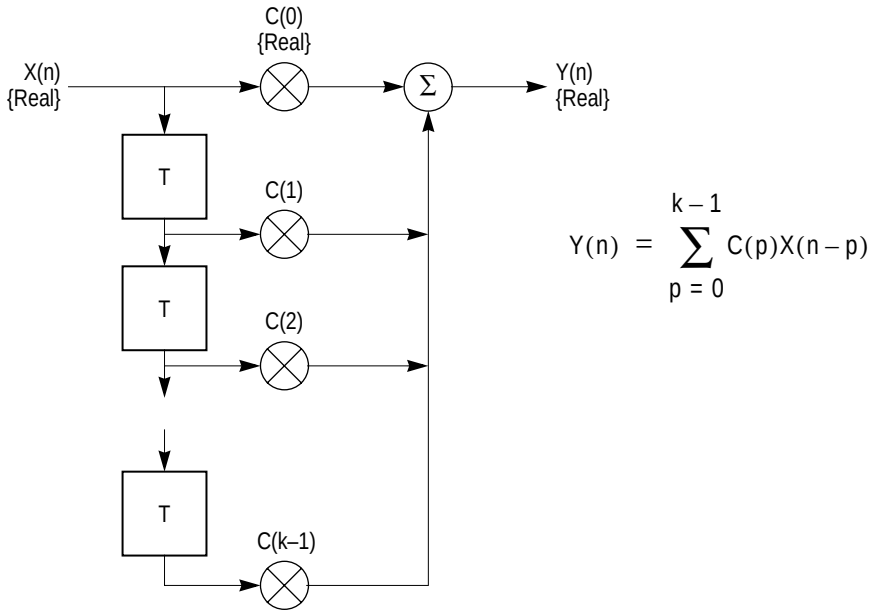


Figure 8. FIR1 Function

11.1.1 FIR1 Coefficient, Input, and Output Buffers

The coefficient vector occupies K 16-bit entries in memory, with C(0) stored in the first location. The 16-bit input samples are stored in order in a circular buffer containing (M+1) bytes. The 16-bit outputs are stored consecutively in a circular buffer containing (N+1) bytes. Table 7 displays the FIR1 coefficient, input, and output buffers.

Table 7. FIR1 Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Output
C(0)	*	*
C(1)	*	*
C(2)	x(n–k+1)	Y(n–k+1)
*	*	*
*	*	*
C(k–1)	x(n–2)	Y(n–2)
	x(n–1)	Y(n–1)
	x(n)	Y(n)

11.1.2 FIR1 Function Descriptor

The FIR1 function descriptor is shown in Figure 9.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0x0	**S**	—	**W**	**I**	—	**IALL**	**INDEX**		**PC**	—	—	**00001**				
Offset + 0x2	**I**															
Offset + 0x4	**K**															
Offset + 0x6	**CBASE**															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 9. FIR1 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The FIR1 parameter packet consists of seven 16-bit entries and is described in Table 8

Table 8. FIR1 Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Number_of_taps – 1. The number of taps should be a multiple of four.
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (4 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size is 4 (2 outputs).
Hword 7	—	Reserved

11.1.3 FIR1 Applications

The FIR1 is used in decimation and Rx interpolation. For example, the partial FD in Figure 10 can be used to implement a 2:1 decimation.

	S	—	W	I	—	IALL	INDEX	PC	—	—	OPCODE
Offset + 0	S	0	W	I	0	1	10	1	0	0	00001
Offset + 2	I=3 (Three iterations)										

Figure 10. FIR1 Decimation Example

11.2 FIR2–Real C, Complex X, and Complex Y

Using the values provided in the parameter packet, the FIR2 implements a basic FIR filter, shown in Figure 11, with K real coefficients, complex input samples, and complex output. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).

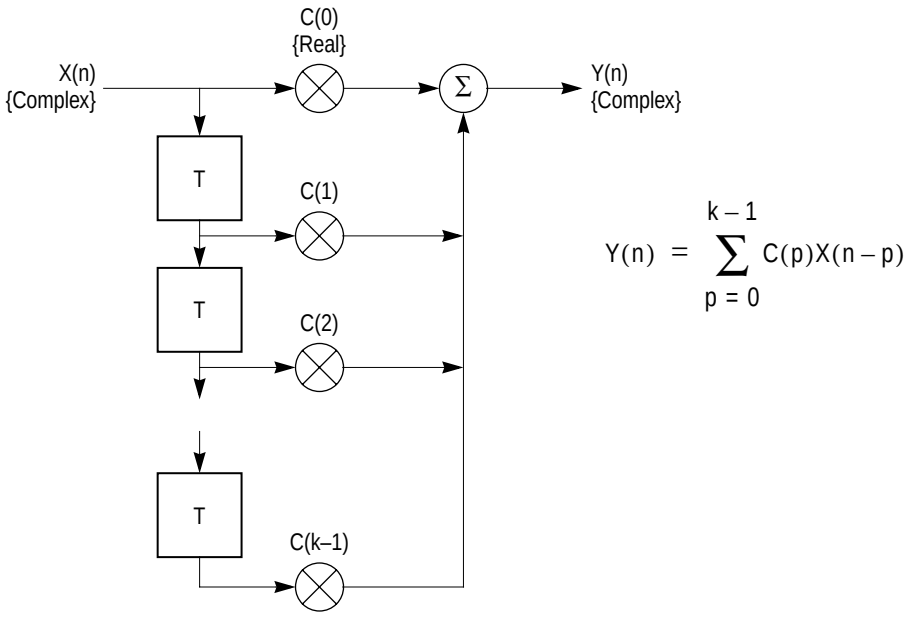


Figure 11. FIR2 Function

11.2.1 FIR2 Coefficient, Input, and Output Buffers

The coefficient vector occupies K 16-bit entries in memory, with C(0) stored in the first location. The input sample buffer is a circular buffer that contains (M+1) bytes; each input sample is two 16-bit entries (real and imaginary components). The next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer containing (N+1) bytes; each output is two 16-bit entries (real and imaginary components). The next output is stored in the address that follows the previous output. Table 9 displays the FIR2 coefficient, input, and output buffers.

Table 9. FIR2 Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Output
C(0)	*	*
C(1)	*	*
C(2)	imaginary {x(n–k+1)}	imaginary{Y(n–k+1)}
*	real {x(n–k+1)}	real{Y(n–k+1)}
*	*	*
C(k–1)	*	*
	imaginary {x(n–2)}	imaginary{Y(n–2)}
	real{x(n–2)}	real{Y(n–2)}
	imaginary{x(n–1)}	imaginary{Y(n–1)}
	real{x(n–1)}	real{Y(n–1)}
	imaginary{x(n)}	imaginary{Y(n)}
	real{x(n)}	real{Y(n)}

11.2.2 FIR2 Function Descriptor

The FIR2 function descriptor is shown in Figure 12.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**S**	—	**W**	**I**	—	**IALL**	**INDEX**		**PC**	—	—	**00010**				
Offset + 0x2	**I**															
Offset + 0x4	**K**															
Offset + 0x6	**CBASE**															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 12. FIR2 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The FIR2 parameter packet consists of seven 16-bit entries and is described in Table 10.

Table 10. FIR2 Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Number_of_taps – 1
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (4 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size is 8 (2 outputs).
Hword 7	—	Reserved

11.2.3 FIR2 Applications

The FIR2 is used for Tx and Rx filters. For example, the partial FD shown in Figure 13 can be used to implement a Tx filter.

	S	—	W	I	—	IALL	INDEX	PC	—	—	OPCODE
Offset + 0	S	0	W	I	0	0	01	0	0	0	00010
Offset + 2	I=3 (Three iterations)										

Figure 13. FIR2 Filter Example

11.3 FIR3–Complex C, Complex X, and Real/Complex Y

Using the values provided in the parameter packet, the FIR3 implements a basic FIR filter, shown in Figure 14, with K complex coefficients, complex input samples, and real or complex output. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).

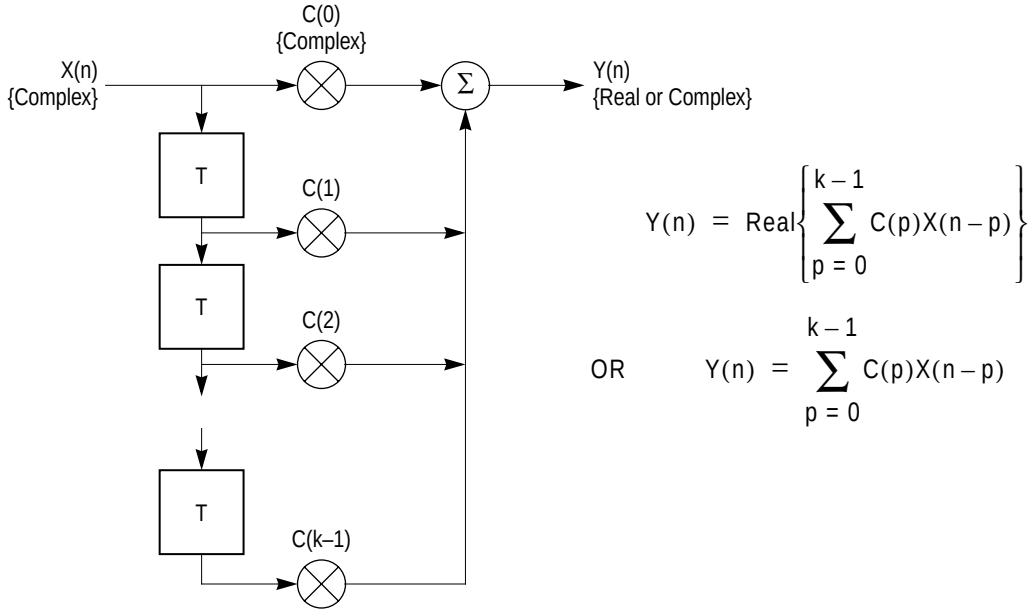


Figure 14. FIR3 Function

11.3.1 FIR3 Coefficient, Input, and Output Buffers

The coefficient vector occupies K pairs of 16-bit entries (real and imaginary components) in memory, with C(0) stored in the first location. The input sample buffer is a circular buffer containing (M+1) bytes; each input sample is two 16-bit entries (real and imaginary components). The next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contains (N+1) bytes; each output is two 16-bit entries (real and imaginary components). The next output is stored in the address that follows the previous output. Table 11 displays the FIR3 coefficient, input, and output buffers.

Table 11. FIR3 Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Complex Output, FD[X]=1	Real Output, FD[X]=0
imaginary{C(0)}	*	*	*
real{C(0)}	*	*	*
imaginary{C(1)}	imaginary {x(n–k+1)}	imaginary{Y(n–k+1)}	Y(n–k+1)
real{C(1)}	real {x(n–k+1)}	real{Y(n–k+1)}	*
*	*	*	*
*	*	*	Y(n–2)
imaginary{C(k–1)}	imaginary {x(n–2)}	imaginary{Y(n–2)}	Y(n–1)
real{C(k–1)}	real{x(n–2)}	real{Y(n–2)}	Y(n)
	imaginary{x(n–1)}	imaginary{Y(n–1)}	

Table 11. FIR3 Coefficient, Input, and Output Buffers (continued)

	real{x(n–1)}	real{Y(n–1)}
	imaginary{x(n)}	imaginary{Y(n)}
	real{x(n)}	real{Y(n)}

11.3.2 FIR3 Function Descriptor

The FIR3 function descriptor is shown in Figure 15.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**S**	—	**W**	**I**	**X**	**IALL**	**INDEX**		**PC**	—		**00011**				
Offset + 0x2	**I**															
Offset + 0x4	**K**															
Offset + 0x6	**CBASE**															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 15. FIR3 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The FIR3 parameter packet consists of seven 16-bit entries and is described in Table 12.

Table 12. FIR3 Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Number_of_taps – 1
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size for FD[X] = 1 is 8 (2 complex outputs). The minimum output buffer size for FD[X] = 0 is 4 (2 real outputs).
Hword 7	—	Reserved

11.3.3 FIR3 Applications

The FIR3 with the real output can be used in echo cancellation as shown in the sample in Figure 16; an equalizer can be implemented using the complex output.

	S	—	W	I	X	IALL	INDEX	PC	—	OPCODE
Offset + 0	S	0	W	I	0	0	01	0	00	00011
Offset + 2	I=3 (Three iterations)									

Figure 16. FIR3 Echo Cancellation Example

11.4 FIR5–Complex C, Complex X, and Complex Y

Using the values provided in the parameter packet, the FIR5 implements a basic FIR filter, shown in Figure 17, with complex coefficients, complex input samples, and complex output. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).

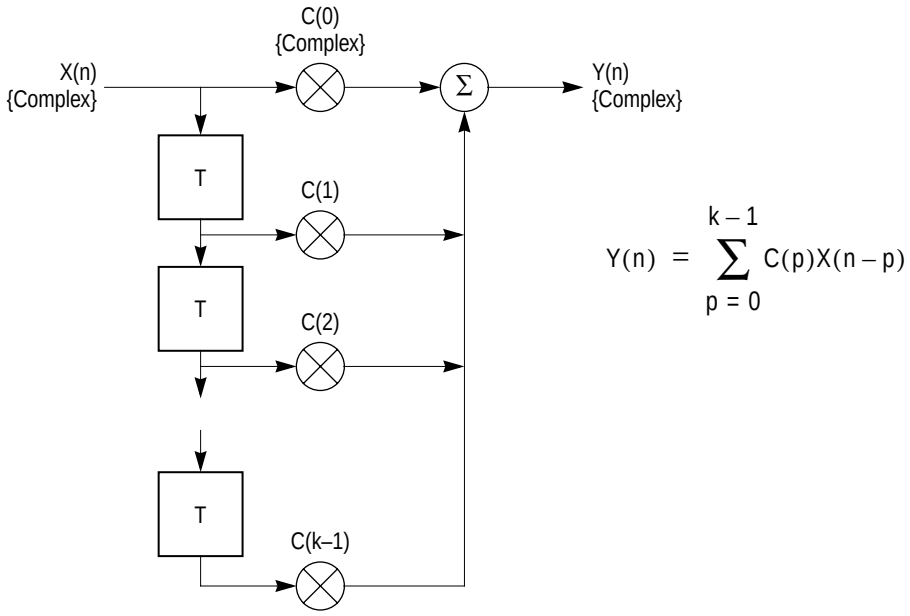


Figure 17. FIR5 Function

11.4.1 FIR5 Coefficient, Input, and Output Buffers

The coefficient vector occupies K pairs of 16-bit entries (real and imaginary components) in memory, with C(0) stored in the first location. The input sample buffer is a circular buffer containing (M+1) bytes; each input sample is two 16-bit entries (real and imaginary components). The next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contains (N+1) bytes, and the next output is stored in the address that follows the previous output. Table 13 displays the FIR5 coefficient, input, and output buffers.

Table 13. FIR5 Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Complex Output, FD[X]=1	Real Output, FD[X]=0
imaginary{C(0)}	*	*	*
real{C(0)}	*	*	*
imaginary{C(1)}	imaginary {x(n–k+1)}	imaginary{Y(n–k+1)}	Y(n–k+1)
real{C(1)}	real {x(n–k+1)}	real{Y(n–k+1)}	*
*	*	*	*
*	*	*	Y(n–2)
imaginary{C(k–1)}	imaginary {x(n–2)}	imaginary{Y(n–2)}	Y(n–1)
real{C(k–1)}	real{x(n–2)}	real{Y(n–2)}	Y(n)
	imaginary{x(n–1)}	imaginary{Y(n–1)}	
	real{x(n–1)}	real{Y(n–1)}	
	imaginary{x(n)}	imaginary{Y(n)}	
	real{x(n)}	real{Y(n)}	

11.4.2 FIR5 Function Descriptor

The FIR5 function descriptor is shown in Figure 18.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	S	—	W	I	X	IALL	INDEX		PC	—	—	00101				
Offset + 0x2	I															
Offset + 0x4	K															
Offset + 0x6	CBASE															
Offset + 0x8	M															
Offset + 0xA	XYPTR															
Offset + 0xC	N															
Offset + 0xE	—															

Figure 18. FIR5 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The FIR5 parameter packet consists of seven 16-bit entries and is described in Table 14.

Table 14. FIR5 Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Number_of_taps – 1
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer size – 1. The minimum output buffer size for FD[X] = 1 is 8 (2 outputs). The minimum output buffer size for FD[X] = 0 is 4 (2 outputs).
Hword 7	—	Reserved

11.4.3 FIR5 Applications

The FIR5 is used for fractionally spaced equalizers. The partial FD shown in Figure 19 can be used to implement a fractionally spaced equalizer.

	S	—	W	I	X	IALL	INDEX	PC	—	—	OPCODE
Offset + 0	S	0	W	I	1	0	11	0	0	0	00101
Offset + 2	I=1 (One Iteration)										

Figure 19. FIR5 Fractionally Spaced Equalizer Example

11.5 FIR6–Complex C, Real X, and Complex Y

Using the values provided in the parameter packet, the FIR6 implements a basic FIR filter, shown in Figure 20, with complex coefficients, real input samples, and complex output. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).

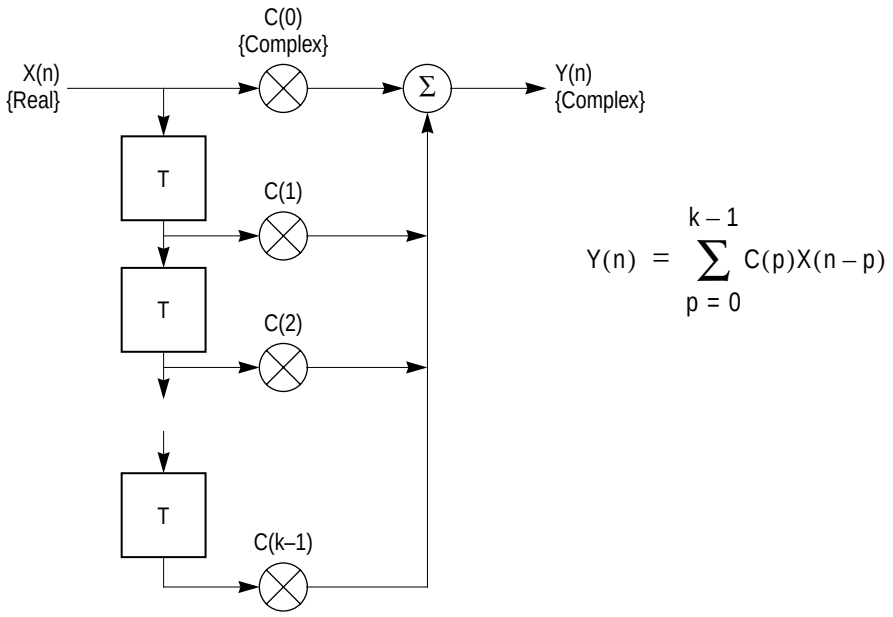


Figure 20. FIR6 Function

11.5.1 FIR6 Coefficient, Input, and Output Buffers

The coefficient vector occupies K pairs of 16-bit entries (real and imaginary components) in memory, and C(0) is stored in the first location. The input sample buffer is a circular buffer containing (M+1) bytes, and each sample is a 16-bit entry. The next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contains (N+1) bytes, and the next output is stored in the address that follows the previous output. Table 15 displays the FIR6 coefficient, input, and output buffers.

Table 15. FIR6 Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Output
imaginary{C(0)}	*	*
real{C(0)}	*	*
imaginary{C(1)}	x(n–k+1)	imaginary{Y(n–k+1)}
real{C(1)}	*	real{Y(n–k+1)}
*	*	*
*	x(n–2)	*
imaginary{C(k–1)}	x(n–1)	imaginary{Y(n–2)}
real{C(k–1)}	x(n)	real{Y(n–2)}
		imaginary{Y(n–1)}
		real{Y(n–1)}
		imaginary{Y(n)}
		real{Y(n)}

11.5.2 FIR6 Function Descriptor

The FIR6 function descriptor is shown in Figure 21.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	S	—	W	I	—	IALL	INDEX		PC	—	—	00110				
Offset + 0x2	I															
Offset + 0x4	K															
Offset + 0x6	CBASE															
Offset + 0x8	M															
Offset + 0xA	XYPTR															
Offset + 0xC	N															
Offset + 0xE	—															

Figure 21. FIR6 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The FIR6 parameter packet consists of seven 16-bit entries and is described in Table 16.

Table 16. FIR6 Parameter Packet

Address	Name	Description
Hword 1	**I**	Number_of_iterations
Hword 2	**K**	Number_of_taps – 1. The number of taps should be a multiple of 2.
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 4 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size is 8 (2 outputs).
Hword 7	—	Reserved

12 IIR–Real C, Real X, Real Y

Using the values provided in the parameter packet, the IIR implements a basic biquad IIR filter, shown in Figure 22, with six real coefficients, real input samples, and real outputs. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1). Several stages of the biquad filter can be cascaded by specifying an iteration count greater than one and concatenating the filter coefficients into one vector.

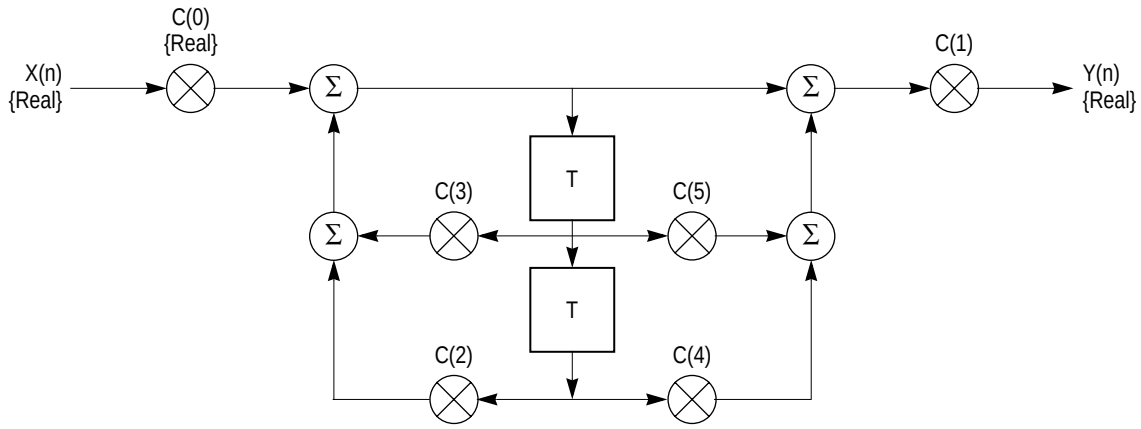


Figure 22. IIR Function

12.1 IIR Coefficient, Input, and Output Buffers

The coefficient vector occupies six 16-bit entries in memory, and C(0) is stored in the first location. C(1) is only used in the last stage of a cascaded IIR filter. The input sample buffer is a circular buffer that contains (M+1) bytes. The next sample is stored in the address that follows the previous one. The output buffer is a circular buffer that contains (N+1) bytes, and the next output is stored in the address that follows the previous one. Table 17 displays the IIR coefficient, input, and output buffers.

Table 17. IIR Coefficient, Input, and Output Buffers

Coefficients	Input Samples	Output
C(0)	*	*
C(1)	*	*
C(2)	x(n–2)	Y(n–2)
C(3)	x(n–1)	Y(n–1)
C(4)	x(n)	Y(n)
C(5)		

12.2 IIR Function Descriptor

The IIR function descriptor is shown in Figure 23.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	S	—	W	I	—		INDEX		—			00111				
Offset + 0x2	I															
Offset + 0x4	TPTR															
Offset + 0x6	CBASE															
Offset + 0x8	M															
Offset + 0xA	XYPTR															
Offset + 0xC	N															
Offset + 0xE	—															

Figure 23. IIR Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The IIR parameter packet consists of seven 16-bit entries and is described in Table 18.

Table 18. IIR Parameter Packet

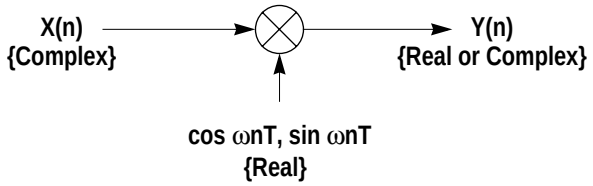
Address	Name	Description
Hword 1	**I**	Number of iterations (= cascaded stages)
Hword 2	**TPTR**	Pointer to a structure of temporary variables used by the delay line blocks. The structure consists of two real numbers and can be left uninitialized.
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 4 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size is 4 (2 outputs).
Hword 7	—	Reserved

12.3 IIR Applications

The IIR is used in timing recovery and interpolating filter, among other things.

13 Modulation (MOD)–Real Sin, Real Cos, Complex X, and Real/Complex Y

Using the values provided in the parameter packet, the MOD implements a basic modulator function, shown in Figure 24, with a modulation table composed of {cos ωnT, sin ωnT} pairs, complex input samples, and real outputs. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1).



$$Real\{Y(n)\} = Real\{X(n)\} \times \cos\omega nT - Image\{X(n)\} \times \sin\omega nT$$

$$Image\{Y(n)\} = Real\{X(n)\} \times \sin\omega nT + Image\{X(n)\} \times \cos\omega nT$$

Figure 24. MOD Function

13.1 Modulation Table, Input, and Output Buffers

The modulation table consists of 16-bit cosine and sine pairs that occupy (K+1) bytes in memory. The input sample buffer is a circular buffer containing (M+1) bytes. Each sample is a pair of 16-bit entries (real and imaginary components), and the next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contain (N+1) bytes, and the next output is stored in the address that follows the previous output. The output buffer can be real or complex, depending on FD[X]. Table 19 shows the modulation table, input, and output buffers.

Table 19. Modulation Table, Input, and Output Buffers

Modulation Table	Input Samples	Complex Output, FD[X]=1	Real Output, FD[X]=0
sin q_1	*	*	*
cos q_1	*	*	*
sin q_2	imaginary{x(n–k+1)}	imaginary{Y(n–k+1)}	real{Y(n–k+1)}
cos q_2	real{x(n–k+1)}	real{Y(n–k+1)}	*
*	*	*	*
*	*	*	real{Y(n–2)}
sin q_n	imaginary{x(n–2)}	imaginary{Y(n–2)}	real{Y(n–1)
cos q_n	real{x(n–2)}	real{Y(n–2)}	real{Y(n)}
	imaginary{x(n–1)}	imaginary{Y(n–1)}	
	real{x(n–1)}	real{Y(n–1)}	
	imaginary{x(n)}	imaginary{Y(n)}	
	real{x(n)}	real{Y(n)}	

13.2 MOD Function Descriptor

The MOD function descriptor is shown in Figure 25.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**S**	—	**W**	**I**	**X**	—						**01000**				
Offset + 0x2	**I**															
Offset + 0x4	**K**															
Offset + 0x6	**MPTR**															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 25. MOD Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The MOD parameter packet consists of seven 16-bit entries and is described in Table 20.

Table 20. MOD Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Modulation table_size – 1. The minimum modulation table size is 8 (2 sin/cos pairs).
Hword 3	**MPTR**	Pointer to modulation table
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size for FD[X] = 1 is 8 (2 outputs). The minimum output buffer size for FD[X] = 0 is 4 (2 samples).
Hword 7	—	Reserved

13.3 MOD Applications

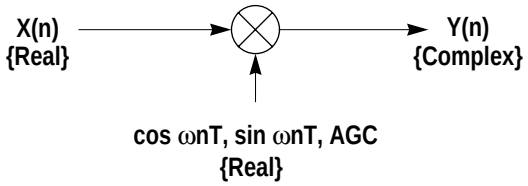
The MOD function is used in modulation. The partial FD shown in Figure 26 can be used to implement a modulator.

	S	—	W	I	X	—	—	—	—	—	—	OPCODE
Offset + 0	S	0	W	I	0	0	0	0	0	0	0	01000
Offset + 2	I=3 (Three iterations)											

Figure 26. MOD Modulation Example

14 DEMOD–Real Sin; Real Cos, Real X, and Complex Y

Using the values provided in the parameter packet, the DEMOD implements a basic demodulator function, shown in Figure 27, with a modulation table composed of (cos ωnT, sin ωnT) pairs, real input samples, and complex outputs. The input data is in a circular buffer with size (M+1), and the output data is in a circular buffer with size (N+1). The AGC parameter controls the demodulator gain.



$$\text{Real}\{Y(n)\} = (1 + AGC) \times X(n) \times \cos\omega nT$$

$$\text{Image}\{Y(n)\} = (1 + AGC) \times X(n) \times (-\sin\omega nT)$$

Figure 27. DEMOD Function

14.1 Modulation Table, Input and Output Buffers, and AGC Constant

The modulation table consists of 16-bit cosine and sine pairs that occupy (K+1) bytes in memory. The input sample buffer is a circular buffer containing (M+1) bytes. The next 16-bit sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contains (N+1) bytes, and the next output is stored in the address that follows the previous output. The AGC constant is in the range -1≤ AGC≤1. Table 21 shows the DEMOD modulation table, input, and output buffers.

Table 21. DEMOD Modulation Table, Input, and Output Buffers

Modulation Table	Input Samples	Output (Complex)
sin q_1	*	*
cos q_1	*	*
sin q_2	x(n–k+1)	imaginary{Y(n–k+1)}
cos q_2	*	real{Y(n–k+1)}
*	*	*
*	x(n–2)	*
sin q_n	x(n–1)	imaginary{Y(n–2)}
cos q_n	x(n)	real{Y(n–2)}
		imaginary{Y(n–1)}
		real{Y(n–1)}
		imaginary{Y(n)}
		real{Y(n)}

14.2 DEMOD Function Descriptor

The DEMOD function descriptor is shown in Figure 28.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**S**	—	**W**	**I**	—							**01001**				
Offset + 0x2	I															
Offset + 0x4	**K**															
Offset + 0x6	**DPTR**															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 28. DEMOD Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The DEMOD parameter packet consists of seven 16-bit entries and is described in Table 22.

Table 22. DEMOD Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	**K**	Modulation table_size – 1. The minimum modulation table size is 8 (2 sin/cos pairs).
Hword 3	**DPTR**	Pointer to a structure consisting of the modulation table pointer and the AGC constant (real)
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XYPTR**	Pointer to a structure composed of the input buffer pointer and the output buffer pointer
Hword 6	**N**	Output buffer_size – 1. The minimum output buffer size is 8 (2 outputs).
Hword 7	—	Reserved

14.3 DEMOD Applications

The DEMOD function is used in modulation. The partial FD shown in Figure 29 can be used for implementations.

	S	—	W	I	X	—	OPCODE
Offset + 0	S	0	W	I	0	0	01001
Offset + 2	I=3 (Three iterations)						

Figure 29. DEMOD Modulation Example

15 LMS1–Complex Coefficients, Complex Samples, and Real/Complex Scalar

The LMS1 implements a basic FIR filter coefficients update. The coefficients and input samples are complex numbers, but the scalar is a real or complex number. Figure 30 shows the LMS1 function.

$$C^{n+1}_i = C^n_i + E \times X_{n-i}$$

Figure 30. LMS1 Function

15.1 Coefficients and Input Buffers

The coefficient vector occupies K pairs of 16-bit entries (real and imaginary components) in memory, with C(0) stored in the first location. The input sample buffer is a circular buffer that contain (M+1) bytes. Each sample is a pair of 16-bit entries (real and imaginary components). The next sample is stored in the address that follows the previous sample.

Table 23. LMS1 Coefficients and Input Buffers

Coefficients	Input Samples
imaginary{C(0)}	*
real{C(0)}	*
imaginary{C(1)}	imaginary{X(n–k+1)}
real{C(1)}	real{X(n–k+1)}
*	*
*	*
imaginary{C(k–1)}	imaginary{X(n–2)}
real{C(k–1)}	real{X(n–2)}
	imaginary{X(n–1)}
	real{X(n–1)}
	imaginary{X(n)}
	real{X(n)}

15.2 LMS1 Function Descriptor

The LMS1 function descriptor is shown in Figure 31.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	S	—	W	I	X	—	INDEX		—	—	—	01010				
Offset + 0x2	—															
Offset + 0x4	K															
Offset + 0x6	CBASE															
Offset + 0x8	M															
Offset + 0xA	XPTR															
Offset + 0xC	EPTR															
Offset + 0xE	—															

Figure 31. LMS1 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The LMS1 parameter packet consists of seven 16-bit entries and is described in Table 24.

Table 24. DEMOD Parameter Packet

Address	Name	Description
Hword 1	—	Reserved
Hword 2	**K**	Number_of_taps – 1
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XPTR**	Pointer to input buffer pointer
Hword 6	**EPTR**	Pointer to scalar
Hword 7	—	Reserved

15.3 LMS1 Applications

The LMS1 is used for updating the coefficients for echo cancellation.

16 LMS2–Complex Coefficients, Complex Samples, and Real/Complex Scalar

The LMS2 implements a basic FIR filter coefficients update. The sample pointer is incremented by two, which is required for fractionally spaced equalizer updates. The coefficients and input samples are complex numbers, but the scalar is a real or complex number. Figure 32 shows the LMS2 function.

$$C^{n+1}_i = C^n_i + E \times X_{n-i}$$

Figure 32. LMS2 Function

16.1 LMS2 Coefficients and Input Buffers

The coefficient vector occupies K pairs of 16-bit entries (real and imaginary components) in memory, with C(0) stored in the first location. The sample input buffer is a circular buffer containing (M+1) bytes. Each sample is a pair of 16-bit entries (real and imaginary components). The next sample is stored in the address that follows the previous sample.

Table 25. LMS2 Coefficients and Input Buffers

Coefficients	Input Samples
imaginary{C(0)}	*
real{C(0)}	*
imaginary{C(1)}	imaginary{X(n–k+1)}
real{C(1)}	real{X(n–k+1)}
*	*
*	*
imaginary{C(k–1)}	imaginary{X(n–2)}
real{C(k–1)}	real{X(n–2)}
	imaginary{X(n–1)}
	real{X(n–1)}
	imaginary{X(n)}
	real{X(n)}

16.2 LMS2 Function Descriptor

The LMS2 function descriptor is shown in Figure 33.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	S	—	W	I	X	—	INDEX		—	—	—	01011				
Offset + 0x2	—															
Offset + 0x4	K															
Offset + 0x6	CBASE															
Offset + 0x8	M															
Offset + 0xA	XPTR															
Offset + 0xC	EPTR															
Offset + 0xE	—															

Figure 33. LMS2 Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The LMS2 parameter packet consists of seven 16-bit entries and is described in Table 26.

Table 26. LMS2 Parameter Packet

Address	Name	Description
Hword 1	—	Reserved
Hword 2	**K**	Number_of_taps – 1
Hword 3	**CBASE**	Filter coefficient vector base address
Hword 4	**M**	Input buffer_size – 1. The minimum input buffer size is 8 (2 samples).
Hword 5	**XPTR**	Pointer to input buffer pointer
Hword 6	**EPTR**	Pointer to scalar
Hword 7	—	Reserved

16.3 LMS2 Applications

The LMS2 is used in updating fractionally spaced equalizer coefficients.

17 Weighted Vector Addition (WADD)–Real X and Real Y

Using two real, scalar coefficients α and β, the WADD function generates a linear combination of two real input vectors. Figure 34 shows the WADD output vector Y as a function of the two input vectors X_1 and X_2.

$$Y(n) = \alpha X_1(n) + \beta X_2(n)$$

Figure 34. WADD Function

17.1 WADD Coefficients and Input Buffers

Each input vector is stored in a circular buffer containing (M+1) bytes. Each sample is a 16-bit entry, and the next sample is stored in the address that follows the previous sample. The output buffer is a circular buffer that contains (N+1) bytes. Each output is 16 bits, and the newest output is stored in the address that follows the previous one.

Table 27. WADD Modulation Table and Sample Data Buffers

X_1 Input Samples	X_2 Input Samples	Output
*	*	*
$x_1(n–k+1)$	$x_2(n–k+1)$	Y(n–k+1)
*	*	*
*	*	*
$x_1(n–1)$	$x_2(n–1)$	Y(n–1)
$x_1(n)$	$x_2(n)$	Y(n)

17.2 WADD Function Descriptor

The WADD function descriptor is shown in Figure 35.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	**S**	—	**W**	**I**	**0**							**01100**				
Offset + 0x2	**I**															
Offset + 0x4	α															
Offset + 0x6	β															
Offset + 0x8	**M**															
Offset + 0xA	**XYPTR**															
Offset + 0xC	**N**															
Offset + 0xE	—															

Figure 35. WADD Function Descriptor

The status and control bits (at offset 0x00) are described in Table 2. The WADD parameter packet consists of seven 16-bit entries and is described in Table 28.

Table 28. WADD Parameter Packet

Address	Name	Description
Hword 1	**I**	Number of iterations
Hword 2	α	X_1 weight coefficient
Hword 3	β	X_2 weight coefficient
Hword 4	**M**	Sample's buffer_size – 1
Hword 5	**XYPTR**	Pointer to a structure composed of X_1 input buffer pointer, the output buffer pointer, and the X_2 input buffer pointer.
Hword 6	**N**	Output buffer_size – 1
Hword 7	—	Reserved

17.3 WADD Applications

Table 29 shows the linear functions available using different α and β values.

Table 29. WADD Applications

α	β	Function
$0 \le \alpha \le 1$	$1-\alpha$	Linear interpolation
α	0	$y(n)=\alpha x(n)$ scalar multiply
1	-1	$y(n)=x_1(n)-x_2(n)$ vector subtract

18 DSP Performance Using the Core Alone Versus Using the CPM

A DSP task on the MPC885 can be done with the core alone or with the help of the built-in DSP capabilities of the CPM. A V.32 modem's Tx data pump flow compares DSP performance using the core alone versus using the CPM. Figure 36 shows the Tx filter example application, which consists of three subfilters.

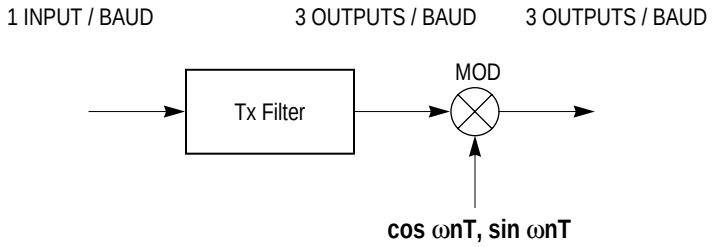


Figure 36. Example DSP Application—Tx Filter

18.1 Tx Filter Example (Core Only)

Implementing the filter using the following C code on the core takes 476 instructions—371 for the filter and 105 for the modulation. The transmission symbol rate requires running the filter 2,400 times a second. Thus, implementing the filter in software alone requires the core to execute 1.14 million instructions per second.

```
void tx_filter ()
{
   S16 *coefr
   S16 *samplr, *sampli
   S16 *coefend;
   S32 filtoutr, filtouti;
   U8 subcount, sampleindex;
   extern S16 mult(S16 p1, S16 p2);     /* in-line invocation */

   coefr=txfiltcoef_str;
   coefend=txfiltcoef_end;
   samplr=&txfiltdly[REAL][txfiltptr];
   sampli=&txfiltdly[IMAG][txfiltptr];
   sampleindex=0;
   while (coefr<coefend) {
      filtoutr=filtouti=0;
      subcount=0;
```

```
      while (subcount<TXSUBFILTLEN) {
         filtoutr+=mult(*coefr, *samplr--);
         filtouti+=mult(*coefr++, *sampli--);
      }
      samplr=&txfiltdly[REAL][txfiltptr];
      sampli=&txfiltdly[IMAG][txfiltptr];
      modbuff[REAL][sampleindex]= filtoutr;
      modbuff[IMAG][sampleindex++]= filtouti;
   }
}

void modulator ()
{

   U8 i;
   S32 termrnd;
   extern S16 mult(S16 p1, S16 p2);     /* in-line invocation */

   i=0;
   while (i<SAMPLE_PER_T) {
      sigout[i]= mult(sn1800[REAL][cosindx], modbuf[REAL][i]) -
               mult(sn1800[IMAG][cosindx], modbuf[IMAG][i]);
      cosindx++;
      if (cosindx==SIN1800TBL_LEN)cosindx=0;
      i++;
   }

void main ()
{
        *
        *
   tx_filter();
```

```
    modulator();
          *
          *
}
```

18.2 Tx Filter Example (Core and CPM)

Implementing the filter using the CPM functions, the user software builds a static FD chain of two functions—an FIR and a MOD. The core activates the CPM to execute the chain with a single write to the CP command register—START DSP. The CPM then signals completion using an interrupt.

The performance load on the core from executing the filter software is negligible. The performance load on the CPM is based on the functions called, the number of clocks required to perform those functions, and the transmission symbol rate. Using the CPM, this filter example consumes 0.55 million clocks per second.

The filter executes three subfilters each time a new sample arrives, invoking the FIR2 function with a three-iteration count and auto-increment of the input sample pointer after the last iteration. FIR2 writes the three subfilter results into the output buffer, which then feeds into the modulation. Modulation invokes the MOD function with a three-iteration count. The MOD function automatically increments the sample pointer on each iteration. Figure 37 shows a conceptual view of the filter implementation followed by example code.

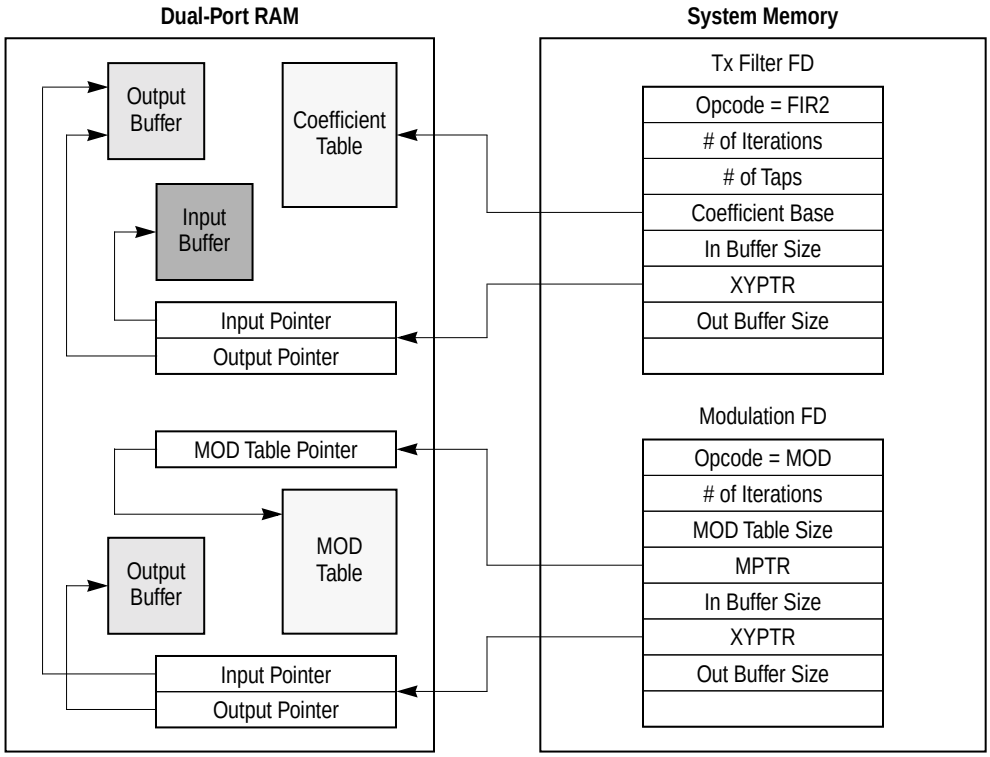


Figure 37. Core and CPM Implementation of Filter Example

```
/* Function Descriptors */
typedef struct dsp_fd {
        unsigned short  status;
        unsigned short  parameter[7];
} DSP_FD;

#define WRAP    0x2000          /* wrap bit */
#define INTR    0x1000          /* interrupt on completion */


/* define for function opcodes */
#define FIR_2   0x0102  /* FIR2  filter */
#define MOD     0x0008  /* Modulation function opcode */

/* Initialize a static fd chain of 2 functions */
DSP_FD  filters[2]= {
                 { FIR_2,P11,P12, …, P17}
                ,{(WRAP | INTR | MOD),P21,P22, …, P27}
};

void main()
{
        *
        *
        *
        /* issue command to CP to start processing the FD chain */
        issue_command( START_FD );
        *
        *
        *
}
```

19 DSP Function Execution Times and CPM Performance Calculation

A DSP function's execution time is directly related to the number of taps and iterations specified. Table 30 lists the execution times for each function, including overhead for context switching, handling the FD, and initialization.

Table 30. DSP Function Execution Times

Function	Execution Time
FIR1	53 + 20 · (i–1) + 1.25 · i · (k+1)
FIR2	47 + 17 · (i–1) + 3 · i · (k+1)
FIR3	44 + 14 · (i–1) + 4 · i · (k+1)
FIR5	44 + 14 · (i–1) + 5 · i · (k+1)
FIR6	50 + 20 · (i–1) + 3 · i · (k+1)
IIR	44 + 11 · i
MOD	44 + 7 · i
DEMOD	47 + 14 · i
LMS1	42 + 7 · (k+1)
LMS2	42 + 7 · (k+1)
WADD	46 + 7 · i

Notes:
Add 1 clock for wrap, 5 clocks for stop, and 4 clocks for interrupt.
i = number of iterations.
k+1 = number of taps.

As seen in Table 30, the CPM loading from DSP applications depends on which functions are called and their parameters. The frequency with which the functions are called also affects CPM loading.

20 Document Revision History

Table 31 provides a revision history for this addendum.

Table 31. Document Revision History

Revision Number	Significant Changes
0.1	Initial release

Home Page:
www.freescale.com
email:
support@freescale.com
USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com
Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, 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
+81 2666 8080
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
support.asia@freescale.com
For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Freescale Semiconductor, Inc.