

Exhibit 1006

Implementation of FIR/IIR Filters with the TMS32010/TMS32020

APPLICATION REPORT: SPRA003A

*Authors: Al Lovrich and Ray Simar, Jr.
Digital Signal Processing – Semiconductor Group*

*Digital Signal Processing Solutions
1989*



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1997, Texas Instruments Incorporated

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

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

Implementation of FIR/IIR Filters with the TMS32010/TMS32020

Abstract

This report discusses the implementation of Finite Impulse Response (FIR)/Infinite Impulse Response (IIR) filters using the TMS32010 and TMS32020. Filters designed with designed processors, such as the TMS320, are superior over their analog counterparts for better specifications, stability, performance, and reproducibility. This report describes a variety of methods for implementing FIR/IIR filters using the TMS320. The TMS320 algorithm execution time and data memory requirements are considered. Tradeoffs between several different filter structures are also discussed. This application report compliments the Digital Filter Design Package (DFDP) discussed in Section 2.



Product Support on the World Wide Web

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

INTRODUCTION

In many signal processing applications, it is advantageous to use digital filters in place of analog filters. Digital filters can meet tight specifications on magnitude and phase characteristics and eliminate voltage drift, temperature drift, and noise problems associated with analog filter components.

This application report describes a variety of methods for implementing Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters with the TMS320 family of digital signal processors. Emphasis is on minimizing both the execution time and the number of data memory locations required. Tradeoffs between several different structures of the two classes of digital filters are also discussed.

In this report, TMS320 source code examples are included for the implementation of two FIR filters and three IIR filters based on the techniques presented. Plots of magnitude response, log-magnitude response, unit-sample response, and other pertinent data accompany each of the filter implementations. Important performance considerations in digital filter design are also included. The methods presented for implementing the different types of filters can be readily extended to any desired order of filters.

Readers are assumed to have some familiarity with the basic concepts of digital signal processing theory.¹ The notation used in this report is consistent with that used in reference [1].

FILTERING WITH THE TMS320 FAMILY

Almost every field of science and engineering, such as acoustics, physics, telecommunications, data communications, control systems, and radar, deal with signals. In many applications, it is desirable that the frequency spectrum of a signal be modified, reshaped, or manipulated according to a desired specification. The process may include attenuating a range of frequency components and rejecting or isolating one specific frequency component.

Any system or network that exhibits such frequency-selective characteristics is called a filter. Several types of filters can be identified: lowpass filter (LPF) that passes only "low" frequencies, highpass filter (HPF) that passes "high" frequencies, bandpass filter (BPF) that passes a "band" of frequencies, and band-reject filter that rejects certain frequencies. Filters are used in a variety of applications, such as removing noise from a signal, removing signal distortion due to the transmission channel, separating two or more distinct signals that were mixed in order to maximize communication channel utilization, demodulating signals, and converting discrete-time signals into continuous-time signals.

Advantages of Digital Filtering

The term "digital filter" refers to the computational process or algorithm by which a digital signal or sequence of numbers (acting as input) is transformed into a second sequence of numbers termed the output digital signal. Digital filters involve signals in the digital domain (discrete-time signals), whereas analog filters relate signals in the analog domain (continuous-time signals). Digital filters are used extensively in applications, such as digital image processing, pattern recognition, and spectrum analysis. A band-limited continuous-time signal can be converted to a discrete-time signal by means of sampling. After processing, the discrete-time signal can be converted back to a continuous-time signal. Some of the advantages of using digital filters over their analog counterparts are:

1. High reliability
2. High accuracy
3. No effect of component drift on system performance
4. Component tolerances not critical.

Another important advantage of digital filters when implemented with a programmable processor such as the TMS320 is the ease of changing filter parameters to modify the filter characteristics. This feature allows the design engineer to effectively and easily upgrade or update the characteristics of the designed filter due to changes in the application environment.

Design of Digital Filters

The design of digital filters involves execution of the following steps:

1. Approximation
2. Realization
3. Study of arithmetic errors
4. Implementation.

Approximation is the process of generating a transfer function that satisfies a set of desired specifications, which may involve the time-domain response, frequency-domain response, or some combination of both responses of the filter.

Realization consists of the conversion of the desired transfer function into filter networks. Realization can be accomplished by using several network structures,^{2,3} as listed below. Some of these structures are covered in detail in this report.

1. Direct
2. Direct canonic (direct-form II)
3. Cascade
4. Parallel
5. Wave⁴
6. Ladder.

Approximation and realization assume an infinite-precision device for implementation. However,

implementation is concerned with the actual hardware circuit or software coding of the filter using a programmable processor. Since practical devices are of finite precision, it is necessary to study the effects of arithmetic errors on the filter response.

TMS320 Digital Signal Processors

Digital Signal Processing (DSP) is concerned with the representation of signals (and the information they contain) by sequences of numbers and with the transformation or processing of such signal representations by numeric-computational procedures. In the past, digital filters were implemented in software using mini- or main-frame computers for non-realtime operation or on specialized dedicated digital hardware for realtime processing of signals.

The recent advances in VLSI technology have resulted in the integration of these digital signal processing systems into small integrated circuits (ICs), such as the TMS320 family of digital signal processors from Texas Instruments. The TMS320 implementation of digital filters allows the filter to operate on realtime signals. This method combines the ease and flexibility of the software implementation of filters with reliable digital hardware. To further ease the design task, it is now possible for engineers to design and test filters using any one of the commercially available filter design packages, some of which create TMS320 code and decrease the design time.

The Texas Instruments TMS320 digital signal processing family contains two generations of digital signal processors. The TMS32010, the first-generation digital signal processor,⁵ implements in hardware many functions that other processors typically perform in software. Some of the key features of the TMS32010 are:

- 200-ns instruction cycle
- 1.5K words (3K bytes) program ROM
- 144 words (288 bytes) data RAM
- External memory expansion to 4K words (8K bytes) at full speed
- 16 x 16-bit parallel multiplier
- Interrupt with context save
- Two parallel shifters
- On-chip clock
- Single 5-volt supply, NMOS technology, 40-pin DIP.

The TMS32020 is the second-generation processor⁶ in the TMS320 DSP family. To maintain device compatibility, the TMS32020 architecture is based upon that of the TMS32010, the first member of the family, with emphasis on overall speed, communication, and flexibility in processor configuration. Some of the key features of the TMS32020 are:

- 544 words of on-chip data RAM, 256 words of which may be programmed as either data or program memory
- 128K words of data/program space
- Single-cycle multiply/accumulate instructions

- TMS32010 software upward compatibility
- 200-ns instruction cycle
- Sixteen input and sixteen output channels
- 16-bit parallel interface
- Directly accessible external data memory space
- Global data memory interface for multiprocessing
- Instruction set support for floating-point operations
- Block moves for data/program memory
- Serial port for multiprocessing or codec interface
- On-chip clock
- Single 5-volt supply, NMOS technology, 68-pin grid array package.

Because of their computational power, high I/O throughput, and realtime programming, the TMS320 processors have been widely adapted in telecommunication, data communication, and computer applications. In addition to the above features, the TMS320 has efficient DSP-oriented instructions and complete hardware/software development tools, thus making the TMS320 highly suitable for DSP applications.

DIGITAL FILTER IMPLEMENTATION ON THE TMS320

For a large variety of applications, digital filters are usually based on the following relationship between the filter input sequence $x(n)$ and the filter output sequence $y(n)$:

$$y(n) = \sum_{k=0}^N a_k y(n-k) + \sum_{k=0}^M b_k x(n-k) \quad (1)$$

Equation (1) is referred to as a linear constant-coefficient difference equation. Two classes of filters can be represented by linear constant-coefficient difference equations:

1. Finite Impulse Response (FIR) filters, and
2. Infinite Impulse Response (IIR) filters.

The following sections describe the implementation of these classes of filters on the TMS32010 and TMS32020.

FIR Filters

For FIR filters, all of the a_k in (1) are zero. Therefore, (1) reduces to

$$y(n) = \sum_{k=0}^M b_k x(n-k) \quad (2)$$

where $(M + 1)$ is the length of the filter.

As a result, the output of the FIR filter is simply a finite-length weighted sum of the present and previous inputs to the filter. If the unit-sample response of the filter is denoted

as $h(n)$, then from (2), it is seen that $h(n) = b(n)$. Therefore, (2) is sometimes written as

$$y(n) = \sum_{k=0}^M h(k)x(n-k) \quad (3)$$

From (3), it can be seen that an FIR filter has, as the name implies, a finite-length response to a unit sample. Denoting the z transforms of $x(n)$, $y(n)$, and $h(n)$ as $X(z)$, $Y(z)$, and $H(z)$, respectively, then

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^M b_k z^{-k} = \sum_{k=0}^M h(k)z^{-k} \quad (4)$$

Equations (3) and (4) may also be represented by the network structure shown in Figure 1. This structure is referred to as a direct-form realization of an FIR filter, because the filter coefficients can be identified directly from the difference equation (3). The branches labeled with z^{-1} in Figure 1 correspond to the delays in (3) and the multiplications by z^{-1} in (4). Equation (3) may be implemented in a straightforward and efficient manner on a TMS320 processor.

TMS32010 Implementation of FIR Filters

Figure 2 gives an example of a length-5 direct-form FIR filter, and Figure 3 shows a portion of the TMS32010 code for implementing this filter.

The notation developed in this section will be used throughout this application report. XN corresponds to $x(n)$, XNM1 corresponds to $x(n-1)$, etc.

In the above implementation, the following three basic and important concepts for the implementation of FIR filters on the TMS320 should be understood:

1. The relationship between the unit-sample response of an FIR filter and the filter structure,
2. The power of the LTD and MPY instruction pair for this implementation, and
3. The ordering of the input samples in the data memory of the TMS320, which is critical for realtime signal processing.

The input sequence $x(n)$ is stored as shown in Figure 4. In general, each of the multiplies and shifts of $x(n)$ in (3) is implemented with an instruction pair of the form

```
LTD  XNM1
MPY  H1
```

The instruction LTD XNM1 loads the T register with the contents of address XNM1, adds the result of the previous multiply to the accumulator, and shifts the data at address XNM1 to the next higher address in data memory. Using the storage scheme in Figure 4, this corresponds to shifting the data at address XNM1 to address XNM2. The instruction MPY H1 multiplies the contents of the T register with the contents of address H1. The shifting is the reason for the storage scheme used in Figure 4. This scheme, critical for realtime digital signal processing, makes certain that the input sequence $x(n)$ is in the correct location for the next pass through the filter.

By comparing (3) with the code in Figure 3, the reason for the ordering of the data and the importance of the shift implemented by the LTD instruction can be seen. To better

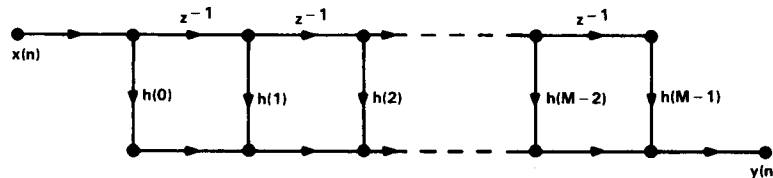


Figure 1. Direct-Form FIR Filter

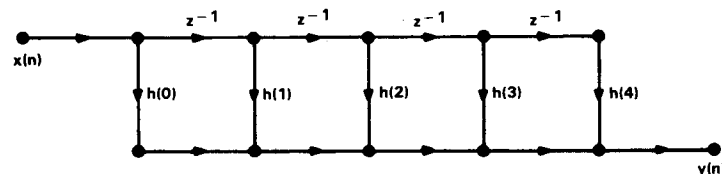


Figure 2. Length-5 Direct-Form FIR Filter

```

* THIS SECTION OF CODE IMPLEMENTS THE FOLLOWING EQUATION: *
* x(n-4)h(4) + x(n-3)h(3) + x(n-2)h(2) + x(n-1)h(1) + x(n)h(0) = y(n) *
*
NXTPT  IN XN,PA2      * GET THE NEW INPUT VALUE XN FROM PORT PA0 *
*
      ZAC              * ZERO THE ACCUMULATOR *
*
      LT XNM4          * x(n-4)h(4) *
      MPY H4
*
      LTD XNM3        * x(n-4)h(4) + x(n-3)h(3) *
      MPY H3
*
      LTD XNM2        * SIMILAR TO THE PREVIOUS STEPS *
      MPY H2
*
      LTD XNM1
      MPY H1
*
      LTD XN
      MPY H0
*
      APAC            * ADD THE RESULT OF THE LAST MULTIPLY TO *
*                     * THE ACCUMULATOR *
*
      SACH YN,1       * STORE THE RESULT IN YN *
*
      OUT YN,PA2      * OUTPUT THE RESPONSE TO PORT PA1 *
*
      B NXTPT        * GO GET THE NEXT POINT *

```

Figure 3. TMS32010 Code for Implementing a Length-5 FIR Filter

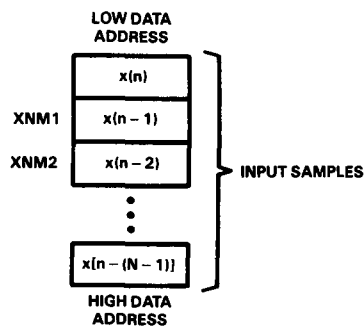


Figure 4. TMS32010 Input Sample Storage for a Length-N FIR Filter

understand the algorithm, the relationship between the input and output of the filter must be considered. Evaluating (3) for a particular value of n , for example, n_0 , yields

$$y(n_0) = \sum_{k=0}^{N-1} h(k) x(n_0 - k) \quad (5)$$

If the next sample of the filter response $y(n_0 + 1)$ is needed, it is seen from (3) that

$$y(n_0 + 1) = \sum_{k=0}^{N-1} h(k) x(n_0 + 1 - k) \quad (6)$$

Equations (5) and (6) show that the samples of $x(n)$ associated with particular values of $h(k)$ in (5) have been shifted to the left (i.e., to a higher data address) by one in (6). This shifting of the input data, illustrated in Figure 5, corresponds to the shifting of the flipped input sequence in relation to the unit-sample response.

Depending on the system constraints, the designer may choose to reduce program memory size by taking advantage of indirect addressing capability provided by the TMS32010. Using either of the auxiliary registers along with the autoincrement or autodecrement feature, the FIR filter program can be rewritten in looped form as shown in Figure 6.

The input sequence $x(n)$ is stored as shown in Figure 4, and the impulse response $h(n)$ is stored as shown in Figure 7. In the looped version, the indirect addressing mode is used with the autodecrement feature and BANZ instruction to control the looping and address generation for data access. While the looped code requires less program memory than the straightline version, the straightline version runs more quickly than the looped code because of the overhead associated with loop control. This design tradeoff should be carefully considered by the design engineer.

It is also possible to use the LTD/MPYK instruction pair to implement each filter tap in straightline code. The MPYK instruction is used to multiply the contents of the T register by a signed 13-bit constant stored in the MPYK instruction word. For many applications, a 13-bit coefficient can adequately implement the filter without significant changes to the filter response. An advantage of using this approach is that the coefficients are stored in program memory and there is no need to transfer them to data memory. This reduces the amount of data memory locations required per filter tap from two to one.

The length-80 FIR filter program in Appendix A implements a linear-phase FIR filter in straightline code. The unit-sample response of the filter is symmetric in order to achieve linear phase. Because of the symmetry, it is necessary to store only 40 (rather than 80) of the samples of the impulse response. This symmetry can often be used to a designer's advantage since it significantly reduces the amount of storage space required to implement the filter.

In summary, by taking advantage of the TMS32010 features, a designer can implement a direct-form FIR filter, optimized for execution time, data memory, or program memory.

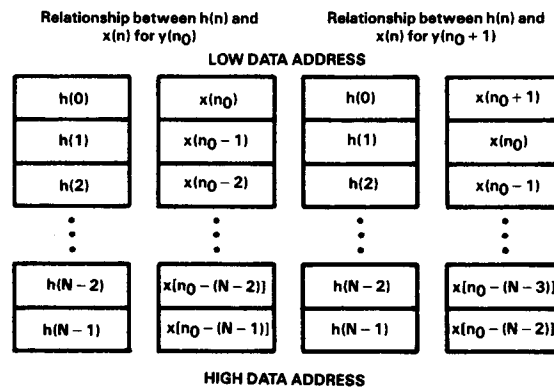


Figure 5. Relationship Between the Contents of Data Registers

```

* THIS SECTION OF CODE IMPLEMENTS THE EQUATION: *
*  $x(n-(N-1))h(N-1) + x(n-(N-2))h(N-2) + \dots + x(n)h(0) = y(n)$  *
*
*      LARP ARO      * AUXILIARY REGISTER POINTER SET TO ARO *
*
* NXTPT  IN XN,PA2   * PULL IN NEW INPUT FROM PORT PA0 *
*
*      LARK ARO,XNMN1 * ARO POINTS TO X(n-(N-1)) *
*      LARK AR1,HN1   * AR1 POINTS TO H(N-1) *
*
*      ZAC           * ZERO THE ACCUMULATOR *
*
*      LT *- ,AR1    *  $x(n-(N-1))h(N-1)$  *
*      MPY *- ,ARO
*
* LOOP   LTD * ,AR1  *  $x(n-(N-1))h(N-1) + x(n-(N-2))h(N-2) + \dots + x(n)h(0) = y(n)$  *
*      MPY *- ,ARO
*
*      BANZ LOOP     * IF ARO DOES NOT EQUAL ZERO, *
*                   * THEN DECREMENT ARO AND BRANCH TO LOOP *
*
*      APAC          * ADD THE P REGISTER TO THE ACCUMULATOR *
*
*      SACH YN,1     * STORE THE RESULT IN YN *
*
*      OUT YN,PA2    * OUTPUT THE RESPONSE TO PORT PA1 *
*
*      B NXTPT      * GO GET THE NEXT INPUT POINT *

```

Figure 6. TMS32010 Code for Implementing a Looped FIR Filter

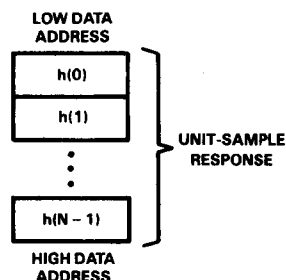


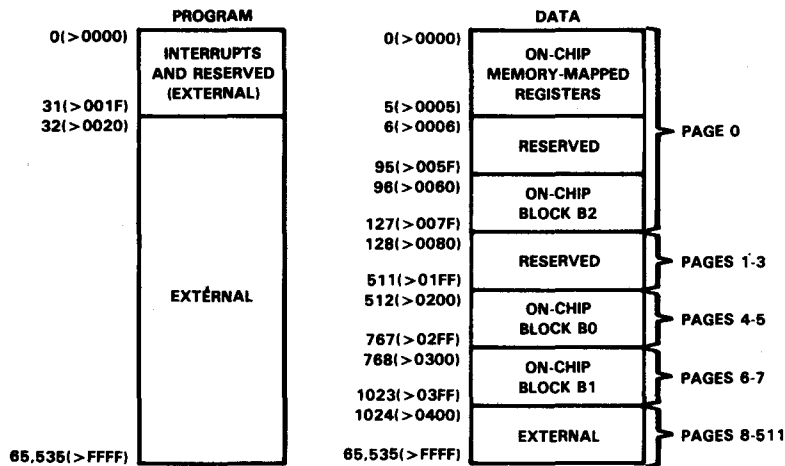
Figure 7. TMS32010 Unit-Sample Response Storage for a Looped FIR Filter

TMS32020 Implementation of FIR Filters

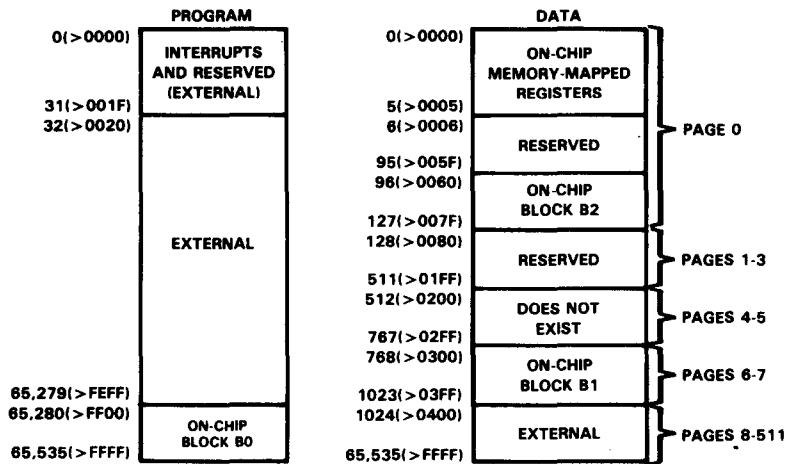
In many DSP applications, realtime processing of signals is very critical. Important choices must be made in selecting a DSP device capable of realtime filtering. For example, in a speech application, a sampling rate of 8 kHz is common, which corresponds to an interval of 125 μ s between consecutive samples. This interval is the maximum

allowable time for realtime operation, corresponding to 625 cycles on the TMS32010. In order to perform the required signal processing tasks in that interval, it is essential to reduce filter execution time. This can be accomplished by a single-cycle multiply/accumulate instruction. The TMS32020, the second-generation DSP device, is a processor with such a capability. A single-cycle multiply/accumulate with data-move instruction and larger on-chip RAM make it possible to implement each filter tap in approximately 200 ns.

The TMS32020 provides a total of 544 16-bit words of on-chip RAM, divided into three separate blocks of B0, B1, and B2. Of the 544 words, 288 words (blocks B1 and B2) are always data memory, and 256 words (block B0) are programmable as either data or program memory. The CNFD (configure block B0 as data memory) and CNFP (configure block B0 as program memory) instructions allow dynamic configuration of the memory maps through software, as illustrated in Figure 8. After execution of the CNFP instruction, block B0 is mapped into program memory, beginning with address 65280. To take advantage of the MACD (multiply and accumulate with data move) instruction, block B0 must be configured as program memory using the CNFP instruction. MACD only works with on-chip RAM. The use of the MACD instruction helps to speed



(a) ADDRESS MAPS AFTER A CNFD INSTRUCTION



(b) ADDRESS MAPS AFTER A CNFP INSTRUCTION

Figure 8. TMS32020 Memory Maps

the filter execution and allows the size of the FIR filter to expand to 256 taps.⁶

The TMS32020 implementation of (3) is made even more efficient with a repeat instruction, RPTK. It forms a useful instruction pair with MACD, such as

RPTK	NM1
MACD	(PMA),(DMA)

The RPTK NM1 instruction loads an immediate 8-bit value N-1 into the repeat counter. This causes the next instruction to be executed N times (N = the length of the filter). The instruction MACD (PMA),(DMA) performs the following functions:

1. Loads the program counter with PMA,
2. Multiplies the value in data memory location DMA (on-chip, block B1) by the

- value in program memory location PMA (on-chip, block B0),
3. Adds the previous product to the accumulator,
 4. Copies the data memory value (block B0) to the next higher on-chip RAM location. The data move is the mechanism by which the z^{-1} delay can be implemented, and
 5. Increments the program counter with each multiply/accumulate to point to the next sample of the unit-sample response.

In other words, the MACD instruction combines the LTD/MPY instruction pair into one. With the proper storage of the input samples and the filter unit-sample response, one can take advantage of the power of the MACD instruction. Figure 9 is a data storage scheme that provides the correct sequence of inputs for the next pass through the filter.

In the TMS32020 code example of Figure 10, data memory values are accessed indirectly through auxiliary register 1 (AR1) when the MACD instruction is implemented. For low-order filters (second-order), using the MACD instruction in conjunction with the RPTK instruction is less effective due to the overhead associated with the MACD instruction in setting up the repeat construct. To take advantage of the MACD instruction, the filter order must be greater than three. For lower-order filters, it is recommended to use the LTD/MPY instruction pair in place of RPT/MACD.

Writing looped code for the TMS32020 implementation of an FIR filter gives no further advantage. Since the MACD instruction already uses less program memory, looped code in this case does not reduce program memory size. Implementing FIR filters of length-3 or higher requires the same amount of program memory (excluding coefficient

storage). For example, an FIR filter of length-256 takes the same amount of program memory space as a FIR filter of length-4.

Since the TMS32020 instruction set is upward-compatible with the TMS32010 instruction set, it is possible to use the LTD/MPYK instruction pair to implement the filter. With the TMS32020, the designer can use either RPTK/MACD or LTD/MPY(K) where appropriate. Depending on the application and the data memory constraints, the use of the LTD/MPYK instruction pair results in less data memory usage at the cost of increasing the program memory storage.

The FIR filter program of Appendix A is an implementation of the same length-80 FIR filter used in the TMS32010 example. In this implementation, it can be seen that the TMS32020 uses less program memory than the TMS32010 with the tradeoff of using more data memory words. The increase in data memory size is indirectly related to the MACD instruction; i.e., in order to take full advantage of the instruction, it is necessary to keep the multiplier pipeline as busy as possible. Therefore, the filter will execute faster when all 80 coefficients are provided in block B0.

The TMS32020 provides a solution for the faster execution of FIR filters. The combination of the RPTK/MACD instructions provides for a minimum program memory and high-speed execution of an FIR filter. If data memory is a concern, the designer can use the LTD/MPYK instruction pair at the cost of increasing program memory and using 13-bit filter coefficients.

IIR Filters

The concepts introduced for the implementation of FIR filters can be extended to the implementation of IIR filters. However, for an IIR filter, at least one of the a_k in (1) is

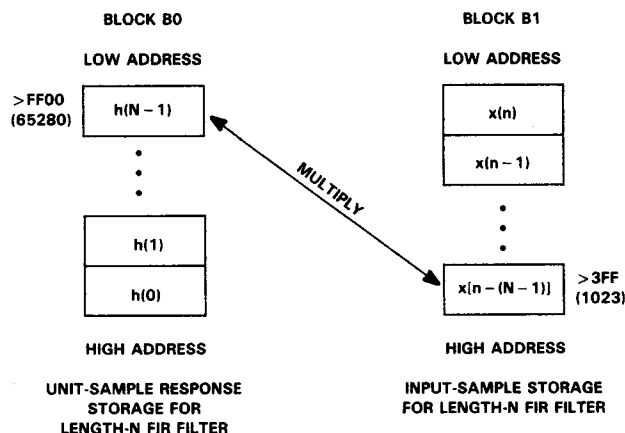


Figure 9. TMS32020 Memory Storage Scheme

```

* THIS SECTION OF CODE IMPLEMENTS THE EQUATION:
* x(n-(N-1))h(N-1) + x(n-(N-2))h(N-2) + ... + x(n)h(0) = y(n)
*
*          CNFP                      * USE BLOCK B0 AS PROGRAM AREA
*
* NXTPT   IN      XN,PA0             * BRING IN THE NEW SAMPLE XN
*
*          LRLK   AR1,>3FF           * POINT TO THE BOTTOM OF BLOCK B1
*          LARP   AR1
*
*          MPYK   0                   * SET P REGISTER TO ZERO
*          ZAC                                * CLEAR THE ACCUMULATOR
*
*          RPTK   NM1                 * REPEAT N-1 TIMES
*          MACD   >FF00,*-           * MULTIPLY/ACCUMULATE
*
*          APAC
*          SACH   YN,1
*
*          OUT    YN,PA1             * OUTPUT THE FILTER RESPONSE y(n)
*
*          B      NXTPT              * GET THE NEXT POINT

```

Figure 10. TMS32020 Code for Implementing a Length-5 FIR Filter

nonzero. It has been shown¹ that the z transform of the unit-sample response of an IIR filter corresponding to (1) is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^M b_k z^{-k}}{1 - \sum_{k=1}^N a_k z^{-k}} \quad (7)$$

where $H(z)$, $Y(z)$, and $X(z)$ are the z transforms of $h(n)$, $y(n)$, and $x(n)$, respectively. Three different network structures often used to implement (7) are the direct form, the cascade form, and the parallel form. Implementation of these structures is discussed in the following sections.

Direct-Form IIR Filter

Equations (1) and (7) may also be represented by the network structure shown in Figure 11. For convenience, it is assumed that $M = N$. This network structure is referred to as the direct-form I realization of an Nth-order difference equation. As was the case for the direct-form FIR filter, the structure in Figure 11 is called direct-form since the coefficients of the network can be obtained directly from the difference equation describing the network. Again, the branches associated with the z^{-1} correspond to the delays in (1) and the multiplications in (7).

The following difference equation:

$$y(n) = \sum_{k=1}^N a_k y(n-k) + \sum_{k=0}^M b_k x(n-k) \quad (8)$$

shows that the output of the filter is a weighted sum of past values of the input to the filter and of the output of the filter. Using techniques similar to those for an FIR filter, this realization can be implemented in a straightforward and efficient way on the TMS32010 and TMS32020.

A network flowgraph equivalent to that in Figure 11 is shown in Figure 12. This system is referred to as the direct-form II structure. Since the direct-form II has the minimum number of delays (branches labeled z^{-1}), it requires the minimum number of storage registers for computation. This structure is advantageous for minimizing the amount of data memory used in the implementation of IIR filters.

In Figures 13 through 17, a second-order direct-form II IIR filter is used as an example for the TMS320 implementation of the IIR filter. The network structure is shown in Figure 13.

The difference equation for this network is

$$\begin{aligned} d(n) &= x(n) + a_1 d(n-1) + a_2 d(n-2) \\ y(n) &= b_0 d(n) + b_1 d(n-1) + b_2 d(n-2) \end{aligned} \quad (9)$$

In this case, $d(n)$, shown in (9) and Figure 13, corresponds to the network value at the different delay nodes. The zero-delay register corresponds to $d(n)$; $d(n-1)$ is the register for the delay of one; and $d(n-2)$ is the register for the delay of two. A portion of the TMS32010 code necessary to implement (9) is shown in Figure 14. Initially all $d(n-i)$ for $i=0,1,2$ are set to zero.

The delay-node values of the filter are stored in data memory as shown in Figure 15. At each major step of the algorithm, a multiply is done, and the result from the previous multiply is added to the accumulator. Also, the past delay-node values are shifted to the next higher location in

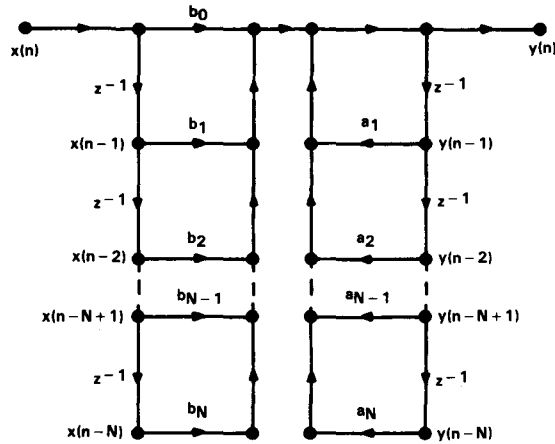


Figure 11. Direct-Form I IIR Filter

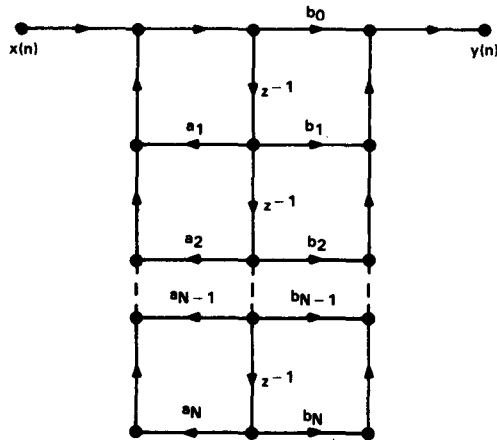


Figure 12. Direct-Form II IIR Filter

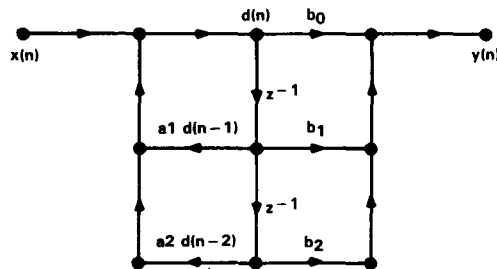


Figure 13. Second-Order Direct-Form II IIR Filter

data memory, thus placing them in the correct position for the next pass through the filter. All of these operations are carried out with instruction pairs, such as

```
LTD  DNM1
MPY  B1
```

where DNM1 corresponds to $d(n-1)$ and B1 corresponds to b_1 as in (9).

When the last multiplication is performed and the result is added to the accumulator, the accumulator contains the result of (9), which is $y(n)$. From (9) and Figure 13, it is evident that the delay-node value $d(n)$ depends on several of the previous delay-node values. This feedback is illustrated by the instruction

```
SACH DN,1
```

and the use of the statements

```
LTD  DNM1
.
.
LTD  DN
```

The ordering of the delay-node values, shown in Figure 15, allows for a simple program structure with minimal computations and minimal data locations. It also accommodates the shifting of the delay-node values in a straightforward way. The feedback of DN makes apparent the underlying structure of the direct-form II filter and (10). This form of the algorithm is flexible and can be extended to higher-order direct-form filters in a straightforward way.

```

* THIS SECTION OF CODE IMPLEMENTS THE EQUATIONS: *
*  $d(n) = x(n) + d(n-1)a_1 + d(n-2)a_2$  *
*  $y(n) = d(n)b_0 + d(n-1)b_1 + d(n-2)b_2$  *
*
*
*      IN XN,PA0      * NEW INPUT VALUE XN *
*
*      LAC XN,15      * LOAD ACCUMULATOR WITH XN *
*
*      LT DNM1
*      MPY A1
*
*      LTA DNM2
*      MPY A2
*
*      APAC
*
*      SACH DN,1      *  $d(n) = x(n) + d(n-1)a_1 + d(n-2)a_2$  *
*                      *
*      ZAC
*
*      MPY B2
*
*      LTD DNM1
*      MPY B1
*
*      LTD DN
*      MPY B0
*
*      APAC
*
*      SACH YN,1      *  $y(n) = d(n)b_0 + d(n-1)b_1 + d(n-2)b_2$  *
*                      *
*
*      OUT YN,PA1     * YN IS THE OUTPUT OF THE FILTER *

```

Figure 14. TMS32010 Code for Implementing a Second-Order Direct-Form II IIR Filter

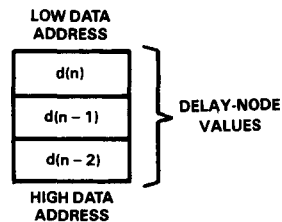


Figure 15. Delay-Node Value Storage for a Second-Order Direct-Form IIR Filter

Figure 16 shows the necessary ordering of the delay-node values for a general direct-form II structure for the case $M \geq N$. Filter order is determined by M or N, whichever is greater.

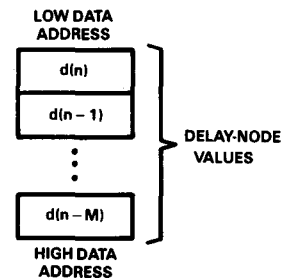


Figure 16. Delay-Node Value Storage for a Direct-Form II IIR Filter

Figure 17 shows a portion of the TMS32020 code for implementing the same second-order direct-form II IIR filter using the MACD instruction. As discussed in the section on FIR filters, using the RPTK/MACD instruction pair is most effective when the filter order is three or higher. The use of the MACD instruction allows the designer to save one word of program memory over the LTD/MPY implementation. The TMS32020 code in Figure 17 is provided only as an example. For a biquad implementation (second-order direct-form II IIR filter), the TMS32010 code and TMS32020 code for the filter implementation are identical. Note that due to larger on-chip RAM of the TMS32020, higher-order IIR filters or sections of IIR filters can be implemented. For the rest of the IIR filter structures, the same discussion applies to both processors.

An example of a TMS32010/TMS32020 program implementing a fourth-order direct-form II structure can be found in Appendix C.

Cascade-Form IIR Filter

In this section, the realization and implementation of cascade-form IIR filters are discussed. The implementation of a cascade-form IIR filter is an extension of the results of the implementation of the direct-form IIR filter.

The z transform of the unit-sample response of an IIR filter

$$H(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 - \sum_{k=1}^N a_k z^{-k}} \quad (10)$$

```
* THIS SECTION OF CODE IMPLEMENTS A SECOND-ORDER DIRECT-FORM II IIR FILTER
* d(n) = x(n) + d(n-1)a1 + d(n-2)a2
* y(n) = d(n)b0 + d(n-1)b1 + d(n-2)b2
*
NEXT    IN    XN,PA2          * NEW INPUT VALUE XN
*
      LAC    XN
      MPYK   0                * CLEAR P REGISTER
*
      LARP   AR1
      LRLK   AR1,>03FF
      CNFP                      * USE BLOCK B0 AS PROGRAM AREA
*
* d(n) = x(n) + d(n-1)a1 + d(n-2)a2
*
      RPTK   1                * REPEAT 2 TIMES
      MACD   >FF00,*+
*
      APAC
      SACH   DN,1              * d(n)
*
* y(n) = d(n)b0 + d(n-1)b1 + d(n-2)b2
*
      ZAC
      MPYK   0                * CLEAR P REGISTER
*
      MPY    >FF02
*
      RPTK   1
      MACD   >FF03,*-
*
      APAC
      SACH   YN,1              * SAVE FILTERED OUTPUT
*
      OUT    YN,PA2            * YN IS THE OUTPUT OF THE FILTER
      B      NEXT
```

Figure 17. TMS32020 Code for Implementing a Second-Order Direct-Form IIR Filter with MACD

may also be written in the equivalent form

$$H(z) = \prod_{k=1}^{N/2} \frac{\beta_{0k} + \beta_{1k}z^{-1} + \beta_{2k}z^{-2}}{1 - \alpha_{1k}z^{-1} - \alpha_{2k}z^{-2}} \quad (11)$$

where the filter is realized as a series of biquads. Therefore, this realization is referred to as the cascade form. Figure 18 shows a fourth-order IIR filter implemented in cascade structure, where the subsections are implemented as direct-form II sections. Each subsection corresponds to one of the terms in the product in (11). Note that any single cascade section is identical to the second-order direct-form II IIR filter described previously.

The difference equation for cascade section i can be written as

$$d_i(n) = y_{i-1}(n) + \alpha_{1i} d_i(n-1) + \alpha_{2i} d_i(n-2) \quad (12)$$

$$y_i(n) = \beta_{0i} d_i(n) + \beta_{1i} d_i(n-1) + \beta_{2i} d_i(n-2)$$

where

$$i = 1, 2, \dots, N/2.$$

$$y_{i-1}(n) = \text{input to section } i.$$

$$d_i(n) = \text{value at a particular delay node in section } i.$$

$$y_i(n) = \text{output of section } i.$$

$$y_0(n) = x(n) = \text{sample input to the filter.}$$

$$y_{N/2} = y(n) = \text{output of the filter.}$$

For the IIR filter consisting of the two cascaded sections shown in Figure 18, there are two sets of equations describing the relationship between the input and output of the filter. The delay-node values for each section are stored as shown in Figure 19. The same indexing scheme used previously

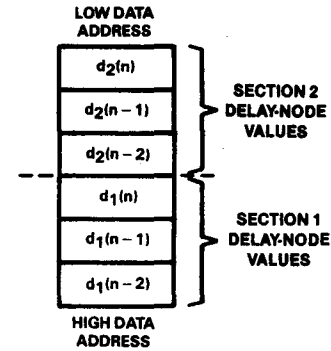


Figure 19. Delay-Node Storage for Cascaded IIR Filter Subsections

is used here (i.e., from the higher address in data memory to the lower address in data memory). In this case, the algorithm can be structured so that the 32-bit accumulator of the TMS320 acts as a storage register and carries the output of one of the second-order subsections to the input of the next second-order subsection. This avoids unnecessary truncation of the intermediate filter values into 16-bit words, and therefore provides better accuracy in the final output.

The implementation of the cascaded fourth-order IIR filter can be summarized as follows:

1. Load the new input value $x(n)$.
2. Operate on the first section as outlined in Figure 12.
3. Leave the output of the first section in the accumulator (i.e., the SACH YN can be omitted for the first-section implementation since the accumulator links the output of one section to the input of the following section).
4. Operate on the second section in the same way as the first section, remembering that

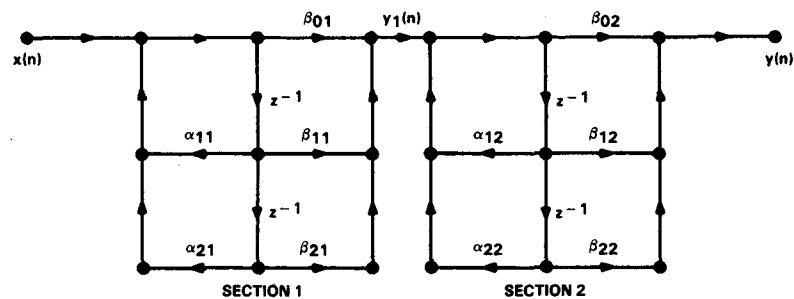


Figure 18. Fourth-Order Cascaded IIR Filter

the accumulator already contains the output of the previous section.

5. The output of the second section is the filter output $y(n)$.

The above procedures can be applied to the IIR filter implementation of higher orders. It can be shown³ that with proper ordering of the second-order cascades, the resulting filter has better immunity to quantization noise than the direct-form implementation, as will be discussed later.

An example of a TMS32010/TMS32020 program that implements a fourth-order IIR cascaded structure is contained in Appendix C.

Parallel-Form IIR Filter

The third form of an IIR filter is referred to as the parallel form. In this case, $H(z)$ is written as

$$H(z) = \sum_{k=0}^{M-N} C_k z^{-k} + \sum_{k=1}^{N/2} \frac{\gamma_{0k} + \gamma_{1k} z^{-1}}{1 - \alpha_{1k} z^{-1} - \alpha_{2k} z^{-2}} \quad (13)$$

If $M < N$, then the term $(C_k z^{-k}) = 0$. The network form is shown in Figure 20, where it is assumed that $M = N = 4$. The multiplication of the input by C (a constant) is trivial. However, for one of the parallel branches of this structure, the difference equation is

$$d_i(n) = x(n) + \alpha_{1i} d_i(n-1) + \alpha_{2i} d_i(n-2) \quad (14)$$

$$p_i(n) = \gamma_{0i} d_i(n) + \gamma_{1i} d_i(n-1)$$

where $i = 1, 2, \dots, N/2$, and $p_i(n)$ = the present output of a parallel branch.

The similarity to the second-order direct-form II network and the single parallel section is apparent. However, in this case, the outputs of all sections are summed to give the output $y(n)$, i.e.,

$$y(n) = Cx(n) + \sum_{i=1}^{N/2} p_i(n) \quad (15)$$

if $M = N$. For the parallel implementation, the delay-node values are also structured in data memory, as shown in Figure 21, thus allowing for an implementation similar to that used previously. After the output of each section stored in the 32-bit accumulator is determined, these outputs are summed to yield the filter output $y(n)$. An example of a TMS32010/TMS32020 program to implement a parallel structure can be found in Appendix C.

PERFORMANCE CONSIDERATIONS IN DIGITAL FILTER DESIGN

In the previous sections, different realizations of the FIR and IIR digital filters were discussed. This section is mainly concerned with the effects of finite wordlength on filter performance.

Some features of FIR and IIR filters, which distinguish them from each other and need special considerations when they are implemented, include phase characteristics, stability, and coefficient quantization effects.

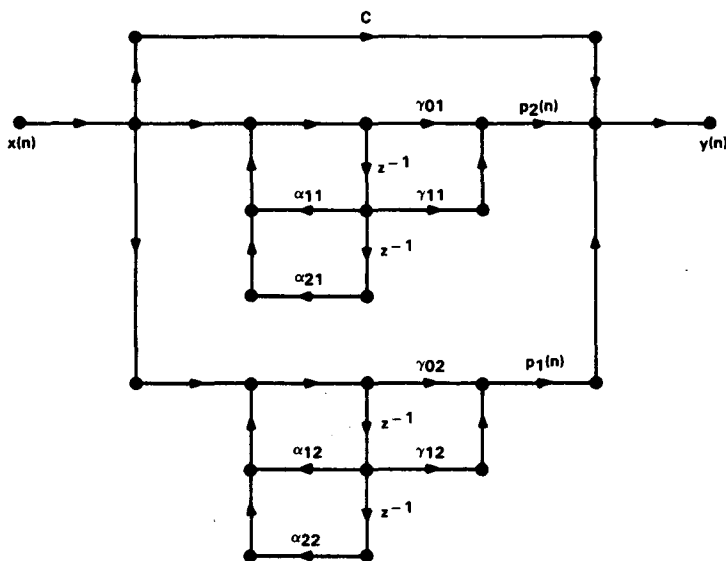


Figure 20. Parallel-Form IIR Filter

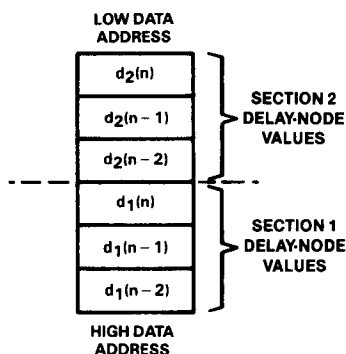


Figure 21. Delay-Node Value Storage for a Parallel IIR Filter

Given a set of frequency-response characteristics, typically a higher-order FIR filter is required to match these characteristics to a corresponding IIR filter. However, this does not imply that IIR filters should be used in all cases. In some applications, it is important that the filter have linear phase, and only FIR filters can be designed to have linear phase.

Another important consideration is the stability of the filter. Since the unit-sample response of an FIR filter is of finite length, FIR filters are inherently stable (i.e., a bounded input always produces a bounded output). This can be seen from (5) where the output of an FIR filter is a weighted finite sum of previous inputs. On the other hand, IIR filters may or may not be stable, depending on the locations of the poles of the filter.

Digital filters are designed with the assumption that the filter will be implemented on an infinite precision device. However, since all processors are of finite precision, it is necessary to approximate the "ideal" filter coefficients. This approximation introduces coefficient quantization error. The net result due to imprecise coefficient representations is a deviation of the resultant filter frequency response from the ideal one. For narrowband IIR filters with poles close to the unit circle, longer wordlengths may be required. The worst effect of coefficient quantization is instability resulting from poles being moved outside the unit circle.

The effect of coefficient quantization is highly dependent on the structure of the filter and the wordlength of the implementation hardware. Since the poles and zeroes for a filter implemented with finite wordlength arithmetic are not necessarily the same as the poles and zeroes of a filter implemented on an infinite precision device, the difference may affect the performance of the filter.

In the IIR filter, the cascade and parallel forms implement each pair of complex-conjugate poles separately. As a result, the coefficient quantization effect for each pair of complex-conjugate poles is independent of the other pairs

of complex-conjugate poles. This is generally not true for direct-form filters. Therefore, the cascade and parallel forms of IIR filters are more commonly used than the direct form.

Another problem in implementing a digital filter is the quantization error due to the finite wordlength effect in the hardware. Sources of error arising from the use of finite wordlength include the following:

1. I/O signal quantization
2. Filter coefficient quantization
3. Uncorrelated roundoff (or truncation) noise
4. Correlated roundoff (or truncation) noise
5. Dynamic range constraints.

These problems are addressed in the following paragraphs in more detail.

Representing instantaneous values of a continuous-time signal in digital form introduces errors that are associated with I/O quantization. Input signals are subjected to A/D quantization noise while output signals are subjected to D/A quantization noise. Although output D/A noise is less detrimental, input A/D quantization noise is the more dominant factor in most systems. This is due to the fact that input noise "circulates" within IIR filters and can be "regenerative" while output noise normally just "propagates" off-stage.

The filter coefficients in all of the routines described in this report are initially stored in program memory, and then moved to data memory. These coefficients are represented in Q15 format; i.e., the binary point (represented in two's-complement form) is assumed to follow the most-significant bit. This gives a coefficient range of 0.999969 to -1.0 with increments of 0.000031. The input is also in Q15 format so that when two Q15 numbers are multiplied, the result is a number in Q30 format. When the Q30 number resides in the 32-bit accumulator of the TMS320, the binary point follows the second most-significant bit. Since the output of the filter is assumed to be in Q15 format, the Q30 number must be adjusted by left-shifting by one while maintaining the most-significant 16 bits of the result. This is accomplished with the step SACH YN,1, which shifts the Q30 number to the left by one and stores the upper sixteen bits of the accumulator following the shift. The result YN is in Q15 format. Note that it is important to keep intermediate values in the accumulator as long as possible to maintain the 32-bit accuracy.

Uncorrelated roundoff (or truncation) noise may occur in multiplications. Even though the input to the digital filter is represented with finite wordlength, the result of processing leads to values requiring additional bits for their representation. For example, a b-bit data sample, multiplied by a b-bit coefficient, results in a product that is 2b bits long. In a recursive filter realization, 2b bits are required after the first iteration, 3b bits after the second iteration, and so on. The fact that multiplication results have to be truncated means that every "multiplier" in a digital structure can be regarded as a noise source. The combined effects of various noise sources degrade system performance.

Truncation or rounding off the products formed within the digital filter is referred to as correlated roundoff noise. The result of correlated roundoff (or truncation) noise, including overflow oscillations, is that filters suffer from "limit-cycle effect" (small-amplitude oscillations). For systems with adequate coefficient wordlength and dynamic range, this problem is usually negligible. Overflows are generated by additions resulting in undesirable large-amplitude oscillations. Both limit cycles and overflow oscillations force the digital filter into nonlinear operations. Although limit cycles are difficult to eliminate, saturation arithmetic can be used to reduce overflow oscillations. The overflow mode of operation on the TMS320 family is accomplished with the SOVM (set overflow mode) instruction, which sets the accumulator to the largest representable 32-bit positive (>7FFFFFFF hex) or negative (>80000000 hex) number according to the direction of overflow.

Dynamic range constraints, such as scaling of parameters, can be used to prevent overflows and underflows of the finite wordlength registers. The dynamic range is the ratio between the largest and smallest signals that can be represented in a filter. For an FIR filter, an overflow of the output results in an error in the output sample. If the input sample has a maximum magnitude of unity, then the worstcase output is

$$y(n) = \sum_{n=0}^{N-1} h(n) = s \quad (16)$$

To guarantee $y(n)$ to be a fraction, either the filter gain or the input $x(n)$ has to be scaled down by a factor "s". Reducing the filter gain implies scaling down the filter coefficients so that the 16-bit coefficient is no longer used effectively. An implication of this scaling is a degradation of the filter frequency response due to higher quantization errors. As an alternative, the input signal may be scaled, resulting in a reduction in signal-to-noise ratio (SNR). In practice, the second approach is preferred since the scaling factor is normally less than two and does not change the SNR drastically. The required scaling on a TMS32020 is achieved by using the SPM (set P register output shift mode) instruction to invoke a right-shift by six bits to implement up to 128 multiply/accumulates without overflow occurring.

For an IIR filter, an overflow can cause an oscillation with full-scale amplitude, thus rendering the filter useless. In general, if the input signal $x(n)$ is sinusoidal, the reciprocal of the gain "s" of the IIR filter is used to prevent output overflows.

For the TMS320 implementation with its double-precision accumulator and P register, scaling down the input sequence by the scaling factor "s" while maintaining a 16-bit accuracy for the coefficients can accomplish the task. For this reason, use of the MPYK instruction for IIR filter implementation is not recommended. Scaling the input signal by a factor "s" results in a degradation in the overall system SNR. Therefore, for IIR filters, it is important to keep the

coefficient quantization errors as small as possible since less accurate coefficients may cause an unstable filter if the poles are moved outside the unit circle. The LAC (load accumulator with shift) instruction on the TMS320 processors easily accomplishes input signal scaling.

In the previous paragraphs, finite wordlength problems associated with digital filter implementation on programmable devices were discussed. The 16-bit coefficients and the 32-bit accumulator of the TMS320 processor help minimize the quantization effects. Special instructions also help overcome problems in the accumulator. These features, in addition to a powerful instruction set, make the TMS32010 and TMS32020 ideal programmable processors for filtering applications.

SOURCE CODE USING THE TMS320

Examples of TMS320 source code for the implementation of two FIR filters and three IIR filters, based on the techniques described in this application report, are contained in the appendixes. Plots of the magnitude response, log-magnitude response, unit-sample response, and other pertinent data precede the filter programs.

Five filter types are presented in the three appendixes as follows:

Appendix A Length-80 bandpass FIR filter
(TMS32010 and TMS32020)

Appendix B Length-60 FIR differentiator
(TMS32010/TMS32020)

Appendix C Fourth-order lowpass IIR filters:
direct-form, cascade, and parallel
types (TMS32010/TMS32020)

The purpose of the source code is to further illustrate the use of the TMS320 devices for filtering applications and to allow implementation and analysis of these filters. The code is based on the programming techniques discussed earlier in this report.

TMS32020 source code is listed in the appendix for a length-80 FIR filter. The TMS32020 source code for the rest of the filter programs is identical to the TMS32010 code, as explained earlier. TMS32010 and TMS32020 instructions are compatible only at the mnemonic level. TMS32010 source programs should be reassembled using a TMS32020 assembler before execution. For more detail about code migration, refer to the TMS32020 User's Guide appendix, "TMS32010/TMS32020 System Migration," for detailed information.⁶

These filters were designed using the Digital Filter Design Package (DFDP) developed by Atlanta Signal Processors Incorporated (ASPI).⁷ This package runs on either a Texas Instruments Professional Computer or an IBM Personal Computer and can generate TMS320 code for the filter designed. DFDP was used to design the FIR filters with the Remez exchange algorithm developed by Parks and McClellan, and to design the IIR filters by bilinear transformation of an elliptic analog prototype. All plots supplied with the filter programs were produced by DFDP.

Filter design packages, such as DFDP, make the design

and implementation of digital filters straightforward. They allow the DSP engineer to quickly examine a variety of filters and understand the tradeoffs involved in varying the characteristics of the filters. Several digital filter design packages and other useful software support from third parties are described in the TMS32010 Development Support Reference Guide.⁸

All of the TMS320 source code examples have several features in common that depend on the implementation and application. These features include the moving of filter coefficients from storage in program memory to data memory, their representation in Q15 format, and the instructions that control the analog interface used for testing.

The hardware configuration that was used to test these filters included a Texas Instruments analog interface board (AIB) to provide an analog-to-digital and digital-to-analog interface. The sampling rate was 10 kHz in all cases. The filters were driven by a white-noise source, and the frequency response was estimated by a spectrum analyzer. Each filter routine contains several lines of code to initialize the analog interface board. The AIB signals the TMS320 that another input sample is available by pulling the BIO pin low. The TMS320 polls this pin using the BIOZ instruction. The AIB houses a TMS32010 device. In order to use the TMS32020 with the AIB (PN: RTC/EVM320C-06), a specially designed adaptor (PN: RTC/ADP320A-06) must be inserted to convert TMS32020 signals to TMS32010 signals. All of these implementation- and application-dependent sections of code are labeled.

Appendix A provides programs for the implementation of a length-80 linear-phase bandpass FIR filter on the TMS32010 and the TMS32020. The filter has been designed using the Parks-McClellan algorithm. Pertinent data for this filter is as follows:

Passband	1.375 - 3.625 kHz
Stopbands	0.0 - 1.0 kHz 4.0 - 5.0 kHz
Attenuation in stopbands	-68.4 dB
Transition regions	1.0 - 1.375 kHz 3.625 - 4.0 kHz

The figures preceding the program show the magnitude response using a linear scale, the log-magnitude response, and the unit-sample response. Both the magnitude response and the log-magnitude response illustrate the equiripple response expected from using the Parks-McClellan algorithm. The unit-sample response possesses the symmetry that is characteristic of linear-phase FIR filters.

A length-60 FIR differentiator, shown in Appendix B, is also designed using the Parks-McClellan algorithm. Characteristics for the FIR differentiator are listed below.

Lower band edge	0.0 kHz
Upper band edge	5.0 kHz

Desired slope	0.4800
Maximum deviation	0.3172 percent

The log-magnitude response is illustrated as well as the unit-sample response, which is antisymmetric for an FIR differentiator. Because the code is written in looped form, there is a dramatic reduction in the amount of program space necessary to implement this filter.

The three filters in Appendix C are fourth-order lowpass IIR filters, designed using the bilinear-transform technique. The first filter is based on a direct-form II structure, the second filter is based on a cascade structure with two second-order direct-form II subsections, and the third filter is based on a parallel structure. These three IIR filters are identical in terms of their frequency response and have the following characteristics:

Passband	0.0 - 2.5 kHz
Transition region	2.5 - 2.75 kHz
Stopband	2.75 - 5.0 kHz
Attenuation in stopband	-25.17 dB

The figures that show the magnitude response, log-magnitude response, phase response, group delay, and the unit-sample response for the three IIR filters are treated as a group and precede the three programs for filter implementation.

Table 1 is a summary of information about the five digital filters that are implemented in the appendixes.

An examination of the length-80 FIR filter implementation reveals the advantages of using a TMS32020 over the TMS32010. The program memory size is reduced by a factor of 15 (11 words vs. 163 words) while execution speed is improved by a factor of 1.8. Since the other filter types do not take advantage of the RPTK/MACD instruction pair, the performance results are the same. For example, a fourth-order cascade-form IIR filter executes at 5.4 μ s using only 27 program memory words.

When implementing linear-phase FIR filters, the designer must choose the right device for the application. If fast execution time and less program memory are essential, then the TMS32020 is the right choice.

The IIR filters are direct transformations of analog filters, exhibiting the same amplitude and phase characteristics as their analog counterparts. IIR filters tend to be more efficient than FIR filters with respect to transitionband sharpness and filter orders required. Although they require less code for implementation than the FIR filters (TMS32010 straightline code), they show great nonlinearity in phase, which limits their use in some applications.

By far the most commonly used IIR structure is the cascade-form realization. It has been shown that proper ordering of the poles and zeroes results in less sensitivity to quantization noise. The Digital Filter Design Package designs IIR filters in cascade form only.

By using a TMS32020 for both FIR and IIR filter implementations, it is possible to design a higher-order filter

Table 1. Summary Table of Filter Programs

LENGTH-80 LINEAR-PHASE BANDPASS FIR (STRAIGHT-LINE CODE)				
CODE	CYCLES	EXECUTION TIME (MICROSECONDS)	PROGRAM MEMORY (WORDS)	DATA MEMORY (WORDS)
Straight Line: TMS32010	163	32.6	163	120
TMS32020 (with RPTK)	90	18	11	161
LENGTH-60 FIR DIFFERENTIATOR (LOOPED CODE)				
CODE	CYCLES	EXECUTION TIME (MICROSECONDS)	PROGRAM MEMORY (WORDS)	DATA MEMORY (WORDS)
Looped: TMS32010/20	243	48.6	11	120
FOURTH-ORDER LOWPASS IIR FILTERS				
STRUCTURE	CYCLES	EXECUTION TIME (MICROSECONDS)	PROGRAM MEMORY (WORDS)	DATA MEMORY (WORDS)
Direct-Form II: TMS32010/20	24	4.8	24	16
Cascade: TMS32010/20	27	5.4	27	18
Parallel: TMS3210/20	28	5.6	28	18

NOTE: The above performance figures are only given as a reference. They should not be taken as benchmarks since programs can always be improved for better speed and memory efficiency.

than with the TMS32010. The TMS32020 is also ideal for higher-order FIR filters that require single-cycle multiply/accumulate operations.

SUMMARY

A brief review of FIR and IIR digital filters has been given to assist in understanding the fundamentals of digital

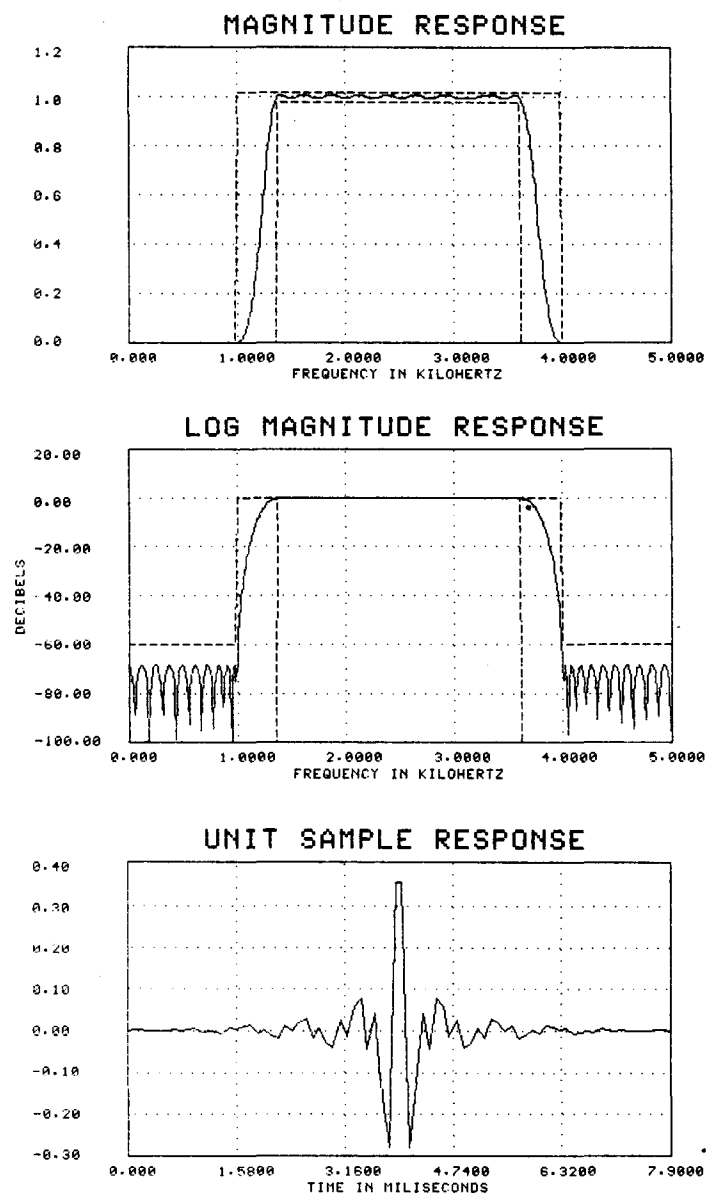
filter structure and their implementations using a digital signal processor. Many design examples have also been included to show the tradeoffs between FIR and IIR structures.

This application report has also described methods for implementing FIR and IIR filters with the TMS32010 and TMS32020. The design engineer can now choose between the two devices, depending on the application.

REFERENCES

1. A.V. Oppenheim and R.W. Schaffer, *Digital Signal Processing*, Prentice-Hall (1975).
2. Andreas Antoniou, *Digital Filters: Analysis and Design*, McGraw-Hill (1979).
3. C.S. Burrus and T.W. Parks, *Digital Filter Design*, John Wiley & Sons (1986).
4. U. Kaiser, "Wave Digital Filters and Their Significance for Customized Digital Signal Processing," *Texas Instruments Engineering Journal*, Vol 2, No. 5, 29-44 (September - October 1985).
5. *TMS32010 User's Guide* (SPRU001B), Texas Instruments (1985).
6. *TMS32020 User's Guide* (SPRU004A), Texas Instruments (1985).
7. *Digital Filter Design Package (DFDP)*, Atlanta Signal Processors Inc. (ASPI), 770 Spring St. NW, Suite 208, Atlanta, GA 30308, 404/892-7265 (1984).
8. *TMS32010 Development Support Reference Guide* (SPRU007), Texas Instruments (1984).

APPENDIX A
LENGTH-80 LINEAR-PHASE PASSBAND FIR FILTER



```
*****  
**      LINEAR-PHASE FIR FILTER  
**      LENGTH-80 BANDPASS FILTER  
**  
**      SAMPLING FREQUENCY = 10 KHZ  
**  
**      FILTER CHARACTERISTICS  
**  
**          BAND 1      BAND 2      BAND 3  
**  
**      LOWER BAND EDGE    0.0000     1.3750     4.0000  
**      UPPER BAND EDGE   -0.0000     3.6250     5.0000  
**      STOPBAND RIPPLE    0.0000     1.0000     0.0000  
**      NOMINAL Ripple     0.0010     0.0200     0.0010  
**      MAXIMUM RIPPLE      0.0004     0.0076     0.0014  
**      RIPPLE IN DB       -68.3965    0.0657    -68.3957  
**  
**      FILTER STRUCTURE  
**  
**  
**          z-1         z-1         z-1  
**      o---o---o---o---o---z---o---z---o---z---o---  
**      x(n)            v h(0) v h(1) v h(2) v h(N-2) v h(N-1)  
**  
**      o---o---o---o---o---z---o---z---o---z---o---  
**      y(n)  
**  
**      CYCLES | EXECUTION TIME | PROGRAM MEMORY | DATA MEMORY  
**             | (MICROSECONDS) | (WORDS) | (WORDS)  
-----|-----|-----|-----  
**      90 | 18 | 10 | 161  
**  
**      IDT 'FIRBPASS'  
**      VN EQU 40  
**      MODE EQU 46  
**      CLOCK EQU 47  
**      XN EQU 48  
**  
**      AORG 0  
**      B START  
**  
**      (EXCLUDING INITIALIZATION AND I/O)  
*****  
**  
**      0020 VN  
**      0045 MODE  
**      0047 CLOCK  
**      0048 XN  
**      0049 XN  
**      0050 XN  
**      0051 0000  
**      0052 0000 FF80  
**      0053 0001 0072  
**  
**      CTABLE AORG 32  
**      CPU DATA >00FF  
**      CHN CHN * -0.107351E+02 *  
**      0025 0021 001E * -0.979376E+01 *
```

```

FIRPASS      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157      16:55:56 08-15-85
                                           PAGE 0003

0114 005B 000B CH59 DATA >000B * 0.346738E-03 *
0115 005C FE05 CH60 DATA >FE05 * -0.860811E-02 *
0116 005D 0187 CH61 DATA >0187 * -0.119391E-01 *
0117 005E 08E6 CH62 DATA >08E6 * 0.704627E-02 *
0118 005F 000B CH63 DATA >000B * 0.342216E-03 *
0119 0060 000B CH64 DATA >000B * 0.621682E-02 *
0120 0061 FF74 CH65 DATA >FF74 * -0.815474E-02 *
0121 0062 FF70 CH66 DATA >FF70 * -0.438169E-02 *
0122 0063 FF6F CH67 DATA >FF6F * -0.348111E-04 *
0123 0064 FF6E CH68 DATA >FF6E * -0.446419E-03 *
0124 0065 00A2 CH69 DATA >00A2 * 0.496452E-02 *
0125 0066 0069 CH70 DATA >0069 * 0.322896E-02 *
0126 0067 FF79 CH71 DATA >FF79 * -0.194902E-03 *
0127 0068 004B CH72 DATA >004B * 0.231021E-02 *
0128 0069 FF84 CH73 DATA >FF84 * -0.229530E-02 *
0129 006A FF8A CH74 DATA >FF8A * -0.212256E-03 *
0130 006B FF86 CH75 DATA >FF86 * -0.771385E-03 *
0131 006C FF89 CH76 DATA >FF89 * -0.675043E-03 *
0132 006D 0051 CH77 DATA >0051 * 0.249065E-02 *
0133 006E 005E CH78 DATA >005E * -0.535353E-03 *
0134 006F FFDC CH79 DATA >FFDC * -0.107231E-02 *
0135
0136 0070 000A MD DATA >000A ; SAMPLING RATE OF 10 KHZ *
0137 0071 01F3 SMP DATA >01F3
0138
0139 0072 START EQU $
0140
0141 * INITIALIZATION OF THE ANALOG INTERFACE BOARD
0142
0143 0072 C807 LDRK MD 7
0144 0073 CA70 LACK MD
0145 0074 582E TBLR MODE
0146 0075 E02E OUT MODE,PA0
0147 0076 CA71 LACK SMP
0148 0077 582F TBLR CLOCK
0149 0078 E12F OUT CLOCK,PA1
0150
0151 * LOAD FILTER COEFFICIENTS
0152
0153 0079 5588 LARP ARO ; USE ARO FOR INDIRECT ADDRESSING
0154 007A D200 LRLK ARO,>200 ; POINT TO BLOCK B0
0155 007C C84F RPTK >4F ; 80 COEFFICIENTS
0156 007D FCA0 BLKP CTABLE,**
0157
0158 007F CE05 * CMFP ; USE BLOCK B0 AS PROGRAM AREA
0159
0160 0080 FA80 WAIT BIOZ ; BIO PIN GOES LOW WHEN A
0161 0081 0000 LRLK B ; NEW SAMPLE IS AVAILABLE
0162 0083 0080 * B ; NEW SAMPLE IS AVAILABLE
0163
0164 0084 8230 MXTPT IN XN,PA2 ; BRING IN THE NEW SAMPLE XN
0165 0085 D100 MXTPT B ; BRING IN THE NEW SAMPLE XN
0166 0086 03FF LRLK ARI,>3FF ; POINT TO THE BOTTOM OF BLOCK B1
0167
0168 0087 5589 * LARP ARI
0169 0088 A000 MPYK 0
0170 0089 CA00 ZAC
0171 008A CB4F RPTK >4F
0172 008B 5C90 MACD >FF00,*-
0173 008C FF00 *
0174 008D CE15 APAC
0175 008E 692D SACH XN,1
0176
0177 008F E22D * OUT XN,PA2 ; OUTPUT THE FILTER RESPONSE Y(n)
0178 0090 FF80 * B WAIT ; GO GET THE NEXT POINT
0179 0091 0080 *
0180
0181 END
NO ERRORS, NO WARNINGS

```


FIRBPASS	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:41:39 08-29-85	PAGE 0005
0228	002A 000A MD DATA >000A			
0229	002B 01F3 SMP DATA >01F3			
0230	002C 01F3 SMP DATA >01F3			
0231	002C 6E00 START LDPK 0			
0232	002D 7E01 LACK 1			
0233	002D 7E01 LACK 1			
0234	002E 507B SACL ONE			
0235	002F 7079 LACK ARG,CLOCK			
0236	002F 7079 LACK ARG,CLOCK			
0237	0030 7129 LACK ARL>29			
0238	0031 712B LACK SMP			
0239	0031 6E88 LACK SMP			
0240	0033 6791 LACK SMP			
0241	0034 107B LACK SMP			
0242	0035 F400 SUB ONE			
0243	0036 0032 BANZ LOAD			
0244	0037 4878 OUT MODE,PA0			
0245	0038 4979 OUT CLOCK,PA1			
0246	0039 F600 WAIT			
0247	0039 F600 WAIT			
0248	003B F900 B WAIT			
0249	003C 0039 * BRING IN THE NEW SAMPLE XN *			
0250	003D 4200 NXTPT IN XN,PA2			
0251	003E 7F89 ZAC			
0252	003E 7F89 ZAC			
0253	003F 6A4F LT XNM79			
0254	003F 6A4F MPY H0			
0255	0040 6D50 * DUE TO SYMMETRY h(0) = h(79) *			
0256	0041 684E * x(n-79) * h(79) *			
0257	0041 684E LTD XNM78			
0258	0042 6D51 MPY H1			
0259	0043 684D LTD XNM77			
0260	0043 684D MPY H2			
0261	0044 6D52 * h(1) = h(78) *			
0262	0045 684C LTD XNM76			
0263	0045 684C MPY H3			
0264	0046 6D53 LTD XNM75			
0265	0047 684B MPY H4			
0266	0048 6D54 LTD XNM74			
0267	0049 684A MPY H5			
0268	0049 684A LTD XNM73			
0269	0049 684A MPY H6			
0270	004A 6D55 LTD XNM72			
0271	004B 6849 MPY H7			
0272	004B 6849 LTD XNM71			
0273	004C 6D56 MPY H8			
0274	004D 684E * h(1) = h(77) *			
0275	004E 6D57 LTD XNM70			
0276	004E 6D57 MPY H9			
0277	004F 6847 LTD XNM69			
0278	004F 6847 MPY H10			
0279	0050 6D58 LTD XNM68			
0280	0050 6D58 MPY H11			
0281	0051 6846 LTD XNM67			
0282	0052 6D59 MPY H12			
0283	0053 6845 LTD XNM66			
0284	0053 6845 MPY H13			
0285	0054 6D5A LTD XNM65			
0286	0054 6D5A MPY H14			
0287	0055 6844 LTD XNM64			
0288	0055 6844 MPY H15			
0289	0056 6D5B LTD XNM63			
0290	0056 6D5B MPY H16			
0291	0057 6843 LTD XNM62			
0292	0057 6843 MPY H17			
0293	0059 6842 LTD XNM61			
0294	0059 6842 MPY H18			
0295	005A 6D5D LTD XNM60			
0296	005B 6841 MPY H19			
0297	005B 6841 LTD XNM59			
0298	005C 6D5E MPY H20			
0299	005C 6D5E LTD XNM58			
0300	005D 6840 MPY H21			
0301	005D 6840 LTD XNM57			
0302	005F 683F MPY H22			
0303	0060 6D60 LTD XNM56			
0304	0061 683E MPY H23			
0305	0061 683E LTD XNM55			
0306	0062 6D61 MPY H24			
0307	0063 683D LTD XNM54			
0308	0063 683D MPY H25			
0309	0064 6D62 LTD XNM53			
0310	0065 683C MPY H26			
0311	0065 683C LTD XNM52			
0312	0066 6D63 MPY H27			
0313	0067 6838 LTD XNM51			
0314	0067 6838 MPY H28			
0315	0068 6D64 LTD XNM50			
0316	0069 683A MPY H29			
0317	0069 683A LTD XNM49			
0318	006A 6D65 MPY H30			
0319	006B 6839 LTD XNM48			
0320	006B 6839 MPY H31			
0321	006C 6D66 LTD XNM47			
0322	006D 6838 MPY H32			
0323	006D 6838 LTD XNM46			
0324	006E 6D67 MPY H33			
0325	006F 6837 LTD XNM45			
0326	006F 6837 MPY H34			
0327	0070 6D68 LTD XNM44			
0328	0071 6836 MPY H35			
0329	0071 6836 LTD XNM43			
0330	0072 6D69 MPY H36			
0331	0073 6835 LTD XNM42			
0332	0073 6835 MPY H37			
0333	0074 6D6A LTD XNM41			
0334	0075 6834 MPY H38			
0335	0075 6834 LTD XNM40			
0336	0076 6D6B MPY H39			
0337	0077 6833 LTD XNM39			

FIRBPASS	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:41:39 08-29-85 PAGE 0007	FIRBPASS	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:41:39 08-29-85 PAGE 0008
0339 0078 6D6C *	MPY H28			0396 009E 6D70 *	MPY H32		
0340	LTD XNM50			0397	LTD XNM31		
0341 0079 6B32	MPY H29			0398 009F 6B1F	MPY H31		
0342 007A 6D6D *				0399 00A0 6D6F *			
0343	LTD XNM49			0400	LTD XNM30		
0344 007B 6B31	MPY H30			0401 00A1 6B1E	MPY H30		
0345 007C 6D6E *				0402 00A2 6D6E *			
0346	LTD XNM48			0403	LTD XNM29		
0347 007D 6B30	MPY H31			0404 00A3 6B1D	MPY H29		
0348 007E 6D6F *				0405 00A4 6D6D *			
0349	LTD XNM47			0406 00A5 6B1C	LTD XNM28		
0350 007F 6B2F	MPY H32			0407 00A6 6D6C *	MPY H28		
0351 0080 6D70 *				0408	LTD XNM27		
0352	MPY H33			0409	MPY H27		
0353 0081 6B2E	LTD XNM46			0410 00A7 6B1B	LTD XNM26		
0354 0082 6D71 *	MPY H34			0411 00A8 6D6B *	MPY H26		
0355	LTD XNM45			0412	LTD XNM25		
0356 0083 6B2D	MPY H35			0413 00A9 6B1A	MPY H25		
0357 0084 6D72 *				0414 00AA 6D6A *			
0358	LTD XNM44			0415 00AB 6B19	LTD XNM24		
0359 0085 6B2C	MPY H36			0416 00AC 6D69 *	MPY H24		
0360 0086 6D73 *				0417	LTD XNM23		
0361	LTD XNM43			0418 00AD 6B18	MPY H23		
0362 0087 6B2B	MPY H37			0419 00AE 6D68 *	LTD XNM22		
0363 0088 6D74 *				0420 00AF 6B17	MPY H22		
0364	LTD XNM42			0421 00B0 6D67 *			
0365 0089 6B2A	MPY H38			0422 00B1 6B16	LTD XNM21		
0366 008A 6D75 *				0423 00B2 6D66 *	MPY H21		
0367 008B 6B29	LTD XNM41			0424 00B3 6B15	LTD XNM20		
0368 008C 6D76 *	MPY H39			0425 00B4 6D65 *	MPY H20		
0369 008D 6B28	LTD XNM40			0426 00B5 6B14	LTD XNM19		
0370	MPY H39			0427 00B6 6D64 *	MPY H19		
0371 008E 6D77 *				0428 00B7 6B13	LTD XNM18		
0372 008F 6B27	LTD XNM39			0429 00B8 6D62 *	MPY H18		
0373	MPY H36			0430 00B9 6B11	LTD XNM17		
0374 008F 6B27	MPY H36			0431 00BC 6D61 *	MPY H17		
0375 0090 6D77 *				0432 00BD 6B10	LTD XNM16		
0376	LTD XNM38			0433 00BE 6D60 *	MPY H16		
0377 0091 6B26	MPY H38			0434 00BF 6B0F	LTD XNM15		
0378 0092 6D76 *				0435 00C0 6D5F *	MPY H15		
0379 0093 6B25	LTD XNM37			0436 00C1 6B0E	LTD XNM14		
0380 0094 6D75 *	MPY H37			0437 00C2 6D5E *	MPY H14		
0381	LTD XNM36			0438 00C3 6B0D	LTD XNM13		
0382	MPY H36						
0383 0095 6B24	LTD XNM35						
0384 0096 6D74 *	MPY H35						
0385	LTD XNM34						
0386 0097 6B23	MPY H34						
0387 0098 6D73 *							
0388 0099 6B22	LTD XNM33						
0389 009A 6D72 *	MPY H33						
0390	LTD XNM32						
0391	MPY H33						
0392 009B 6B21	LTD XNM31						
0393 009C 6D71 *	MPY H31						
0394	LTD XNM30						
0395 009D 6B20	MPY H30						

```

FIRBPASS      32010 FAMILY MACRO ASSEMBLER      PC2.1 84.107      20:41:39 08-29-85
                                                    PAGE 0009

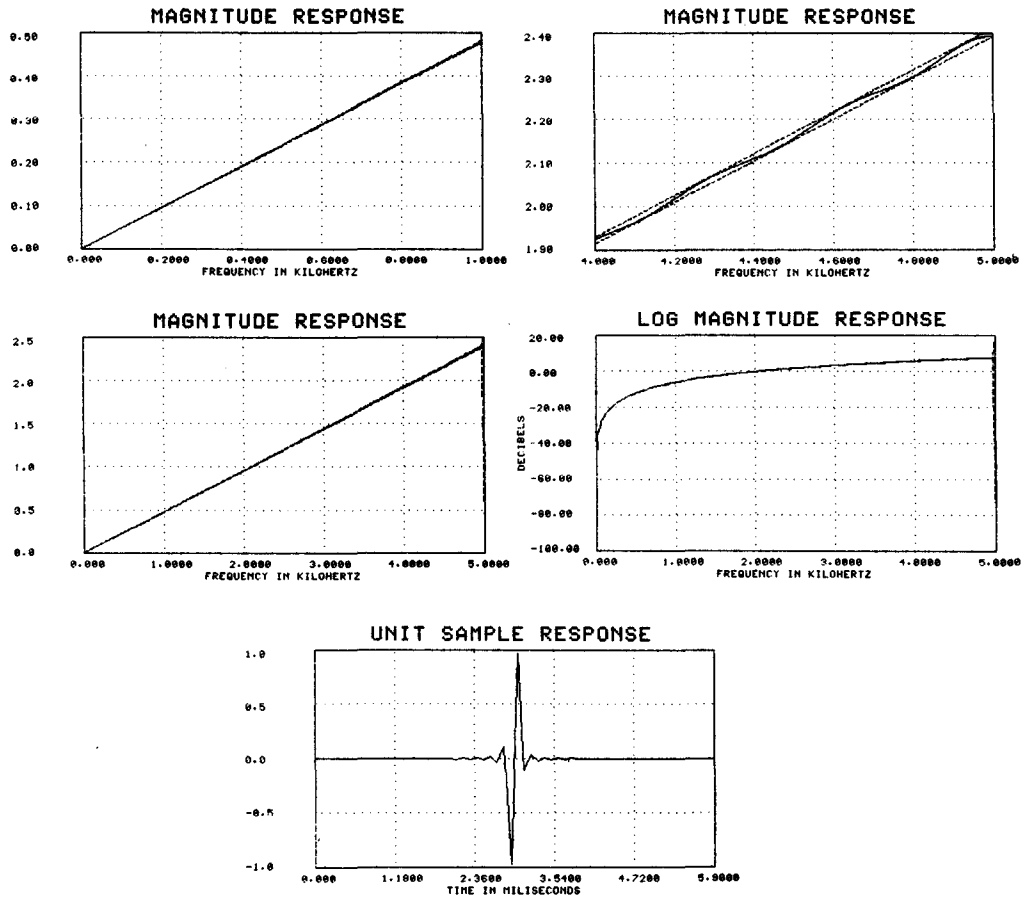
0453 00C4 6D5D      *      MPY H13
0454      *      *      *      *      *      *      *      *      *      *
0455 00C5 6B0C      *      LTD XNM12
0456 00C6 6D5C      *      MPY H12
0457      *      *      *      *      *      *      *      *      *      *
0458 00C7 6B0B      *      LTD XNM11
0459 00C8 6D5B      *      MPY H11
0460      *      *      *      *      *      *      *      *      *      *
0461 00C9 6B0A      *      LTD XNM10
0462 00CA 6D5A      *      MPY H10
0463      *      *      *      *      *      *      *      *      *      *
0464 00CB 6B09      *      LTD XNM9
0465 00CC 6D59      *      MPY H9
0466      *      *      *      *      *      *      *      *      *      *
0467 00CD 6B08      *      LTD XNM8
0468 00CE 6D58      *      MPY H8
0469      *      *      *      *      *      *      *      *      *      *
0470 00CF 6B07      *      LTD XNM7
0471 00D0 6D57      *      MPY H7
0472      *      *      *      *      *      *      *      *      *      *
0473 00D1 6B06      *      LTD XNM6
0474 00D2 6D56      *      MPY H6
0475      *      *      *      *      *      *      *      *      *      *
0476 00D3 6B05      *      LTD XNM5
0477 00D4 6D55      *      MPY H5
0478      *      *      *      *      *      *      *      *      *      *
0479 00D5 6B04      *      LTD XNM4
0480 00D6 6D54      *      MPY H4
0481      *      *      *      *      *      *      *      *      *      *
0482 00D7 6B03      *      LTD XNM3
0483 00D8 6D53      *      MPY H3
0484      *      *      *      *      *      *      *      *      *      *
0485 00D9 6B02      *      LTD XNM2
0486 00DA 6D52      *      MPY H2
0487      *      *      *      *      *      *      *      *      *      *
0488 00DB 6B01      *      LTD XNM1
0489 00DC 6D51      *      MPY H1
0490      *      *      *      *      *      *      *      *      *      *
0491 00DD 6B00      *      LTD XN
0492 00DE 6D50      *      MPY H0
0493      *      *      *      *      *      *      *      *      *      *
0494 00DF 7F8F      *      APAC
0495      *      *      *      *      *      *      *      *      *      *
0496 00E0 597A      *      SACH YN,1
0497      *      *      *      *      *      *      *      *      *      *
0498 00E1 4A7A      *      OUT VN,PA2
0499      *      *      *      *      *      *      *      *      *      *
0500 00E2 F900      *      B WAIT
0501      *      *      *      *      *      *      *      *      *      *
0502      *      *      *      *      *      *      *      *      *      *
NO ERRORS, NO WARNINGS
END

```

* OUTPUT THE FILTER RESPONSE Y(n) *

* GO GET THE NEXT POINT *

APPENDIX B **LENGTH-60 FIR DIFFERENTIATOR**



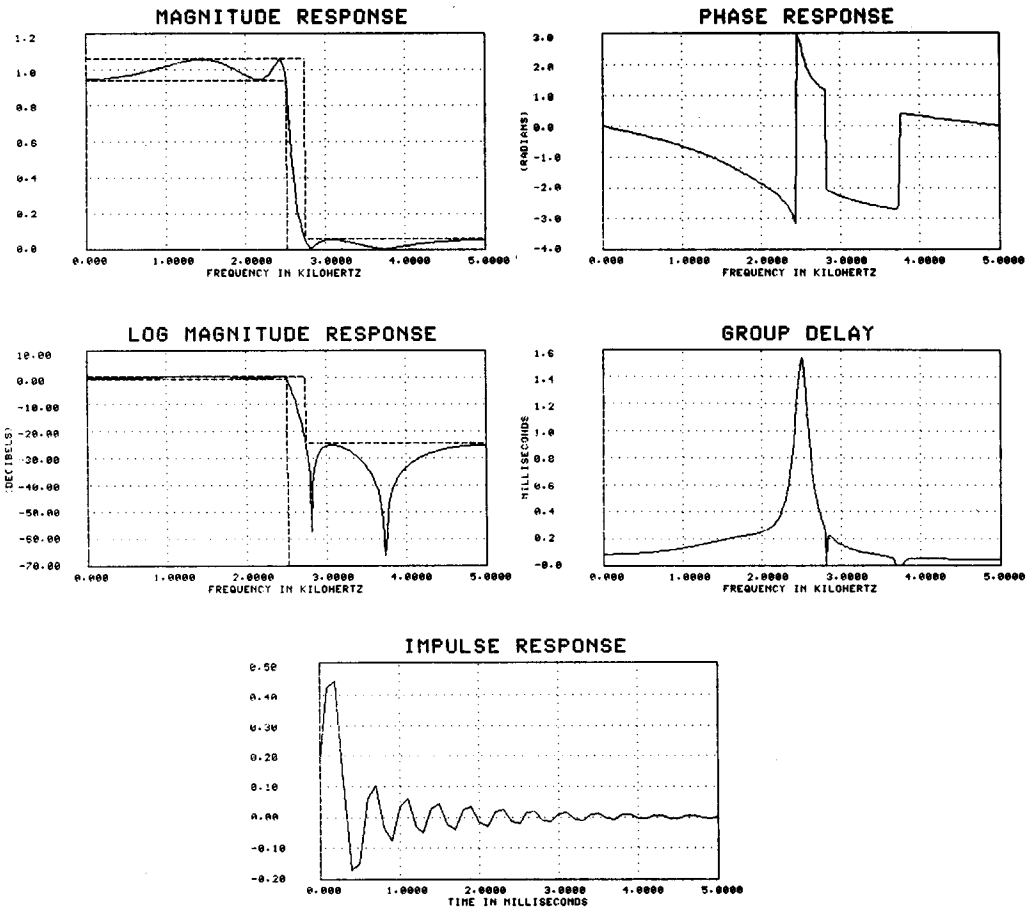
	0000E	XNM14	EQU 14
	0000F	XNM15	EQU 15
	00010	XNM16	EQU 16
	00011	XNM17	EQU 17
	00012	XNM18	EQU 18
	00013	XNM19	EQU 19
	00014	XNM20	EQU 20
	00015	XNM21	EQU 21
	00016	XNM22	EQU 22
	00017	XNM23	EQU 23
	00018	XNM24	EQU 24
	00019	XNM25	EQU 25
	0001A	XNM26	EQU 26
	0001B	XNM27	EQU 27
	0001C	XNM28	EQU 28
	0001D	XNM29	EQU 29
	0001E	XNM30	EQU 30
	0001F	XNM31	EQU 31
	00020	XNM32	EQU 32
	00021	XNM33	EQU 33
	00022	XNM34	EQU 34
	00023	XNM35	EQU 35
	00024	XNM36	EQU 36
	00025	XNM37	EQU 37
	00026	XNM38	EQU 38
	00027	XNM39	EQU 39
	00028	XNM40	EQU 40
	00029	XNM41	EQU 41
	0002A	XNM42	EQU 42
	0002B	XNM43	EQU 43
	0002C	XNM44	EQU 44
	0002D	XNM45	EQU 45
	0002E	XNM46	EQU 46
	0002F	XNM47	EQU 47
	00030	XNM48	EQU 48
	00031	XNM49	EQU 49
	00032	XNM50	EQU 50
	00033	XNM51	EQU 51
	00034	XNM52	EQU 52
	00035	XNM53	EQU 53
	00036	XNM54	EQU 54
	00037	XNM55	EQU 55
	00038	XNM56	EQU 56
	00039	XNM57	EQU 57
	0003A	XNM58	EQU 58
	0003B	XNM59	EQU 59
	0003C	*	EQU 60
	0003D	H1	EQU 61
	0003E	H2	EQU 62
	0003F	H3	EQU 63
	00040	H4	EQU 64
	00041	H5	EQU 65
	00042	H6	EQU 66
	00043	H7	EQU 67
	00044	H8	EQU 68
	00045	H9	EQU 69

FIRDF	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:43:03 08-29-85	20:43:03 08-29-85	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:43:03 08-29-85	20:43:03 08-29-85
			PAGE 0005	PAGE 0006				
0228 0034 0014	CH50	DATA	>0014		* -CH9 *			
0229 0035 0015	CH51	DATA	>FFED		* -CH8 *			
0230 0036 0016	CH52	DATA	>0011		* -CH7 *			
0231 0037 0017	CH53	DATA	>FFED		* -CH6 *			
0232 0038 0018	CH54	DATA	>FFED		* -CH5 *			
0233 0039 0019	CH55	DATA	>FFED		* -CH4 *			
0234 003A 0010	CH56	DATA	>0010		* -CH3 *			
0235 003B 0011	CH57	DATA	>FFEA		* -CH2 *			
0236 003C 0012	CH58	DATA	>003D		* -CH1 *			
0237 003D 0013	CH59	DATA	>FFCF		* -CH0 *			
0238								
0239 003E 000A	MD	DATA	>000A					
0240 003F 01F3	SMP	DATA	499		* SAMPLING RATE OF 10 KHZ *			
0241								
0242 0040 6500	START	LDPK 0						
0243								
0244 0041 7801	LACK 1				* CONTENT OF ONE IS 1 *			
0245 0042 507B	SACL ONE							
0246								
0247 0043 7079	LARK ARO,CLOCK				* THIS SECTION OF CODE LOADS *			
0248 0044 713C	LARK ARI,60				* THE FILTER COEFFICIENTS AND *			
0249 0045 7E3F	LACK SMP				* OTHER VALUES FROM PROGRAM *			
0250 0046 6880	LARP ARO				* MEMORY TO DATA MEMORY *			
0251 0047 6791	TBLR *-ARI							
0252 0048 007B	SUB ONE							
0253 0049 7400	BANZ LOAD							
0254								
0255 004B 4878	OUT MODE,PA0				* INITIALIZATION OF ANALOG *			
0256 004C 4979	OUT CLOCK,PA1				* INTERFACE BOARD *			
0257								
0258 004D 6880	LARP ARO				* SET ARP TO ARO *			
0259								
0260 004E F600	WAIT				* BIO PIN GOES LOW WHEN A *			
0261 0050 F900	B WAIT				* NEW SAMPLE IS AVAILABLE *			
0262								
0263 0052 4200	NXTPT				* BRING IN THE NEW SAMPLE XN *			
0264								
0265 0053 703B	LARK ARO,XIM59				* ARO POINTS TO THE INPUT SEQUENCE *			
0266 0054 7177	LARK ARI,H59				* ARI POINTS TO THE IMPULSE RESPONSE *			
0267								
0268 0055 7F89	ZAC							
0269								
0270 0056 6A91	LT *-ARI							
0271 0057 8D90	MPY *-ARO							
0272								
0273 0058 6881	LTD *-ARI							
0274 0059 6D90	MPY *-ARO							
0275								
0276 005A F400	BANZ LOOP							
0277								
0278 005C 7F8F	APAC				* ACCUMULATE LAST MULTIPLY *			
0279								
0280 005D 597A	SACH YN,1							

0281
 0282 005E 4A7A *
 0283 005F F900 *
 0284 0060 808E *
 0285
 0286
 NO ERRORS, NO WARNINGS
 END

* OUTPUT THE FILTER RESPONSE Y(n) *
 * GO GET THE NEXT POINT *

APPENDIX C **FOURTH-ORDER LOWPASS IIR FILTERS**



[illegible]


```

0020:43:59  08-29-85
PAGE 0002

```

IIR4CAS	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:43:59 08-29-85	PAGE 0003
0114 0014 6880	LOAD			
0115 0015 6791	TBLR *-ARI			
0116 0016 1014	SUB ONE			
0117 0017 F400	BANZ LOAD			
0118 0018 0014	*			
0119 0019 7F89	ZAC			
0120 001A 5000	SACL D2N			
0121 001B 5001	SACL D2NM1			
0122 001C 5002	SACL D2NM2			
0123 001D 5003	SACL DIN			
0124 001E 5004	SACL DINM1			
0125 001F 5005	SACL DINM2			
0126	*			
0127 0020 4810	OUT MODE,PA0			
0128 0021 4911	OUT CLOCK,PA1			
0129 0022 8600	WAIT			
0130 0023 0026	B WAIT			
0131 0024 F900				
0132 0025 0022	*			
0133 0026 4213	NXTPT			
0134	*			
0135 0027 2F13	LAC XN,15			
0136	*			
0137 0028 6A04	LT DINM1			
0138 0029 6D09	MPY A11			
0139	*			
0140 002A 6C05	LTA DINM2			
0141 002B 6D0A	MPY A21			
0142	*			
0143 002C 7F8F	APAC			
0144	*			
0145 002D 5903	SACH DIN,1			
0146	*			
0147 002E 7F89	ZAC			
0148	*			
0149 002F 6D08	MPY B21			
0150	*			
0151 0030 6804	LTD DINM1			
0152 0031 6D07	MPY B11			
0153	*			
0154 0032 6803	LTD DIN			
0155 0033 6D06	MPY B01			
0156	*			
0157	*			
0158	*			
0159 0034 6C01	LTA D2NM1			
0160 0035 6D0E	MPY A12			
0161	*			
0162 0036 6C02	LTA D2NM2			
0163 0037 6D0F	MPY A22			
0164	*			
0165 0038 7F8F	APAC			
0166	*			
0167 0039 5900	SACH D2N,1			

IIR4CAS	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:43:59 08-29-85	PAGE 0004
0168	*			
0169 003A 7F89	ZAC			
0170	*			
0171 003B 6D0D	MPY B22			
0172 003C 6801	LTD D2NM1			
0173 003D 6D0C	MPY B12			
0174	*			
0175 003E 6800	LTD D2N			
0176 003F 6D0B	MPY B02			
0177	*			
0178 0040 7F8F	APAC			
0180	*			
0181 0041 5912	SACH YN,1			
0182	*			
0183	*			
0184 0042 4A12	OUT YN,PA2			
0185	*			
0186 0043 F900	B WAIT			
0187 0044 0022	*			
0188	END			
NO ERRORS, NO WARNINGS				

[illegible]

IIIR4PAR	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:45:18 08-29-85 PAGE 0063	IIIR4PAR	32010 FAMILY MACRO ASSEMBLER	PC2.1 84.107	20:45:18 08-29-85 PAGE 0064
0114 0008 FEF7 CA12	DATA >FEF7	* -0.008080 *		0168 0030 6D06 *	MPY G01		
0115 0009 90E5 CA22	DATA >90E5	* -0.867723 *		0169 0031 7F8F *	APAC		
0116 0010 5988 CC	DATA >5988	* 0.699476 *		0170 0032 5914 *	SACH P1,1	* FINISHED FIRST PARALLEL SECTION *	
0117 0011 000A MD	DATA >000A			0171 0033 2F12 *	LAC XN,15	* START SECOND PARALLEL SECTION *	
0118 0012 01F3 SMP	DATA >01F3	* SAMPLING RATE OF 10 KHZ *		0172 0034 6A02 *	LT D2NM2		
0119 0013 6E00 START	LDPK 0			0173 0035 6D0D *	MPY A22	* $d_2(n-2) * a_{22}$ *	
0120 0014 7E01 LACK 1	LACK 1	* CONTENT OF ONE IS 1 *		0174 0036 6B01 *	LTD D2NM1		
0121 0015 5013 SACL ONE	SACL ONE			0175 0037 6D0C *	MPY A12		
0122 0016 7010 LARK AR0,CLOCK	LARK AR0,CLOCK	* THIS SECTION OF CODE LOADS *		0180 0038 7F8F *	APAC		
0123 0017 710A LARK AR1,10	LARK AR1,10	* THE FILTER COEFFICIENTS AND *		0181 0039 5900 *	SACH D2N,1		
0124 0018 6880 LACK SMP	LACK SMP	* OTHER VALUES FROM PROGRAM *		0182 003A 2F14 *	LAC P1,15		
0125 0019 6880 LOAD	LOAD	* MEMORY TO DATA MEMORY *		0183 003B 6D0B *	MPY G12		
0126 0020 6791 TBLR **,AR1	TBLR **,AR1			0184 003C 6B00 *	LTD D2N		
0127 0021 6013 SUB ONE	SUB ONE	* THIS SECTION SETS THE *		0185 003D 6D0A *	MPY G02		
0128 0022 6013 BANZ LOAD	BANZ LOAD	* INITIAL STATE OF THE *		0186 003E 6C0E *	LTA C		
0129 0023 480F ZAC	ZAC	* FILTER TO ZERO *		0187 003F 6D12 *	MPY XN		
0130 0024 4910 SACL D2N	SACL D2N			0188 0040 7F8F *	APAC		
0131 0025 5001 SACL D2NM1	SACL D2NM1	* INITIALIZATION OF ANALOG *		0189 0041 5911 *	SACH YN,1	* FINISHED SECOND PARALLEL SECTION *	
0132 0026 5002 SACL D2NM2	SACL D2NM2	* INTERFACE BOARD *		0190 0042 4A11 *	OUT YN,PA2	* AND FINISHED FILTER	
0133 0027 5003 SACL DIN	SACL DIN	* BIO PIN GOES LOW WHEN A *		0191 0043 F900 *	B WAIT	* OUTPUT THE FILTER RESPONSE Y(n) *	
0134 0028 5004 SACL DINM1	SACL DINM1	* NEW SAMPLE IS AVAILABLE *		0192 0044 0021 *		* GO GET THE NEXT POINT *	
0135 0029 5005 SACL DINM2	SACL DINM2			0200			
0136 0030 480F OUT MODE,PA0	OUT MODE,PA0	* BRING IN THE NEW SAMPLE XN *		0201			
0137 0031 4910 OUT CLOCK,PA1	OUT CLOCK,PA1	* START FIRST PARALLEL SECTION *		0202			
0138 0032 4910 WAIT	WAIT			0203			
0139 0033 4212 NXTFT	NXTFT			0204			
0140 0034 2F12 *	LAC XN,15			0205			
0141 0035 6A05 *	MPY A21	* $d_1(n-2) * a_{21}$ *					
0142 0036 6D09 *	LT DINM2						
0143 0037 6B04 *	LTD DINM1						
0144 0038 6D08 *	MPY A11						
0145 0039 7F8F *	APAC						
0146 0040 5903 *	SACH DIN,1						
0147 0041 7F89 *	ZAC						
0148 0042 6D07 *	MPY G11						
0149 0043 6B03 *	LTD DIN						

NO ERRORS, NO WARNINGS