Performance Estimation Methods for XP32 MAXL

Ian J. Curington and Rex W. Tracy
Floating Point Systems, U.K., Ltd.

Abstract
XP32 application programs depend heavily on program structure for optimal execution. Methods of designing XP32 programs are introduced in this paper, using as an example an interpolated spectral analysis application. A performance analysis technique is introduced and applied to the example.

Introduction

This paper illustrates a concise approach to performance estimation of XP32 MAXL program modules. A structured application design methodoloy is introduced, showing the manner in which code structure affects performance. This methodology is a tool for the analysis of a wide variety of application programs on the XP32, as well as a model for the application structure.

Problem Statement

To illustrate the analysis techniques discussed below, an application problem is posed for execution on the XP32 co-processor in the FPS-5000 Series array processor. The problem considered is one of interpolated spectral estimation using multiple XP32 co-processors. The problem common to the design of this, and many other potential applications, is identifying the maximum system throughput achievable for a given number of co-processors. Additionally, it is useful to know whether or not an XP32 can achieve high performance on this application.

The example is centered around a specific spectrum estimation process which is illustrated in Figure 1. As shown in the block diagram, data originates in the FPS-5000 Series System Common Memory (SCM), is processed in an XP32 co-processor, and results are returned to SCM. One block of time-ordered input data is processed at a time. The process expands the input block by zero filling and windowing before applying the forward Fast Fourier Transform (FFT). The frequency domain data is then scaled and a magnitude estimate made at each of the frequencies from the FFT. The process continues for a large number of data blocks.

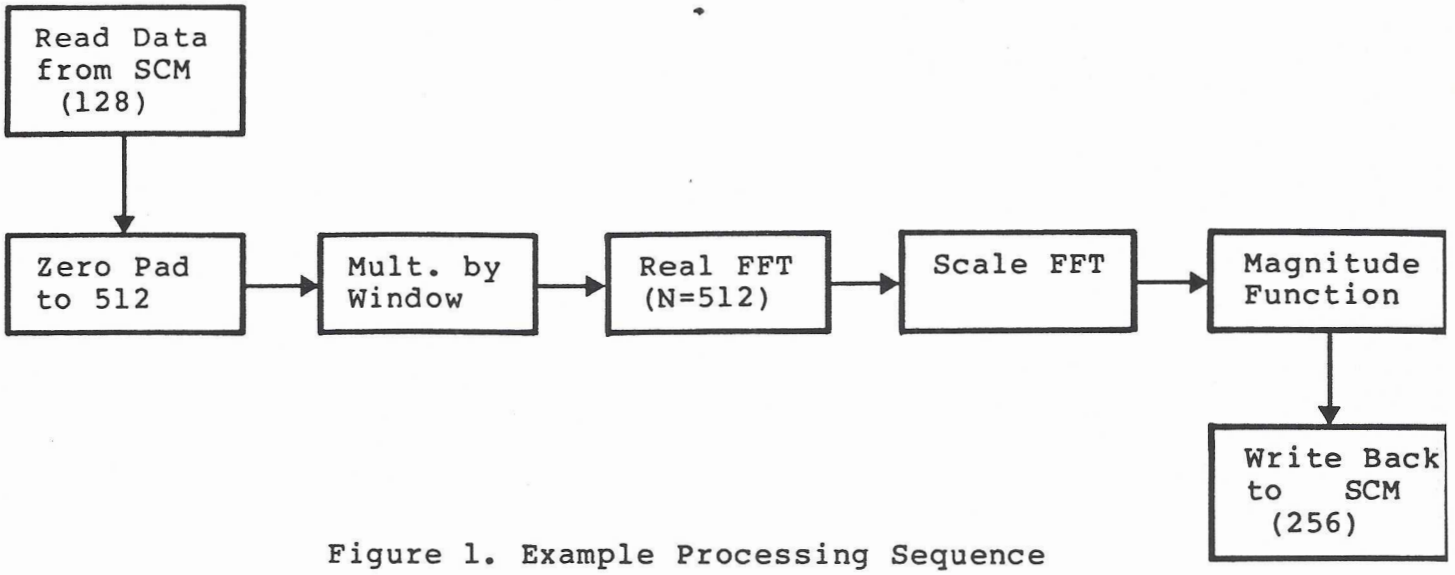

Figure 1. Example Processing Sequence

In this example, the specific parameters of the application lead to the use of an interpolated spectral estimate and system parameters given by

```
NINPUT = 128
NFFT   = 512
NOUTPT = 256
```

where NINPUT represents the number of acquired data samples per processing block, NFFT is the size of the FFT data block, and NOUTPT is the number of spectral estimates sent back to SCM.

Analysis Methodology

Developing an application package for use in the FPS-5000 involves a series of decisions determining the structure of the code to be developed. The first consideration is how to partition the application between the various system processors. In this example, a portion of the spectral processing could be performed by the XP32, while the remainder is processed by the control processor. However, in order to facilitate extending the application code to multiple XP32 co-processors, in this example processing is performed entirely within one XP32 co-processor.

Limiting the problem to one XP32, without using the control processor arithmetic capabilities, performance estimation and optimization proceeds in the following manner:

1. Analyze the processing requirements of the computations in a single XP32 co-processor as shown in the following example analysis.

2. Analyze the I/O requirements necessary to support a single XP32 and determine if the SCM bandwidth can support the single XP32, or multiple XP32s, also shown in the following example analysis.

3. Determine the relationship between processing time and I/O time between the XP32 local main data (LMD) and SCM.
   1. If I/O time is greater than processing time, then little will be gained by optimizing the structure of the XP32 MAXL code. However, repartitioning the application to include more intensive XP32 calculations would be helpful.
   2. If I/O time is less than processing time, XP32 MAXL code should be organized to take advantage of overlapped data transfer and efficient executive queue management.

4. Create a preliminary draft of the XP32 MAXL code to estimate the number of XP32 co-processors appropriate for this application.

These steps lead to an efficient first order estimate of both the best way in which to structure the XP32 code (given a particular problem partitioning), and to an estimate of the system performance on this application.

Example Analysis

For the purposes of this paper, the computational analysis begins by developing a table associating XP32 math library (XPMLIB) functions with the required processing tasks. An overhead estimate must be added to account for the processing time spent queueing and dequeueing the operations to be performed by the arithmetic section of the XP32. Further entries in the table are included for each of the data transfers between SCM and XP32 Local Main Data (LMD). An illustration for this specific application is given in Table 1. In estimating data transfer times, an FPS-5100/5400 Series Array Processor is assumed.

From the analysis shown in Table 1, a number of conclusions may be drawn from this application. First, the processing time is considerably longer than the I/O transfer time, indicating that this application is a good candidate for using multiple XP32 co-processors. Because any one XP32 requires only 10 percent of the SCM bandwidth, transfer times are not likely to be extended beyond these estimates. Second, the upper bound on throughput with a single XP32 is limited by the processing time of 1.128 milliseconds (if the code is structured to overlap I/O and processing). Finally, if the code is written in a simplistic manner, such as input, process, output, the XP32 time will equal the sum of I/O and processing time (approximately 1.32 milliseconds). This time is approximately 20 percent longer than if the code were fully optimized.

Processing Function	Vector Size	XP32 Math Lib Routine	Function Start-up (cycles)	Function Execution (cycles/pt)	Total Execution Time (microseconds)
Zero Pad	NFFT	ZVCLR	20	0.5	46.00
Window	NINPUT	ZVMUL	19	1.5	35.16
Real FFT	NFFT	ZRFFT	90	7.5	603.10
Scale FFT	NFFT	ZSCLRF	19	3.0	259.16
Magnitude	NOUTPT	ZCVMGS	20	1.5	131.33
Subtotal Arithmetic Processing Time					1074.76
Queue/Dequeue overhead (5%)					53.74
Total Processing Time					1128.50 usec

Transfer SCM → LMD	NINPUT	XPDMAR		3.0 (2MHz)	64.00
Transfer LMD → SCM	NOUTPT	XPDMAR		3.0 (2MHz)	128.00
Total Data Transfer Time					192.00 usec

Table 1. Processing and Data Transfer Timing Estimates.

Because vector sizes are in the range of 128 to 512, a greater than usual amount of overhead may occur in queue management. This amount is estimated to be approximately 5 percent, and is shown as an addition to the processing subtotal in Table 1.

A Simple Program Structure

This section describes a MAXL program which implements the spectral estimation process as described in the previous section. Initially, no attempt is made to take advantage of the XP32 capability of overlapping data transfers with computation.

In constructing the example code, it is assumed that there are a number of input data blocks available in SCM when the XP32 code is entered. The address of the data, the number of data blocks, and the address of the resultant buffer in SCM are passed to the XP32 MAXL routine when it is initially started by the control processor. The XP32 processes all available data blocks before returning control to the control processor.

In order to introduce the code included in this section, the overall structure is outlined in Figure 2. This structured

technique is useful in organizing the flow of control in XP32 applications.

A. Do for all data blocks available:

1. **Input Buffer from SCM to Local Main Data (LMD).**
2. **Wait for LMD buffer to be filled.**
3. **Process Spectral Estimation Arithmetic Operations.**
4. **Wait for computations to finish.**
5. **Output results to SCM.**
6. **Wait for buffer to empty.**

B. Pass control back to the control processor.

Figure 2. Example Non-overlapped MAXL Structure.

```
C  XP32 MAXL SPECTRAL ESTIMATION EXAMPLE
C
        SUBROUTINE SPECTR( INAD, NBLOCKS, OUTAD )
C
C$APMATH ZVCLR,ZVMUL,ZRFFT,ZSCRLF,ZCVMGS
        PARAMETER ( FROMXP=1, TOXP=2, NFFT=512, NINPUT=128 )
        PARAMETER ( NOUTPT=256, FLOAT=2 )
C
        INTEGER BUFFER,J,WORK,WINDOW
C
C LMD LOCATIONS:
C   BUFFER=0, WINDOW BUFFER=8K, WORK AREA=12K
        DATA BUFFER,WINDOW,WORK/ 0, 8192, 12288 /
C
C FOR ALL DATA BLOCKS,
        DO 600 J=1,NBLOCKS
C
C GET DATA IN FROM SCM
        CALL XPDMAR ( TOXP, BUFFER, 1, INAD, 1, FLOAT, NINPUT )
        CALL XPISNC
C
C RUN THE PROCESSING FUNCTIONS
        CALL ZVCLR ( WORK, 1, NFFT )
        CALL ZVMUL ( BUFFER, WINDOW, WORK, NINPUT)
        CALL ZRFFT ( WORK, NFFT, 1 )
        CALL ZSCLRF( WORK, NFFT, 3, 1)
        CALL ZCVMGS( WORK, BUFFER, NOUTPT )
      . CALL XPISNC
C
C NOW SEND THE RESULT BACK TO SCM.
        CALL XPDMAR ( FROMXP, BUFFER, 1, OUTAD, 1, FLOAT, NOUTPT )
        CALL XPISNC
        INAD  = INAD  + NINPUT
        OUTAD = OUTAD + NOUTPT
600     CONTINUE
```

```
        RETURN
        END
C END OF EXAMPLE
```

This code sequence can be combined with the numbers obtained from Table 1 to generate a timing diagram for the execution of this example. As shown in Figure 3, the total processing time is approximately 1.32 milliseconds.

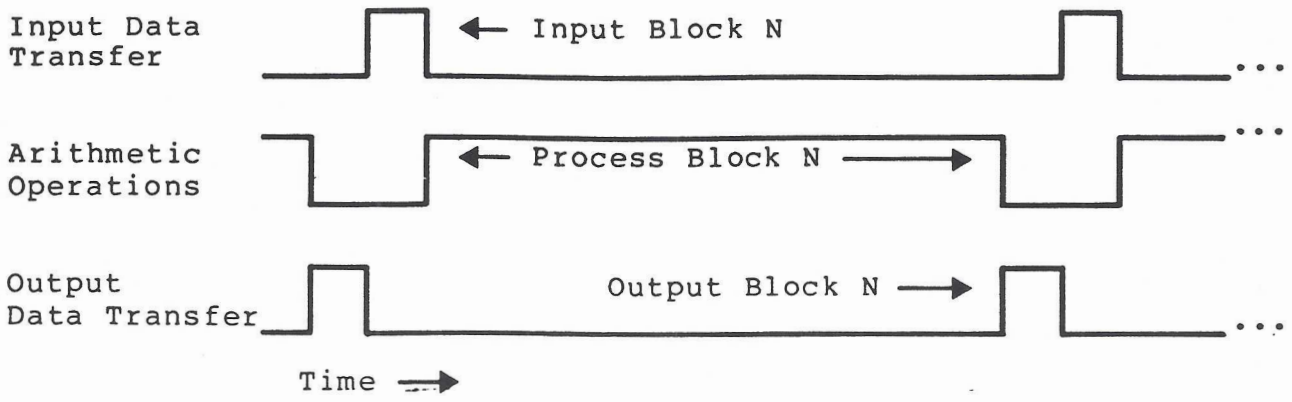

Figure 3. Timing Diagram for Non-Overlapped Example.

Given that the FPS-5100/5400 architecture can support an aggregate data transfer bandwidth of 4Mhz, a second XP32 co-processor could easily be included (to process half of the required blocks) with no throughput degradation caused by memory contentions. Even with two XP32s in the application, the total requirement for memory access is .384 milliseconds (at 2Mhz), out of a frame time of 1.32 milliseconds. Thus opportunity exists to add a third XP32 (which increases I/O time to .576 milliseconds), and continue to have sufficient SCM bandwidth available to support additional computations by the control processor.

Viewed from the perspective of overall system throughput, the effective frame time is 1.32/3 or 440 microseconds per data block. If a real time interface, such as a GPIOP, is involved in data acquisition, the I/O rate translates into 3.44 microseconds per input data word with block size of 128. The additional I/O load of the GPIOP at 290 KHz continues to leave a wide margin for SCM access by the control processor.

Adding Overlap of Data Transfer and Computation

Starting with the same overall assumptions, overlapping data transfers with computation, the first step is to revise the structural outline of the XP32 MAXL code. Figure 4 shows a double buffered control scheme surrounding exactly the same processing steps. To further simplify the control structure, assume that an even number of data blocks are to be processed. An odd number of data blocks can easily be accomodated by skipping the last buffer B processing and output steps 7, 9, and 10 in Figure 4.

1. Input Buffer A
2. Wait for Buffer A to fill
3. Start Process on Buffer A (In Place)
4. Input Buffer B
5. Do J=1 to (NBLOCKS/2)-2 the following:
   A. Wait for Buffer B full and Processing Complete
   B. Start Process on Buffer B
   C. Output Buffer A results
   D. Input new Buffer A
   E. Wait for Buffer A full and Processing Complete
   F. Start Process on Buffer A
   G. Output Buffer B
   H. Input new Buffer B
6. Wait for Buffer B full and Processing Complete
7. Start Process on Buffer B
8. Output Buffer A results
9. Wait for Processing Complete
10. Output Buffer B results
11. Wait for Buffer B emptied
12. Pass Control back to the control processor

Figure 4. Double Buffered Control Outline.

Translating the structure into MAXL is accomplished as shown in the following MAXL code module. The processing calls could be split off into a separate MAXL module for symbolic independence and more structured programming style, but are shown merged in-line to minimize MAXL interpretation overhead.

```
C DOUBLE BUFFERED OVERLAPPED I/O VERSION OF
C  XP32 MAXL SPECTRAL ESTIMATION EXAMPLE
C
      SUBROUTINE SPECTR( INAD, NBLOCKS, OUTAD )
C
C$APMATH ZVCLR,ZVMUL,ZRFFT,ZSCRLF,ZCVMGS
      PARAMETER ( FROMXP=1, TOXP=2, NFFT=512, NINPUT=128 )
      PARAMETER ( NOUTPT=256, FLOAT=2 )
C
      INTEGER N,TEMP,BUFFRA,BUFFRB,IADDR,OADDR,J,WORK,WINDOW
C
C LMD LOCATIONS:
C   BUFFER A =0, BUFFER B =4K, WINDOW BUFFER=8K, WORK AREA=12K
      DATA BUFFRA,BUFFRB,WINDOW,WORK/ 0, 4096, 8192, 12288 /
C
C INITIALIZE LOCAL POINTERS
      IADDR = INAD
      OADDR = OUTAD
C
C START FIRST READ OPERATION
      CALL XPDMAR ( TOXP, BUFFRA, 1, IADDR, 1, FLOAT, NINPUT )
      CALL XPISNC
```

```
          IADDR = IADDR + NINPUT •
C
C PROCESS BUFFER A
          CALL ZVCLR ( WORK, 1, NFFT )
          CALL ZVMUL ( BUFFRA, WINDOW, WORK, NINPUT)
          CALL ZRFFT ( WORK, NFFT, 1 )
          CALL ZSCLRF( WORK, NFFT, 3, 1)
          CALL ZCVMGS( WORK, BUFFRA, NOUTPT )
C
C START READING BUFFER B
          CALL XPDMAR ( TOXP, BUFFRB, 1, IADDR, 1, FLOAT, NINPUT )
          IADDR = IADDR + NINPUT
C
C COMPUTE THE NUMBER OF TIMES THROUGH THE DO-LOOP
          TEMP = NBLOCKS **2*-1
          N    = TEMP -2
C
C MAIN LOOP THROUGH ALL BUT THE LAST TWO BUFFERS
          DO 600 J = 1, N
C
          CALL XPISNC
          CALL ZVCLR  ( WORK, 1, NFFT )
          CALL ZVMUL  ( BUFFRB, WINDOW, WORK, NINPUT)
          CALL ZRFFT  ( WORK, NFFT, 1 )
          CALL ZSCLRF ( WORK, NFFT, 3, 1)
          CALL ZCVMGS ( WORK, BUFFRB, NOUTPT )
          CALL XPDMAR ( FROMXP, BUFFRA, 1, OADDR, 1, FLOAT, NOUTPT )
          CALL XPDMAR ( TOXP,   BUFFRA, 1, IADDR, 1, FLOAT, NINPUT )
          OADDR = OADDR + NOUTPT
          IADDR = IADDR + NINPUT
C
          CALL XPISNC
          CALL ZVCLR ( WORK, 1, NFFT )
          CALL ZVMUL ( BUFFRA, WINDOW, WORK, NINPUT)
          CALL ZRFFT ( WORK, NFFT, 1 )
          CALL ZSCLRF( WORK, NFFT, 3, 1)
          CALL ZCVMGS( WORK, BUFFRA, NOUTPT )
          CALL XPDMAR ( FROMXP, BUFFRB, 1, OADDR, 1, FLOAT, NOUTPT )
          CALL XPDMAR ( TOXP,   BUFFRB, 1, IADDR, 1, FLOAT, NINPUT )
          OADDR = OADDR + NOUTPT
          IADDR = IADDR + NINPUT
C
  600     CONTINUE
C
          CALL XPISNC
          CALL ZVCLR  ( WORK, 1, NFFT )
          CALL ZVMUL  ( BUFFRB, WINDOW, WORK, NINPUT)
          CALL ZRFFT  ( WORK, NFFT, 1 )
        . CALL ZSCLRF ( WORK, NFFT, 3, 1)
          CALL ZCVMGS ( WORK, BUFFRB, NOUTPT )
          CALL XPDMAR ( FROMXP, BUFFRA, 1, OADDR, 1, FLOAT, NOUTPT )
          OADDR = OADDR + NOUTPT
          CALL XPISNC
          CALL XPDMAR ( FROMXP, BUFFRB, 1, OADDR, 1, FLOAT, NOUTPT )
          CALL XPISNC
C
```

```
RETURN
END
```

In the same method as the previous non-overlapped case, a timing diagram can be built combining the MAXL code structure with the timing analysis performed earlier. In this case, the only time the arithmetic section is not performing useful calculations is while communicating syncronization information to the XP32 executive or is performing request queue management operations. The timing is the same as for the previous analysis and is summarized graphically in Figure 5. The effective frame time has been reduced to 1.13 milliseconds.

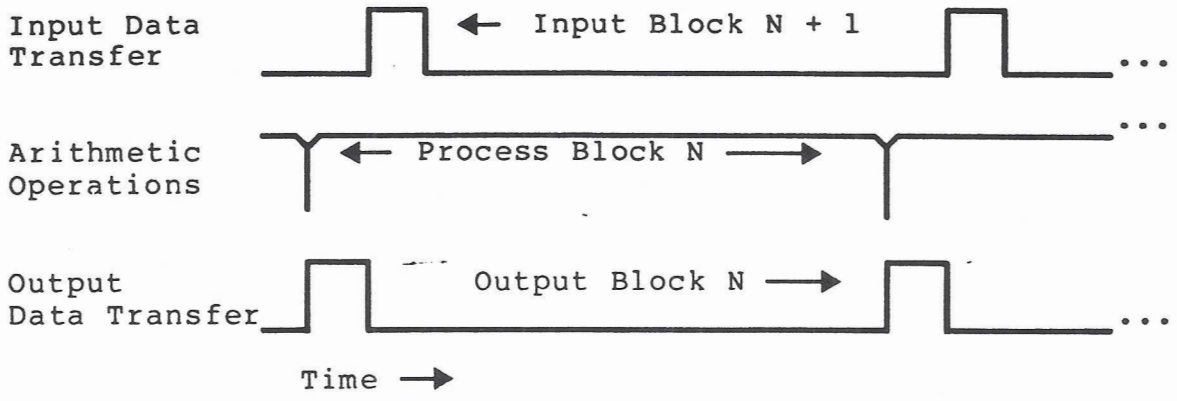

Figure 5. Timing Diagram for Overlapped Double-Buffered Example.

Again, the memory bandwidth requirements for this program structure are found to be small (192 microseconds at 2Mhz out of every 1128 microseconds), leading to the conclusion that multiple XP32s are able to operate in this environment. Using three XP32s, the effective frame time is 377 microseconds, or approximately 2.95 microseconds per data sample. At continuous real-time rates, the throughput requirement of the I/O channel is 340 kHz and falls within the capabilities of the FPS-5000 architecture.

XP32 Executive Performance Considerations

The preceding analysis has ignored the effects of the XP32 executive on overall throughput. In this specific application, it may be valid to ignore the executive because of the MAXL code structure. The executive operates in a fashion designed to minimize the overhead in starting operations, and performs as much MAXL interpretation as possible, overlapped with other XP32 functions. In the above examples the only area in which MAXL interpretation may affect performance is the updating of address pointers.

If a MAXL program is structured to leave the arithmetic section idle while performing control operations, a more thorough analysis technique is required. The speed of MAXL code interpreted by the executive is slower than the same code executing on the control processor, where MAXL is compiled to APAL rather than interpreted. Due to this consideration, It is therefore advantageous to retain

as much of the time consuming MAXL operations on the control processor as possible.

The executive performs numerical operations on 24-bit integers by performing several distinct steps. First, it fetches the code word from SCM, storing it internally in a scratch area. Second, it decodes and validates the requested operation, and reads the parameter list. In the third step, it executes the requested operation. The time required for these steps can be summarized as

1. Fetch Operation = 1.5 cycles
2. Validate, Decode, Fetch N Arguments = 2.5 * N + 3.5 cycles
3. Execute Operation = 1 to 50 cycles

The time required for execution of the operation varies considerably with the type of operation. Figure 6 shows the approximate cost of various operations (one cycle in the XP32 takes .167 microseconds). The times may be longer if SCM bandwidth is restricted, and if the internal queues in the XP32 become full.

Operation	Approximate Time (Cycles)
I = J+K	6
I = 47	6
I = I+1	6
I = J+73	6
I = J*K	50
I = ISCM(K)	20
I = ILMD(K)	13
DO ...	1
Continue	1
CALL NAME(1,2,...,N)	40 + 3/2*N

Figure 6. XP32 Executive Operation Times.

Future Directions

To confirm the validity of these performance evaluation techniques, testing of application codes such as those shown in this paper on production XP32s is required. As experience is gained, the model can be refined to produce more accurate results. The optimizations present in the XP32 executive may improve the execution times beyond those shown. More advanced tools are expected to become available to allow automatic performance evaluation, such as an XP32 simulator, run-time process monitor, or program development tools appropriate for multi-processor environments. Hopefully the techniques presented in this paper will prove useful. More advanced techniques will always be a welcome improvement.

Conclusion

The carefully applied structuring techniques for XP32 MAXL programs has been shown to yield dramatic performance improvements over traditional methods. The FPS-5000 Series architecture is capable of sustaining high throughput when the application is structured correctly. Through analysis techniques discussed in this paper the achievable performance can be estimated ahead of the actual implementation. Future developments will aid in refining the analysis tools and allow for direct performance estimates.

References

[1] Curington, I. "Power Spectrum Analysis with the FPS-5000 Series", CHECKPOINT Vol. 1, Issue 7, August 1983, Floating Point Systems, Portland Oregon.

[2] Tracy, R. W. "A Distributed Architecture for High Throughput Array Processors" FPS-5000 Series Application Note, June 1983, Floating Point Systems, Portland Oregon.

[3] "FPS-5000 Preliminary User's Guide" Publication number 800-1013-000A, January 1984, Floating Point Systems, Portland Oregon.