A MULTI-PROCESSOR BASED DIGITAL BEAMFORMING SYSTEM

by

Kapriel Karagozyan

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements

for the degrees of

BACHELOR OF SCIENCE

and

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1988

Signature of Author.....
Department of Electrical Engineering and Computer Science, May 5, 1988

Certified by.....
Dr. Arthur B. Baggeroer, Professor
Department of Electrical Engineering and Computer Science
Department of Ocean Engineering

Certified by.....
Robert Fine, Manager of Applications Engineering
Digital Signal Processing Division, Analog Devices Inc.

Accepted by.....
Arthur C. Smith, Chairman
Committee on Graduate Students

A MULTI-PROCESSOR BASED
DIGITAL BEAMFORMING SYSTEM

by

KAPRIEL KARAGOZYAN

Submitted to the Department of Electrical Engineering and
Computer Science on May 10, 1988 in partial fulfillment
of the requirements for the Degrees of Bachelor of
Science and Master of Science in Electrical Engineering

ABSTRACT

Today's real-time digital beamformers do not make full use of the existing architectural and computational advantages offered by the modern VLSI digital signal processors. An efficient and modular architecture using distributed processing techniques was designed and developed. A hardware prototype that implements this architecture was successfully built and tested.

The design of this passive sonar beamforming system is based on several ADSP-2100 digital signal processors that independently perform the beamforming calculations under the supervision of another ADSP-2100 processor. The modular architecture allows the user to tailor the size of the system to his performance needs. The system samples its inputs at a rate of 10KHz. and is capable of taking inputs from 32 or fewer hydrophones with an arbitrary arrangement. It can form up to 32 simultaneous beams with 32 inputs. It is also capable of forming a larger number of beams using fewer inputs. The array parameters are user configurable through an IBM PC/AT interface. The output beams are available in analog and digital formats for further processing.

This experimental beamforming system will be used in sonar systems for oceanographic research in the Arctic areas.

Thesis Supervisor : Dr. Arthur B. Baggeroer
Title : Professor of Ocean Engineering and Electrical Engineering

TABLE OF CONTENTS

CHAPTER 1 : Introduction

The beamforming concept (in general and specifically in sonar systems) is discussed in an introductory manner in this chapter. Beamforming system issues are presented, and the specifications for a desired sonar beamforming system are defined.

1.1 Sonar Systems:

A basic sonar system can use two different methods to analyze and evaluate possible targets in the water. The first is called " active sonar ". This method involves the transmission of a well defined acoustic signal which can reflect from objects in water. This provides the sonar receiver with a basis for detecting and locating the targets of interest. The limitations of this method are mainly due to the loss of the signal strength during propagation through the water and reverberation caused by the signal reflections. Simplistically, active sonar can be thought of as the underwater equivalent of radar.

The second method is called "passive sonar". This one bases its detection and localization on sounds which are emitted from the target itself (machine noise, flow noise, transmissions of its active sonar). Its limitations are due to the imprecise knowledge of the characteristics of the target sources and to the dispersion of the target signals by the undersea medium. A generic active sonar system and a generic passive sonar system are shown in figures (1-1) and (1-2).

Sonar systems have a wide variety of military and commercial uses. Some of the military applications include detection, localization, classification, tracking, parameter estimation, weapons guidance, countermeasures and communications. Some of the commercial applications include fish location, bottom mapping, navigation aids, seismic prospecting and acoustic oceanography. More detailed information about sonar technology can be found in references [1],[2].

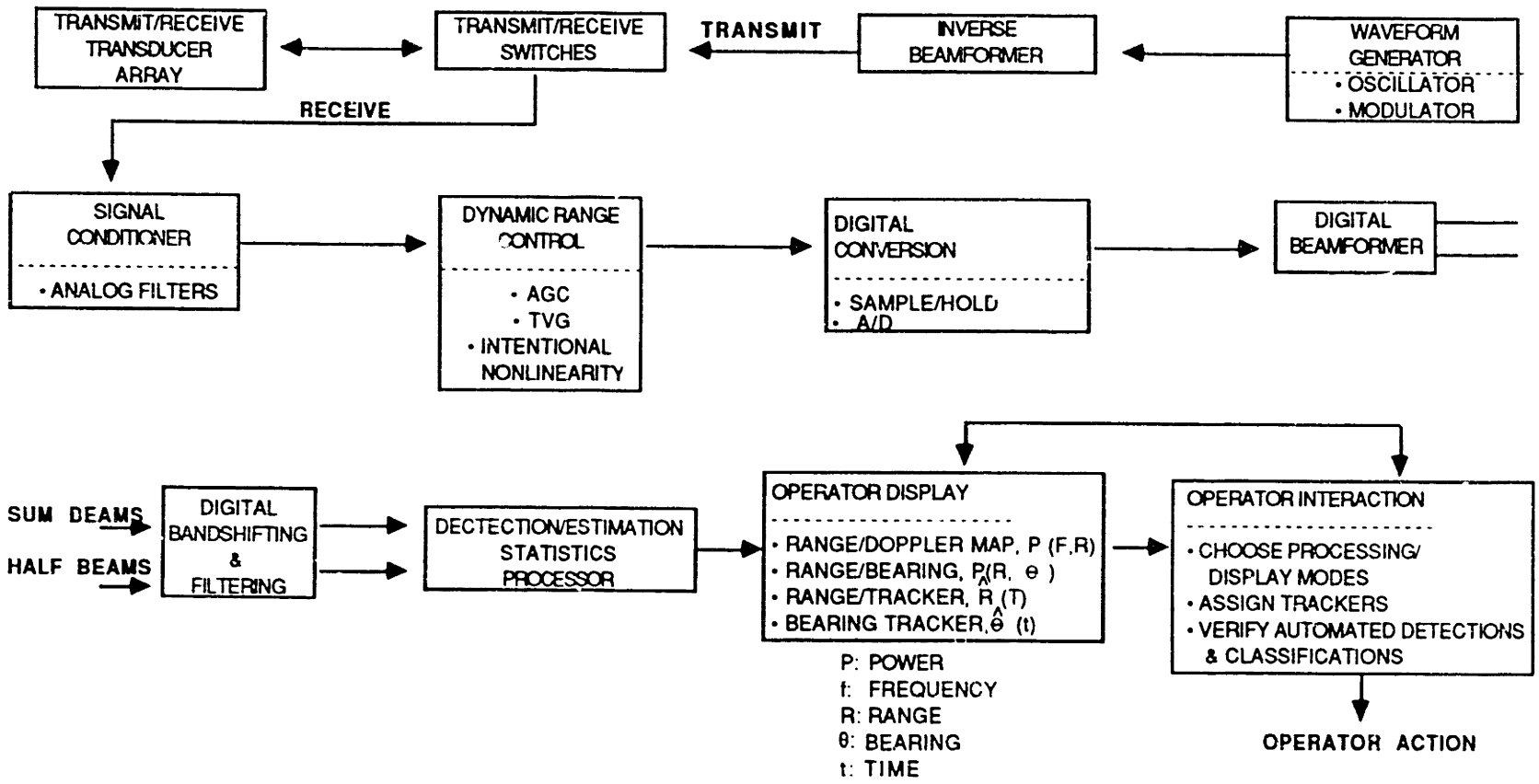


Figure 1-1 : Generic active sonar system block diagram

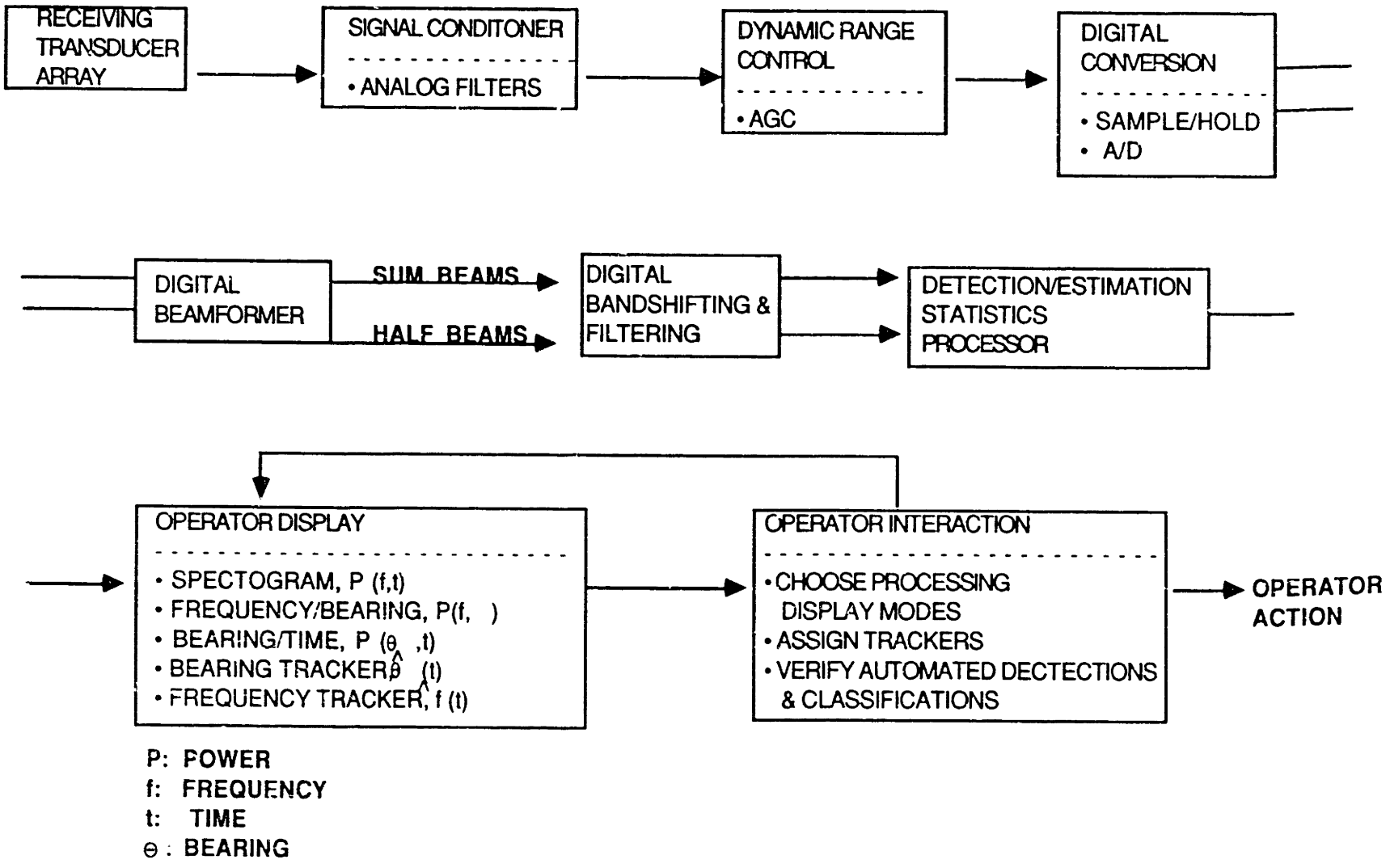


Figure 1-2 : Generic passive sonar system block diagram

1.2 Sonar Beamforming :

One of the important building blocks of a sonar system is the beamformer. The concept, techniques and some implementations of sonar beamforming will be briefly discussed in the following sections.

1.2.1 The Beamforming Concept :

In its simplest form, sonar beamforming can be defined as " the process of combining the outputs from a number of omnidirectional transducer elements, arranged in an array of arbitrary geometry, so as to enhance signals from some defined spatial location while suppressing those from other sources " [3]. Thus, a beamformer may be considered to be a spatial filter. It is generally assumed that the waves arriving to the transducers all propagate with the same speed c , so that the signals of interest lie on the surface of the cone defined by $\omega = c\,|k|$ in (k, ω) space. Ideally the passband of the beamformer is the intersection of this cone with the plane containing the desired direction vector, as shown in figure (1-3).

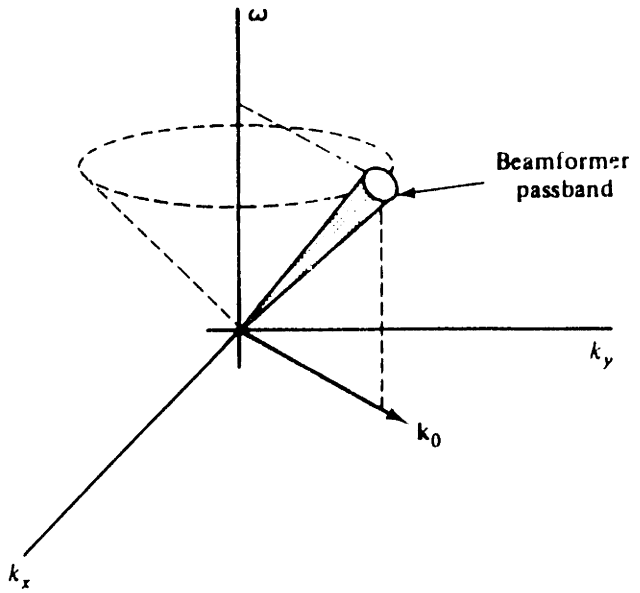


Figure 1-3 : Passband of an ideal beamformer

The beamforming operation is accomplished through a series of operations that involve the weighting, delay, and summation of the signals received by the spatial elements. The summed output that contains information about a particular direction is called a " beam ". This output is then sent to a signal processor and/or a display for frequency and temporal discrimination. A beamforming system can employ analog or digital components and techniques; these are to be discussed in further sections.

Beamformers are used both in passive and active sonar systems. In passive sonar, the beamformer acts on the received waveforms. Active sonar also utilizes a conventional beamformer which acts on the waveforms that are reflected from the targets (most active sonars use the same array for receiving/transmitting). There are several well known techniques that can be utilized in forming beams from receiver arrays. The discussion in the following section will concentrate on the weighted delay-and-sum beamforming technique which is very common and is also referred as time-delay beamforming. Discussion on other techniques, such as FFT beamforming or phase-shift beamforming may be found in references [2],[4].

1.2.2 Time-delay Beamforming :

In time-delay beamforming, beams are formed by averaging weighted and delayed versions of the receiver signals. Each receiver has a known location and samples the incoming signals spatially. In order to steer the beams (i.e. to choose beamforming directions), each receiver's output has to be delayed appropriately relative to the other receivers. An overall block diagram for a conventional time delay beamformer is shown in Figure(1-4). The mathematical details of time delayed beamforming are discussed in this section.

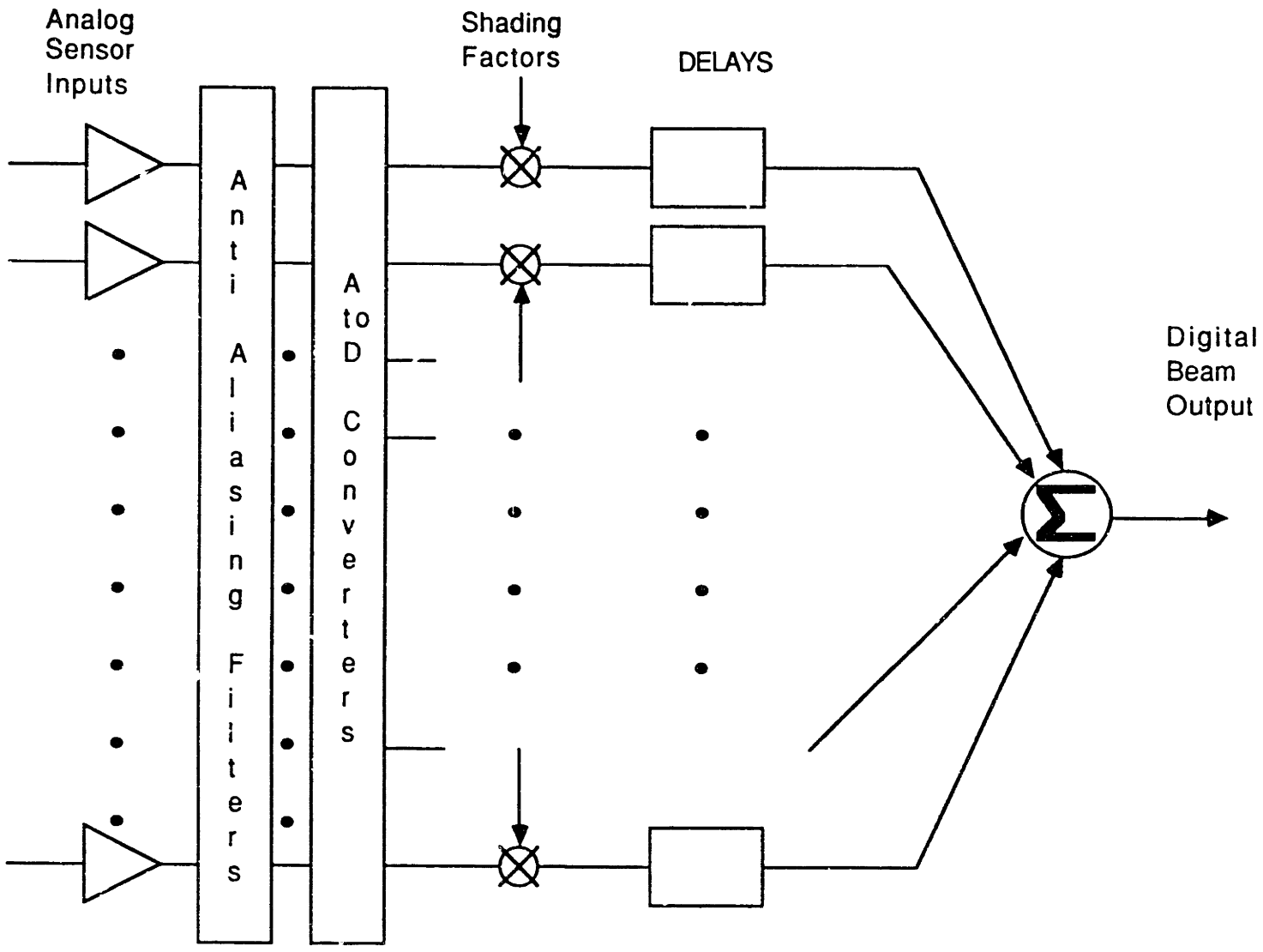


Figure 1-4 : Conventional, time-domain digital beamforming concept

In order to describe this operation mathematically, let us assume that the array of receivers is composed of a three dimensional distribution of equally weighted omnidirectional sensors. Their spatial locations are specified in the Cartesian coordinate system of figure (1-5). The beamforming task consists of generating the waveform $b_m(t)$ (or its corresponding sample sequence) for each desired steered beam direction B_m. Each $b_m(t)$ consists of the sum of suitably time delayed replicas of the individual sensor outputs $e_n(t)$. The time delays compensate for the differential travel time differences between sensors for a signal from the desired beam direction.

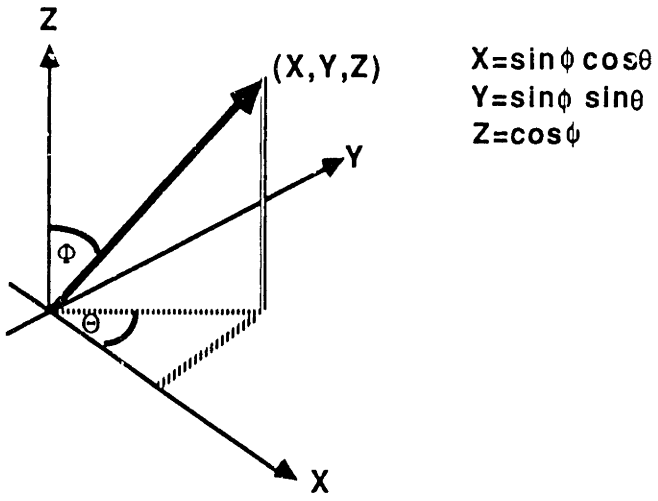


Figure 1-5 : Coordinate system definition

Let the output of an element located at the origin of coordinates be s(t). Under the assumption of plane wave propagation, a source from direction S_l (the lth source direction unit vector) produces the following sensor outputs

$$e_n(t) = s\left(t + \frac{E_n \cdot S_1}{v}\right)$$

(1.2.2-1)

where E_n is the nth element position vector and v is the speed of propagation ($v \approx 1500$ m/s) for acoustic waves in the ocean. Appropriately delaying the individual sensor outputs, to point a beam in the direction B_m, yields the beamformer output

$$b_m(t) = \sum_{n=1}^{N} e_n\left(t - \frac{E_n \cdot B_m}{v}\right)$$

(1.2.2-2)

The operation defined in (1.2.2-2) is known as beamforming. The complexity of the beamforming operation arises from the need to carry out this summation in real-time for a large number of sensors and a large number of beams.

At this point, for simplicity and further understanding of the beamforming concept, the discussion will be limited to one dimensional (line) arrays with regularly spaced hydrophones (i.e. underwater omnidirectional acoustic sensors). It will also be assumed that the beamforming is performed digitally. This discussion can be easily generalized to multi-dimensional arrays if necessary.

Assume that the presence of a plane wave signal s [t - (E . B) / v] needs to be detected. It is propagating with a known direction B , and is measured at E in a background of spatially white noise (B and E are vectors). The line array of figure (1-6) is used. The signal has the same value at each wavefront and the noise is uncorrelated from sensor to sensor. Thus, in order to enhance the signal from the noise, the sensor outputs are delayed and summed. The delays account for the propagation delay of the wavefront to each sensor. This yields

$$b(n\Delta) = \frac{1}{N}\sum_{i=0}^{N-1} X_i\left(n\Delta + \frac{E_i \cdot B}{v}\right)$$

(1.2.2-3)

where $X_i(n\Delta)$ is the sampled output of the ith sensor. Note that for the ith sensor the following relationship holds:

$$\frac{E_i \cdot B}{v} = -i\,\frac{d}{v}\sin\Theta$$

(1.2.2-4)

The input to the beamformer is a set of time series, usually one set of time

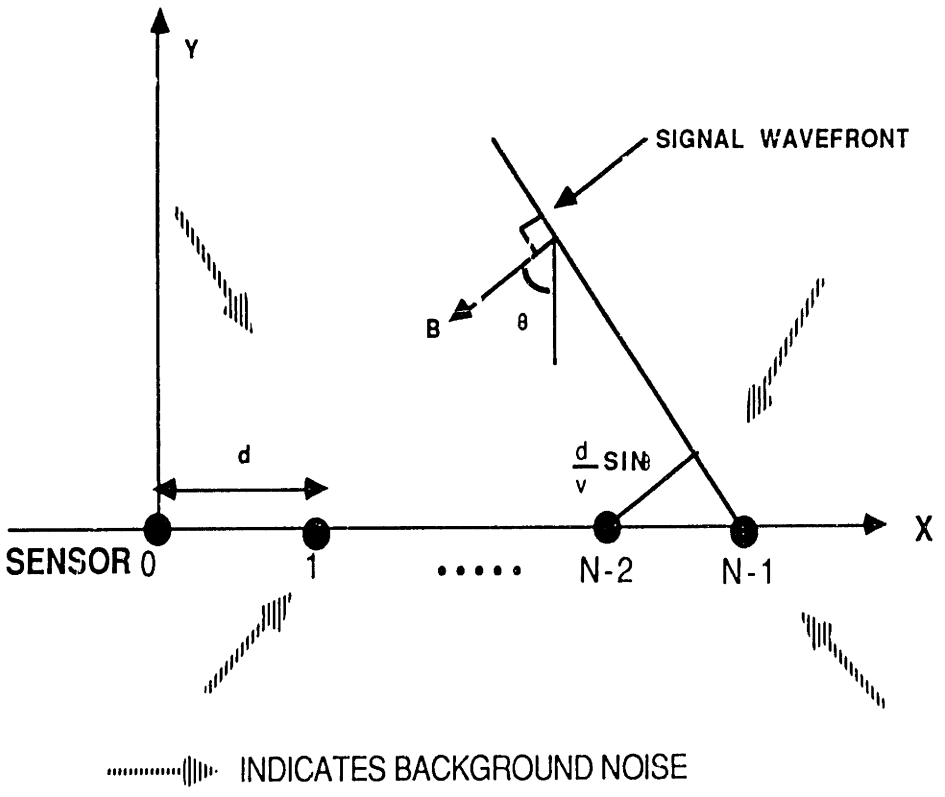


Figure 1-6 : line array of equispaced hydrophones

series for each sensor, while the output of the beamformer is another set of time series, generically referred to as beams.

The beamformer is spatially discriminating because for a plane wave with a propagation direction Θ, different than Θ_0 which assumed by the beamformer, the sensor outputs will not be coherently combined. This leads to partial cancellation of the incoming signals with $\Theta \neq \Theta_0$. Thus, for a plane wave signal

$$X_i(n\Delta) = e^{j(\omega n\Delta + \omega id/v \sin\Theta)} \tag{1.2.2-5}$$

and

$$b(n\Delta) = \frac{1}{N}\sum_{i=0}^{N-1} e^{j\omega id(\sin\Theta - \sin\Theta o)/v} e^{j\omega n\Delta} \tag{1.2.2-6}$$

where d is the spacing between the sensors. Thus the amplitude of the plane wave arriving from a direction Θ at the output of a beamformer steered to Θ_0 has an attenuation given by

$$|B(\omega,\Theta)| = \left| \frac{\sin[N(\omega/2)d(\sin\Theta - \sin\Theta 0)/v]}{N\sin[(\omega/2)d(\sin\Theta - \sin\Theta o)/v]} \right| \tag{1.2.2-7}$$

This function is known as the beam pattern of the array. Various beam patterns for line arrays are shown in figures (1-7), (1-8) and (1-9). It can be seen clearly that, for a given wave frequency, all plane waves with $\Theta \neq \Theta_0$ are attenuated, leading to the interpretation of a beamformer as a spatial filter. In most applications, it is desirable to have a beam pattern with a very narrow main lobe and very low level sidelobes for maximum noise rejection.

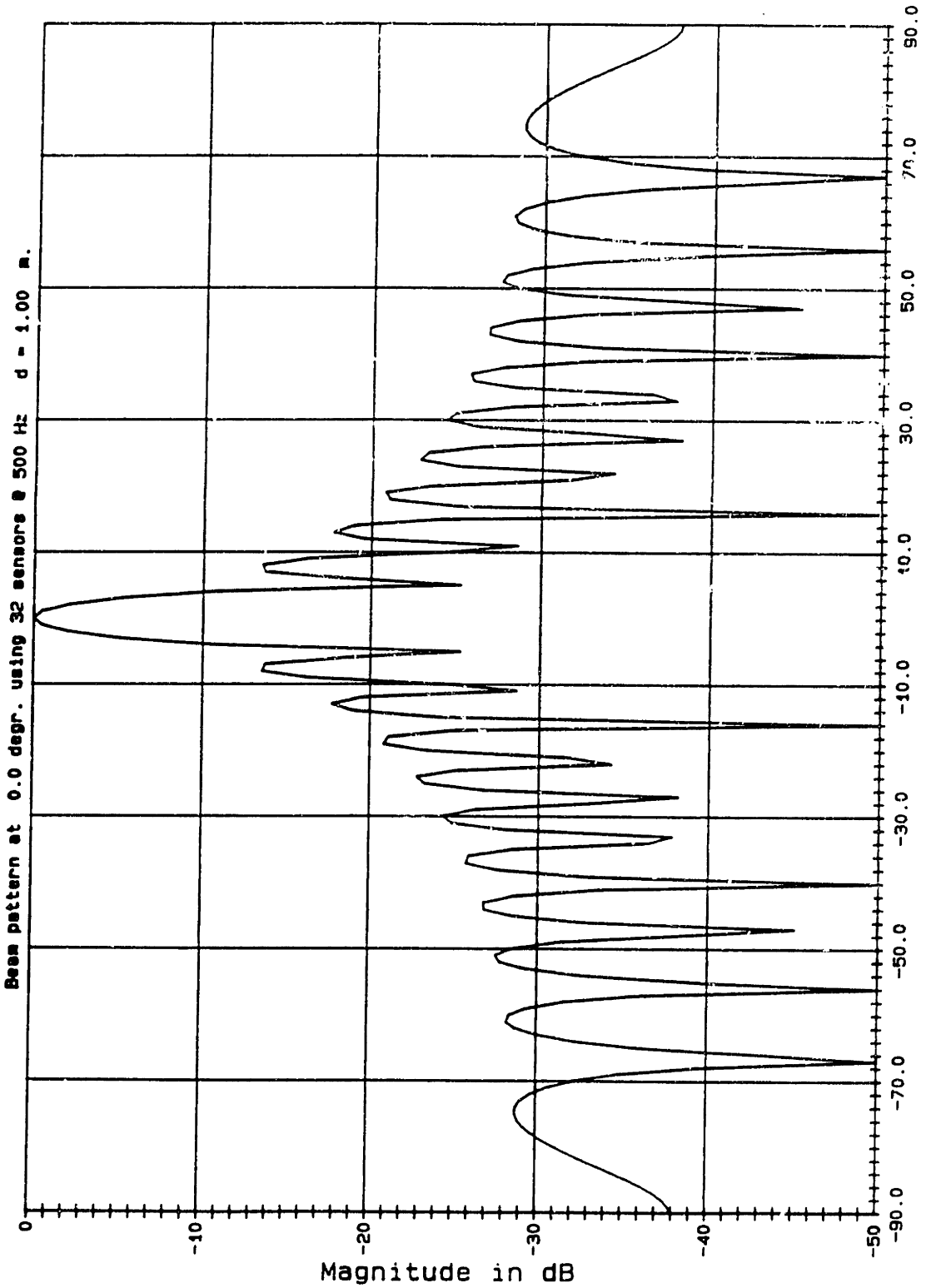


Figure 1-7 : Line array beam pattern example

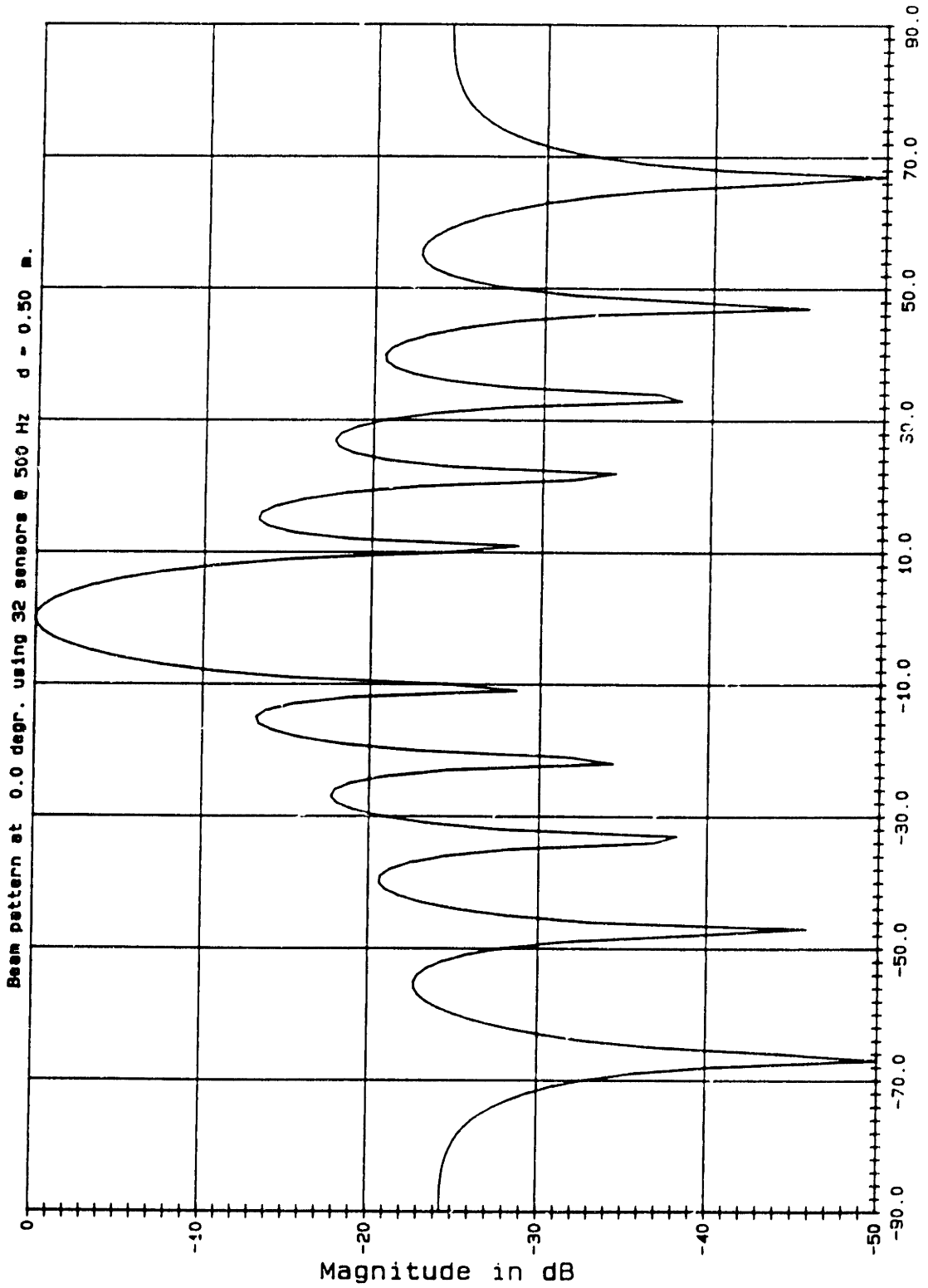


Figure 1-8 : Line array beam pattern example

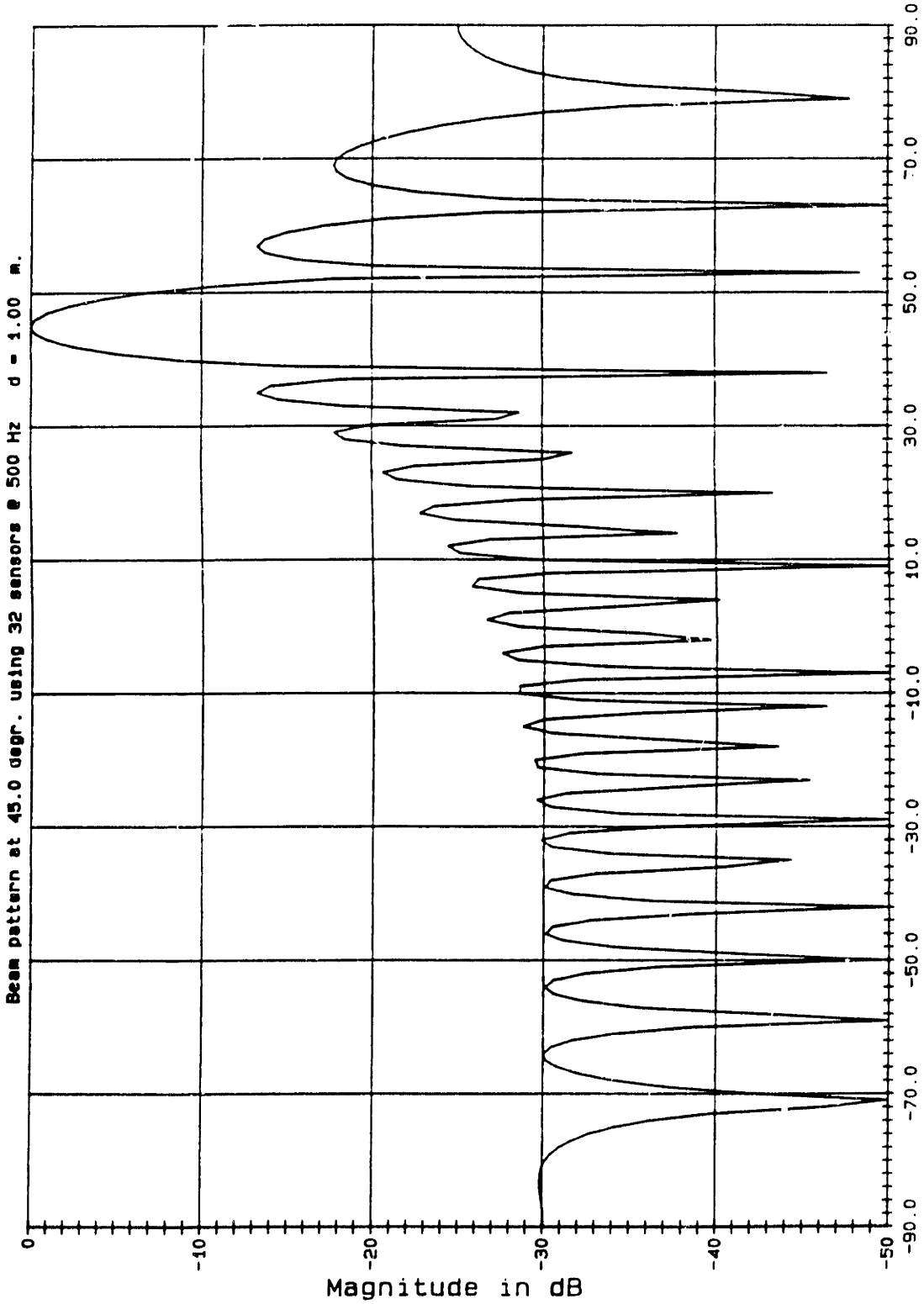


Figure 1-9 : Line array beam pattern example

There are various ways to approach the ideally desired beam pattern: Increasing ω (the operating frequency) and/or N results in narrower main lobes even though the side lobe level does not change. Increasing the sensor spacing also results in narrower main lobes. But this is limited by the fact that spatial aliasing will occur for $\Delta x > \lambda_{min}/2$ [4], where λ_{min} is the signal wavelength for the highest frequency of interest. Spatial aliasing exhibits itself in terms of extra main lobes near the endfire region as shown in figure (1-10). In order to reduce the sidelobe levels, the sensor outputs must be weighted. This procedure is known as " shading ". Thus, in equation (1.2.2-6), we replace the $1/N$ factor by w_k. The corresponding beam pattern is

$$|B(\omega,\Theta)| = \left| \sum_{i=0}^{N-1} w_i e^{j\omega id(\sin\Theta - \sin\Theta 0)/v} \right| \qquad \textbf{(1.2.2-8)}$$

The usual windowing techniques of Fourier transform theory can be used to reduce the sidelobes[4]. By employing a Hamming window, for example, the sidelobes may be reduced to -40 db at the expense of widening the main lobe.

Another problem, in a line array with fixed shading, is the quantization errors that are introduced while inserting the appropriate delays. In a digital beamformer, for ideal operation, the beamforming directions will be limited such that the delays "t" are multiples of the sampling interval. Any other choice of directions (delays) will introduce errors onto the beam pattern [5]. One method that is used to reduce such errors involves the interpolation of the incoming samples. Further discussion on this topic can be found in reference [6].

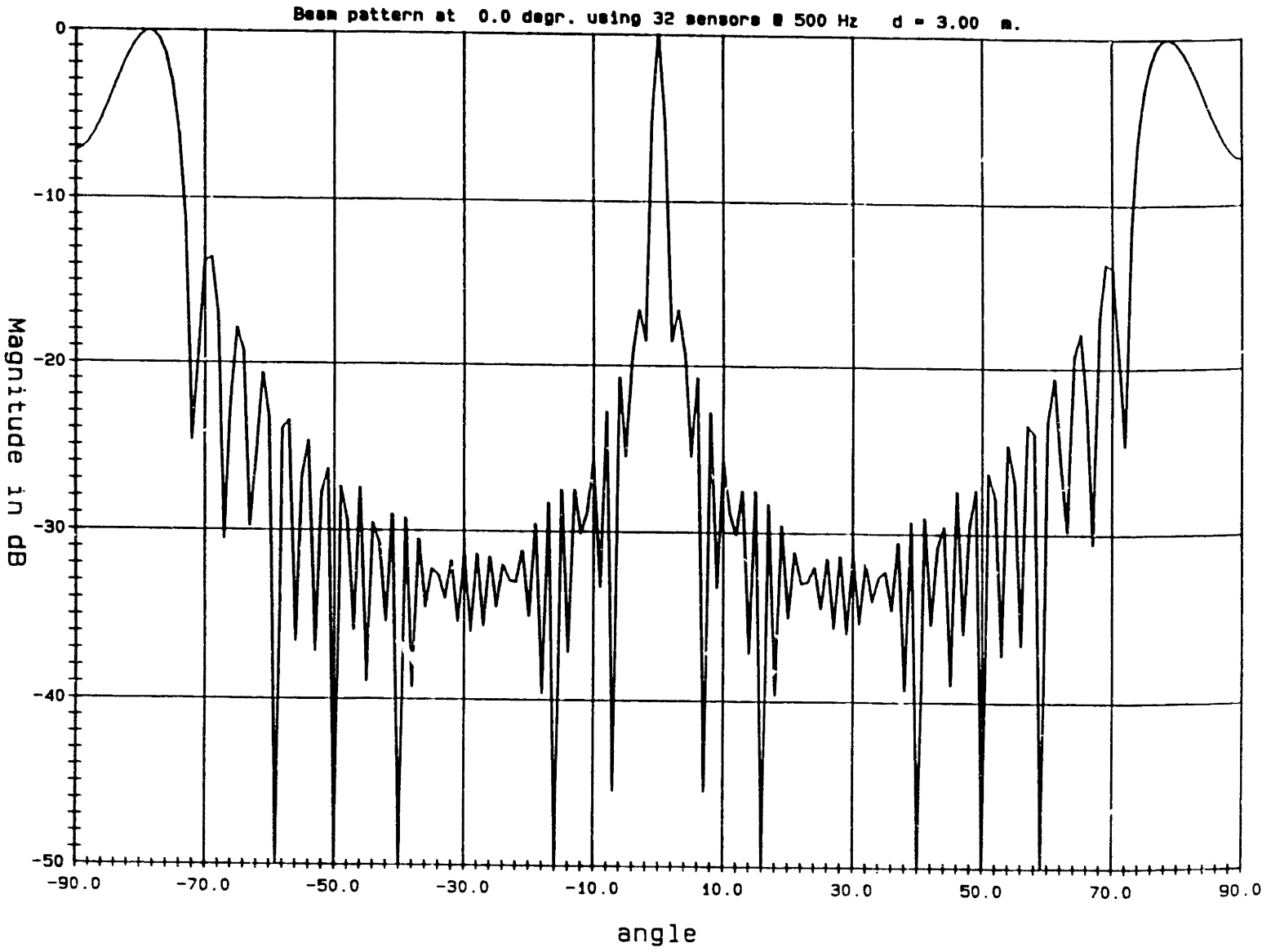


Figure 1-10 : Spatial aliasing example on the beam pattern of a line array

1.2.3 Digital Beamforming System Issues :

The beamforming task can be performed using analog or digital systems. Implementing analog tapped delay lines and attempting to form multiple beams in real-time using analog hardware results in big, non-flexible and cumbersome beamforming systems. The use of a digital beamformer results in a smaller, more accurate, and much more flexible hardware/software unit than an analog beamforming system. In this section, some of the important issues in digital time-delay beamforming systems will be discussed. The section will concentrate on the real-time beamformers.

The computational capacity required by a real-time beamformer can be roughly computed as Nf_s multiply and adds per second per beam from (1.2.2-3), where N is the number of sensors and f_s is the input sampling frequency. This does not seem very computationally demanding in its current form. However, multiple simultaneous beams are usually needed in order to span the space around the sensors and consequently, the computation demand rises to N_bNf_s multiply and adds per second, where N_b is the number of beams formed. A typical passive sonar system may utilize ≈30 sensors, require ≈30 or more beams and use sampling rates up to 10 KHz or higher, which results in a need for at least 12 million multiply and adds per second. New generations of quieter sources will require the use of more sensors (for higher resolution) and more beams, which will cause the computation demand to increase very rapidly in the coming years.

Another concern in designing real-time beamformers is the amount of storage that is needed in order to implement the digital delays. For example, in the case of a line array with sensor spacings d, the storage necessary to form all the synchronous beams is in the order of N^2f_sd/v. The " synchronous beams " are all the beams that can be formed using delays which are multiples of the input

sampling period. In the typical beamforming system that we considered earlier, the size of storage required is in the order of 10000 memory locations. The chosen memory word width in a particular application could be ≈16 bits or more, which would require at least 20 Kbytes of RAM per beam (unless shared). The demand for fast accessible storage will be rising in the coming years along with the demand for computation power. Advances in VLSI memory technology have managed to somewhat keep up with the demand so far.

Current real-time digital beamforming systems in the market can be classified into two groups: One class uses hardware and software that has been designed for a very specific application. The result is that those systems tend to be very compact and efficient but also inflexible and expensive. The other class uses general purpose array processors and software that are generally very flexible but yet very costly, bulky and inefficient. An efficient compromise between flexibility, computational power and cost is almost non-existent.

There are a number of applications that require beamforming systems which can handle sensor arrays of arbitrary arrangement. This increases the demand for a powerful real-time beamformer that is flexible enough to let the user specify the arrangement of the sensors in water. In addition to this feature, a beamforming system which also lets the user pick a large number of arbitrary beamforming directions is desirable. A friendly user interface, obviously, would be helpful during user's system configuration in a particular application. A low price tag and small physical size are also very important factors that would satisfy all possible users. Further discussion on beamforming system issues can be found in references [4],[7],[8].

1.3 Specifications for an Experimental Beamformer for Oceanographic Research :

Today, there are a large number of oceanographic and military experiments that are being performed by collecting acoustical underwater data. One area for such applications is the Arctic. The passive sonar systems used in these hostile environments need to be small and power efficient due to a number of logistical reasons. Given their physical constraints, these systems need to maximize their computational power. All the building blocks for such a system must comply with these demands.

A beamformer is needed for use in an Arctic passive sonar system. It must be able to take inputs from an arbitrary array of up to 32 sensors, form multiple beams in real time (≈ 30 beams) and be user configurable from an IBM AT personal computer (or compatible). The acoustic frequencies of interest, in this case, range from 0 Hz to 2000 Hz. The input sampling rate is required to be 10 KHz. The rate is higher than the Nyquist rate because of the need for a higher resolution in the beamforming directions. The acoustic data is collected by hydrophones and the analog data must be digitized to 12 bits. The gain applied to the analog inputs must be manually selectable (1 - 1000). The output beams must be sent to another system for further processing. This has to be done over a specific parallel interface. The output beams must be also available at an analog output, one at a time, for testing, monitoring and some other signal processing tasks. The beamformer must be mountable on a 19" rack.

CHAPTER 2 : System Design and Operation

The details of the design for the experimental high performance research beamformer that was specified in 1.3 will be discussed in this chapter. The discussion includes hardware, firmware and software design issues as well as system test procedures and results.

2.1 System Architecture :

The architectural goals, the building blocks and the overall operation of the system are presented in this section.

2.1.1 Architectural Considerations :

There are several important architectural issues to be considered before designing this beamforming system :

One important consideration is the need for modularity. Some users may be interested in using fewer than 32 sensors and/or forming only a few beams. Such users should not be forced to buy a system which exceeds their needs, but they should also have the option of expanding their system's capabilities if necessary.

Another consideration is the demand for speed. As discussed earlier, in order to form multiple simultaneous beams, a large number of summations have to be computed in real-time. This requirement, along with modularity, leads one to consider distributed processing architectures.

One more consideration is flexibility and ease of use. The users must be able to specify any array configuration that they desire and form any beam that they may need. This requires a friendly and interactive user interface, through which the users can specify many different system parameters.

The architecture of this beamforming system must be able to accomodate all

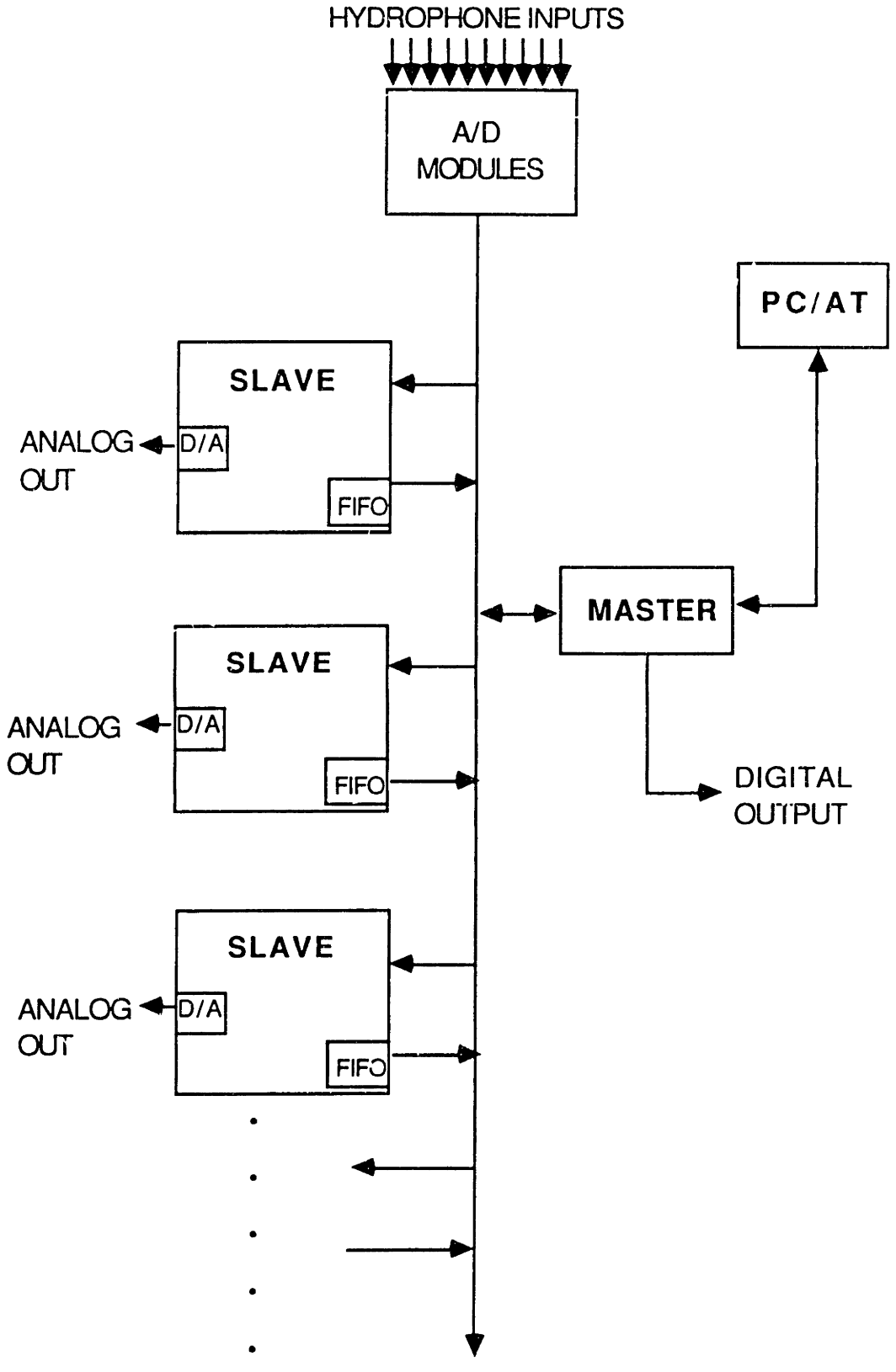


Figure 2-1 : General architecture for the experimental sonar beamformer

of the considerations and goals listed above.

2.1.2 The Building Blocks :

The experimental beamformer consists of several building blocks that are explicitly shown in figure (2-1). There are several parallel buses in the system for data transfers and communication: A wide common bus is used for the subsystems to communicate and exchange data with each other. Another bus facilitates the communications with the IBM AT. This bus is used to download the user system configuration data from the AT into the beamformer. Finally, a bus is used to send the output beams to a sonar signal processing system which performs further processing on the data. There are three main types of subsystems in this beamformer: One is the analog to digital conversion (A/D) module. This module takes the hydrophone outputs as inputs, is responsible for sampling the incoming analog signals and converting them into a digital format. Each A/D module can handle a limited number of hydrophones, but more modules may be attached to the common bus in order to handle a larger number of inputs. Another subsystem is the "Master" module, which is responsible for controlling the data exchange among all internal modules and the data flow over the I/O buses. The final type of subsystem is the "Slave" module, which is responsible for the actual beamforming task. Each slave can beamform in multiple directions and more slaves may be added in order to form a larger number of beams.

2.1.3 System Operation :

The internal operation of the system is strictly controlled by the master. The handling of the incoming samples and the I/O exchanges are also under the master's control. The operation of the system is synchronized to the input sampling clock which has a period of 100 microseconds (10 KHz). This implies

that the calculated beam samples have to be sent out every 100 microseconds. This duration can be called a system cycle. Thus, the master has to go through its duties within one system cycle and be ready to handle the next set of incoming samples. In addition to its regular cyclic duties, the master has to accomplish a one time task which is the handling of the system configuration data sent from the AT. The sequence of the system events is illustrated on figure (2-2) and it is as follows:

(1) The system configuration variables (e.g. number of sensors, beam directions etc.) are entered into the AT by executing an interactive program. The same program, after doing some calculations, downloads the necessary data to the master. The master/AT communications are accomplished using a simple ad hoc protocol over a parallel interface card located on the AT's backplane. The master keeps the necessary configuration variables in several of its internal registers. The next step is to send this information to all the slaves in order for each of them to identify the beams for which they are responsible.

(2) The master, after having sent this information, waits for a signal from each one of the slaves confirming that they are ready to beamform. Once all the slaves are ready, the master starts the cyclic operation of the beamformer by responding to an interrupt initiated by an A/D conversion (the sampling clock had been running but the conversions were not being recognized until now).

(3) At this point, the master starts to read the results of the A/D conversions, which correspond to a simultaneous snapshot of the incoming waveforms at the hydrophone locations. It reads all the results in sequence and writes them into the memories of all the slaves. Thus, all slaves receive identical copies of the incoming waveform samples. These samples are kept in circular buffers (255 slots) which are located in each slave's data memory space. New

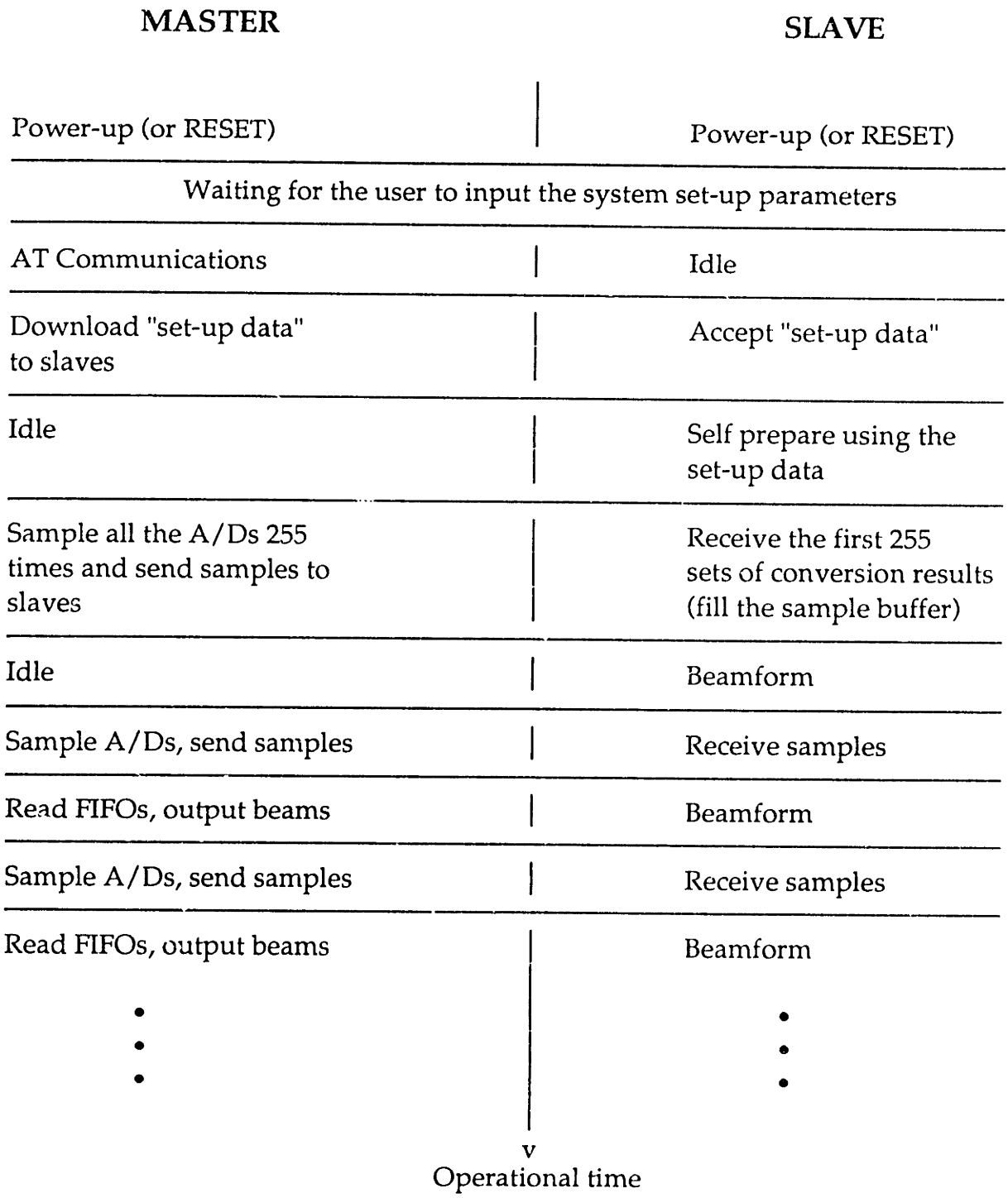


Figure 2.2 : The sequence of system events

incoming samples are put into consecutive locations in this buffer. Once the end of the buffer is reached, the oldest snapshot of samples get written over by the newest samples and this cyclic process goes on. The consecutive snapshots of the waveform provide the slave with the delayed samples that are needed in time-delay beamforming.

(4) As soon as all the new current set of samples are stored into the sample buffer, the master initiates an interrupt which orders all the slaves to start beamforming.

Each slave performs the beamforming task by reading the appropriate locations in its sample buffer, by multiplying those values with a shading factor and by keeping a running sum of these weighted samples until the summation is finished. Once a summation is finalized (i.e. a beam sample is formed), the result is shifted into a First-in First-out memory (FIFO) for further collection by the master. Each slave has its own dedicated FIFO and only the master can shift the beam samples out of the FIFOs. Each slave computes and stores its own beam samples independent of other slaves. Once a slave has completed its assigned set of beams, it waits for the next interrupt (initiated by the master) that orders all the slaves to beamform again. While the slaves are busy forming and storing the current set of beam samples, the master reads the set of beam samples that were formed during the previous system cycle. The master reads and sends each beam sample, one at a time, over its output bus. After finishing its output duties, the master waits for the next A/D interrupt.

The user must realize that the first 255 sets of beam samples produced are invalid! This is due to the fact that the sample buffer is not full until the end of the 255th system cycle. Therefore, during that period, the locations read by the slaves contain meaningless data. The actual valid system outputs are produced

≈25.5 ms. later (this is negligible compared to long hours of system operation).

There are a number of important operational timing issues due to the length of the system cycles. The number of different beam samples that can be formed by each slave is limited by the system cycle length. This constraint exists because the slaves have to release the control of their individual memory buses in order to allow write operations by the master. Another constraint is that the master needs to read all the incoming samples and also send all the beam samples out within a system cycle. The maximum number of beams that can be formed in this system are directly limited by these timing constraints. The beam allocations per slave must be calculated carefully during the system configuration phase. Otherwise, incomplete beams and invalid outputs may result because of the master not having enough time to send out all the beam samples or other complications.

Another feature that is provided is the ability to observe some beams through analog outputs. Digital to analog (D/A) converters are included on each slave board and the desired analog beam output can be selected using a thumbwheel switch. Each slave sends the desired beam sample to its D/A before shifting it into the FIFO. The overhead caused by adding this analog port is minimal and it is a very useful test point for system debugging.

As one can conclude, the minimum system configuration consists of a master, a slave and an A/D module. On the other hand, the maximum possible system configuration is limited by the speed of the internal hardware and the maximum data rate capability of the output port. All buses, clearly, must be present in any system configuration. The exact capability of different system configurations will be discussed in later sections.

2.2 System Hardware :

Because of the high performance requirements, the system hardware includes state of the art VLSI digital signal processors, high speed hybrid A/D converters and very high speed CMOS and Bipolar LSI components. Hardware selection, design, timing, operation and interfacing issues in the system are discussed in the following sections. An estimate of the total system cost is given in appendix E.

2.2.1 Component Evaluations and Selection :

The system needs several critical fast components that require careful selection. The first such component is the Central Processing Unit (CPU) that is needed in the slave modules. Due to space limitations on the slave boards, the microcoded control units are not feasible options and the selection will be made among single chip processors. There are a large number of VLSI single chip processors on the market, and we can roughly classify them into three groups: The first are the general purpose micro-processors; the second are the general purpose micro-controllers and finally the digital signal processors (DSP). For this application, the DSPs are clearly the CPU of choice due to their fast multiply and add times. The decision among such processors is the next step. In this application, almost all of the commercially available DSPs are eligible since the required resolution on the incoming data is only 12 bits. Speed is the criterion that eliminates the eligibility of most DSPs for this application. The choice, finally, is narrowed down to Texas Instruments' TMS-32020, NEC's upd77C20 and Analog Devices' ADSP-2100. Other DSPs and vector processors that can handle wider data word widths are not considered because those luxuries are not beneficial in this case. The following is the list of the features that were compared before choosing the ADSP-2100:

(1) The cycle times are: 200 ns. for TMS-32020; 250 ns for upd77C20 and 125 ns for ADSP-2100.

(2) ADSP-2100 is the only among the three processors to have single cycle access to all of its external memories.

(3) All processors have single cycle multiply-accumulate instructions but only the ADSP-2100 has a 40 bit result field.

(4) ADSP-2100 provides powerful circular addressing capabilities, whereas the other two processors need special techniques in order to implement similar addressing modes.

(5) The assembly languages used by the Texas Instruments and the NEC processors are cumbersome. The one for ADSP-2100 is clear and easy to learn.

(6) The projected cost for 1988 (on 6/87) was $150 for the ADSP-2100, $100 for the TMS-32020 and $60 for the upd77C20.

Clearly the TMS-32020 and the upd77C20 are lacking a number of strengths when compared to the Analog Devices' processor. The extra cost of the ADSP-2100 is appropriate since it outperforms the other contestants. ADSP-2100 also manages to fulfill the CPU requirements of the master module. It provides easy handling of memory mapped peripherals and can handle four external interrupts. These features and the savings in the design time (it already is being used as the slave CPU), again, balance its extra cost versus other available high speed micro-controllers. More detailed information about ADSP-2100 is available in appendix D and references [9],[10].

The next set of critical components to be selected are the ones to be used in the A/D modules. These include 12 bit fast A/D converters, high precision sample and hold circuits, very low noise operational amplifiers to be used in the input gain section and the anti-aliasing filters. The prototyping of a front-end

analog signal conditioning and A/D conversion section using discrete components would be a very difficult task in such a noisy digital environment. A hybrid A/D converter with on board voltage references along with sample and hold circuits can perform better than any discrete prototype circuit with similar functionality. Thus, after some research the Analog Devices' AD-1332 is found to be most appropriate and convenient. It integrates several of the necessary components inside : Its central element is a 12 bit 5µs AD-7672 A/D converter. A voltage reference for the converter is included. The converter is preceded by an AD-585 sample and hold circuit which itself is preceded by an optional 4 pole Butterworth low-pass filter (anti-aliasing filter). The converter output is fed into a 12 bit latch with tri-state output buffers (an optional integrated FIFO is also available for the temporary storage of the conversion results). The AD-1332 is very easily addressable from a micro-processor, which makes it ideal for this application. Some more detailed specifications for the AD-1332 is given in appendix D. The availability of this hybrid also helps reduce the system design and prototyping time, since this circuit must be duplicated for all incoming sensor inputs. Multiplexing the sample and hold outputs into fewer converters is not desirable because of system performance considerations.

The selection of the rest of the high speed VLSI and LSI components in the system is relatively easier. Several levels of address decoding and buffering that are present in the system result in high demands on the memory components. Integrated Device Technologies' (IDT) 2K x 8 CMOS static RAMs with 25 ns. access times are used as the data memory components on the slaves. The program memories for the master and the slaves also need to be very fast. Cypress Semiconductor's 2K x 8 CMOS EPROMs with 35 ns. access times are used as the program memory components for all CPUs. The next set of important high speed

components to be chosen are the slave FIFOs. IDT's 72413L35, 64 x 5 CMOS FIFOs are used for this purpose. Some more details about the FIFOs can be found in appendix D. Spatially convenient high speed 64 x 8 FIFOs were not available at that time. In order to provide an analog output port on the slaves, Analog Devices' 16 bit D/A converter AD-569 is used. The rest of the LSI components that are used are off the shelf Advanced Schottky (Fairchild's FAST & Texas Instruments' 74AS series), Advanced Low Power Schottky (Texas Instruments' 74ALS series) or very high speed CMOS (IDT's 74FCT series) integrated circuits. More detailed information and specifications for these components are available in their manufacturers' data books.

Another evaluation that had to be made was for the interface card between the AT and the master. A parallel I/O card that can easily be plugged into the AT's backplane and that can easily be addressed from a high level program was desirable. There are a large number of such I/O boards but most of them have very well defined I/O protocols that have to be obeyed. The allowed development time for the design was relatively short and this discouraged spending effort on a complicated protocol. A simple non-standard protocol and a very flexible I/O card was needed. Analog Devices' RTI-817 parallel I/O board was the appropriate choice. The board contains three 8 bit bi-directional ports that accept user configured directions. These ports are memory mapped and addressable from high level programs. More detailed information about the card can be found in the RTI-817 User's Manual published by Analog Devices' IAD Division.

2.2.2 Master Board Hardware :

The details of the circuitry on the master module (board) is discussed in this section. The hardware schematics for the master module and its interfaces are

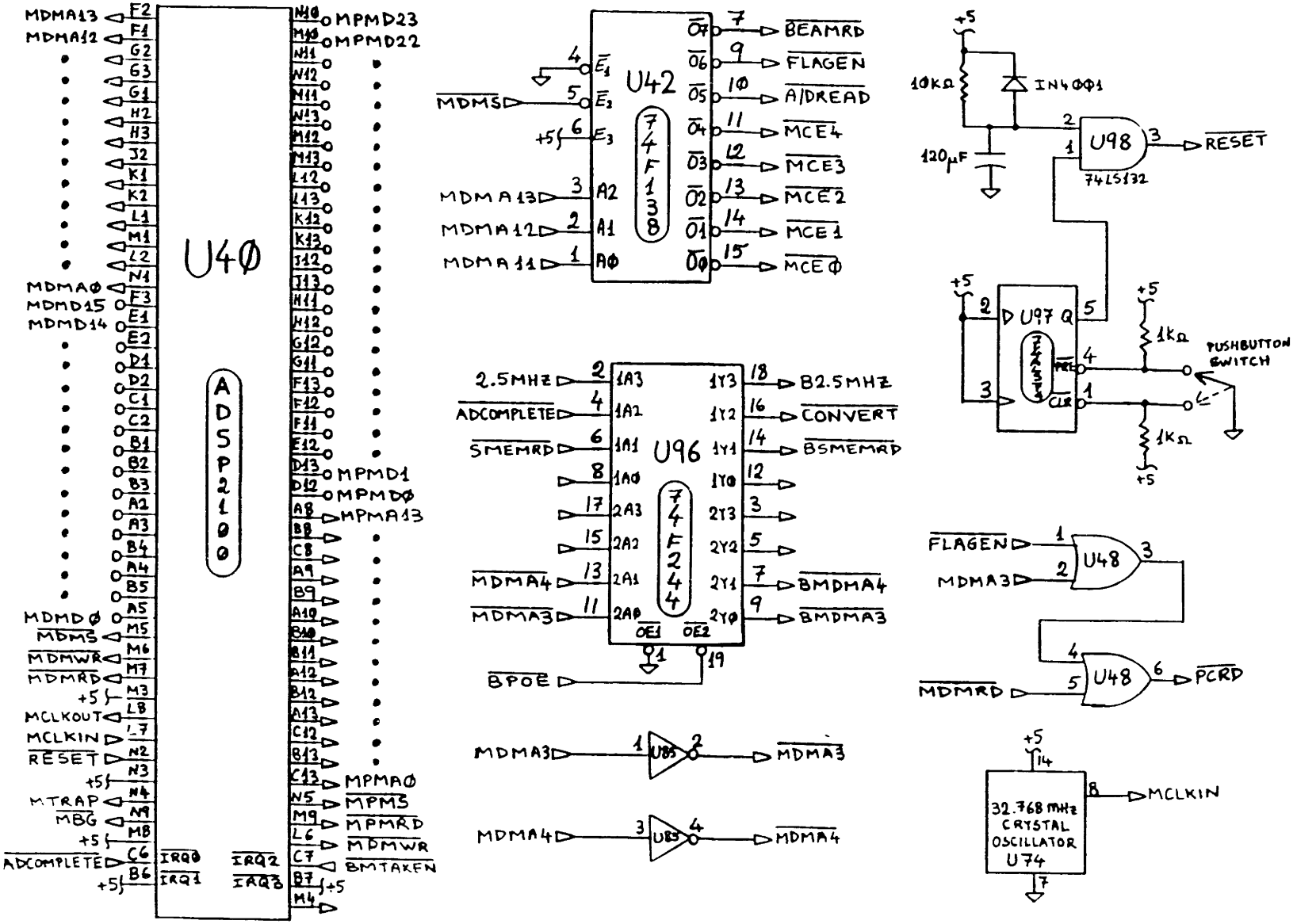


Figure 2-3 : Master CPU and its main decoders

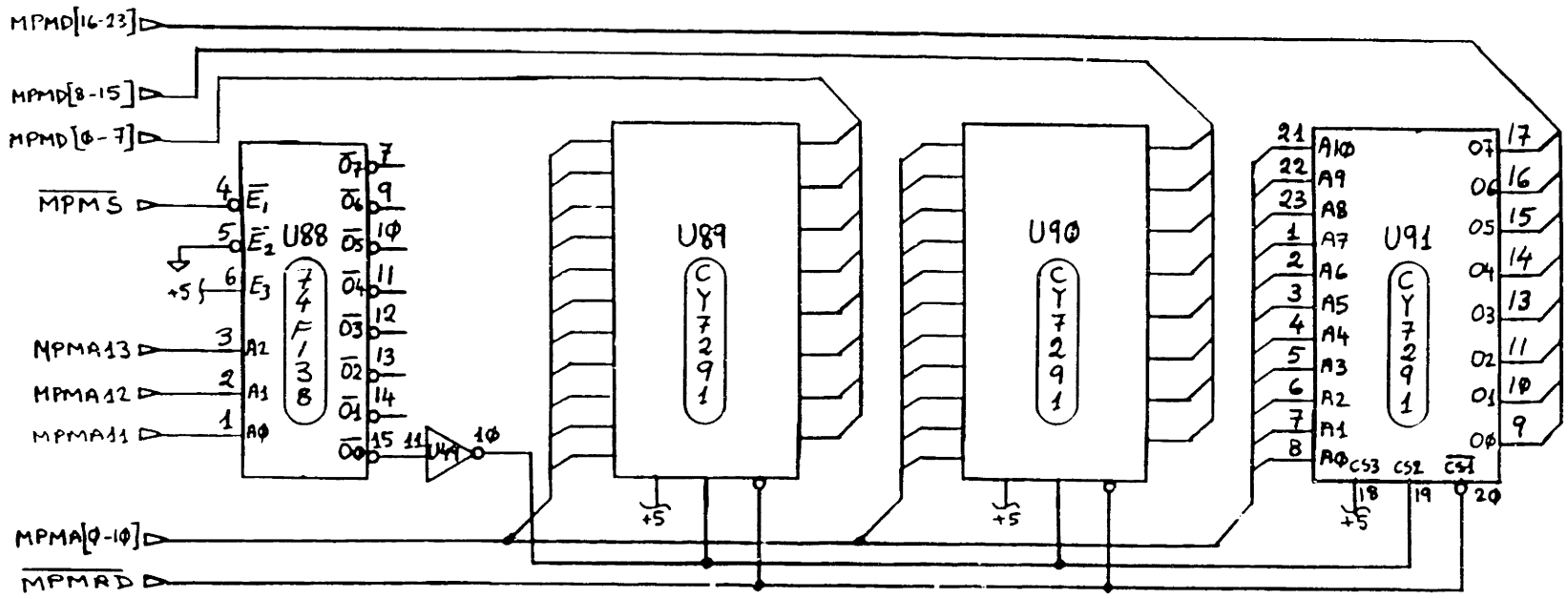
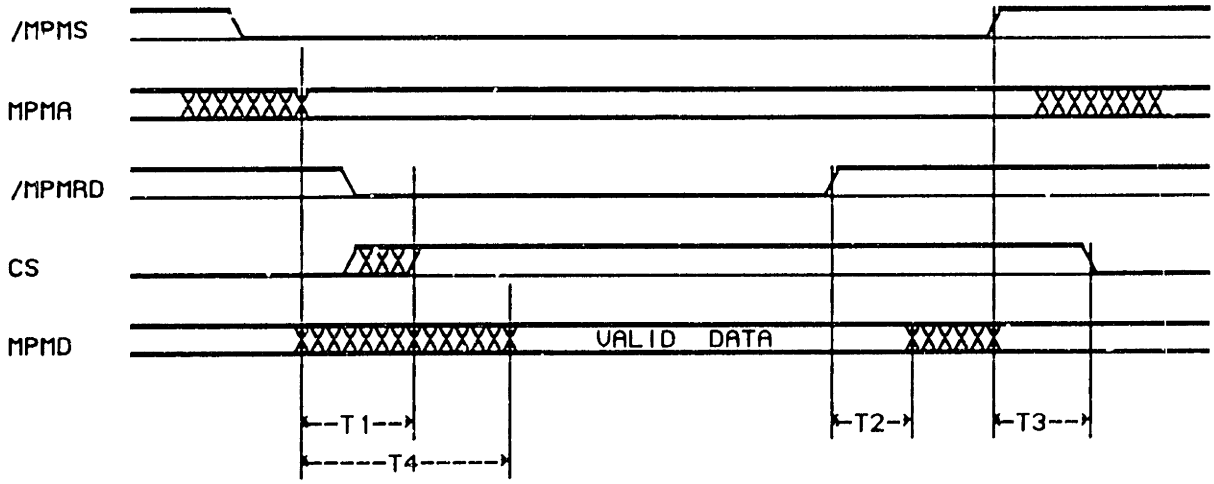


Figure 2-4 : Master program memory (ROM)

shown in several figures among the pages of this section. All components shown on the schematics have a unique " U " number associated with them for ease of referencing.

The circuitry on the master board is centered around the CPU which is an ADSP-2100 (U40). The master CPU takes its instructions from three CY7291-35 2Kx8 EPROM (U89-U91) over the Program Memory Data (PMD) bus. These devices are mapped to the CPU's Program Memory Adress (PMA) space. A 74F138 decoder (U88) serves as an address decoder on the PMA bus. The timing diagrams for the execution of a CPU program memory read cycle from the EPROMs is shown in figure (2-5). The master board does not contain any data memory. The CPU's internal registers are sufficient for most operations except during the configuration phase. The details of how this problem is handled will be discussed later in the firmware design section.



T1 = 15ns max. T2 = 25ns max. T3 = 14ns max. T4 = 35ns max.

Figure 2-5 : Master CPU program memory read cycle (from ROM)

The master recognizes two interrupts : The first one is the /ADCOMPLETE interrupt, which occurs every 100 µs. and notifies the master about the availability of new A/D conversion results. This interrupt comes directly from the sampling clock. The second one is the /BMTAKEN interrupt which notifies the master that the external sonar signal processor has received a beam sample and is ready to receive the next one. The inputs /MBR (master bus request) and /MHALT (master halt) are tied high because no other component is allowed to control the master

Another set of devices generate the sampling clock that is used to sample the analog sensor inputs and to initiate the A/D conversions (see figure 2-6). These are three 4 bit counters (U66 - U68) and a 5.12 MHz oscillator (U75). There is also another 4 bit counter (U70) and a 5.0 MHz oscillator which generate the conversion clock input for the AD-1332s. The /CONVERT (10KHz) and CLKIN (2.5MHz) are sent over the backplane to the A/D boards. The /ADCOMPLETE is the same signal as /CONVERT, different names are used for easy understanding of component functions.

The master board also has three 74F138s (U41,U42&U94) used for data memory address bus (DMA) decoding. These decoders provide the master CPU with decoded DMA lines (MCE0 - MCE4) in order for it to access the slave data memories. The /A/DREAD signal is used to enable some decoders on the A/D boards which enable the output latches of the AD-1332s. /BEAMRD is used to read the FIFOs on the slave boards. /FLAGEN is used to generate a series of control signals and flags which are the following : /SBR which is used to signal the slave CPUs to surrender the control of their DMA and Data Memory Data (DMD) buses. This is necessary in order for the master CPU to take control.

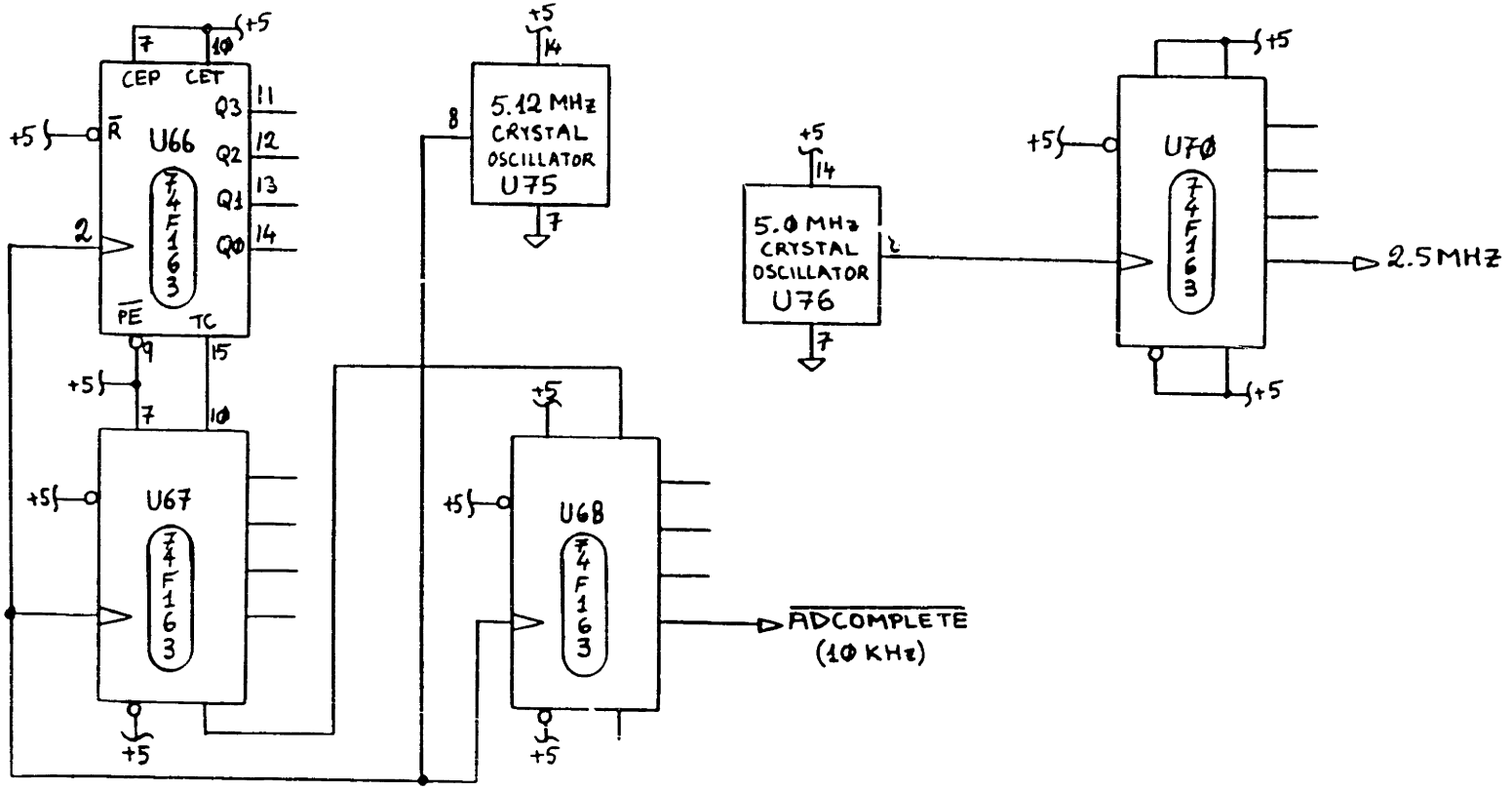


Figure 2-6 : Sampling clock generation on the Master board

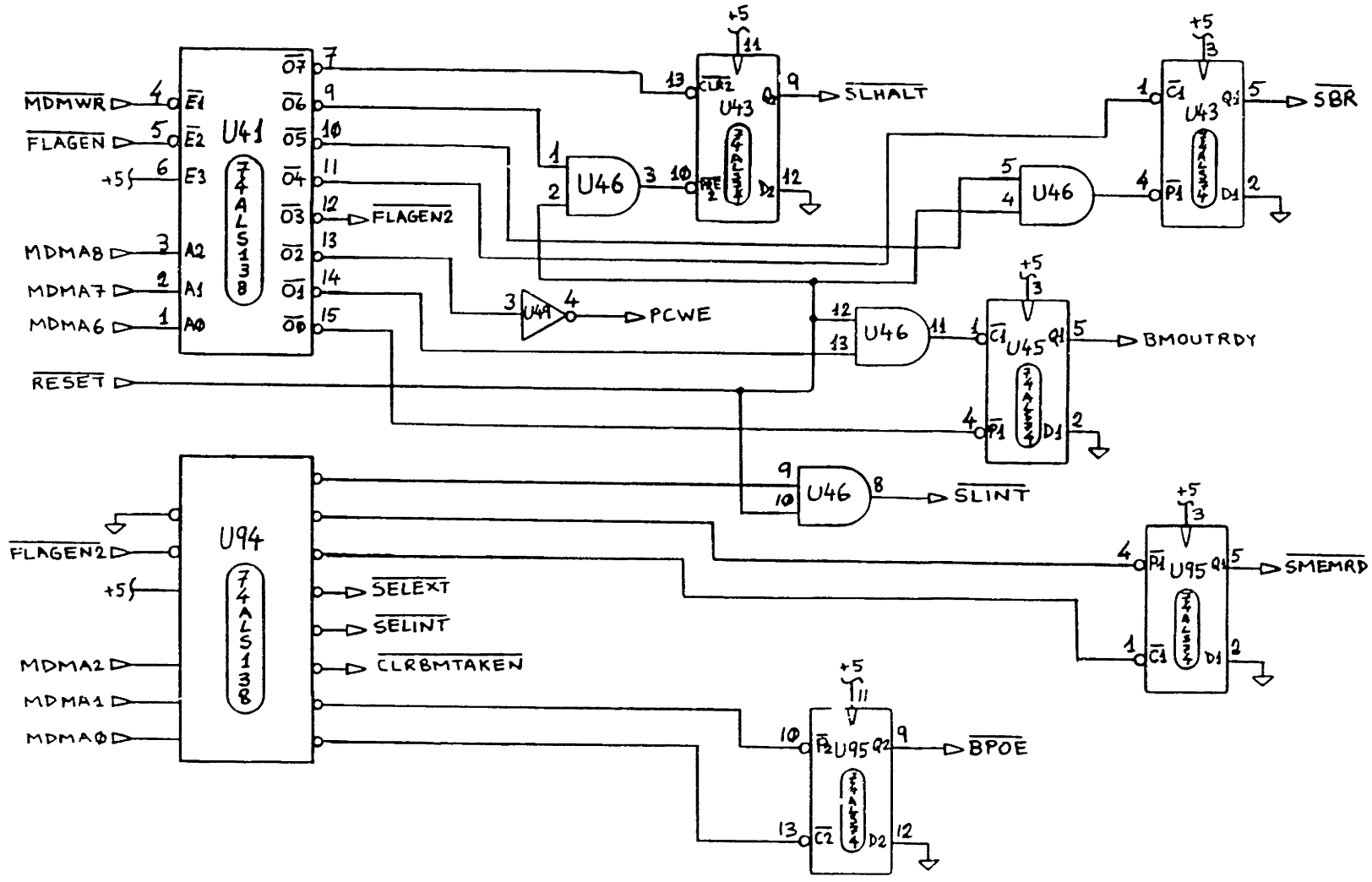


Figure 2-7 : Control flag generation on the Master board

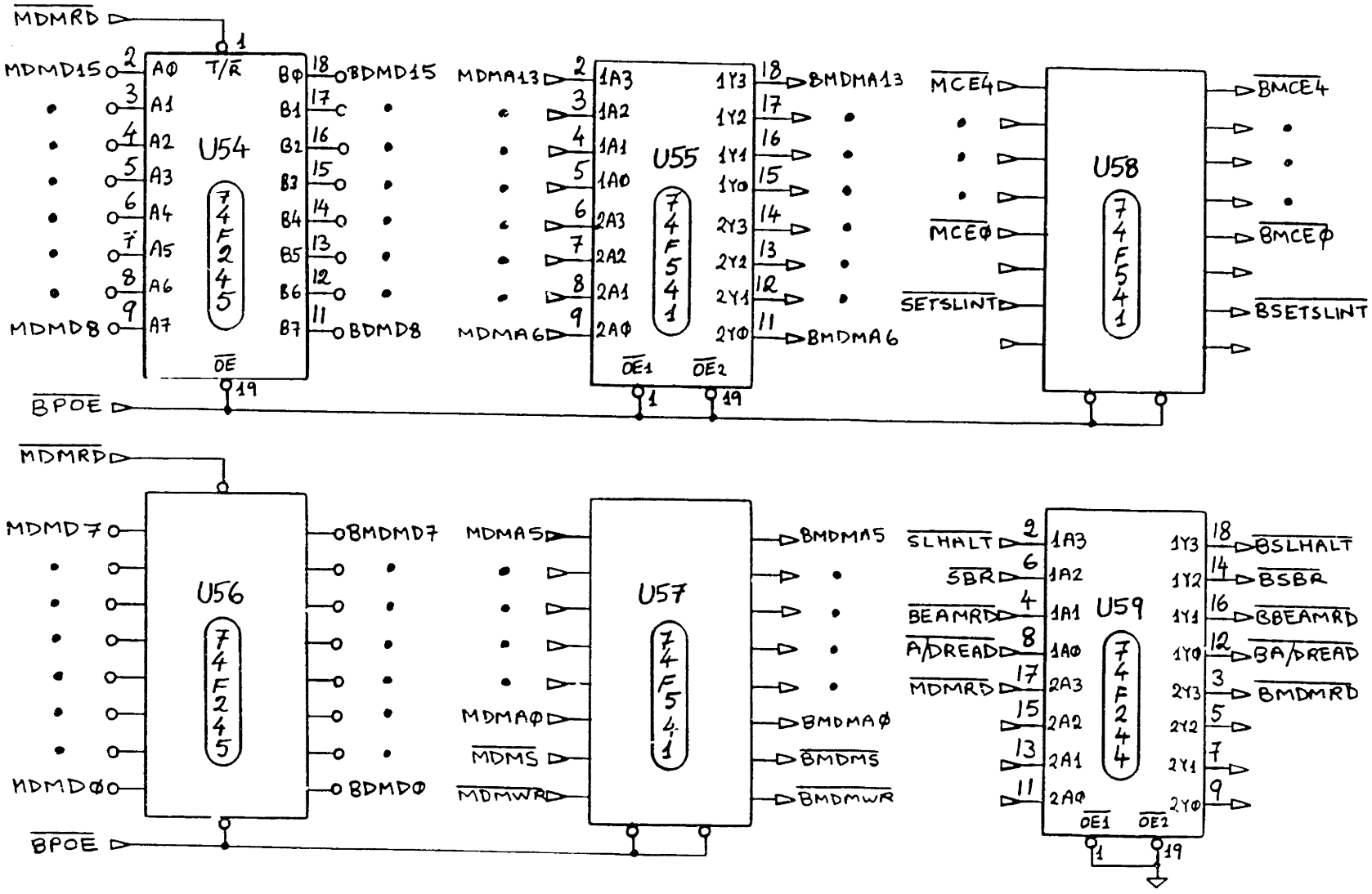


Figure 2-8 : Backplane drivers on the Master board

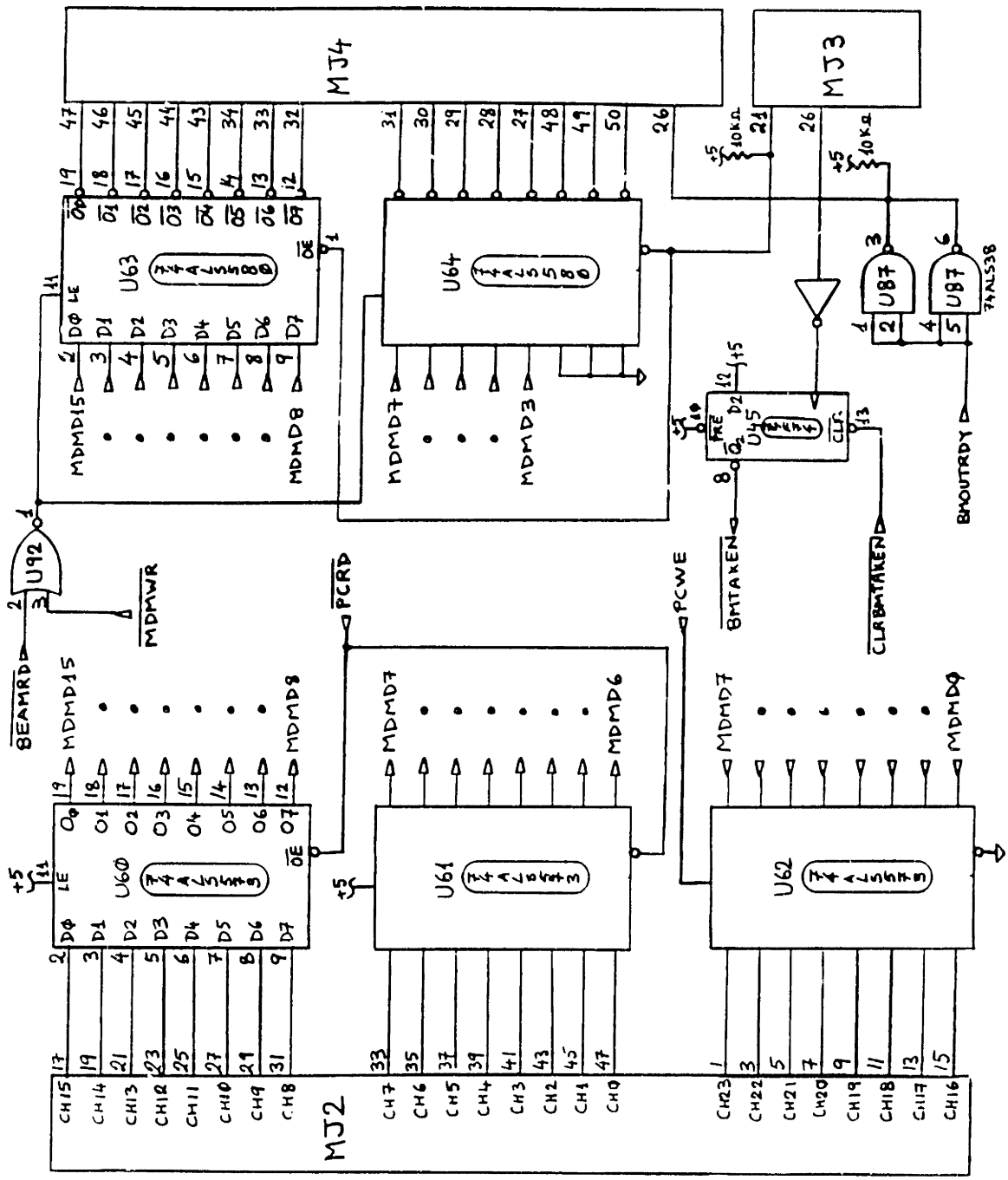


Figure 2-9 : Master/AT & Master/External Processor interfaces

/SLHALT is used to halt all the slaves or to bring them back into operation if they were in a TRAP state. BMOUTRDY is a flag sent over the output bus to notify that a beam sample is ready to be transmitted. /CLRBMTAKEN is a signal that is used to clear the BMTAKEN interrupt which is set by the external sonar processor. /PCRD and PCWE allow the master to read and write to the AT communications bus during system configuration downloading. These signals enable the input and output latches on the AT communications bus. The /SMEMRD signal sets the direction of the data transfers between the slaves and the backplane bus (common bus). /BPOE is the signal that enables or disables the backplane bus drivers on the master board. /SLINT line is used to interrupt the slaves. The /SELEXT and /SELOSC lines are not currently used. These lines are reserved for selecting a possible external sampling clock option.

The master board also contains a large number of devices dedicated for the CPU's external bus interfaces. A bank of bus drivers and transceivers (U54 - U59) are used to provide the necessary buffering for the signals that are traveling over the backplane bus. The names of the buffered signals (on the backplane side) are always preceded with a " B ". For example, MDMD0 becomes BMDMD0 and so on. The AT communications are handled through three octal latches (U60-U62). These latches provide direct interfacing (over a flat parallel cable) to the parallel I/O board that is plugged into the AT's backplane. This board also carries two inverting octal latches (U63 & U64) which facilitate the beam sample transfers over the external output bus.

The master also houses the system reset circuitry (U97 & U98) which generates the /RESET signal. This signal is used to initialize all CPUs , all A/D converter hybrids and all latches that are used for flag setting purposes. A push-button switch is placed on the board to facilitate a manual system reset.

Finally, the interfacing to the three separate buses requires 4 separate connectors to be placed on the master board : A 96 pin Eurocard connector and is the connection to the backplane, a 50 pin flat cable connector (MJ2) which is used between the master and the AT, a 28 pin (MJ3) and a 50 pin (MJ4) flat cable connector which are used on the output bus connections between the master and the external sonar signal processor. The master only requires a +5 Volt digital power supply for proper operation.

2.2.3 Slave Board Hardware :

The details of the circuitry on the slave boards (modules) are discussed in this section. Hardware schematics for the slave board and its interfaces are shown in several figures on the following pages.

The circuitry on the slave board is centered around the CPU, which is an ADSP-2100 (U27). The CPU takes its instructions from three CY7291-35 2Kx8 EPROMs (U5 - U7). These are mapped into the PMA space of the slave CPU and the instructions are made available over the 24 bit wide PMD bus. Two 2Kx8, 35ns static RAMs (U3 & U4) are also mapped onto the PMA space and available for data storage using the upper 16 bits of the PMD bus. A 74F138 (U8) decoder provides the necessary address decoding for the PMA lines. The timing diagrams for the slave program memory access cycles are given in figures (2-10), (2-11) and (2-12). There are ten IDT6116LA-25 2Kx8 static RAM chips (U1,U2 &U10 - U17) on the slave board. These devices are mapped into the DMA space of the slave CPU. The first 8K locations of this space are dedicated to the circular sample buffer. The remaining 2K locations are used for additional data storage. The 74F138 (U18) decoder provides the necessary DMA decoding (SCE0 - SCE4) for the static data RAMs. It is also used to provide the DMA mapping for some additional devices

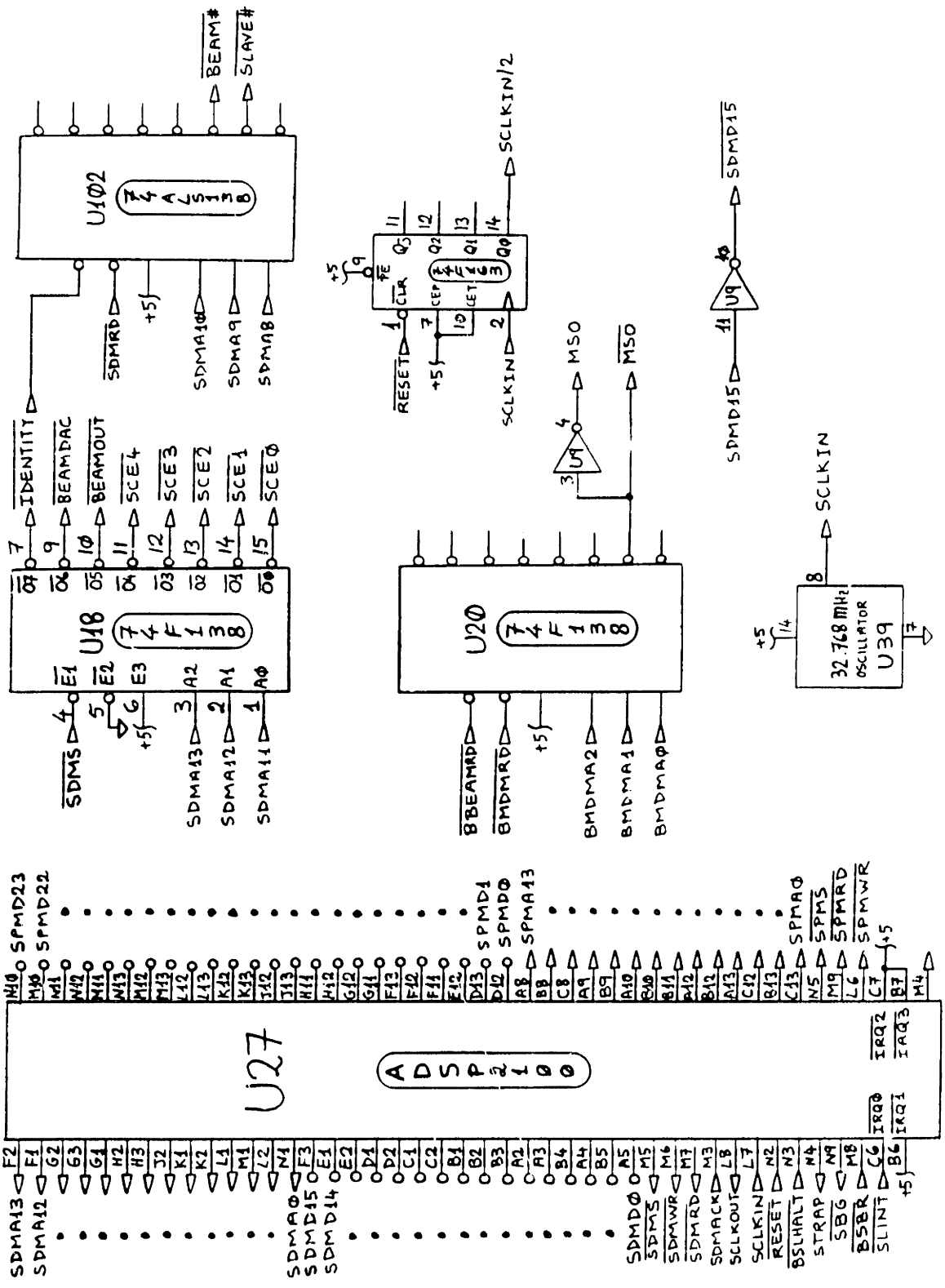
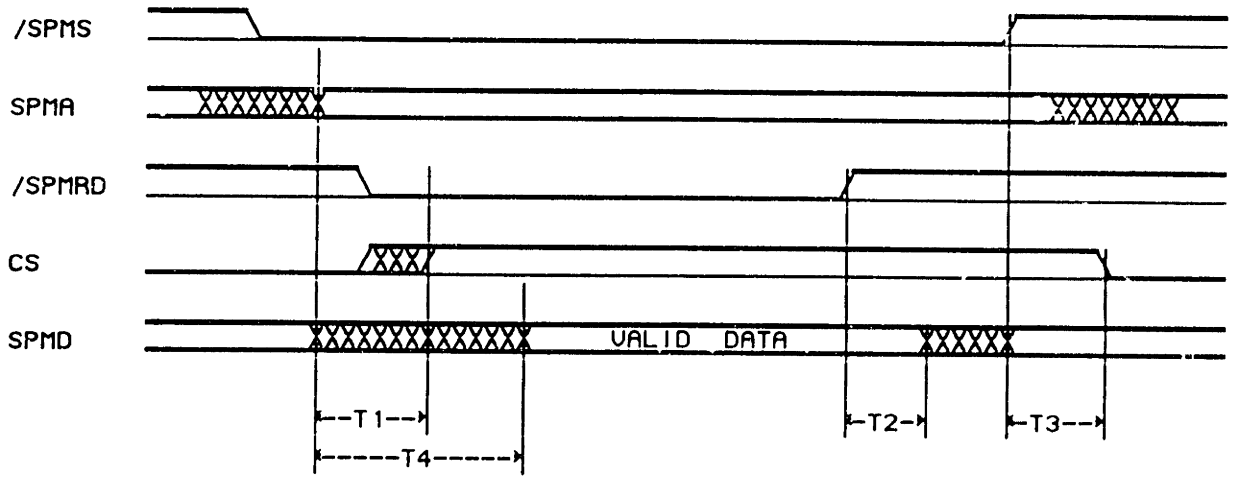
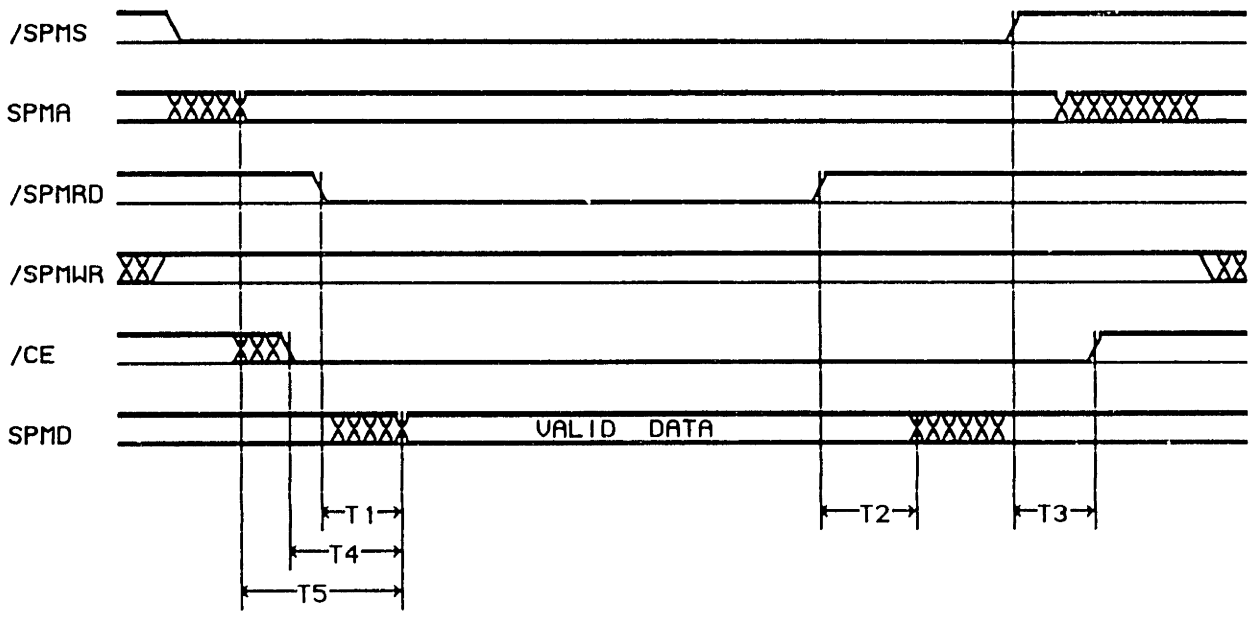


Figure 2-10 : Slave CPU and its main decoders



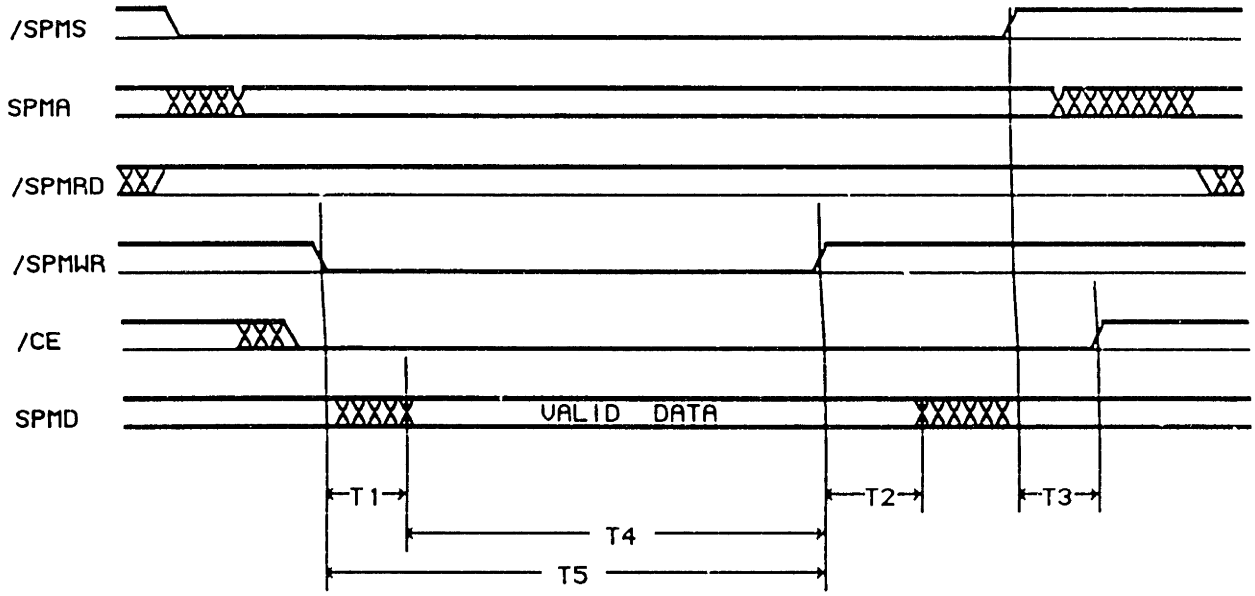
T1 = 15ns max. T2 = 25ns max. T3 = 14ns max. T4 = 35ns max.

Figure 2-11 : Slave CPU program memory read cycle (from ROM)



T1 = 12ns max. T2 = 12ns max. T3 = 8ns max. T4 = 25ns max. T5 = 25ns max.

Figure 2-12 : Slave CPU program memory read cycle (from RAM)



T1 = 32ns max. T2 = 18ns min. T3 = 8ns max. T4 = 25ns min. T5 = 45ns min.

Figure 2-13 : Slave CPU program memory write cycle

and flags. The output FIFO for the slave module is built using 4 IDT72413L35 64x5 FIFO chips (U21 - U24). These chips can be loaded (written to) from the slave DMD bus by issuing a "shift-in" (SSI) command. SSI is generated by combining the BEAMOUT and SDMWR signals. The contents of the FIFO chips can be read from the FDMD bus, which is tied to the backplane BDMD bus through buffers. Reading the FIFO is accomplished by issuing a "shift-out" (MSO) signal. MSO is produced by a 74F138 decoder (U20), which decodes some control lines generated by the master. This decoder is placed on each slave board. But a different one of its outputs is jumper selected depending on the slave's identity. This allows each FIFO to have a unique location in the master's DMA space. The timing diagrams for SSI and MSO cycles are shown in figures (2-14) and (2-15).

The slave CPU only recognizes one external interrupt. This interrupt is the /SLINT signal generated by the master, which is used to start the beamforming

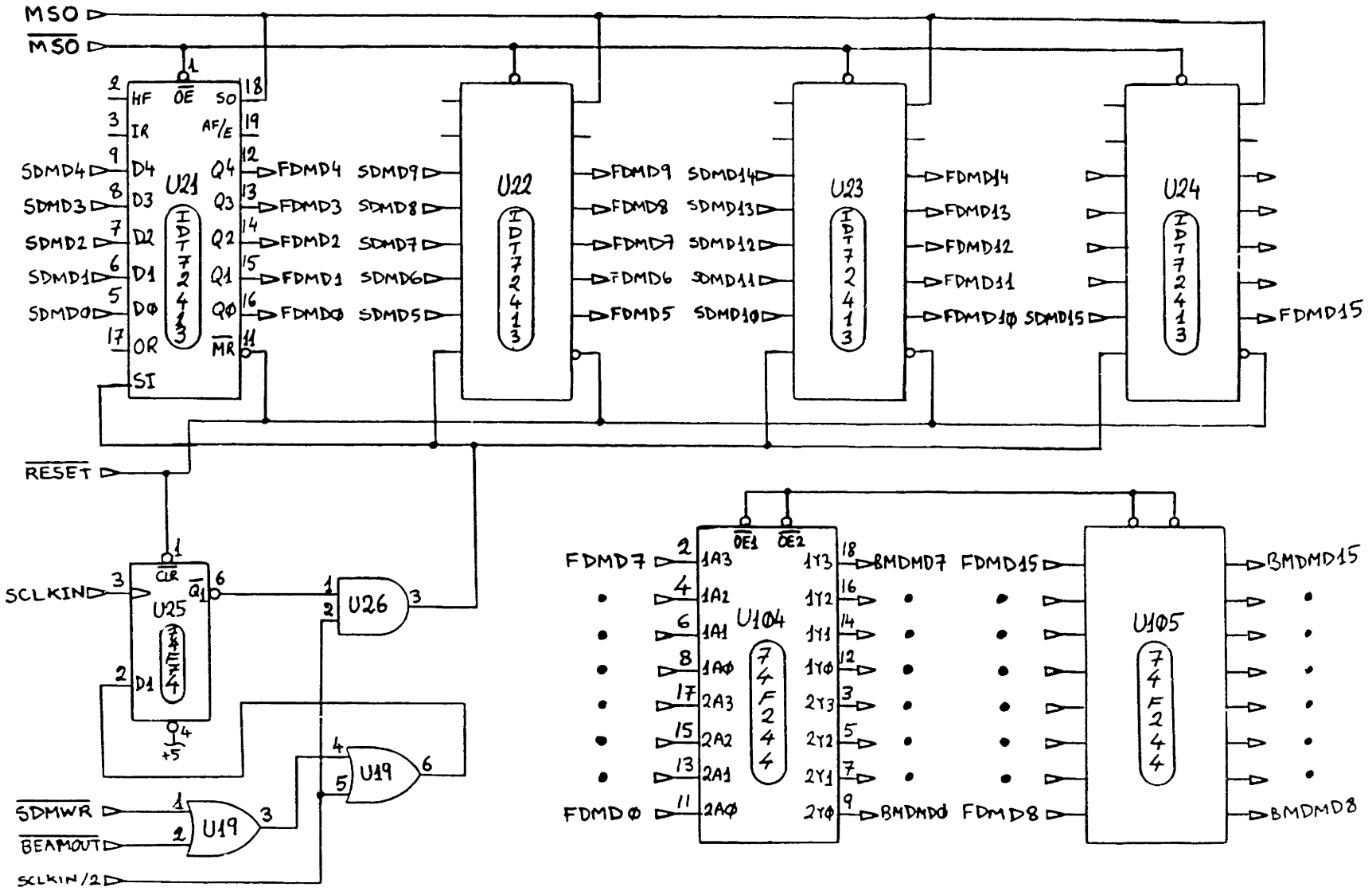
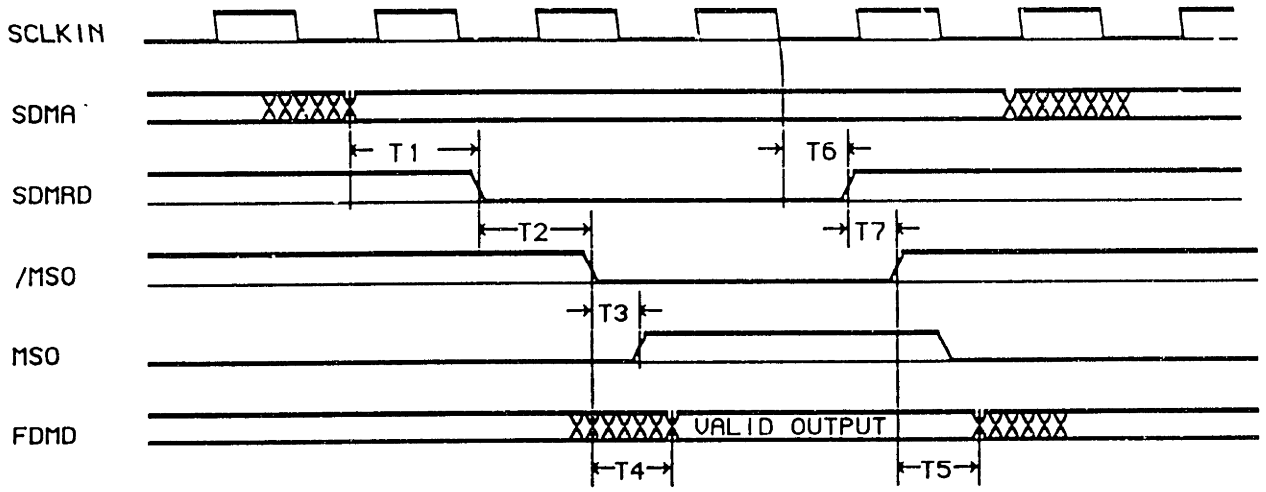
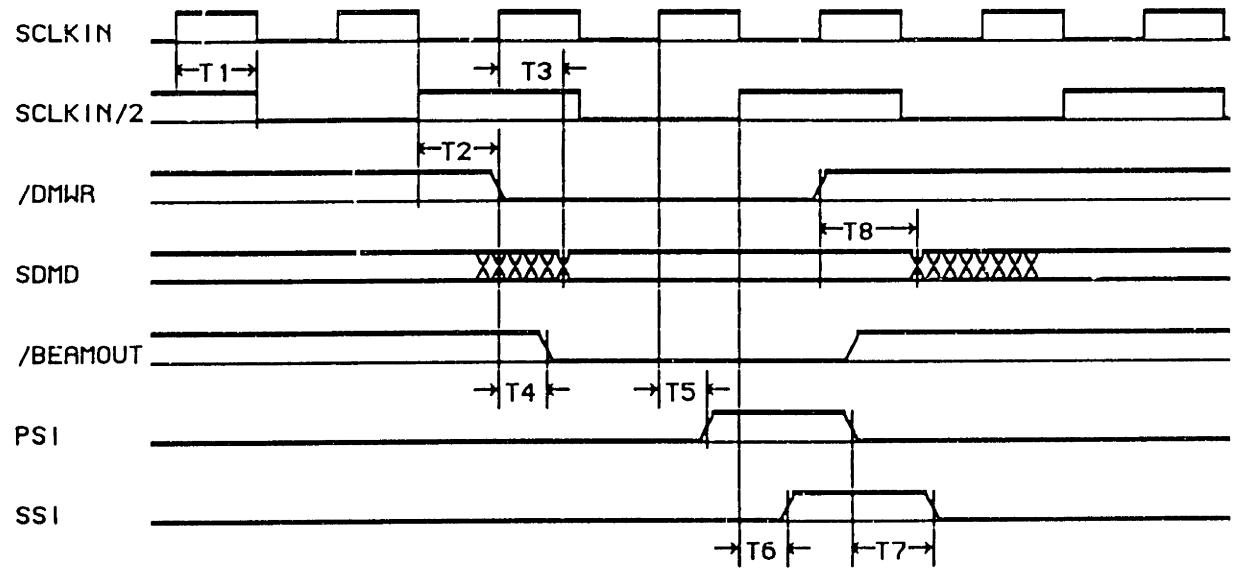


Figure 2-14 : The FIFO output circuitry on the Slave board



T1=16ns min; T2=8ns max; T3=5ns max; T4=15ns max; T5=12ns max; T6=21ns max; T7=8ns max

Figure 2-14 : FIFO shift-out timing for the slave CPU



T1 = 17ns max 13ns min T2 = 28ns max 11ns min T3 = 40ns max T4 = 6ns max 1ns min T5 = 10ns max 5ns min T6 = 6ns max 1ns min

T7 = 6ns max 1ns min T8 = - - - - - 11ns min

Figure 2-15 : FIFO shift-in timing for the Slave CPU

operation after a new circular sample buffer update. The slave CPU clears this interrupt, by issuing the /CLRSLINT signal, immediately after it finishes forming its assigned beams.

The slave board, like the master, contains a large number of bus drivers and transceivers for easy interfacing with the backplane bus. Two of these devices (U104 & U105) are dedicated for data transfers from the FIFO outputs onto the backplane. These are controlled by the master generated MSO signal. Five of these drivers (U32,33 & U35 - U37) are used to buffer the data, address and control lines arriving from the master. These are all controlled by the /SBG signal which is only enabled if the slave CPU surrenders the control of its buses (thus allowing the master to gain control over these buses). Another buffer (U34) is used to isolate the outputs of the slave DMA decoder (U18) from the slave DMA bus when the master has access to it. The timing diagrams for slave data memory access cycles initiated by the master CPU are given in figures (2-19) and (2-20). The timing for the Slave CPU's data memory access cycles can be directly followed from the ADSP-2100 data sheet.

Another component located on the slave board is the AD-569, 16 bit D/A converter (U100). This converter is mapped onto the slave DMA space. The /BEAMDAC signal generated at U18, along with /DMWR, is used to latch the data into the converter. An AD-588, ±5 Volt voltage reference (U101) is used with the D/A converter. Since the D/A converter is a slow responding peripheral, the slave /DMWR cycle must be extended using the SDMACK signal (this is an input to the slave CPU). A small circuit (U103,U19,U9) is used to generate the SDMACK during a write cycle to the D/A converter. A timing diagram for SDMACK with the extended data memory write cycle is shown in figure (2-23).

The slave board also houses a 16 position thumbwheel switch which is

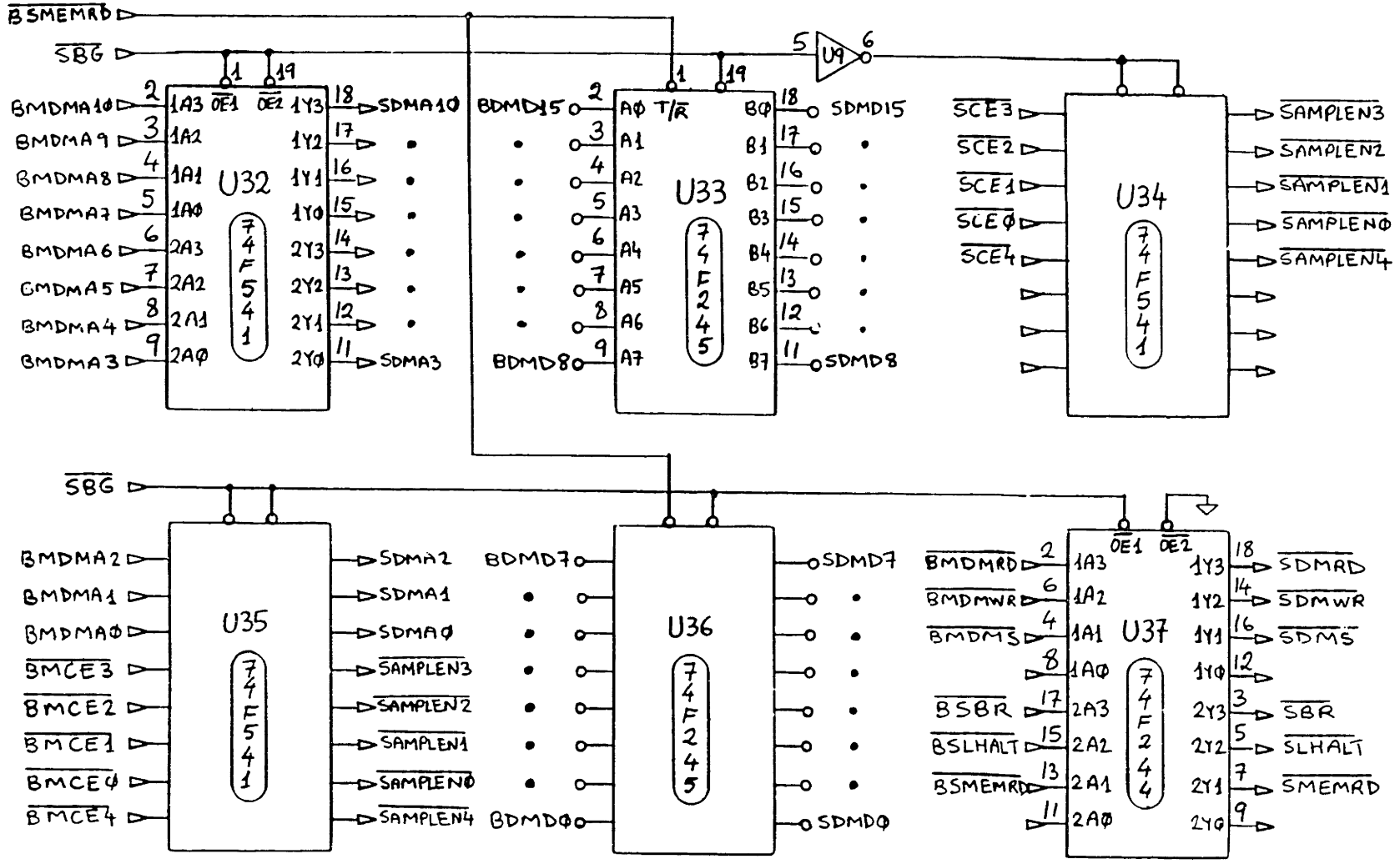


Figure 2-16 : Backplane drivers and receivers on the Slave board

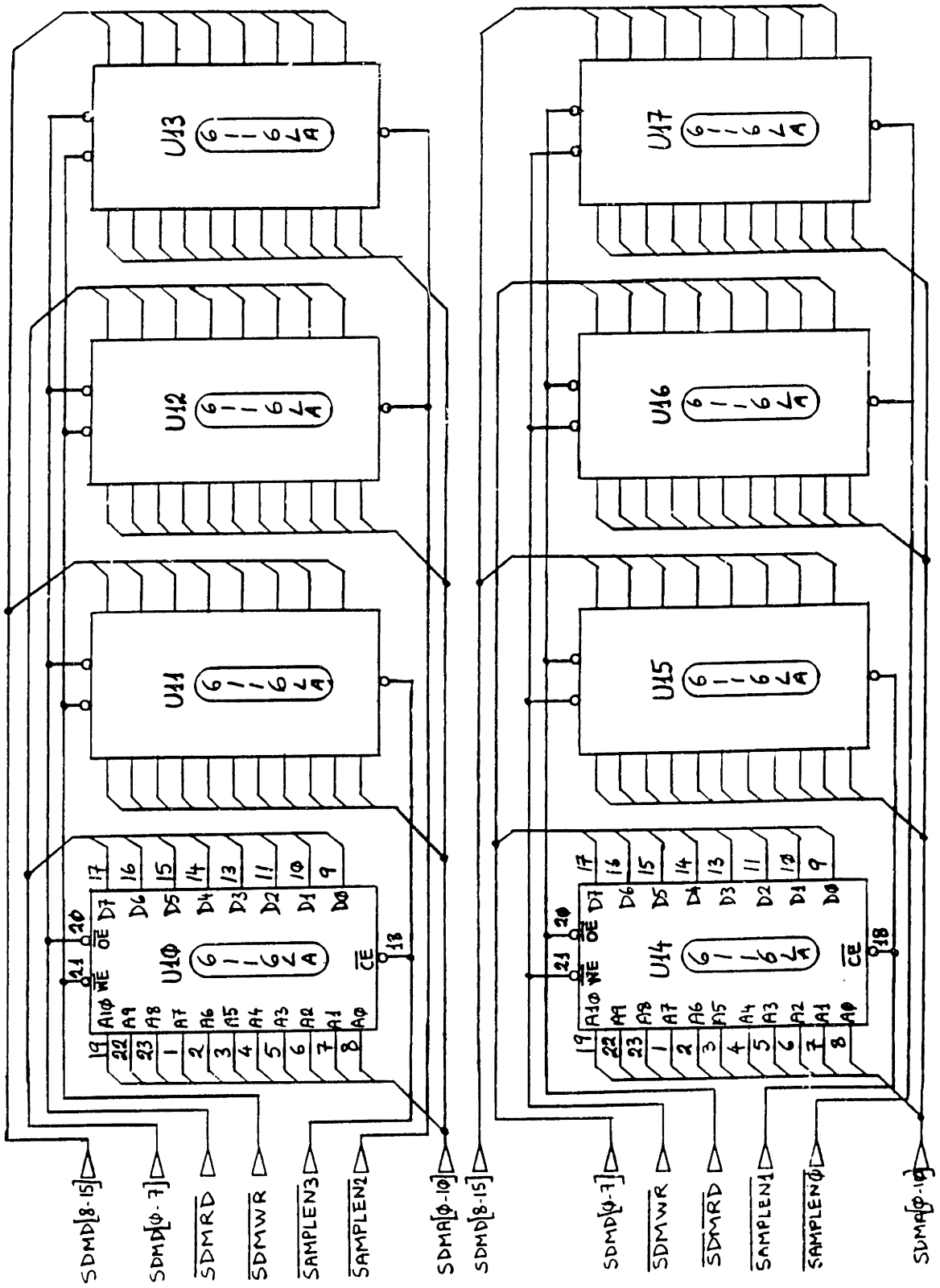


Figure 2-17 : Slave sample buffer (Data Memory)

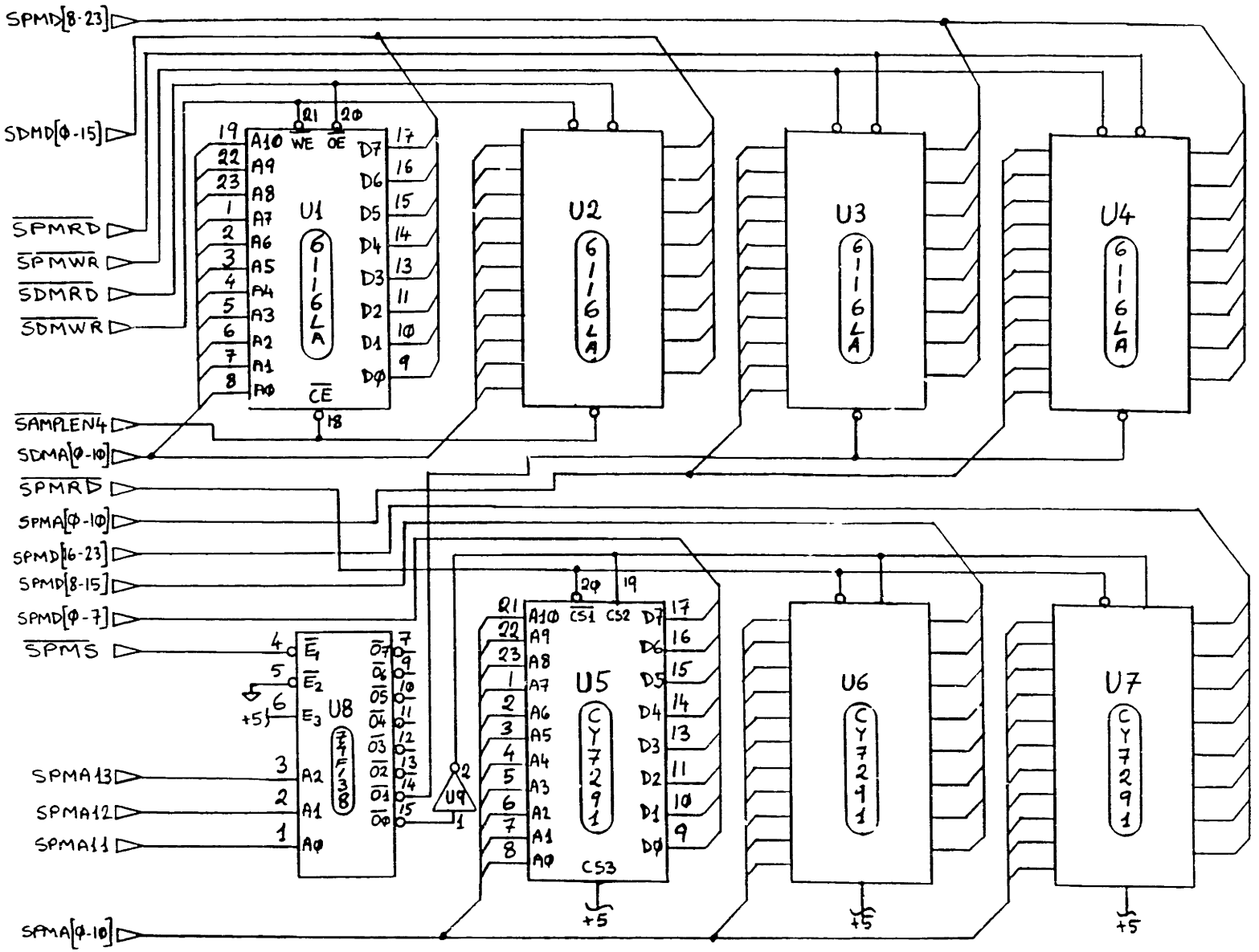
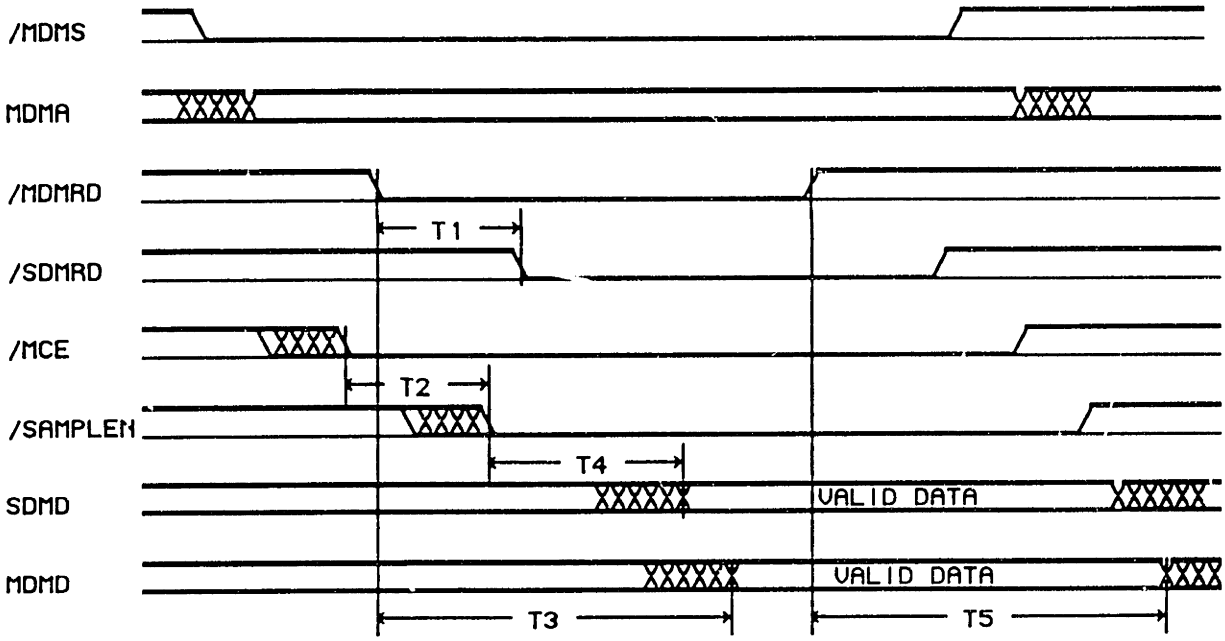
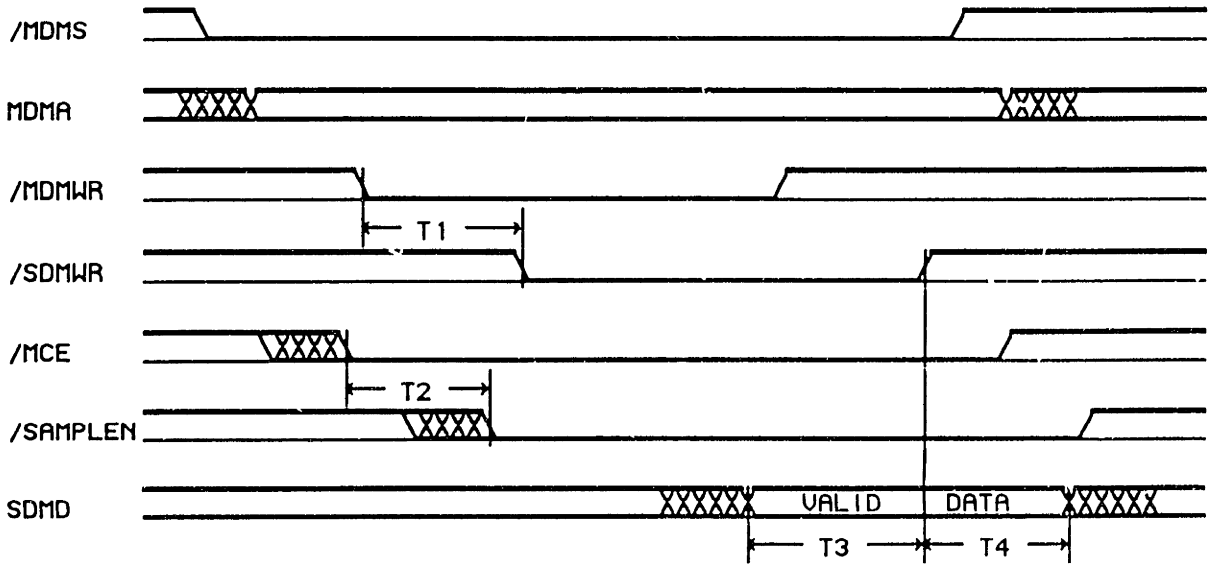


Figure 2-18 : Slave program memory (RAM & ROM)



T1 = 13ns max. T2 = 13ns max. T3 = 40ns max. T4 = 25ns max. T5 = 12ns min.

Figure 2-19 : Sample buffer read cycle by the Master CPU



T1 = 13ns max. T2 = 13ns max. T3 = 13ns min. T4 = 16ns min.

Figure 2-20 : Sample buffer write cycle by the Master CPU

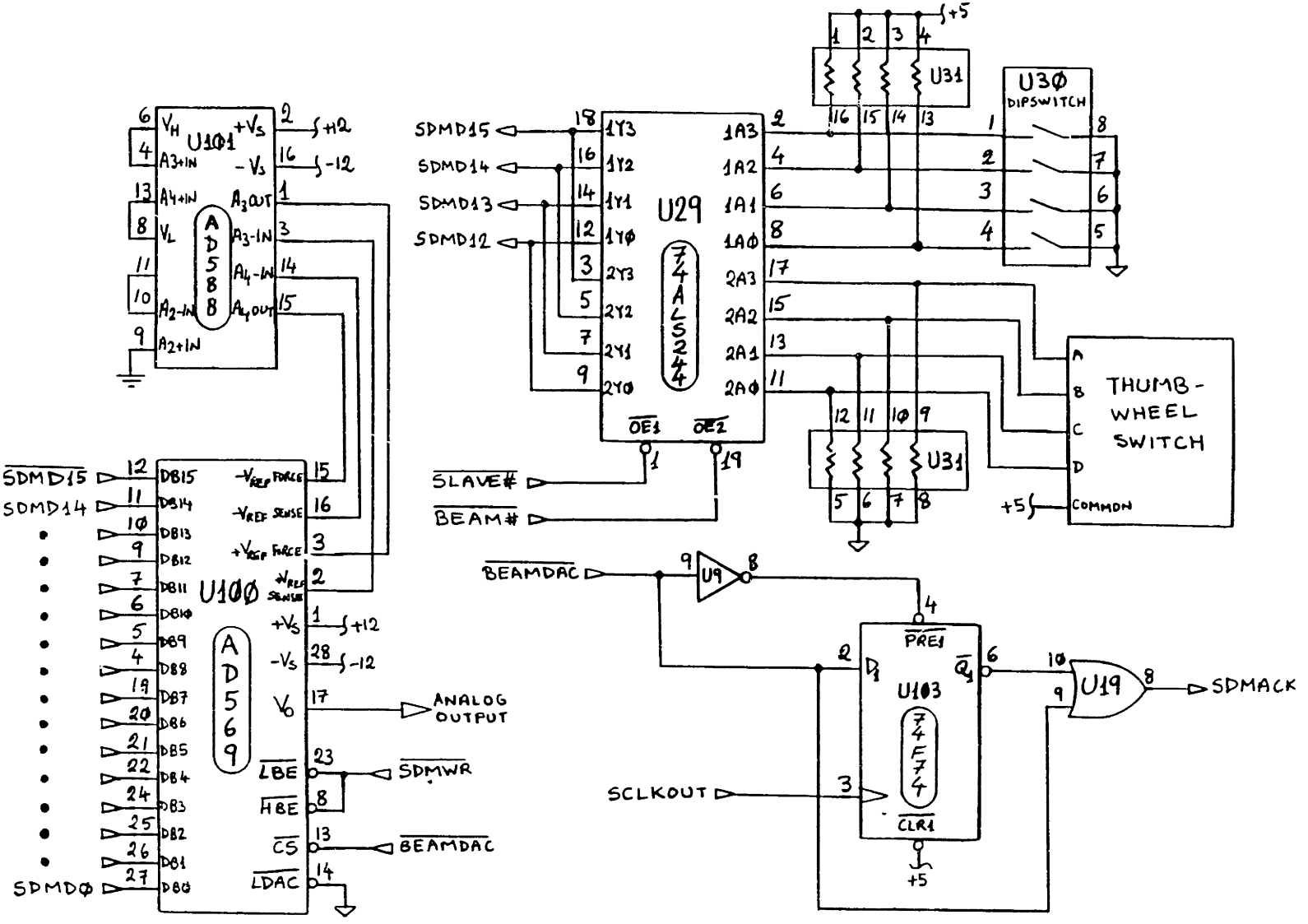


Figure 2-21 : D/A converter circuitry on the Slave board

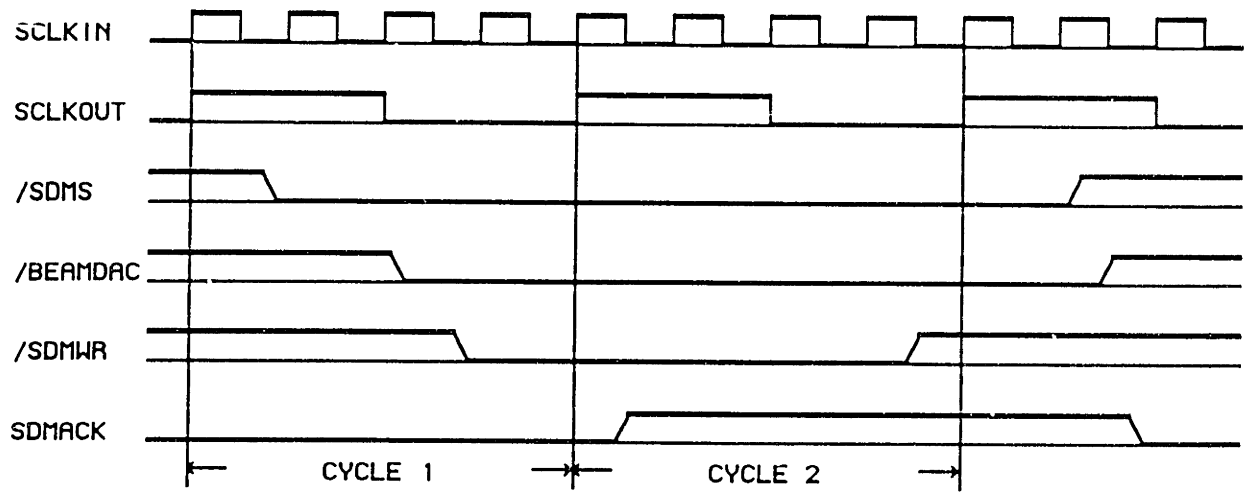


Figure 2-22 : Slave CPU's write cycle to the D/A (extended by SDMACK)

used to select the beam that will be sent out through the D/A converter. It also contains a set of 4 dipswitches that are used to give each slave board its own identity. Setting these switches allow the slave CPU to recognize the beams that are under its responsibility and to know the beam that is requested for the analog output. A decoder (U102) and an octal buffer allow the slave CPU to "read" these switches. These switches are also mapped onto the slave DMA space. The /IDENTITY signal generated at U18 and the /BEAM# and /SLAVE# signals generated at U102 provide the mapping.

There are two connectors on the slave board: A 96 pin Eurocard connector and a male BNC connector. The first one is used to interface to the backplane bus, and the second one is connected to the analog output of the D/A converter. The user can utilize the BNC connector to analyze the switch selected output beam. The slave board requires +5 Volt digital and ±12 Volt analog power supplies for proper operation.

2.2.4 A/D Board Hardware :

The details of the circuitry on the A/D boards (modules) are discussed in this section. Hardware schematics for the A/D boards and their interfaces are shown in the following pages.

Each A/D board can receive up to 4 analog inputs ranging between ±5 Volts and can convert them into a 16 bit signed digital fixed point format (1.15 format) with 12 bit accuracy. In order to accomplish the conversion task, the circuitry on the boards is designed around four AD-1332, 12 bit hybrid A/D converters. Some specifications and a block diagram of AD-1332 is provided in appendix D. A gain stage for each input is included on the A/D boards in order to provide the necessary amplification of the hydrophone outputs. The gain stage utilizes AD-OP07 operational amplifiers in inverting configurations. The available gains (1, 10, 100, 1000) are selected using an external 4 position rotary switch. AD-7590 analog CMOS switches are used to select the desired resistor combination for the amplifier's feedback loop. The CMOS switches are used because of the future possibility of using CPU generated signals to make gain selections. The board also contains two octal bus drivers (U71 & U72) in order to provide adequate buffering for the AD-1332 outputs. The outputs of these buffers are tied to the backplane DMD bus. The lowest 4 bits of the buffer inputs are tied low because the A/D converters have only 12 bit resolution. The A/D output, which is in offset binary format, has to have its most significant bit inverted because of the fixed point format that is used in the system. Another component that is on the board is a decoder (U53) which provides a unique location for each AD-1332 in the master's DMA space. The configuration of this decoder would be different in other A/D boards in order for each hybrid to have a unique location. Another component that is present is a AD-964 DC-DC converter, which is used to provide ±15 Volts to

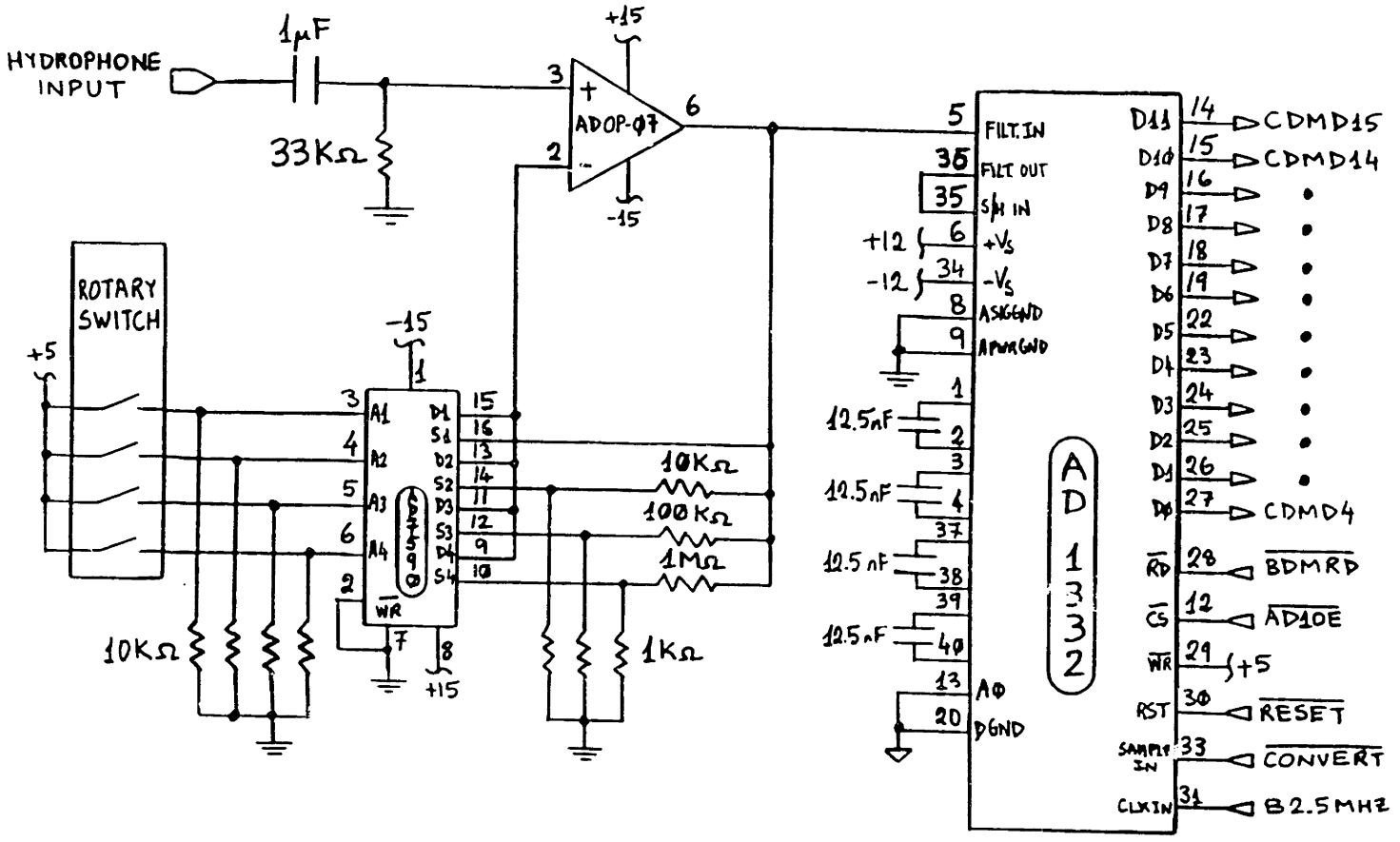


Figure 2-23 : Input gain and A/D conversion circuitry for a single input channel .

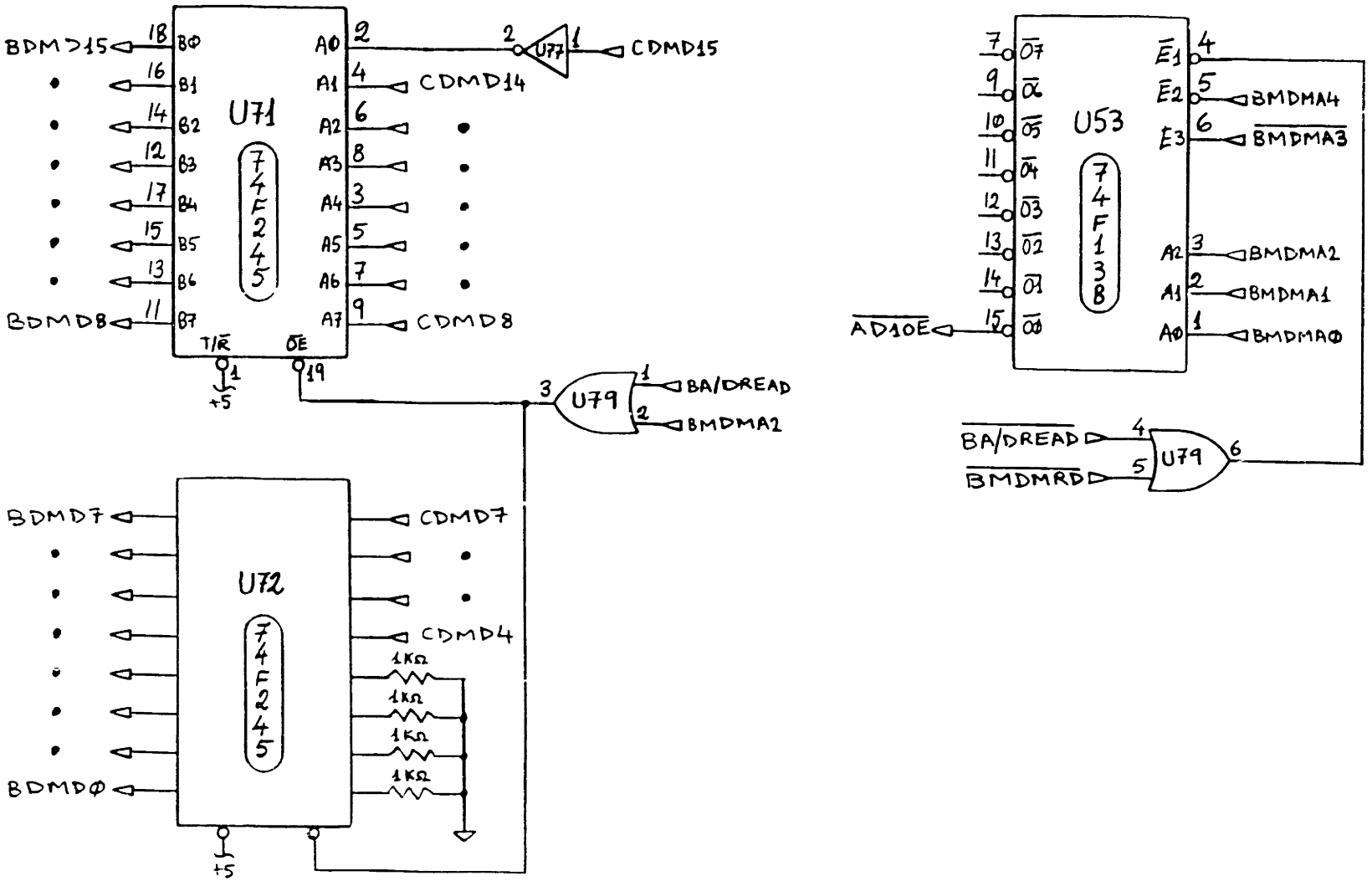


Figure 2-24 : Decoding and buffering on the A/D board

the input gain amplifiers and the analog CMOS switches by converting the incoming +12 Volt analog supply. +5 Volt digital and ±12 Volt analog supplies are needed for proper operation. There are 5 connectors on this board. One is the 96 pin male Eurocard connector that is used to interface to the backplane bus. The other four are male BNC connectors that accept the hydrophone outputs.

2.3 System Firmware :

The assembly code that is responsible for the master and slave CPUs' operation will be discussed in this section. The code is written in the ADSP-2100's easy to understand assembly language. Details of this language can be found in the ADSP-2100 user's guide.

2.3.1 Master Firmware :

The firmware that runs in the master CPU is of moderate length, its listing is given in appendix A. The listing consists of two sections: the first one is the architecture description file, which describes the different memory sections and ports that are necessary for proper system operation (this file is used by the development tools of the ADSP-2100 for code assembly and prom coding). The second section is the assembly file which contains the actual code that will run on the CPU. The architecture description file is self-explanatory, the following discussion will concentrate on the assembly file.

The master assembly code can be divided into two subdivisions: the AT communications section and the A/D result handling section. All the code sections will be briefly described in the following paragraphs.

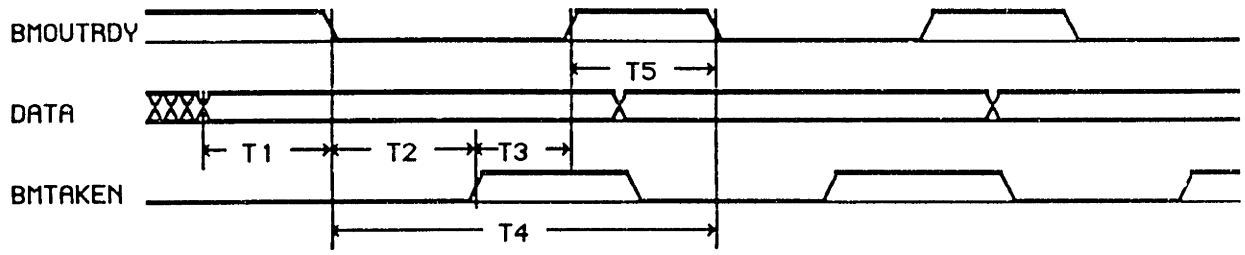
The AT communications section of the master code starts at the beginning of the file and ends at the " WAIT " routine. The beginning of the file contains a series of port, variable and interrupt declarations. The port names correspond to

the actual signal names that were discussed in the "master board" section in chapter 2 (2.2.2). The interrupt names are self-explanatory. The program starts by clearing certain flags and the PC_INIT_WAIT routine is entered. This routine forces the master CPU to wait until the AT is ready to download the system configuration data. Once the AT is ready, the PC_COMM routine is executed. This routine sets some of the internal CPU registers to be used during the communication. At the end of this routine, the AT is notified that the master is ready and the master CPU enters the PC_WAIT1 loop. The first six pieces of data downloaded by the AT are handled differently than the rest. These first six pieces contain information about the number of sensors in the system, the number of beams to be formed, the total number of indexes to be used (the indexes are used by the slave CPUs to pick the desired input samples from the circular buffer), the number of slaves in the system, the number of beams assigned to each slave and the number of beams assigned to the last slave. This data will be stored in the master CPU's data memory (which actually is the same as the slave data memories) in consecutive locations. These locations are declared at the beginning of the program. This storage task is accomplished by the INIT_STO routine. At this point, the indexes that were calculated by the AT are next to be downloaded. These values are received and stored into data memory by the INDEX_STORE routine. After receiving all of the necessary data, the master executes the CHECKSUM routine. This routine compares the number of pieces of data that was received to a new value sent by the AT. This new value is the number of pieces of data that was sent by the AT. If these values are equal, that implies that the downloading was successful. After establishing this, the master enters the COMM_END routine. This routine prepares the master, using the downloaded data, for its next set of tasks (responding to A/Ds etc.). This routine and the

INIT_STO routine are the only times the master CPU reads a value from a slave data memory. This type of read operation, as mentioned before, is only allowed to be done from a single slave which must be plugged into a designated slot on the backplane. Another piece of data that is calculated during this routine is the shading coefficient that will be used in the current set of beams. The system currently can only handle a rectangular window with a magnitude of 1. Some additional code that can handle several different shading windows will be added later.

After preparing itself, the master CPU issues a /SLHALT signal which commands the slaves to also prepare themselves using the recently downloaded data (the slave CPUs were in a TRAP state until now). The master waits for 1000 cycles, in order to give more than enough time for the slaves to get prepared. It finally enables two nested interrupts and enters the WAIT loop.

The second major section of the code starts at the WAIT loop. The master CPU waits for the /ADCOMPLETE interrupt to occur and as soon as that is received it jumps to the ADCOMPLETE routine. During this routine, the master CPU reads the output latches (conversion results) of each AD-1332 and writes these values into all of the slaves' sample buffers. The master CPU keeps track of a few pointers in order to address the circular buffers and the A/D boards properly. Once the sample storage is completed, the execution of the SENDBEAM routine gets started. During this loop, the master CPU reads all the slave FIFOs in sequence and writes them to the system output port. Output beam samples are sent one at a time and the handshaking with the external sonar signal processor is executed for each beam sample. The handshaking starts with the assertion of the BMOUTRDY signal and requires a wait for the BMTAKEN interrupt (set by the external processor). The arrival of this interrupt means that the external



T1 = 250ns min. T2 = 1.5μs max. T3 = 250ns max. T4 = 3μs max. T5 = 625ns min.

Figure 2-25 : Beam output timing

processor is ready to receive the next value. Consequently, the interrupt is cleared by the master CPU and the next beam sample is sent out in the same manner. The timing diagrams for the beam sample output are shown in figure (2-26). The ONESLAVE routine serves a similar purpose to the SENDBEAM routine, except it only handles the output FIFO of the last slave. This is necessary because the number of beams formed by the last slave is different than the amount that the rest of the slaves are each responsible for. This inequality exists, since for a given number of desired beam directions, it is not always possible to divide the job evenly among the slaves. Once the beam output tasks are completed, a " rti " (return from interrupt) instruction is executed and the program returns to the WAIT loop where the master CPU waits for the next ADCOMPLETE interrupt.

The master firmware continues its cyclic, double interrupt driven operation until the assertion of RESET or a system power-down.

2.3.2 Slave Firmware :

The slave board requires less firmware coding than the master, its listing is given in the appendix B. Its listing, again, consists of two sections: an architecture description file and an assembly file. The following discussion will concentrate on the assembly file.

The slave firmware, like the master's, can be divided into two subdivisions: the system set-up section and the beamforming section. Both code sections will be briefly described in the following paragraphs.

The set-up section of the code starts at the beginning of the file and ends at the WAIT routine. The beginning of the file contains a series of port, variable and interrupt declarations. The port names, as in the master's case, correspond to the actual signal names that were discussed in the "slave board hardware" section in chapter 2 (2.2.3). The only recognized interrupt, /SLINT, is initiated by the master to start the beamforming operation. After the necessary system parameter declarations are done, the program stops itself and the slave CPU enters a TRAP state. The slave CPU gets re-activated after the master is finished communicating with the AT. At that point in time, the master CPU asserts the /SLHALT signal, which wakes up the slave and causes the program execution to continue from the location following the "trap" instruction. The first part of the program lets the slave CPU move the indexes from its data memory into its " index memory " which is located in its program memory space (the indexes were in the data memory until now, because that was the only place that the master could have stored them temporarily). Once this is accomplished, the slave executes a series of instructions in order to set up its address registers. In order to do this, it uses the downloaded beamforming information. Following the preparation, it enters the WAIT loop and is now ready to respond to beamforming interrupts to be issued

by the master.

The slave CPU constantly monitors the D/A beam selection switch while it is in the WAIT loop. Since the slave CPU returns to the WAIT loop every 100μs , the monitoring allows it to make a decision, in real time, as to which beam to send out through the analog port.

The second major section of the program starts at the WAIT loop. The slave CPU waits for the /SLINT interrupt to occur and as soon as that is received, it jumps to the BEAM_FORM routine. The BEAM_FORM routine allows the slave CPU to read the index memory and pick the indexed samples from the sample buffer. The samples determined by the indexes are the delayed samples that are needed to perform beamforming. The frame of the circular buffer is virtually rotated every time a new set of fresh samples come in. Therefore the indexes that are read must be modified before their use, since they are referenced to the absolute origin of the circular buffer. The samples that are read are multiplied by the shading factor (which is currently 1) and accumulated in the " mr " register. The resulting beam sample is finally written into the FIFO. In the meantime, if the beam sample belongs to the beam that is requested at the analog port, it is then written to the BEAMDAC port (the D/A converter). The slave CPU, before returning from the interrupt, also clears the /SLINT flag. The program finally returns to the WAIT loop where it will wait for the next /SLINT interrupt that will be issued.

The slave firmware continues its cyclic, single interrupt driven operation until the assertion of /RESET or a system power-down.

2.4 System Software :

There is only one program in the system that can be classified as software. This is the program that runs on the AT and that is responsible for the user interface and the downloading of the system configuraton data. The code is written in the " C " language and compiled on the Microsoft C Compiler. The listing for the program is given in appendix C. The program is comprised of several function call and definition sections. The details of these sections will be briefly described in the following paragraphs.

The short declaration section at the beginning of the program includes certain useful libraries, defines a number of variables and declares the 8 bit parallel I/O port addresses. These ports are located on an Analog Devices RTI-817 parallel I/O card which is plugged into the AT's backplane. Following this section there are a series of function definitions starting with main (). The duty of the main() function is to call all the necessary sub-functions in correct order for proper code execution. The user_io () function is the interactive function that intakes the system variables from the user. This function prints questions on the screen, which in turn have to be answered by the user via the keyboard. The values that have to be entered are the following : the number of sensors, the number of beams to be formed, the number of slaves in the system, the cartesian coordinates for the sensor locations and the spherical coordinates for the desired beams. The user_io() function assigns these values to variables and arrays in order for other functions to calculate the necessary delays for beamforming.

The job of the index_calculate () function is explained in its name. It first converts the spherical coordinate beam directions to cartesian coordinates. Once this is accomplished, it calculates the necessary delays using the formula 1.2.2-2. The propagation speed of the sound in water is assumed to be 1470 m/s which is

typical for the Arctic waters (this value can be changed easily in the code depending on the application environment). Another duty of this function is to identify the beams that the system is unable to produce with the given array configuration. This task is accomplished by checking whether any one of the required delays falls outside of the sample buffer length.

The next function listed is slave_capacity () whose task is to determine the maximum number of beams that can be formed with the given system configuration. It also calculates the number of beams to be assigned to each slave and the number of beams to be assigned to the last slave. These values are stored as variables to be downloaded later to the master.

The next important function to be discussed is download (). As the name suggests, this function is responsible for downloading the system configuration information to the master. This download () function utilizes some other conveniently defined utility functions which are shown on the rest of the program listing. Some of these are send(x), wt(x), linv(w), iinv(w) and rnd(x). The descriptions for these utility functions are clearly given, with comments, on the program listing. The download () function first sends the initial six pieces of system set-up data to the master, as explained in the "master firmware" section in chapter 2 (2.3.1). The next task is to send all the calculated indexes to the master. At the end of that operation, a checksum is sent to the master. This checksum, as explained earlier, corresponds to the number of pieces of data (number of indexes + 6) that was just downloaded. Upon the master's acknowledgement, the downloading operation is completed by printing a positive message on the screen. If the master sends an incorrect message the downloading operation is terminated by printing a failure message on the screen. At this point, the user is given the choice of aborting or retrying the downloading process.

The downloading process is easily executable by the user. Once the system parameters are entered, it takes at most a few seconds for the AT to download all the information to the master.

2.5 System Enclosure :

The main enclosure constraint for the system is that it has to be mountable on a 19" rack. Considering this constraint and considering the number of integrated circuit packages to be used, a 19" subrack cage with the lowest available height (13 cm.) was chosen to house the beamformer. Since this is only a prototype system, a backplane is not used. Nine 96 pin female Eurocard wire-wrap connectors are mounted in the rear of the subrack and the signals are bussed using wire-wrap wires. The power and ground buses that feed all the boards are bussed across the back of the cage. In a production version, in order to have room for the maximum system configuration, a printed circuit board backplane with 21 connectors and buried power and ground planes must be used. All of the system prototype boards are wire-wrapped. A production version, again, would require printed circuit boards with power and ground planes for proper operation and maximum space utilization. A liftable cover is used to protect the wires in the back of the cage. The input amplifiers' gain selector switch is placed on the back cover. The input BNC connectors of the A/D boards and the D/A output BNC connectors are located in the front of the cage. They are mounted on the A/D and slave boards on the opposite side of the Eurocard connectors. The AT communications cable and the output cables are extended out of the front of the cage. The system is very compact compared to its existing lower performance counterparts and has a very low cost per formed beam.

2.6 System Utilization Procedure :

This section describes the required equipment and the proper set-up procedure for using the system in a passive sonar application.

The experimental research beamformer can be mounted on a 19" rack. The power supplies that are required to be provided on the rack are: One digital power supply (+5 Volts, 12 A) and one analog power supply (±12 Volts, 4A). The sensor inputs to the system are standard male BNC connectors, therefore all hydrophone outputs must be brought in coaxial cables with standard female BNC endings. The D/A output also requires the same type of coaxial cables for proper usage. The input gain switch is located behind the cage. The flat cable connectors for the output bus are located on the master board and their wire terminals are perpendicular to the board's plane. The flat cable connector used for the AT communications is also located on the master and it is oriented in the same direction of the BNC connectors (facing outward). The master, slaves and the A/D boards may be plugged into any slot on the backplane with the exception of one slot which is reserved for a slave (it has to be always occupied, even in the minimum system configuration). The dipswitches on the slave boards can be set to a value between binary 0000 to 1111. These specify a unique identity for each slave. The first slave is always set to 0000 (this slave must always be in the reserved slot). The rest of the slaves should each have an identity incremented by one; the second slave will get 0001, the third will get 0010 and so on. The "open" position on the switches corresponds to a "1". The thumbwheel switch on the slave boards allows the observation of up to 16 beams in analog form.

In order to install the user interface, several steps must be taken : The parallel I/O card (RTI-817) has to be installed in one of the slots on the AT's backplane. The AT to be used must have a Microsoft C Compiler installed. The

library routines that are called at the beginning of the user interface program also must be installed along with the compiler. The name of the user interface program is "Comm.c" and it must be included into one of the user directories in the AT.

The operation of the system starts with power-up. The next step is to invoke the user interface program in order to enter the desired system parameters. The "Comm.c" program is invoked by typing "Comm" and carriage return into the AT.

At this point, the program will print " Enter the number of sensors :". The user must type in the exact number of hydrophones and hit carriage return. The next questions are about the number of beams to be formed and about the number of slaves in the system. These parameters must be entered in a similar manner to the number of sensors. The following questions ask the user to enter the cartesian coordinates for each sensor location(in meters). The choice for the origin of the cartesian space is left to the user. Once the origin is established, the sensor locations relative to it should be used to answer these questions. The entered coordinates must be separated by commas and followed by a carriage return. The next set of questions ask the user to enter the desired beamforming directions (in degrees). The angles for elevation (Φ), and azimuth (Θ) are to be determined relative to the user chosen origin. The angles must be defined as they were shown in figure (1-5). The entered angles, again, must be separated by commas. The numbers assigned to each beam (e.g. beam 2, beam 3 etc.) determine the order of the beam samples that will be in the digital output beam stream. This implies that, every system cycle, the external sonar processor will receive the beam samples in the order that they were assigned during the user interactions.

Once the questions are completed, the program calculates the delay indexes

and other data that will be downloaded to the master. In case of the impossibility to form a certain beam with the given sensor array, the program will print an error message such as : "beam x will not have a valid output". This notifies the user that the output beam stream will still have a slot reserved for "beam x", but that it will not contain useful data. Another message that will be printed lists the beam assignments for the slaves. This listing is in such order that the analog beam outputs can easily be selected using the thumbwheel switches on each slave. Some further messages that might appear are used to notify the user about checksum errors. In case of an error, the user is then given a choice to retry downloading the parameters or to restart the entire operation. Finally, if the downloading is successfully completed, the user is notified on the screen and the beamformer starts its operation immediately. Any further changes that need to be made in the system parameters can only be accomplished by resetting the system and re-executing "Comm.c" with the new information.

As mentioned earlier, a prototype of the system is built at the minimum possible configuration level: one A/D board, a master and only one slave. The prototype can be powered and used by following the instructions provided in the previous paragraphs. The prototype is lacking one of the features that the finalized production level system would have: The slave beam assignments are not yet computed and printed on the screen by "Comm.c". All the beams are assigned to the single slave in the prototype and the formed beams are still available at the D/A port. By selecting different positions on the thumbwheel switch, the currently formed beams can be observed. Furthermore, the position numbers selected on the switch correspond to the numbers that are assigned to the current beams by "Comm.c". This enables the user to very easily reference and test the output beams produced by the prototype.

CHAPTER 3 : System testing

The debugging and testing procedures for the system prototype, along with the test and characterization results, are presented in this chapter.

3.1 Prototype Construction and Debugging :

The construction and the debugging of the prototype hardware and firmware were done in a modular fashion. Once the internal details of the system functionality were determined, the hardware design of all the boards was started. But in order to have the proper hardware, the firmware design also had to be started at the same time. The system design was improved by an iterative process that determined the needs of both the hardware and the firmware.

The hardware construction was started immediately following the first completed version of the hardware schematics. The slave board was the first to be built. Its debugging was accomplished by using the ADSP-2100 in-circuit emulator. This emulator provides the designers with the ability to run ADSP-2100 assembly code routines in real-time or in single-step mode within the circuit that is being developed. Some short test routines were developed and ran on the slave board. These simple test routines helped isolate and debug different functional parts of the board. Longer routines were used to test the interactions of these separate functional parts. A high speed logic analyzer and a digital oscilloscope was used to evaluate the waveforms. The slave firmware was being developed in parallel on the ADSP-2100 simulator. The simulator provides software simulations for the ADSP-2100 assembly code programs. The actual slave firmware was not used during this stage of slave development.

The user interface program on the AT was well under way of development at this time. Some simple communication routines were written in " C " to test

the input and output ports of the RTI-817 I/O card. These ports were tested by extensive use of the logic analyzer. At this point, the ad hoc communications protocol was determined and it was added to the user interface program. The program was going to be fully tested after the master was operational.

The next step in the system development was the construction and debugging of the master board. The master board was tested and debugged in a similar fashion to the slave board. The emulator, again, was used extensively during this stage of development. A different aspect of the master board that needed to be tested was its AT interface. This interface was tested by writing and running some simple I/O routines on the emulator and by observing their interaction with the simplified user interface program on the AT. Once the hardware and the firmware for the interface were debugged, a final version for the master's I/O routines and a final version for the communications program on the AT were produced.

The construction of the enclosure was also going on during the initial phases of the system development. The cage and the backplane were ready for the master and the slave to be plugged in before the debugging of the master was finished. The master and the slave boards were plugged into the backplane and tested together using two separate emulators instead of their CPUs. Their backplane interfaces, as well as their message passing capabilities were debugged and tested during this phase of development. Real-time data transfers were accomplished and the inadequacies of this particular prototype construction style at such high speeds were discovered. The most serious problem that prevented proper system functionality was the lack of a strong ground plane on both boards. Fast edges of the signals caused the integrated circuits to demand very large amounts of instantaneous current. This demand and the lack of a ground plane

caused considerable voltage differences among different parts of the boards. These differences were large enough to disturb the valid logic levels of the signals and caused the boards to malfunction. This problem was finally solved by adding a ground grid, using thick metal bars, to both boards. Finally, some sections of the actual master and slave firmware were also used during this phase. This effort enabled the redesign for some parts of the firmware.

The next phase in the system development is the construction and testing of the A/D boards. Because of the previous problems associated with not having ground planes, the A/D board was constructed on a plastic Eurocard with built in power and ground planes. The planes needed to be etched away in certain areas in order to allow for proper component placement. In addition, isolated power and ground planes had to be created for the analog and digital portions of the circuitry. The debugging and testing of this board required the use of a function generator, two emulators and the previously constructed boards. Some short test routines allowed the input sine waves to be sampled, read by the master and sent to the slave for D/A output. Large voltage differences among separate boards' grounds lead to the discovery of a new series of malfunctions. These problems were mainly caused by the lack of a solid built-in ground bus on the backplane. Most of these problems required mechanical corrections as well as electrical ones. For example, the overall power requirement was reduced by using more CMOS integrated circuits and additional thick bus wires were used to reduce the voltage differences.

Once all of the above phases were completed, the master and slave firmware programs were ready for final modifications. Some preliminary beamforming tests were executed. These tests involved feeding a sine wave with a constant amplitude and phase into the sensor inputs. The sine wave was input

to one A/D at a time and the system was asked to beamform in the broadside direction. The output was observed at the D/A output on the slave board. Once that test proved successful, the number of A/D's receiving the same input was increased one at a time. All the A/Ds were finally fed the same sinewave, thus simulating a planar wavefront. This qualitative test phase was completed by coding the finalized versions of the slave and master firmware programs into the EPROMS in their respective PMA spaces. The emulators, again, were the most important tools during these stages of the system development. At the end of this phase, the emulators were replaced with actual ADSP-2100s, and the prototype system was ready for further quantitative beamforming tests.

The only section of the system that did not get thoroughly tested is the digital beam output circuitry. The reason is that the sonar processor which will be receiving the output beams is still under development (but its input interface is already specified). The testing of this section was limited to some analysis on a logic analyzer and a digital oscilloscope.

3.2 Prototype Testing and Characterization :

Once the construction and the qualitative functional testing were completed, the next step was to test the system's beamforming capabilities more thoroughly and characterize them. The essential testing technique that was used during this stage of the system development will be described in this section.

In order to test that this system can beamform accurately in the desired steering angles, it is necessary to simulate the arrival of wavefronts from a given direction. The simulated hydrophone locations have to be determined and entered into the system by using the AT interface. The test consists of evaluating the beam outputs while beamforming in the known direction of wavefront

arrival and a number of other steering directions. The decrease in the magnitude of the beam output, as the beamforming direction is varied, can be plotted as a function of the steering direction. This plot corresponds to the actual beam pattern of the simulated hydrophone array for the given direction of plane wave arrival. By comparing this plot to the ideal beam pattern of an array with identical parameters, it is possible to characterize the performance of this beamforming system and of the beamforming algorithm.

In order to simplify the testing task, the hydrophone array to be simulated will consist of four elements which are equally spaced along a line. The formula for the beam pattern of such an array is given in Chapter 1 (equation 1.2.2-6). Simulating a waveform with a desired arbitrary arrival angle is a very difficult task. Therefore the arrival angle of the incoming wavefront is decided to be perpendicular to the array. The perpendicular arrival angle implies that the input wavefront will have identical levels at each sensor. The magnitude of the beams formed in several directions, with the broadside presence of such a wavefront, can be plotted as the beam pattern of the four element array.

The waveform that will be used to test the system is a sine wave that is produced by an analog function generator. The output of the generator is connected to all of the sensor inputs of the prototype system in order to simulate a broadside wavefront arrival. The magnitude of the input waveform is kept in the order of tens of millivolts. The manual input gain switch is used to amplify the input and to keep it within ±5 Volts. The desired beam directions and the sensor locations are entered into the AT as they are needed. The output beams are observed and measured at the D/A output. They are selected, one at a time, using the thumbwheel switch on the front plane of the slave board. The output attenuations are recorded for a fixed input frequency and the same procedure is

repeated for different frequencies.

The resulting beam patterns for several test cases are presented in the figures on the following pages. The ideal beam patterns for each case are also shown. For a given array, as the input frequency is raised, one can observe an increasing number of discrepancies between the ideal beam pattern and the actual results. These are mainly due to the quantization of the time delays. In this case, the time delay resolution is 100 μs. which is increasingly inadequate for higher frequencies (shorter wavelengths). A detailed discussion on this topic can be found in reference[5]. Some of the lesser discrepancies in the attenuation levels are related to the imperfections of the measurements due to the high noise levels in the D/A circuitry. The D/A, in the constructed prtotype, only achieves a real resolution of 8 bits in the proximity of a very noisy wire-wrapped digital environment. Another cause of the minor attenuation discrepancies is the unreliable sine-wave oscillator that generates the input waveforms. The voltage output of this particular old oscillator is not stable and fluctuates by ±50 mV over time.

It can be safely concluded that better results can be achieved with a properly constructed production quality system.

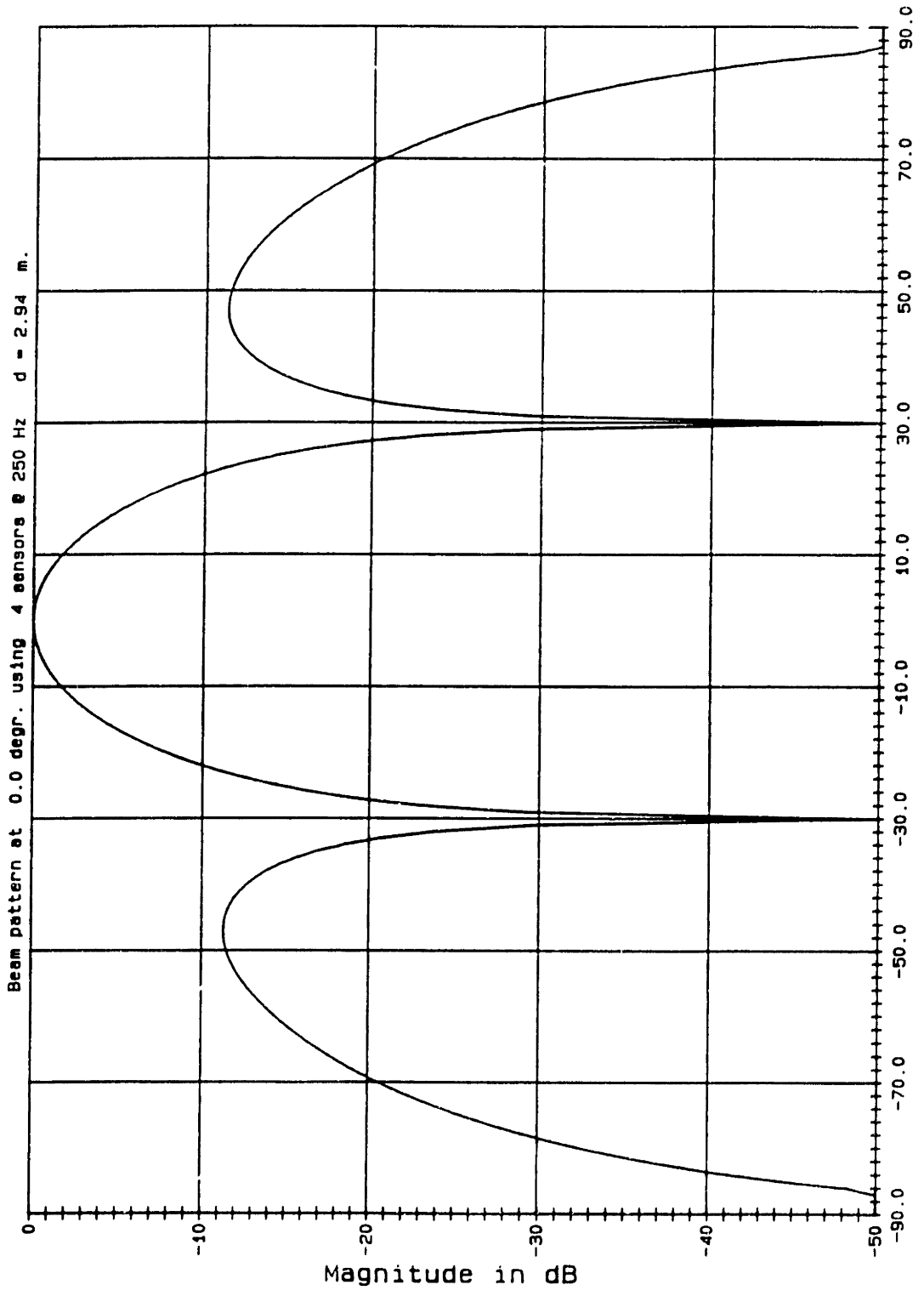


Figure 3-1 : Ideal beam pattern for test case @ 250 Hz with 4 sensors

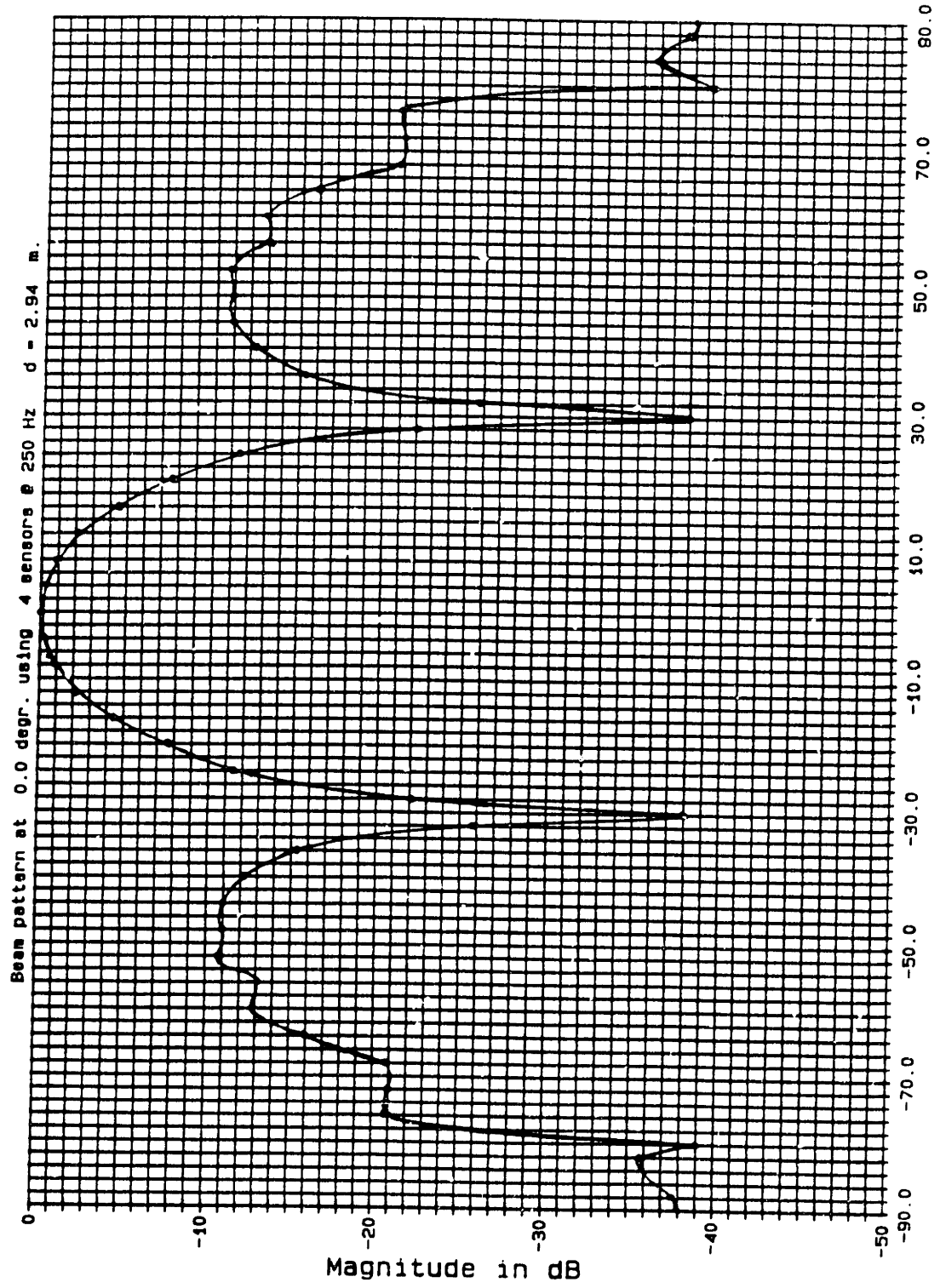


Figure 3-2 : Measured beam pattern for test case @ 250 Hz with 4 sensors

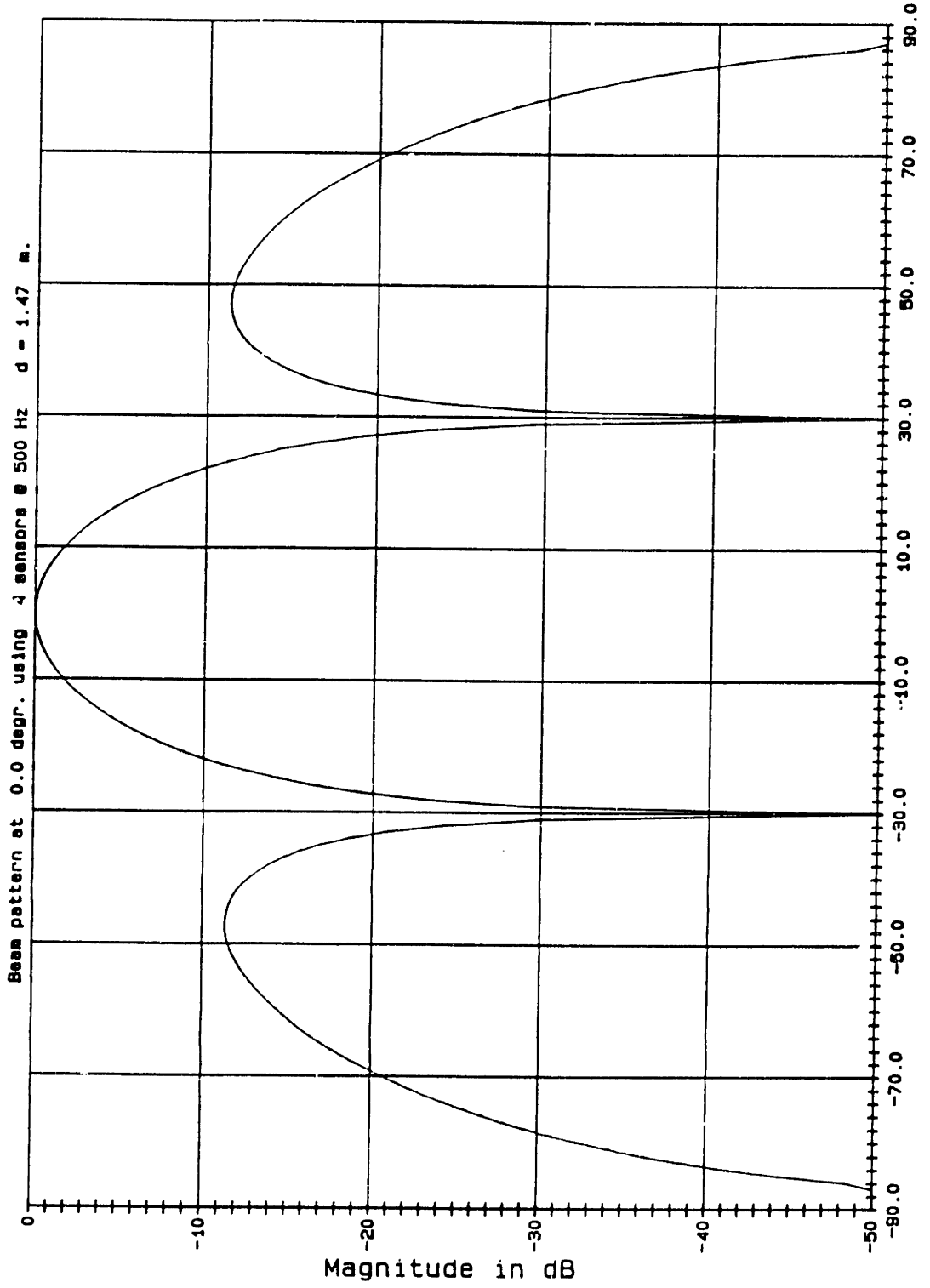


Figure 3-3 : Ideal beam pattern for test case @ 500 Hz with 4 sensors

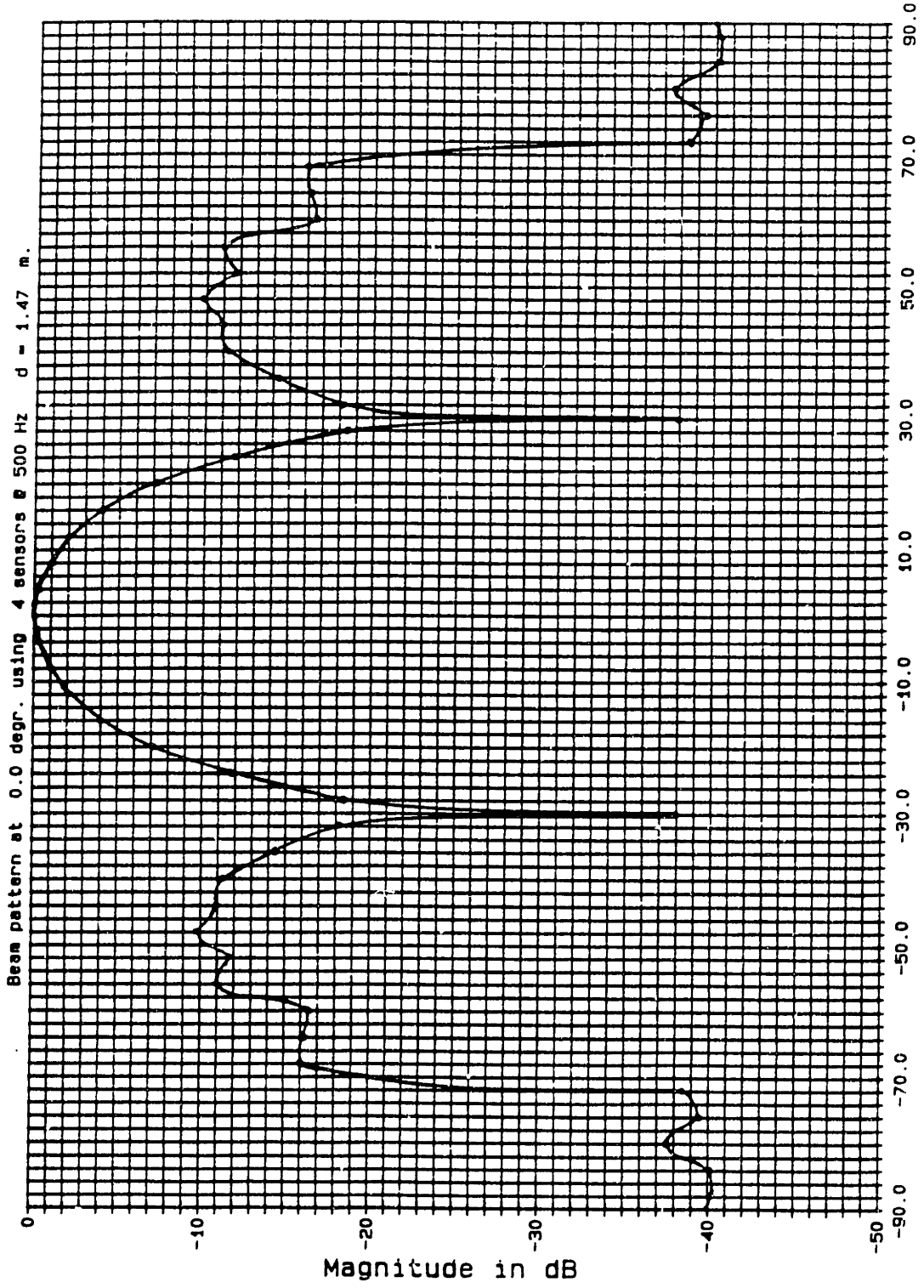


Figure 3-4 : Measured beam pattern for test case @ 500 Hz with 4 sensors

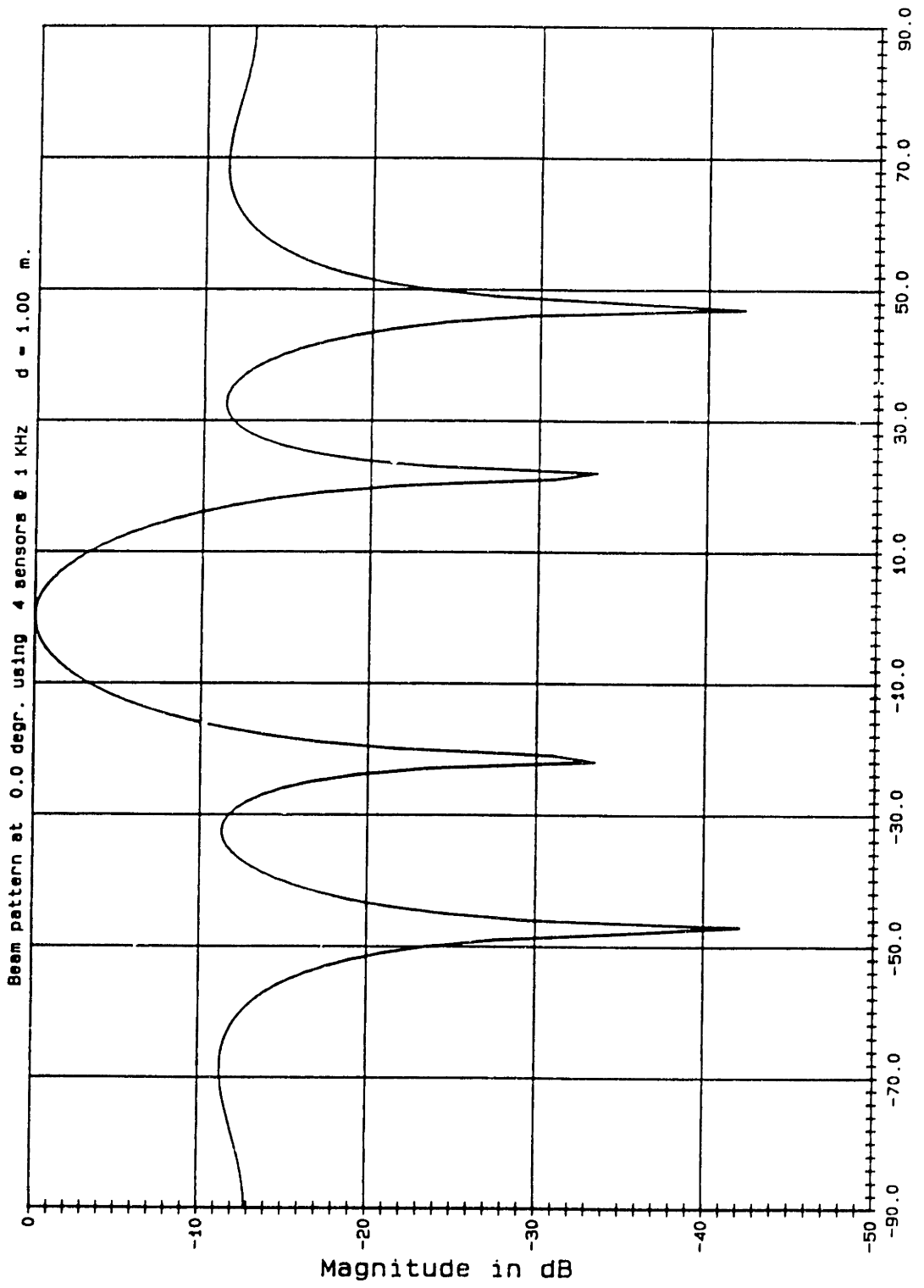


Figure 3-5 : Ideal beam pattern for test case @ 1 KHz with 4 sensors

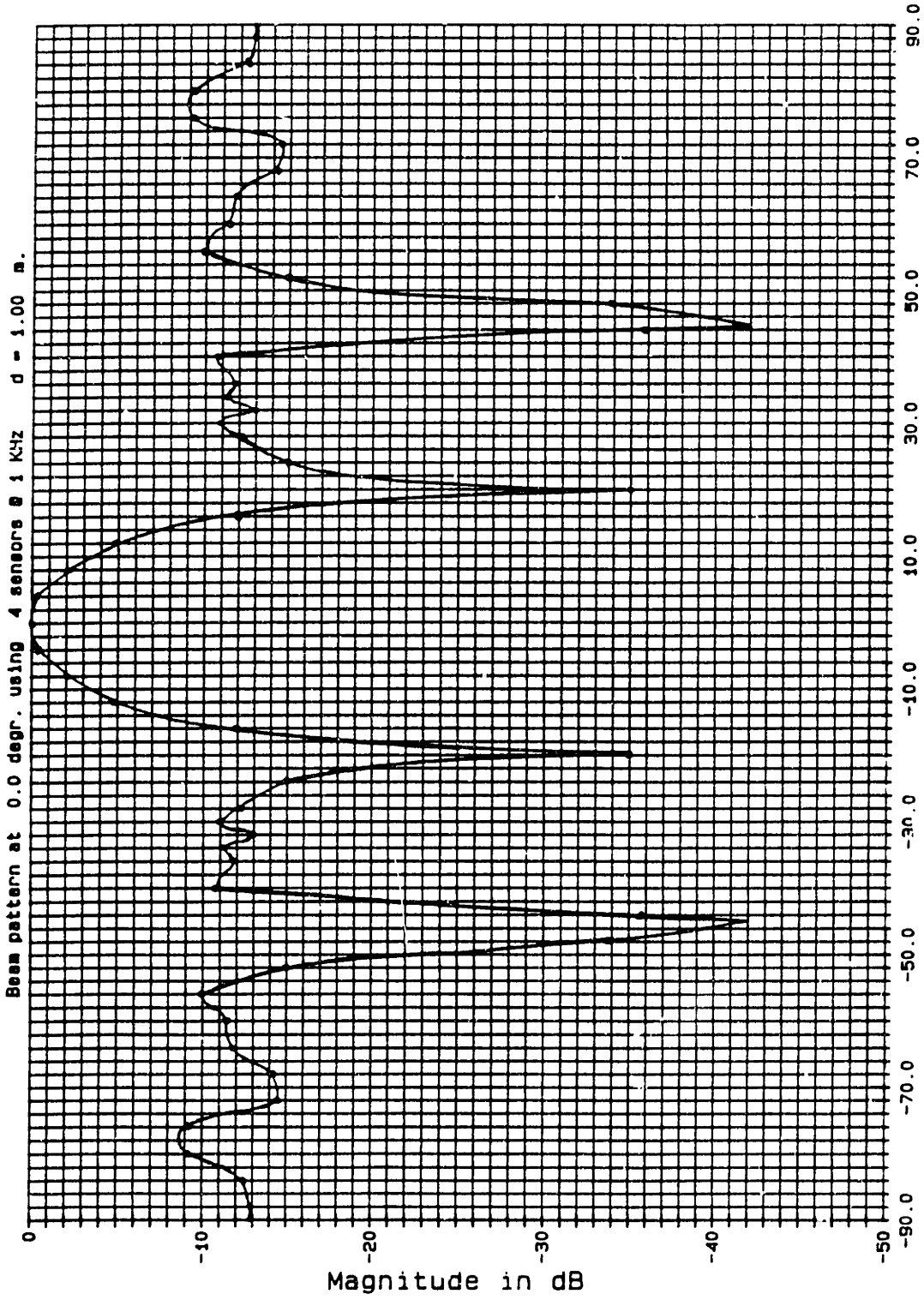


Figure 3-6 : Measured beam pattern for test case @ 1 KHz with 4 sensors

CHAPTER 4 : Concluding Notes

There are several ways to improve the performance, functionality and the user interface of the experimental research beamformer. Possible additional features, some architectural and circuit level enhancements and some production recommendations are briefly discussed in this chapter.

4.1 Possible Additional Features :

A large number of features can be added to this system without great difficulty. One possible feature is to provide the user with the option to choose the frequency of the input sampling clock. It is possible to incorporate additional hardware on the master board in order to allow an external clock to be routed to the A/D boards. This can be accomplished by adding the circuit shown on figure (3-1). /SELEXT and /SELINT are already existing decoder outputs that can be firmware activated. A male BNC connector, that would receive the external clock,could be added onto the master board. The external sampling clock option could be asked as an additional question by "Comm.c" during the system parameter configuration. The program would download the necessary information to the master which, in turn, activate the needed signals for the proper clock selection. Another duty of the program would be to modify the beam assignments, since the system cycle may be shorter or longer depending on the new sampling frequency.

A software feature that can be added is the ability to save the current system parameters into a special file. The system could be re-started by invoking "Comm.c" and instructing it to boot up the system using the parameters that were saved in this special file. This would also allow the user to edit this file and re-start the system with a new set of configuration parameters in a very short

time. Some software can be written in "C" to accomplish this task. This feature would make "Comm.c" even more user-friendly than it already is.

One final and important feature could be the addition of a filtering and smoothing circuit for the output of the D/A converter. By smoothing the output of the D/As, it is possible to feed the analog output beams into a spectrum analyzer, or a general purpose data acquisition system and perform some simple analysis without using the external sonar processor.

4.2 Possible Performance Improvements :

There are two essential aspects of a real-time beamforming system that always demand improvement. These aspects are the beam throughput and the maximum number of sensor inputs. There are several ways to improve these aspects by keeping the basic existing architecture and making relatively minor modifications to the circuitry, the firmware and the software. The modifications with ascending levels of complexity are discussed in this section.

The main performance issue for a real-time beamformer is the beam throughput. The main goal of such a system is to form as many simultaneous beams as possible using the existing technology. There are several ways to improve the beam throughput performance of the experimental research beamformer within its existing distributed processing architecture. The most obvious and relatively easiest way to achieve this goal is to replace the current ADSP-2100s with ADSP-2100As. The ADSP-2100A has an 83 ns. instruction cycle time as opposed to 125 ns. cycle time of the ADSP-2100. This upgrade would result in ≈50 % increase in system beam throughput. The ADSP-2100A is pin and source code compatible with the ADSP-2100. This allows the easy upgrade of the system with no firmware changes. The modifications that are needed for the CPU

upgrade are mostly in hardware. The timing requirements during the ADSP-2100A's data and program memory access operations must be analyzed carefully and faster devices should be placed on the critical data paths. It will probably be possible to just upgrade the current memory components in order to compensate for the new required shorter data and program memory access cycles. 2Kx8 Static RAMs with 15 and 20 ns. access times can probably be sufficient. These memory chips will be commercially available very soon. Another modification that would be necessary is in "Comm.c"; because it would have to be able to assign a larger number of beams per slave. At this stage, the input bandwidth of the external sonar processor would have to be carefully evaluated. This is necessary since it is very likely that the output bandwidth of the upgraded system, in maximum configuration, could be higher than the input capacity of the external processor. If such an incompatibility resulted, this would possibly result in using fewer number of slaves in order to match the output bandwith requirements. The overall consequence would be a cost reduction for the less demanding users and higher performance for the more demanding users.

The next important aspect to be considered for improvement is the maximum number of input sensors to be handled. It is possible to add more input channels to the experimental research beamformer. But, each added A/D converter would have to be placed in a unique location in the master CPU's data memory address space and this would require some additional address decoding circuitry on the A/D boards. There is enough room for more digital components on these boards and the changes in the wiring would be relatively simple. One important effect of such an addition is that the maximum number of beams that can be formed simultaneously might be reduced. This effect might take place because the master will have to read more inputs within one system cycle and

consequently might not have enough time to read all the results from the slave FIFOs. But it is also possible that this addition might not affect the system's beamforming capabilities, especially if ADSP-2100As are used as CPUs. These performance trade-offs must be carefully calculated before attempting to expand the A/D capabilities of the system.

The availability of various shading windows for the inputs could be another useful feature. The necessary work to implement this task would have to be done in Comm.c and the system firmware programs. The option to pick a particular shading window could be presented to the user during the user interface phase of the system operation. The shading coefficients could be calculated by Comm.c on the fly, and downloaded to the master. The master could send these coefficients to the slaves instead of sending the current unit rectangular window factor. The shading factors could reside in the program memory space of each slave and could ultimately used by the slaves during beamforming. The downloading overhead would be minimal but the additional program memory accesses during the beamforming loop might result in system performance degradation. The number of simultaneous beams that can be formed might get reduced because of the added overhead of reading the shading factors from program memory. Careful calculations are needed to determine the exact consequences of such a system modification.

Another improvement that can be done to the system is to redesign the A/D boards with multi channel A/D converter hybrids. Analog Devices' AD-1334, which was not available at the initial prototyping phase, would be the optimal choice for this purpose. The AD-1334 contains 4 sample and hold circuits, a 4 to 1 analog multiplexer, an AD-7672 12 bit, 5μs A/D converter and an output FIFO. The 4 sample and hold circuits (AD-585) are the same as the one used in

AD-1332. The A/D converter is also identical to the one in AD-1332. The output FIFO is 12 bits wide and 64 locations deep. It is possible to use this hybrid in a mode where all of the AD-585s would sample the inputs simultaneously. Some overhead analog circuitry must be added externally because of the lack of on board low-pass filters in the AD-1334. AD-1334 has the same package as the AD-1332, so it would be possible to fit as many as 3 A/D hybrid packages on the same board even with the addititonal digital and analog overhead circuitry that is needed. Such a construction strategy would allow each A/D board to handle up to 12 sensor inputs. Some minor modifications in the master firmware would also be needed in order to properly address the AD-1334s. The system redesign effort that is necessary to implement the substitution of AD-1334s is of moderate difficulty. The backplane slot savings would prove this redesign effort to be very valuable, especially if the need for more than 32 input channels is expected in the future.

4.3 Production Recommendations :

This section discusses several electrical and mechanical issues to be considered in order to send this system into production.

The system enclosure should be an industry standard 19" wide and 3U card cage capable of housing 20 Eurocards (110mm x 220mm).

The first major issue to be decided for production is the choice of the circuit boards. All the separate system modules should be constructed on printed circuit boards with buried power and ground planes. The power planes are absolutely necessary because of the large number of fast advanced schottky and advanced CMOS chips. A separate ground plane should be used for the analog components wherever necessary. The analog and digital ground planes should have a common connection as near to the A/D and D/A converters as possible. There

should be a single board entry point for each power and ground bus. Soldering should be used to mount the devices onto the printed circuit board. The backplane to be used should also be constructed as a printed circuit board. Buried power and ground buses should be routed over its whole length and brought to the 96 pin female connectors in order to provide power and grounds to the boards.

Power supply decoupling also should not be neglected. Each integrated digital circuit on the circuit boards should have their power and ground pins decoupled by at least 0.1μF ceramic capacitors. The +5 Volt power entry points for each board should be decoupled from the digital ground by polarized 300μF tantalum capacitors. Analog supply decoupling must be done as shown in the schematics.

The D/A converters on the slaves are more susceptible for noise interference than the A/D hybrids. This necessitates great care in routing the digital signals and the digital power planes around the D/A circuits. Some additional protection, such as a metal case around the D/A converters might be appropriate.

In order to aid the user, some LEDs (light emitting diodes) should be mounted in certain parts of the system. LEDs can be mounted on the front panel of each board that would be lit when all the power supplies for that board are functioning properly. Additional LEDs can be used to display the hex code selected by each thumbwheel switch on each slave. This would allow the user to detect whether the thumbwheel switches are functioning properly and selecting the correct output beam. The system /RESET switch, also, should be mounted in the front of the enclosure within user's easy reach.

Finally the last issue to be considered is the design and mounting of

termination resistor networks for each signal line that is routed over the backplane. This should be done after the decisions for all the boards, integrated circuits and connectors are completed. The reason is that they all contribute to the different parameters that will affect the necessary termination resistances.

3.4 Concluding Remarks :

The goals that were set in section 1.2.4 for an experimental beamformer for oceanographic research have been met. The resulting beamformer design, with no modifications, can take inputs from up to 32 hydrophones and form up to 32 simultaneous beams. The number of beams formed will increase if fewer number of hydrophones are used. The system is compact and relatively inexpensive to manufacture and has a friendly user interface. Its architecture is open and will allow the system to perform better by easy substitution of technologically more advanced components in the future. The system is ready for production and field testing as soon as the missing slave_capacity() function in the Comm.c program is completed.

BIBLIOGRAPHY

[1] A. A. Winder, "Sonar System Technology", IEEE Transactions on Sonics and Ultrasonics, Vol. SU-22, No. 5, September 1975.

[2] A. B. Baggeroer, "Sonar Signal Processing", in *Applications of Digital Signal Processing* , A. V. Oppenheim, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1978, Ch. 6.

[3] T. E. Curtis and R. J. Ward, "Digital Beam Forming for Sonar Systems", IEE Proc., Vol. 127, Pt. F, No. 4, August 1980.

[4] W. C. Knight, R. Pridham and S. M. Kay, "Digital Signal Processing For Sonar", Proc. IEEE, Vol. 69, No. 11, November 1981.

[5] D. A. Gray, "Effect of Time-Delay Errors on the Beam Pattern of a Linear Array", IEEE Jour. of Oceanic Engineering, Vol. OE-10, No. 3, July 1985.

[6] R. G. Pridham and R. A. Mucci, "A Novel Approach to Digital Beamforming", Jour. Acoust. Soc. Am. 63(2), February 1978.

[7] R. J. Janssen "Sonar Beamforming and Signal Processing", Electronic Progress, Vol. XXVIII No. 1, Raytheon Co., 1987.

[8] W. S. Hodgkiss and V. C. Anderson, "Hardware Dynamic Beamforming", Jour. Acoust. Soc. Am. 69(4), April 1981.

[9] Analog Devices, DSP Division Technical Staff, *ADSP-2100 User's Manual*. Analog Devices Inc., 1986.

[10] Analog Devices, DSP Division Technical Staff, *ADSP-2100 Data Sheet*. Analog Devices Inc., 1986.

APPENDIX

Appendix A1

```
.system                          master_system;

.seg/rom/abs=0/pm/code           rom_program_storage[2048];

.seg/ram/abs=0/dm/data           sample_mem[8160];
.seg/ram/abs=8160/dm/data        system_info[32];
.seg/ram/abs=8192/dm/data        shading_coeff_mem[32];
.seg/ram/abs=8224/dm/data        scratch_mem[2016];
.seg/ram/abs=10240/dm/data       ad_read[32];
.seg/ram/abs=14337/dm/data       fifo_shift_out[7];

.port/abs=h#300f                 setbmoutrdy;
.port/abs=h#304f                 clrbmoutrdy;
.port/abs=h#308f                 pcwe;
.port/abs=h#310f                 setsbr;
.port/abs=h#314f                 clrsbr;
.port/abs=h#318f                 clrslhalt;
.port/abs=h#31cf                 setslhalt;
.port/abs=h#30c8                 setbpoe;
.port/abs=h#30c9                 clrbpoe;
.port/abs=h#30ca                 clrbmtaken;
.port/abs=h#30cd                 setsmemrd;
.port/abs=h#30ce                 clrsmemrd;
.port/abs=h#30cf                 setslint;
.port/abs=h#3007                 pcrd;
.port/abs=14336                  beamsend;


.endsys;
```

Appendix A2

```
.module/rom/abs=0       master_code;

.port                   setbmoutrdy;
.port                   clrbmoutrdy;
.port                   pcwe;
.port                   clrsbr;
.port                   setsbr;
.port                   clrslhalt;
.port                   setslhalt;
.port                   setbpoe;
.port                   clrbpoe;
.port                   clrbmtaken;
.port                   setsmemrd;
.port                   clrsmemrd;
.port                   setslint;
.port                   pcrd;
.port                   beamsend;

.var/dm/ram/abs=8160    sensor_num;
.var/dm/ram/abs=8161    beam_num;
.var/dm/ram/abs=8162    index_num;
.var/dm/ram/abs=8163    slave_num;
.var/dm/ram/abs=8164    beams_per_slave;
.var/dm/ram/abs=8165    last_slave_beam_num;


                        jump adcomplete;        {vectored addr. for interrupt 0}
                        rti;
                        jump beamtaken;                     {for interrupt 2}
                        rti;                                {for interrupt 3}


                        imask=b#0000;
                        icntl=b#00000;
                        ay0=h#0fff;
                        dm(clrbmoutrdy)=mx0;
                        dm(clrbmtaken)=mx0;
                        dm(clrsbr)=mx0;
                        dm(clrslhalt)=mx0;
                        dm(clrbpoe)=mx0;
                        dm(clrsmemrd)=mx0;
                        nop;

pc_init_wait:           ax0=dm(pcrd);           {wait for the ready message}
                        ar=ax0-ay0;             {from the PC}
                        if eq jump pc_comm;
                        jump pc_init_wait;

pc_comm:                mx0=h#ff0f;             {initial set up before the}
                        ay0=5;                  {PC communications}
                        af=ay0+1;               {a message is sent to the PC}
                        ay0=h#1fff;             {at the end of this routine}
                        ax1=h#ffff;             {signaling that the master is}
                        mx1=h#ff0f;             {ready}
                        i1=8160;
                        m1=1;
                        l1=6;
                        i2=0;
                        m2=1;
```

```
                        l2=8160;
                        m3=-1;
                        i4=8166;
                        m4=1;
                        i5=9000;
                        m5=0;
                        l5=1;
                        i6=0;
                        m6=1;
                        l6=2055;
                        dm(setsbr)=ay0;
                        nop;
                        nop;
                        dm(pcwe)=mx0;
                        jump pc_wait1;

pc_comm_end_check:      ay1=i3;                 {counter register to check for}
                        ar=ay1-1;               {the end of the index}
                        if lt jump pc_end;      {downloading}

pc_wait1:               ay1=dm(pcrd);           {wait for ffff from the PC}
                        ar=ay1-ax1;
                        if eq jump pc_read;
                        jump pc_wait1;

pc_read:                modify(i6,m6);
                        af=af-1;                {write the initial 6 pieces}
                        if lt jump index_store; {of data and then start the}
init_read:              ax0=dm(pcrd);           {index storage}
                        ay1=h#e000;
                        ar=ax0 and ay1;         {mask out the bottom 13 bits}
                        ay1=h#a000;
                        ar=ar-ay1;
                        if eq jump init_sto;    {compare to h#a000}
                        jump init_read;
init_sto:               ar=ax0 and ay0;         {mask out the top 3 bits}
                        dm(setbpoe)=mx0;
                        nop;
                        dm(i1,m1)=ar;
                        dm(setsmemrd)=mx0;
                        nop;
                        nop;
                        nop;
                        nop;
                        nop;
                        l4=dm(beam_num);
                        l4=dm(beam_num);
                        i3=dm(index_num);
                        dm(clrsmemrd)=mx0;
                        l3=i3;
                        nop;
                        nop;
                        nop;
                        dm(clrbpoe)=mx0;
                        jump pc_wait1;

index_store:            ax0=dm(pcrd);           {read the pc output}
```

```
                    ay1=h#e000;
                    ar=ax0 and ay1;
                    ay1=h#a000;
                    ar=ar-ay1;
                    if eq jump sto;          {compare to h#a000}
                    jump index_store;
sto:                ar=ax0 and ay0;
                    dm(s:tbpoe)=mx0;
                    nop;
                    dm(i2,m2)=ar;            {store index}
                    modify(i3,m3);
                    dm(clrbpoe)=mx0;
                    jump pc_comm_end_check;

pc_end:             ax1=h#f0ff;
pc_end_loop:        ay1=dm(pcrd);
                    ar=ax1-ay1;
                    if eq jump checksum;
                    jump pc_end_loop;
checksum:           ax1=i6;
checksum_loop:      ay1=dm(pcrd);
                    ar=ax1-ay1;
                    if eq jump comm_end;
                    jump checksum_loop;

comm_end:           mx0=h#f0;
                    dm(pcwe)=mx0;
                    dm(setbpoe)=mx1;
                    ay0=h#0201;
                    af=ay0-1;        {af=1}
                    ay0=0;
                    dm(setsmemrd)=mx0;
                    nop;
                    nop;
                    nop;
                    nop;
                    nop;
                    si=dm(sensor_num);
                    si=dm(sensor_num);
                    dm(clrsmemrd)=mx0;
                    nop;
                    nop;
                    nop;
                    nop;
                    ay0=si;
                    ar=ay0-1;
                    if eq jump onesensor;
                    ay0=0;
                    sr=lshift si by 9 (lo);
                    ax1=sr0;
                    astat=0;
                    divq ax1;
                    divq ax1;divq ax1;divq ax1;
                    divq ax1;divq ax1;divq ax1;
                    divq ax1;divq ax1;divq ax1;
                    divq ax1;divq ax1;divq ax1;
                    divq ax1;divq ax1;divq ax1;
                    dm(8192)=ay0;            {shading coefficient}
                    jump allsensor;
onesensor:          ay0=h#7fff;
```

```
                    dm(8192)=ay0;
allsensor:          dm(setsmemrd)=mx1;
                    nop;
                    nop;
                    nop;
                    nop;
                    nop;
                    ay0=dm(slave_num);
                    ay0=dm(slave_num);              {do all the necessary}
                    ax1=dm(beams_per_slave);        {DMreads before the}
                    my1=dm(last_slave_beam_num);    {release of SBR}
                    mx0=dm(sensor_num);
                    dm(clrsmemrd)=mx0;
                    nop;
                    my0=255;
                    mr=mx0*my0(uu); {total # of samples to be put into}
                    si=mr0;         {the sample buffer}
                    sr=lshift si by -1 (hi);
                    l2=sr1;
                    ar=ay0-1;
                    dm(clrsbr)=mx1;
                    nop;
                    ax0=ar;
                    i1=10240;
                    m1=1;
                    l1=mx0;
                    i2=0;
                    m2=1;
                    if eq jump fix_base;
                    i3=h#3800;              {base addr. for the FIFOs}
                    m3=1;                   {with multiple slaves}
                    ar=ay0+1;
                    l3=ar;
                    jump normal;
fix_base:           i3=h#3801;              {base addr. for the FIFO}
                    m3=0;                   {with single slave}
                    l3=0;
normal:             dm(setslhalt)=mx1;      {HALT the slave}
                    nop;
                    dm(clrslhalt)=mx1;
                    dm(clrbpoe)=mx1;        {enable the slvtrap interrupt}
                    cntr=2000;
                    do slave_wait until ce;
slave_wait:         nop;
                    ar=1;
                    ay1=1;
                    icntl=b#00101;
                    imask=b#0001;

wait:               jump wait;

adcomplete:         ar=ar-ay1;                  {we ignore the first}
                    if eq jump first_adcomp;    {IRQ0 by using this}
                    dm(setsbr)=mx1;
                    af=ay0-1;
                    dm(setbpoe)=mx1;            {this routine is used}
                    cntr=mx0;                   {to fill up the sample}
                    do sample_store until ce;   {memory after each}
                    mx1=dm(i1,m1);              {A/D conversion}
sample_store:       dm(i2,m2)=mx1;
```

```
                    dm(clrsbr)=mx1;
                    nop;
                    dm(setslint)=mx1;
                    dm(clrbpoe)=mx1;


sendbeam:           if eq jump oneslave;     {read out the beams in a}
                    m3=1;
                    modify(i3,m3);
                    m3=0;
                    cntr=ax0;                {sequential manner within each}
                    do beamout until ce;     {FIFO in order of numbering}
                    cntr=ax1;
                    do fifo_out until ce;
                    dm(setbpoe)=mx1;
                    nop;
                    mx1=dm(i3,m3);
                    dm(clrbpoe)=mx1;
                    dm(beamsend)=mx1;
                    imask=b#0100;
                    dm(setbmoutrdy)=mx1;     {set a flag for the external}
                    cntr=6;                  {sonar processor}
                    do resp_wait until ce;
resp_wait:          nop;
fifo_out:           nop;
                    m3=1;
                    modify(i3,m3);
beamout:            m3=0;


oneslave:           cntr=my1;
                    do endfifo until ce;
                    dm(setbpoe)=mx1;
                    nop;                     {this routine is for single}
                    mx1=dm(i3,m3);           {slave configurations and}
                    dm(clrbpoe)=mx1;         {is also used for handling}
                    dm(beamsend)=mx1;        {the last FIFO to be read out}
                    imask=b#0100;            {it copes with the irregularity}
                    dm(setbmoutrdy)=mx1;     {of the last set of beams}
                    cntr=6;                  {i.e. possibly fewer # of beams}
                    do resp_wt until ce;
resp_wt:            nop;
endfifo:            nop;
first_adcomp:       ar=0;
                    rti;

beamtaken:          dm(clrbmoutrdy)=mx1;     {int. routine to handle}
                    dm(clrbmtaken)=mx1;      {KVDH's confirmation}
                    rti;

.endmod;
```

Appendix B1

```
.system                          slave_system;

.seg/rom/abs=0/pm/code           rom_program_storage[2048];
.seg/ram/abs=2048/pm/data        index_mem[2048];

.seg/ram/abs=0/dm/data           sample_mem[8160];
.seg/ram/abs=8160/dm/data        system_info[32];
.seg/ram/abs=8192/dm/data        shading_coeff_mem[32];
.seg/ram/abs=8224/dm/data        scratch_mem[2016];

.port/abs=h#2800                 beamout;
.port/abs=h#3000                 beamdac;
.port/abs=h#3800                 clrslint;
.port/abs=h#3900                 slave_id;
.port/abs=h#3A00                 dac_beam_sel;
.endsys;
```

```
.module/rom/abs=0       slave_code;

.port                   beamout;
.port                   beamdac;
.port                   clrslint;
.port                   slave_id;
.port                   dac_beam_sel;

.var/dm/ram/abs=8160    sensor_num;
.var/dm/ram/abs=8161    beam_num;
.var/dm/ram/abs=8162    index_num;
.var/dm/ram/abs=8163    slave_num;
.var/dm/ram/abs=8164    beams_per_slave;
.var/dm/ram/abs=8165    last_slave_beam_num;

                        jump beam_form;
                        rti;
                        rti;
                        rti;

                        imask=b#0000;
                        icntl=b#00000;
                        dm(clrslint)=mx0;

                        trap;                   {waiting for pc_comm to end}

                        i1=0;                   {this initial routine is used}
                        m1=1;                   {to transfer the indexes from}
                        l1=dm(index_num);       {the sample_mem into the}
                        i4=2048;                {index_mem in PM}
                        m4=1;
                        l4=l1;
                        cntr=l1;
                        do index_store until ce;
                        mx0=dm(i1,m1);
index_store:            pm(i4,m4)=mx0;

                        mx0=dm(sensor_num);     {<-- # of sensors}
                        my0=255;
                        mr=mx0*my0(uu);
                        si=mr0;
                        sr=lshift si by -1 (hi);
                        i1=mx0;
                        m1=mx0;
                        l1=sr1;             {<-- length of circular buffer}
                        si=dm(slave_id);            {determine this slave's}
                        sr=lshift si by -12 (hi);   {i.d. and the starting}
                        ax0=dm(slave_num);          {location of the first}
                        ay0=sr1;                    {index.Also determine}
                        af=ax0-ay0;                 {the # of indexes}
                        af=af-1;                    {for this slave}
                        if eq jump last_slave;
                        se=dm(beams_per_slave);
                        jump all_slave;
last_slave:             se=dm(last_slave_beam_num);
all_slave:              mx0=dm(sensor_num);
                        my0=dm(beams_per_slave);
                        mr=mx0*my0(uu);
                        si=mr0;
                        sr=lshift si by -1 (hi);
```

```
                        mx0=sr1;
                        ar=ax0-ay0;
                        ay1=ar;
                        ar=ay1-1;
                        my0=ar;
                        mr=mx0*my0(uu);
                        si=mr0;
                        sr=lshift si by -1 (hi);
                        ax1=2048;
                        ay1=sr1;
                        ar=ax1+ay1;
                        i5=ar;      {<-- starting addr. of the indexes}
                        m5=1;                          {for this slave}
                        mx0=se;
                        my0=dm(sensor_num);      {<-- index per beam}
                        mr=mx0*my0(uu);
                        si=mr0;
                        sr=lshift si by -1 (hi);
                        l5=sr1;          {<-- total # of indexes to be used}
                        m3=0;                          {by this slave}
                        l3=l1;
                        i6=8192;
                        m6=0;
                        l6=1;
                        my1=dm(i6,m6);      {<-- shading coefficient , only one}
                        ax1=dm(sensor_num);      {for now: rectangular window}
                        icntl=b#00001;
                        imask=b#0001;

wait:                   si=dm(dac_beam_sel);     {setup the down counter to be}
                        sr=lshift si by -12(hi);{used in the dac_write process}
                        ay1=sr1;
                        af=ay1+1;
                        jump wait;       {wait for a circular buffer update}

beam_form:              nop;
                        cntr=se;
                        do beam_end until ce;
                        mr=0;
                        cntr=ax1;
                        do single_beam_sample until ce;
                        ax0=pm(i5,m5);           {read index}
                        m3=ax0;
                        i3=i1;
                        modify(i3,m3);           {<-- modified index}
                        mx0=dm(i3,m3);           {<-- get the desired sample}
single_beam_sample:     mr=mr+mx0*my1(rnd);      {<-- MAC it !!}
                        af=af-1;
                        if eq jump dac_write;
beam_end:               dm(beamout)=mr1;         {<-- write result to fifo}
                        modify(i1,m1);     {<-- advance the circ. buffer pointer}
                        dm(clrslint)=mx0;
                        nop;
                        rti;

dac_write:              dm(beamdac)=mr1;         {<-- write result to DAC}
                        nop;
                        jump beam_end;

.endmod;
```

Appendix C

```
#include <stdio.h>
#include <math.h>
#include <setjmp.h>
#define input 0x303
#define out0 0x301
#define out1 0x302

jmp_buf env;
int index_num = 0;
int sensor_num , beam_num , slave_num , beams_per_slave , last_slave_bm_num;
float loc[32][3];
float ang[64][2];
float dir[64][3];
int ind[32][64];

main()          /*this is the program for the user-master communications*/

{
        setjmp(env);
        user_io();
        index_calculate();
        slave_capacity();
        download();

}


user_io()
{
extern int sensor_num , beam_num , slave_num;
extern float loc[32][3];
extern float ang[64][2];
int i = 0;
int sens_cntr , beam_cntr;
float a,b;

        while (++i < 12){
                printf("\n");
        }
        printf("THIS IS THE SETUP INTERFACE FOR THE BEAMFORMER\n");
        printf("PLEASE INPUT THE REQUESTED INFORMATION:\n\n");
        printf("Enter the number of sensors: ");
        scanf("%d", &sensor_num);
        printf("\nEnter the number of beams to be formed: ");
        scanf("%d", &beam_num);
        printf("\nEnter the number of slaves in the system: ");
        scanf("%d", &slave_num);
        sens_cntr = sensor_num;
        sens_cntr++;
        while (sens_cntr-- >1){

                printf("\nEnter the x,y,z coordinates for sensor#%d: "
                        ,sens_cntr);
                scanf("%f,%f,%f", &loc[sens_cntr][1], &loc[sens_cntr][2]
                        ,&loc[sens_cntr][3]);
        }
        beam_cntr = beam_num;
        beam_cntr++;
        while (beam_cntr-- >1){
```

```
                printf("\nEnter the elevation,azimuthal angles for beam#%d: "
                        ,beam_num - beam_cntr);
                scanf("%f,%f",&a,&b);
                ang[beam_cntr][1] = a * 0.0174533; /* convert degrees to */
                ang[beam_cntr][2] = b * 0.0174533; /* radians            */
        }
        printf("\n........Thank you! Please wait for further instructions.");
}


index_calculate()
{
extern int sensor_num ,  beam_num , slave_num;
extern float loc[32][3];
extern float ang[64][2];
extern float dir[64][3];
extern int ind[32][64];
int sens_cntr , beam_cntr , cntr;
int s = 1470;           /* speed of propagation for sound in the ocean */
float a,b,c,d;
float half_buffer_size;
        cntr = beam_num;
        cntr++;
        while (cntr-- >1){/* determine the cartesian coord. for beam direction*/

                dir[cntr][1] = cos(ang[cntr][2]) * sin(ang[cntr][1]);
                dir[cntr][2] = sin(ang[cntr][2]) * sin(ang[cntr][1]);
                dir[cntr][3] = cos(ang[cntr][1]);

        }
        sens_cntr = sensor_num;
        sens_cntr++;
        while (sens_cntr-- >1){/*determine delay indexes per beam per sensor*/

                beam_cntr = beam_num;
                beam_cntr++;
                while (beam_cntr-- >1){

                        index_num++;
                        a = (loc[sens_cntr][1] * dir[beam_cntr][1] +
                             loc[sens_cntr][2] * dir[beam_cntr][2] +
                             loc[sens_cntr][3] * dir[beam_cntr][3])/s;
                        b = sens_cntr - 1 + sensor_num * rnd(a * 10000);
                        half_buffer_size = 127 * sensor_num;
                        c = half_buffer_size;

                        if (b > -c && b < c)
                                ind[sens_cntr][beam_cntr] = b + c;
                        else{
                                printf("\n beam#%d won't have a valid output!",
                                        beam_cntr);
                                ind[sens_cntr][beam_cntr] = 0;
                             }
                        }
                }
}


slave_capacity()
/*****************************************************************************/
```

```
/*THIS ROUTINE WILL CALCULATE THE BEAM-FORMING CAPACITY OF THE SYSTEM.*/
/*IT WILL ALSO TELL THE USER WHICH BEAMS ARE BEING FORMED BY WHICH SLAVE AND*/
/*THEIR OUTPUT ORDER.*/
/*****************************************************************************/
{
extern int beam_num , beams_per_slave , last_slave_bm_num;

beams_per_slave = 5;
last_slave_bm_num = beam_num;
}


rnd(x)    /*this function rounds a real number to its closest integer*/
float x;
{
int z;
        if (x - (int) x >= 0.5)
                z = (int) x + 1;
        else
                z = (int) x;
        return z;
}



download()
{
extern int sensor_num , beam_num , slave_num;
extern int beams_per_slave , last_slave_bm_num;
extern int ind[32][64];
int choice , go;
int mrdy = 0x0f;        /*master sends this when*/
int pchigh = 0xff;      /*it is ready to receive data*/
int sens_cntr , beam_cntr;
lo; a;
int b , c;

begin:  printf("\n Downloading beamformer initialization data....");

        outp(out0 , iinv(0xff));/*pc is ready to transmit,sends h#0fff*/
        outp(out1 , iinv(0xf));
        wt(mrdy);               /*the pc first downloads 6 pieces of*/
        send(sensor_num);       /*setup data*/
        send(beam_num);
        send(index_num);
        send(slave_num);
        send(beams_per_slave);
        send(last_slave_bm_num);

        beam_cntr = beam_num;
        beam_cntr++;
        while (beam_cntr-- >1) {

                printf("*");
                sens_cntr = sensor_num;
                sens_cntr++;
                while (sens_cntr-- >1) {

                        send(ind[sens_cntr][beam_cntr]);
                }
```

```
        }

        outp(out0 , iinv(0xff));        /*End of data is signaled*/
        outp(out1 , iinv(0xf0));
        a = index_num + 6;      /*checksum is about to be sent*/
        b = (int) a/256;
        c = a - b * 256;
        outp(out0 , iinv(c));
        outp(out1 , iinv(b));
        a = inp(input);
        if (a == 0x0f)          /*check master's response to checksum*/
                {
                printf("\n\n DOWNLOADINNG SUCCESSFULLY COMPLETED !! ");
                }
        else
                {
                printf("\n\n********** CHECKSUM ERROR ******************");
error:          printf("\n\nDo you want to... 1: retry downloading?");
                printf("\n                   2: abort the operation?");
                printf("\n Enter 1 or 2  :");
                scanf("%d", &choice);
                if (choice == 1)
                        {
                        printf("\n Reset the beamformer and hit 1 again :");
wait:                   scanf("%d", &go);
                        if (go == 1)
                                {
                                printf("\n\n RETRYING !!!....");
                                goto begin;
                                }
                        else
                                goto wait;
                        }
                else
                        {
                        if (choice == 2)
                                {
                                printf("\n\n ABORTING !!!....");
                                printf("\n Reset the beamformer!");
                                longjmp(env,0);
                                }
                        else
                                goto error;
                        }
                }

}


send(x)         /*this routine splits up the data to be sent into its lower*/
int x;          /*and higher words and outputs them to the ports*/
{
int y,z;                /*all valid data that will be downloaded is preceded*/
int pchigh = 0xff;      /*by b#101 in the highest three bits*/

                outp(out0 , iinv(pchigh));
                outp(out1 , iinv(pchigh));
                if (x - 255 <= 0)
                        {
```

```
                        outp(out0 , iinv(x));
                        outp(out1 , iinv(0xa0));
                        }
                else
                        {
                        y = (int) x/256;
                        z = x - y * 256;
                        outp(out0 , iinv(z));
                        outp(out1 , iinv(0xa0 + y));
                        }
}



wt(x)           /*this function will make the pc wait until the argument*/
int x;          /*appears at the input port from the master*/
{
        while ( inp(input) != iinv(x));
}


linv(w)         /*this function inverts all the bits of its long argument*/
long w;
{
long z;
        z = 0xffff - w;
        return z;
}

iinv(w)         /*this function inverts all the bits of its int argument*/
int w;
{
int z;
        z = 0xff - w;
        return z;
}
```

ANALOG DEVICES

DSP Microprocessor

ADSP-2100

FEATURES

- Separate Program and Data Buses, Extended Off-Chip
- Single-Cycle Direct Access to 16K × 16 of Data Memory
- Single-Cycle Direct Access to 16K × 24 (Expandable to 32K × 24) of Program Memory
- Dual Purpose Program Memory for Both Instruction and Data Storage
- Three Independent Computational Units: ALU, Multiplier/Accumulator and Barrel Shifter
- Two Independent Data Address Generators
- Powerful Program Sequencer
- Internal Instruction Cache
- Provisions for Multiprecision Computation and Saturation Logic
- Single-Cycle Instruction Execution
- Multifunction Instructions
- Four External Interrupts
- 125ns Cycle Time
- 475mW Maximum Power Dissipation with CMOS Technology (J and K Grades)
- 100-Pin Grid Array, 100-Lead PLCC

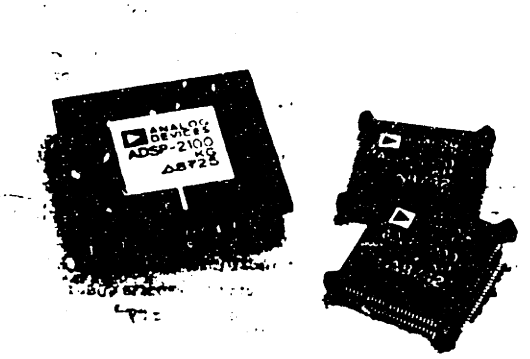
APPLICATIONS

- Optimized for DSP Algorithms Including
  - Digital Filtering
  - Fast Fourier Transforms
- Applications Include
  - Image Processing
  - Radar, Sonar
  - Speech Processing
  - Telecommunications

GENERAL DESCRIPTION

The ADSP-2100 is a single-chip microprocessor optimized for digital signal processing (DSP) and other high-speed numeric processing applications. It integrates computational units, data address generators and a program sequencer in a single device.

The ADSP-2100 makes efficient use of external memories for program and data storage, freeing silicon area for increased processor performance. The resulting architecture combines the functions and performance of a bit-slice/building block system with the ease of design and development of a general-purpose microprocessor. The ADSP-2100 (K and T grades) operates at 8.192MHz. Every instruction executes in a single 125ns cycle. Fabricated in a high-speed 1.5 micron double-layer metal CMOS process, the ADSP-2100 dissipates less than 475mW (J and K grades).



The ADSP-2100's flexible architecture and comprehensive instruction set support a high degree of operational parallelism. In one cycle the ADSP-2100 can:

- generate the next program address
- fetch the next instruction
- perform one or two data moves
- update one or two data address pointers
- perform a computational operation.

DEVELOPMENT SYSTEM

The ADSP-2100 is supported by a complete set of tools for software and hardware system development. The Cross-Software System provides a System Builder for defining the architecture of systems under development, an Assembler, a Linker and a Simulator. The Simulator provides an interactive instruction-level simulation. A PROM Splitter generates PROM burner compatible files. An Emulator is available for hardware debugging of ADSP-2100 systems.

ADDITIONAL INFORMATION

For additional information on the architecture and instruction set of the processor, refer to the *ADSP-2100 User's Manual*. For more information about the Development System, refer to the *ADSP-2100 Cross-Software Manual* and the *ADSP-2100 Emulator Manual*. For examples of a variety of ADSP-2100 applications routines, refer to the *ADSP-2100 Applications Handbook, Volume 1* or *Volume 2*. Manuals are available only from your local Analog Devices sales office. See ordering information.

One Technology Way; P. O. Box 9106; Norwood, MA 02062-9106 U.S.A
Tel: 617/329-4700 Twx: 710/394-6577
Telex: 924491 Cables: ANALOG NORWOODMASS

ANALOG DEVICES

COMPLETE
12-BIT SAMPLING A/D CONVERTER
FOR DIGITAL SIGNAL PROCESSING

AD1332

PRELIMINARY

FEATURES

Complete A/D System for DSP Includes:
- 4th-Order Anti-Aliasing Filter
- 12-Bit Sampling A/D Converter
- 32-Word Deep FIFO Memory Buffer
- Fully Asynchronous High Speed Digital Interface

Sample Rate up to 125KHz
Entire System is Dynamically Characterized
15ns Data Access Time Allows "no wait state" Interface to:
ADSP2100, TMS320C25
DSP56000, NECμPD77230

APPLICATIONS

Sonar Signal Processing
Vibration Analysis
Ultra Sound Imaging

PRODUCT DESCRIPTION

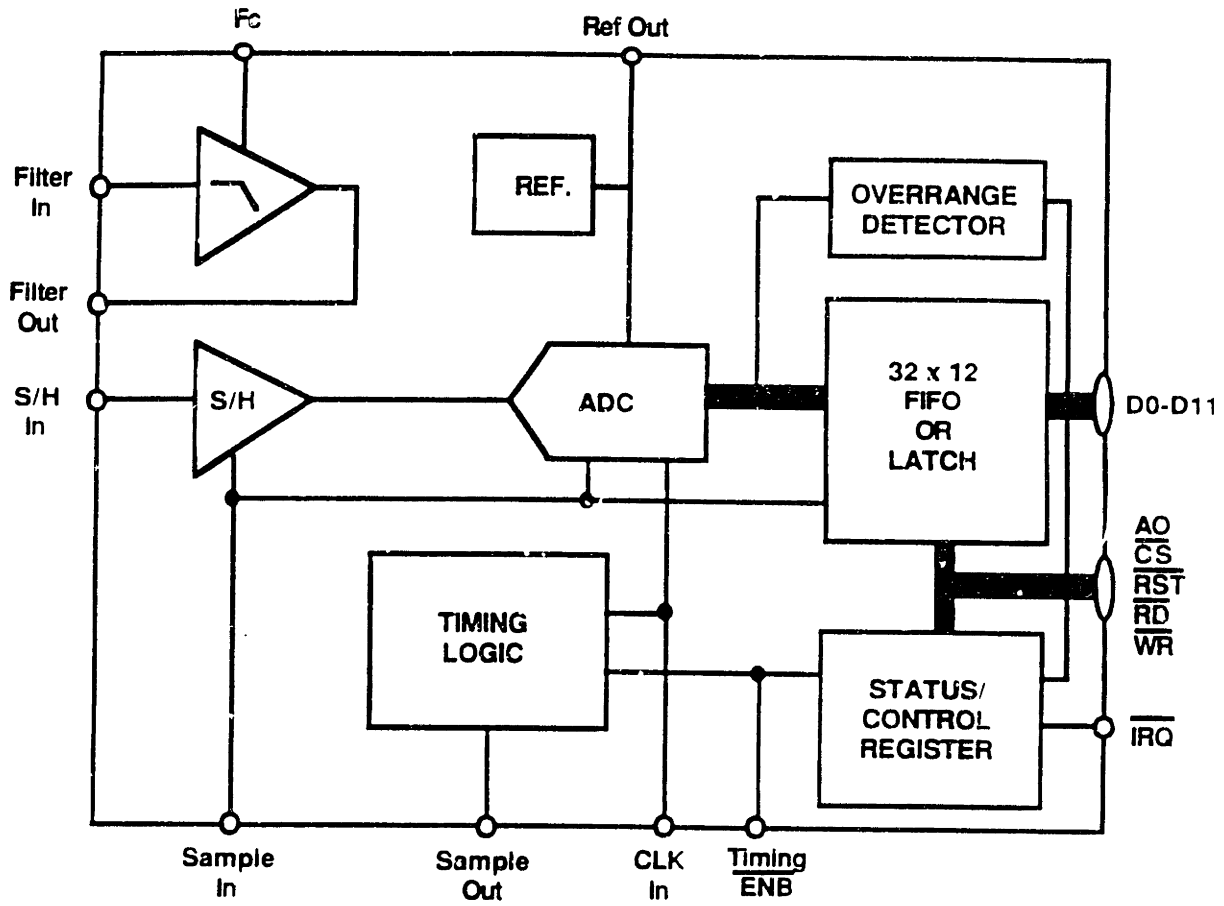
The AD1332 is a complete, 12-bit A/D converter system optimized for use in Digital Signal Processing (DSP) applications. The device consists of a fourth-order anti-aliasing filter, a 12-bit sampling A/D, a 32-word FIFO buffer, and a fully asynchronous high speed digital interface. The product is manufactured using highly reliable advanced hybrid circuit assembly techniques and is packaged in a 40 pin hermetic DIP.

The anti-aliasing filter is an active four-pole Butterworth. Cut-off frequencies (f_C) are user-selectable (capacitor programmable) and operation is specified for f_C up to 50 KHz. The filter may be bypassed entirely if desired.

The 12-bit sampling A/D converter can convert ±5 Volt full-scale signals at sample rates up to 125KHz. The rate is programmable by means of a single external clock. The entire converter system is specified and tested for Signal-to-Noise Ratio and Total Harmonic Distortion.

The digital interface provides a true asynchronous link between the A/D and a high-speed microprocessor. Data transfer is controlled by generating an interrupt signal when data is available. Interrupts can be generated when the FIFO is full (32 words), half-full (16 words), or when a single word of data is ready (FIFO bypassed). In addition, the AD1332 can generate an interrupt signal when the A/D conversion results are over-range.

The AD1332 provides a completely specified and tested system that bridges the interface and specification gap between A/D converters and high-speed DSP.



AD1332 BLOCK DIAGRAM

Integrated Device Technology Inc

CMOS PARALLEL 64 x 5-BIT FIFO WITH FLAGS

PRELIMINARY IDT72413

FEATURES:

- First-In, First-Out dual-port memory—35 MHz
- 64 x 5 organization
- Low-power consumption
  —Active: 200mW (typical)
- RAM-based internal structure allows for fast fall-through time
- Asynchronous and simultaneous read and write
- Expandable by bit width
- Cascadable by depth at 25MHz (IDT72413L25) not cascadable at 35MHz (IDT72413L35)
- Half-full and Almost-full/Empty status flags
- IDT72413 is pin and functionally compatible with MMI67413
- High-speed data communications applications
- Bidirectional and rate buffer applications
- High-performance CEMOS™ technology
- Available in DIP and LCC
- Military product available, 100% screened to MIL-STD-883, Class B

DESCRIPTION:

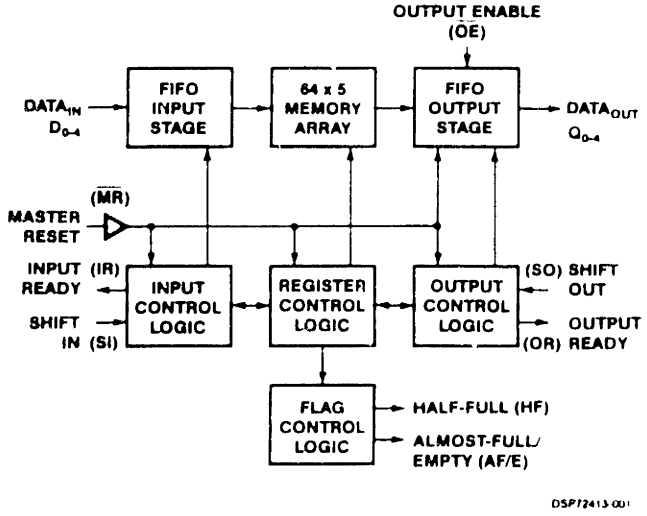
The IDT72413 is a 64 x 5, high-speed First-In, First-Out (FIFO) that loads and empties data on a first-in, first-out basis. It is expandable in bit width. The IDT72413L25 (25MHz) is cascadable in depth. The IDT72413L35 (35MHz) is not cascadable in depth.

The FIFO has a Half-full flag, which signals when it has 32 or more words in memory. The Almost-Full/Empty flag is active when there are 56 or more words in memory, or when there are 8 or less words in memory.

The IDT72413 is pin and functionally compatible to the MMI 67413. It operates at a shift rate of 35MHz. This makes it ideal for use in high-speed data buffering applications. The IDT72413 can be used as a rate buffer, between two digital systems of varying data rates, in high-speed tape drivers, hard disk controllers, data communications controllers and graphics controllers.

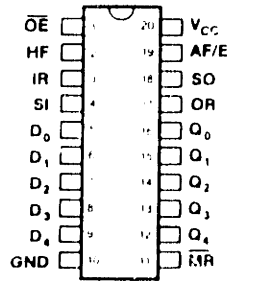
The IDT72413 is fabricated using IDT's high-performance CEMOS process. This process maintains the speed and high output drive capability of TTL circuits in low-power CMOS.

FUNCTIONAL BLOCK DIAGRAM



DSP72413-001

PIN CONFIGURATION



DIP
TOP VIEW

LCC
CONSULT FACTORY

CEMOS is a trademark of Integrated Device Technology, Inc.

ANALOG DEVICES

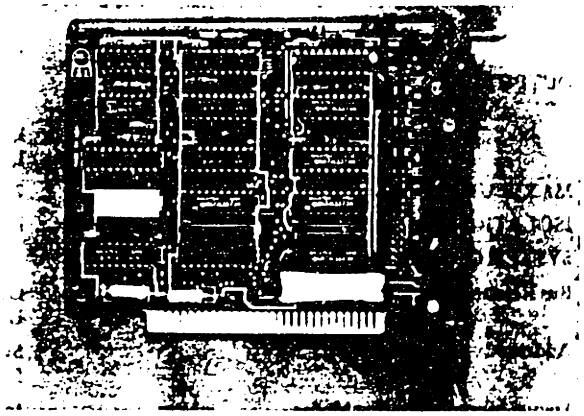
IBM PC/XT/AT™ Compatible Digital Input/Output Board

RTI-817

FEATURES
24 Channels of Digital Input/Output
Three 8-Bit Groups Independently Selectable for I/O
Interrupt Generation on Change of State
Compatible to 16 and 24 Position Solid-State Relay Subsystems

GENERAL
Compatible to IBM PC/XT/AT or Equivalent
Optional Screw Termination Panels

APPLICATIONS
Parallel Data Transfer to PC
Digital I/O Control
AC or DC Monitoring and Control of Voltages
Relay Control



GENERAL DESCRIPTION

An IBM-compatible member of the RTI™ Interface family, the RTI-817 is a 24-channel (bit) input and output board that plugs into one of the expansion slots in the IBM PC/XT/AT. The board can be used with TTL low-level input/output circuitry or with solid-state relay subsystems (16- or 24-channel versions) to provide 2500V isolation for interfacing with high-level ac and dc signals.

The 24-channel capability of the RTI-817 is divided into three ports (or groups) with 8 bits per group. These eight bit ports can be configured for either a digital input or output function.

There are two unique features associated with the RTI-817: an eight-bit latching capability and an interrupt on change of state. The latching capability is software or hardware selectable. It stores the state of eight digital input lines in a register which can then be read from the PC data bus. Interrupt generation occurs when one of the eight digital input channels changes state in a single port. This feature frees up the PC to do other activities since there is no need to poll the digital input port for an event to occur.

The RTI-817 can be installed in either a long or short slot in the IBM PC/XT/AT. The board maps into the I/O channel address structure as 4 consecutive bytes, addressable in an unoccupied 4-byte boundary using a DIP switch. The board operates from the bus +5V power source.

Typical applications of the RTI-817 include sensing and control of high-level signals, sensing low-level (TTL) switches or signals, driving indicator lights or controlling recorders, and parallel data transfer (via software) to computers or panel meters.

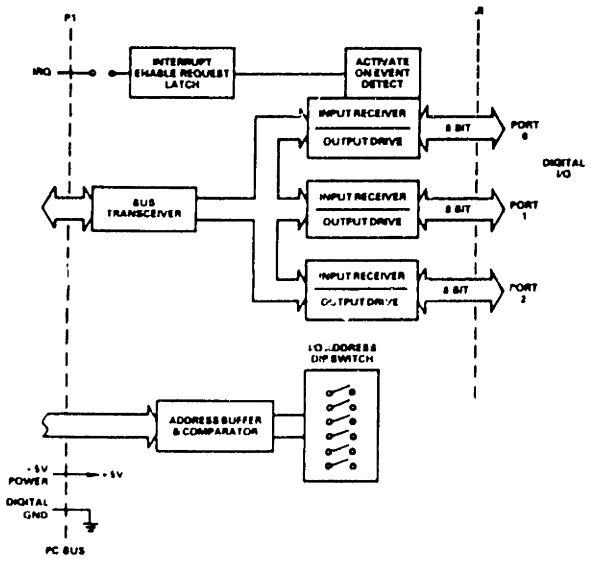


Figure 1. RTI-817 Block Diagram

RTI™ is a trademark of Analog Devices, Inc.
IBM PC/XT/AT™ is a trademark of International Business Machines, Inc.

Two Technology Way; Norwood, MA 02062-9106 U.S.A.
Tel: 617/329-4700 **Twx: 710/394-6577**
Telex: 174059 **Cables: ANALOG NORWOODMASS**