COMPUTER GENERATION OF DIGITAL FILTER BANKS

by

P. A. Ruetz

Memorandum No. UCB/ERL M84/94

30 October 1984

COMPUTER GENERATION OF DIGITAL FILTER BANKS

by

Peter A. Ruetz

Memorandum No. UCB/ERL M84/94

30 October 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TABLE OF CONTENTS

Computer Generation of Digital Filter Banks

Peter A. Ruetz

University of California, Berkeley

Electronics Research Lab

Berkeley, California 94720

I INTRODUCTION

Fully automated design of complex integrated circuits has often resulted in limited usefulness because of poor performance or inefficient silicon space utilization. If few restrictions are placed on the function of the ICs to be generated, then the optimization problem becomes difficult, yielding circuits far inferior to custom designs. Another important aspect of using automated design systems is the time required to develop the software and its reliability. There is no advantage in reducing hardware design time if the resultant software development effort becomes equally time consuming and error prone.

It has become apparent that tradeoffs between development time, generality and final circuit performance must be made. The design system described here was based on an emphasis on high performance with minimal software effort. Instead of treating the software and hardware designs as distinct problems, the hardware architecture and layouts were designed in a way that made automation simpler while maintaining performance.

Some automated design systems have been developed which allow the user to interact only at the highest level. If this is incompatible with the requirements of the user, then the entire design system is of no use. If however, the software is designed to allow the user to operate at a lower level, more jobs can be accomplished. Our design system has been developed in a hierarchical manner. For those wishing to generate filter banks, the task can be accomplished from the highest level, i.e. totally automated. The user can also use the lower levels of the system (i.e only partially automated) for other applications. Further, the system can be extended at the highest level for the specific needs of the user.

The scope of applications that has been chosen is digital filter banks which are a parallel and/or cascade connection of filter sections. Digital filter banks are found in applications as diverse as MODEMs and spectrum analyzers for speech recognition, channel vocoders, consumer stereo and EEG analysis. Decimation and rectification are required in addition to digital filtering in the spectrum analysis applications.

II THE HIERARCHY

Currently, the hierarchy is four levels deep. At the lowest level are the circuit 'cells'. These cells consist of basic building blocks such as counters, adders, RAM cells, ROM cells, etc. The cells can be used without any automation for a totally manual design. At the next level, the layout generator assembles these cells into more complex blocks such as data paths and controllers from hardware descriptions. This would be useful for users that desire a signal processor, but need a few additional circuits that have not been designed or are not handled by the layout generator. The user would only have to specify the hardware specifications including the RAM size and ROM contents and add the new circuit blocks to form a completed chip. At the third level, the layout generator assembles the data path and the controller into a completed chip. For those that have a non-filter digital signal processing application, a chip could be generated completely from this hardware description. Finally, at the highest level, the filter compiler generates the hardware description from a digital filter description. At this level, digital filter banks can be generated completely automatically.

To generate filter bank chips, the design procedure shown in figure 1 is followed. The digital filter bank structure and coefficients are specified in a input file. The filter compiler converts the input file to the hardware description. To check the algorithms before the circuit is fabricated, the hardware description can be used as an input to a real-time tester. When the designer is satisfied, the layout generator is used to create a layout file.

III THE CELLS

The basic architecture of the hardware is shown in figure 2. There are two main blocks: the controller consisting of the program counter, the ROM and the address index register and the data path consisting of the ALU and RAM.

There are several reasons for having few large circuit blocks. The block division was chosen to minimize assembly difficulty while retaining adequate generality. With few blocks, the automatic assembly is simplified. Routing difficulty is reduced by having fewer blocks that need to be routed together. The blocks are also made up primarily of abutting circuit cells which are very simple to assemble.

The large blocks were chosen to be functionally complete. That is, the blocks can be easily used to perform some complex function. The blocks would be complicated to use, except at the lowest level of cells, in a partially assembled form. The program counter may be useful without the ROM but it is easily assembled from counter cells so that it need not be a separate block.

There is also a somewhat natural division. As all cells in the data path could be designed with the same bit slice pitch, the data path could be made a single block requiring no data bus routing. The ROM was designed to minimize area that determined the pitch of ROM cells. The ROM cell pitch is, however, vastly different from the pitch of the control lines entering the data path. That makes it more efficient to optimize each as separate blocks with routing between the two than to stretch the ROM to the pitch of data path control lines.

III-1 The Controller

The controller was designed to be small with high performance. To achieve these goals it was made very simple with a minimum number of features. For example, there is no branching capability or micro coded instructions. Adding complexity can result in vastly increased area as the extra registers and routing are a significant fraction of the controller. ROM bits are very small, regularly spaced and hence very efficient. Instead of putting the convenience of micro coded instructions in hardware, it is put in the software (at the highest level) where it does not add to the silicon area.

Every cycle the controller outputs a valid horizontal control word. This horizontal control word specifies the value of every data path control line. Each controller output comes directly from the ROM with the exception of some of the RAM address lines when decimation is used. With decimation, the index register modifies the RAM address. Although this increases controller complexity, it saves ROM space and averts the need to perform address computations in the data path. The data path is never used for any control operation, allowing continuous signal processing. The circuit is also more compact since busses are not needed to connect the control and data path.

III-2 The Data Path

Figure 3 is a block diagram of the data path. As in most signal processors there is a RAM, adder, accumulator, some form of negation/absolute value logic and i/o. However, no array multiplier is included.

Again, only a minimum of features are provided. In this way the size can be kept small making room for additional data paths on a single chip for greater throughput. The cell circuit design problem is also reduced, while programming the data path is more complicated. This is not a problem when automation is used, as the filter compiler generates and optimizes the micro code.

Since there is no array multiplier, fixed coefficient multiplies are implemented in a serial-parallel manner [1]. This is accomplished with the use of the barrel shifter, adder and accumulator. Since a restriction of fixed coefficient multiplies is placed on the system, less that N cycles are required for an MxN multiply, where M is the signal width and N is the multiply coefficient length, by programming the ROM properly. Because a barrel shifter can shift several (0 to 5 in this case) places in a single cycle, multiplies require only a number of cycles equal to the number of '1's in the coefficient.

By using coefficients encoded in canonical signed digit format [2], it is possible to save more cycles in a serial-parallel multiply when there are more than two consecutive ones in the coefficient. This arises because in hardware it is just as easy to subtract as it is to add a partial product. For example:

to perform: (0.01111)Yn

rewrite: =((0.1)-(0.00001))Yn

=(0.1000(-1))Yn

=(0.1)Yn - (0.000001)Yn

Therefore, only two cycles are necessary to perform the multiplication, whereas four cycles would be required for the direct implementation.

The adder is a ripple-carry type. To ensure high speed, different even and odd cells are used which minimizes the delay through the carry chain. This results in very compact circuits which can operate at rates over 4 MHz. The ripple carry adder is also particularly well suited for a bit slice design which makes the automatic layout very straight forward. The output of the adder saturates instead of simply overflowing to prevent limit cycles. This is easily incorporated in the hardware but would require several cycles per computation to implement in software.

Pipelining in the data path increases the performance of the circuit by making higher clock rates feasible. The pipeline registers are at the output of the RAM, the input of the RAM and at output of the adder (the accumulator). With pipelining, the RAM and the barrel shifter, adder combination both get a full cycle for operation. Although pipelining makes micro coding more difficult, it is transparent to the user when the filter compiler is used.

The memory input register is the only register of the three which can be selectively written. In some cases, the result of a computation can be held in this register until the RAM is inactive during a serial-parallel multiply. At this point, the result can be written into the RAM without requiring an extra cycle. Proper use of this register reduces the length of the micro code by preventing the data path from becoming memory bound.

The RAM is a four transistor dynamic type with a schematic shown in figure 4. A dynamic memory was chosen over static designs because the dynamic RAM is smaller with lower power consumption. The RAM is automatically refreshed as long as the sample rate is kept over 1 KHz because every location is both written and read each sample.

Three possible choices for the RAM design were the one, three of four transistor cells. The four transistor cell was chosen over one transistor designs to minimize process sensitivity. To avoid running busses between the RAM and ALU, it was desired to have the same pitch for both so they could be attached directly, simplifying automatic layout. To use space efficiently, this required that the RAM have a single column decode as the optimized cell pitch was approximately half that of the ALU bit slice. The three transistor design is more difficult to column decode so the four transistor design was chosen.

IV LAYOUT GENERATOR

The layout generator assembles the cells into a data path and a controller block from hardware descriptions. If desired these blocks are then assembled into a complete chip. The hardware is described by several parameters including: data path word width, RAM size, decimation ratio and ROM contents.

IV-1 Layout Generation Issues

Before starting development of the layout generator, several aspects of automated layout were identified as difficult problems. General placement of the major circuit blocks requires sophisticated optimization algorithms to generate space efficient designs. A two level router would be needed to rout between these blocks. An extensive data base would be needed to store the necessary data for the router and placer. The data base would contain the terminal locations on each block and the available routing area.

Other aspects of the automated layout were found to be easily handled problems. It is not difficult to assemble blocks (ie the ROM, ALU and RAM) from abutting cells since the relationships involved are all well determined by the hardware parameters specified by the user and the cell characteristics. The way the cells go together is determined by the cell designer so that proper cell design can help the automated layout. For example, by including the signal routing within the cells, the need for inter-block routing by the program is avoided. Fixed routing, where the routing terminals have a constant relationship throughout all changes in hardware parameters, can be accomplished by inserting a cell with the appropriate wires in it. That is, no algorithm for routing is required at all. Regular routing, where the routing terminals are evenly spaced throughout changes in the hardware parameters, is implemented by a simple program loop.

IV-2 The Floor Plan

In order to avoid the more difficult problems, two major restrictions were made. The first was to use a fixed floor plan, the relative placement of circuit blocks, pads and routing areas on the chip. The floor plan was chosen to reduce the complexity of the algorithms used and the number of layout decisions that must be made by the program. With the chosen floor plan all routing is either fixed or regular.

The decision was also made to have the program 'know all'. That is, all information regarding the cells and their connection was coded directly into the algorithms. Using specific information of the application avoids having to solve the general problem and reduces the software design time. Software reliability is enhanced when the simplest algorithms are used instead of complex general algorithms with obscure failure modes. This approach obviously makes the programs very specific to the particular cells which are used so that changes in the cells may require changes in the software. Therefore, one should not expect to make major upgrades without significant software changes with a system such as this. However, because the software development time is relatively small, new software can be written when significant changes are made.

IV-3 Examples of Generated Circuits

The circuit remains easy to assemble over the large changes in hardware parameters shown in figure 5a-d. The hardware parameters for each is listed in table 1. From the figure the fixed floor plan can be seen. The controller, data path, pads and routing areas are always in the same relative position. The I/O parallel buss at the top of the chip is an example of regular routing. The routing area does not change shape or relative position as the parameters change. The routing between the controller and the data path is a function only of the RAM size and whether decimation is used. As there are only a few cases, each is treated as fixed routing and a cell with the appropriate wires is simply inserted. Wiring from the PC to the ROM is handled similarly. The wiring of supplies and clocks requires little jumping (except in the fixed routing cells) and a minimal amount of decision making.

The silicon area is also used efficiently over the range of parameter changes. Virtually the only wasted area is near the pads or due to differences in length of the controller and data path (see figures 5b,5c). The RAM gets longer as the number of states in the filter bank increases. The controller increases in length with the program length. Since adding states requires a longer program to process these states, the ROM and RAM tend to get larger together. In figures 5a-c the ROM is not column decoded and the waste area is not too large. In figure 5d the ROM length increased significantly so that a column decoded ROM was used to minimize the unused space.

Some waste of space is allowed if the waste is not large while the savings in effort is. For example, when decimation is used, the ROM width is constant regardless of the RAM size. Up to 3 bits of RAM are unused but the routing is simplified. The data buss routing area between the data path and pads on the right side of the chip is of constant size. These simplifications reduce the number of cases to be handled with some space wasted for the very small chips. However, the designs would likely not be used anyway, due to the large overhead involved.

IV-4 Output File Format

The output of the layout generator are KIC format [3] files. This format was chosen because layout stations are being used which read this format making visual checks convenient. The KIC format also supports the hierarchical organization of the hardware. The CIF format [4] is used for actual fabrication but does not allow arrays as the KIC format.

IV-5 Block Assembly

As mentioned previously, the assembly of blocks from cells is a straightforward task. An output file is written that lists the cells with the appropriate offsets and orientation. This information is calculated from the hardware parameters and cell parameters (eg size).

The controller is a connection of many cells that makes its manual layout difficult. Most variations in the controller are functions of the ROM width and length (found from the binary listing), and the decimation ratio all of which the user specifies explicitly. The ROM length determines how many bits will be used in the PC and decoder and how the decoder is programmed. Since there are only 5 different PC sizes, each is a cell with appropriate routing wires. The decimation ratio determines which type of ROM output register will be used and how the index register itself is configured. If there is no decimation, all output registers are the same and no index register is used. If there is decimation the output registers that feed the index register input are of a different type and an index register must be included and programmed to decimate properly.

The data path assembly is quite simple due to the bit sliced nature and the small number of blocks. The entire ALU only requires one line in a KIC file specifying an array of bit slices. The entire RAM array is similarly specified. The RAM decoder can be generated in the same way as the ROM decoder with each cell being described by one line in the KIC file.

V THE FILTER COMPILER

The filter compiler generates hardware descriptions from digital filter descriptions. This allows the automatic generation of filter banks with virtually no knowledge of the final hardware.

V-1 Filter Specification

The compiler reads an input file specifying the filter bank organization in terms of a parallel connection of channels. Each channel is a cascade connection of sections. Variations on this format are allowed that have been found useful in some applications. A section can be factored out and used by different channels. An example of this is the direct form band pass filter. The zeros are the same for all channels and can be factored out and computed only once. Figure 6 shows an example of a filter bank organization. In this example there are 16 parallel channels, each consisting of a 4th order BPF section, rectifier, 1 pole LPF section, decimation by 8 and a 2nd order LPF section.

Each section is a single input, single output structure with delays, multiplications and additions. Diagrams of some of the currently programmed sections are shown in figure 7.

All multiplies defined in the sections use fixed coefficients of canonical signed digit format. Use of this format, which was described earlier, optimizes the usage of the adder by minimizing the number of cycles required to perform multiplication.

There are several options allowed in each section in the bank. The user can full wave rectify the input of any section. This is useful in spectrum analyzer applications. Decimation is also handled but in a somewhat restricted way. A number of channels can have their outputs decimated and modified by some specified filter. The post decimation filter is the same for all channels being decimated and the decimation ratio is always the same as the number of channels being decimated. These restrictions were applied simply to reduce the development time and could be relaxed in future systems. The user can also specify that the output of any section be sent off chip through the parallel buss while setting an output strobe. To implement multiple inputs, the input of any section can be taken from any channel output or any channel input. Being able to specify a channel output, allows the use of a filter by many other channels (described above) and really allows very arbitrary filter organizations. Normally, the default (no specification) results in the parallel channels operating on the same input data.

The format of the input file is tailored to filter banks and was chosen to simplify the compiler. The format is as follows:

1. Input channels (one or more)
   These sections receive data from off chip and may perform
   some filtering (eg zeros of direct BPF).

2. Standard channels (one or more)
   These are just the regular channels, ie some cascade
   connection of sections. These channels will be decimated
   if a decimation channel is specified.

3. Decimation channel (optional)
   This is the channel that operates on the output of all
   standard sections above after decimation.

4. Non-decimated Standard channels (optional)
   More regular channels that are not to be decimated.

The format for the the sections is as follows:

1. Section identifier (2 letters), <options, if any>, N coefficients

The format specifies the order that micro code is generated and stored in the ROM and hence the order that it is executed. This save the compiler from having to determine this information.

V-2 The Filter Library

The compiler references a filter library which contains pertinent information about the allowed sections. A file contains a list of valid section identifiers along with the number of memory locations and coefficients required for each section.

For each section there is also a file containing the macro for that section. The macro file contains the symbolic micro code that implements a section without the coefficients or options inserted. Symbolic micro code is just a description of data path control lines that have been grouped functionally. The symbolic micro code has fields to describe the following:

memory operation (read or write)
relative memory address (actual address computed by compiler)
barrel shifter input mux selection (memory or BS output)
number of shifts (constant or taken from input file)
adder a input mux select
adder b input mux select
i/o operation

Currently, this micro code must be written by hand for each section. This involves a detailed knowledge of the timing and architecture that the average user would not have. Although software could generate the micro code from difference equations, this was not chosen because higher performance code could be generated by hand. For a second order section the length of the micro code is typically only 8 words.

V-3 Compiler Operation

The operation of the filter compiler is shown in figure 8. On the first pass, the input file is checked for errors and hardware requirements such as RAM size and decimation ratio are determined. On the second pass, symbolic micro code for the entire bank is generated. The symbolic micro code is then compressed. Finally, the symbolic micro code is assembled to binary micro code.

During the first pass, several errors are checked for, the amount of RAM is determined and each state is assigned a RAM location. The error check will locate syntax errors, undefined sections, filter library errors or the use of the wrong number of coefficients. If decimation is used, the decimation ratio is determined by counting the number of standard sections before the decimation section. The RAM requirements can then be determined. Without decimation, the amount of RAM required can be found by simply adding up the memory requirements for each section. With decimation, things are not as simple. The index register supplies the high order RAM address lines when the decimation is performed so that some RAM may not be used.

Amount of RAM accessed A= (RAM needed for input and standard sections)
+(RAM needed for decimation channel)
*(number of decimated channels)

Amount of RAM included on chip B= ${}_2$(int(log$_2$(A-1))+1)

Each state is then assigned to a RAM location. The states are assigned to sequential RAM locations as they are encountered in the input file if there is no decimation. That is, the first state of the first filter is stored in the first RAM location while the last state of the last section is stored in the last location. With decimation, the states accessed by the decimation filter are assigned first at fixed increments. The remaining states are then filled in sequentially.

The symbolic micro code for the entire bank is generated during the second pass. To accomplish this the input file is scanned until a section declaration is found. The macro for that section is read from the library and expanded into complete micro code by inserting the coefficients and options from the input file. This process is repeated until the end of the input file is found.

The symbolic code generated in the second pass is compressed by looking for sequences of code that can be shortened. There are three cases that are optimized. First, and most important, is the performing of the first memory access during the final computation of the previous section. This appears at nearly every section boundary and utilizes the pipelining of the data path. Another case is the utilization of both adder inputs when the accumulator is empty. Normally the B input is zeroed and data is brought in through the A input. If a coefficient has certain properties, additional data can be brought in through the B input, saving one cycle. One cycle can be save if data needed for the next operation is found to be left in the adder during the previous calculation. This occurs for certain filter structures with some coefficients.

These optimizations help produce code with essentially the same efficiency as that done by hand. For a speech recognition filter bank, the number of micro instructions was reduced from 480 to 384 words. The optimization is performed on the symbolic code because the cases are easier to identify than when the code has been assembled to binary.

With the symbolic code optimized, it is converted to binary for use by the layout generator. This is a simple operation because for each symbolic field value there is exactly one binary pattern for one or more control lines. This data is written directly to a file for the layout generator or tester control data is included for use by a real time tester.

VI THE TESTER

The tester set-up in shown in figure 9 is quite valuable in producing designs which work the first time fabricated. The filter compiler running on the VAX 11-780 generates tester code that is down loaded to a pattern generator. The pattern generator performs exactly the same function as the controller block included in the complete chip. It sends the horizontal control words in real time to a data path that is the same as that used in a final chip. The spectrum analyzer generates digital input data and examines the filter outputs. In this way, the filter designer can check the input file for errors and the effects of finite data word with and coefficient truncation on the filter responses. If there is a problem, it is found before fabrication. Further, this set up will verify that the compiler is working properly and that the filter library data is correct.

VII FABRICATED CIRCUITS

Several circuits have been fabricated using this system. A single band pass filter chip was fabricated to determine the efficiency of a small chip. A 16 channel filter bank for the front end of a speech recognition system and a 16 channel consumer stereo spectrum analyzer have been generated and fabricated. Table 2 gives a summary of the performance of these chips.

All circuits have been fabricated with a four micron NMOS depletion load process and are designed to work with a single 5 V supply. Although a 3 MHz non-overlapping clock is sufficient for the chips to operate at the designed sample rates, they can be run reliably with clocks up to 4 MHz.

VII-1 A Small Chip

The 4 pole band pass filter chip die photo is shown in figure 10a and measured frequency response in figure 11a. Although the area per pole for this chip is quite high it might be useful when data is in digital form so that a switched capacitor or other analog filter would not be appropriate because of the high overhead in including the A/D and D/A. The circuit shown has a word width of 10 bits and a dynamic range of 48 dB. Since each additional bit increases the dynamic range 6 dB, a chip with 100 dB of dynamic range would only be 30 %

larger.

VII-2 A Spectrum Analyzer for a Speech Recognition System

A block diagram of 112 pole speech recognition system [5] chip is shown in figure 6. Each channel consists of a 4 pole Butterworth band pass filter, followed by a full wave rectifier and the first pole of a 3 pole Butterworth low pass filter. The output of the 1 pole anti-aliasing filter is decimated and low pass filtered with the rest of the Butterworth filter. A photo of the die is shown in figure 10b. The frequency response of all 16 channels is shown in figure 11b.

The number of cycles available to perform all filtering is given by:

number of micro instr= (number of processors)*(max clock rate)/(sample rate)

To ensure that this maximum number of operations was not exceeded, several steps were taken. Filter structures were carefully chosen. The state variable form shown in figure 7a has a relatively complex structure compared to the direct form (figure 7b). However, the state form is less sensitive to coefficient truncation than the direct form when there is a large ratio of sample frequency to filter band edge frequency. For low frequency filters, the insensitivity to coefficient truncation makes the state form filter more efficient than the direct form. At high frequencies, the direct form becomes more efficient due to its simpler structure. Therefore, the five lowest frequency filters are state form while the upper eleven are direct form. To save more cycles, the zeros of the direct form were factored out of each channel and computed only once. In the state form, at low frequency the zero at 1/2 the sample frequency has little effect and was left out.

VII-3 A Spectrum Analyzer for Consumer Stereo

The structure of the 16 channel consumer stereo spectrum analyzer is very similar to that of the speech recognition chip. The sample rate was increased to

20 KHz to allow higher frequency filters and the band pass filters were limited to 2 poles. The 1/2 octave filters range in center frequency from 45 Hz to 8 KHz. The ratio of the lowest frequencies of interest to the sample rate is extremely small (much worse than the speech recognition chip) indicating that the state form will be better at lower frequencies. The photo of the die is shown in figure 10c with the log-log frequency response shown in figure 11c.

The design of this chip was automated one more level than the others. Instead of specifying the digital filters, a program was written to generate the digital filter specifications from desired 3 dB frequencies. The program picked the most suitable structure and determined and truncated all coefficients.

VIII CONCLUSIONS

The tools discussed here have been extremely valuable in the development of the circuits that have been fabricated. These tools not only shortened the hardware design time, but provided testing that found all errors before fabrication. Minor changes, such as increasing the width of the data path and fine tuning the gains of the channels, were made by simply editing the filter description file. Normally this would be a tedious task prone to careless mistakes. By careful design of the circuit cells and restricting the applications to filter banks, the software complexity was reduced with a development time of one man-month.

Table 1

	circuit 5a	circuit 5b	circuit 5c	circuit 5d
	1 channel	8 channel	16 channel	16 channel
RAM length (words)	8	64	64	64
data path word width	10	16	16	20
ROM length	32	128	128	192
Decimation ratio	-	8	8	8
number of processors	1	2	2	2

Table 2

	single	16 channel	16 channel
	4 pole	speech recognition	consumer hi-fi
data path word width	10	20	20
size	2.8mm x 2.5 mm	7.2mm x 3.7mm	6.7mm x 3.6mm
power dissipation	260 mW	570 mW	570 mW
SNR	48 dB	80 dB	80 dB
number of poles	4	112	80
sample rate	84 KHz (max)	14 KHz	20 KHz

References

[1] Rabiner L., Gold B., Theory and Applications of Digital Signal Processing, Prentice Hall, 1975.

[2] Schmidt L., "Designing programmable Digital Filters for LSI implementation", Hewlett Packard Journal, Vol 29, no 13, p. 15-23.

[3] Keller K., Newton A., "KIC 2: A low-Cost, Interactive Editor for Integrated Circuit Design", Proc. 24th COMPCON, Feb 1982.

[4] Mead C., Conway M., Introduction to VLSI Systems, Addison-Wesley, 1980.

[5] Lowy M., et al, "An Architecture for a Speech Recognition System", ISSCC DIGEST OF TECHNICAL PAPERS, p. 118-119, Feb 1983.

[6] Agarwal R., Burns C., "New Recursive Filter Structures Having very low Sensitivity and Roundoff Noise", IEEE J. Circuits and Syst., vol CAS-22, pp. 921-927, Dec 1975.

[7] Ruetz P., et al, "Computer Generation of Digital Filter Banks", ISSCC DIGEST OF TECHNICAL PAPERS, p. 20-21, Feb 1984.

ACKNOWLEDGEMENTS

This research was sponsored by DARPA under contract number N00034-K-0251.

Steve Pope and Bjorn Solberg deserve credit for a large portion of the circuit design and layout of the macrocells.

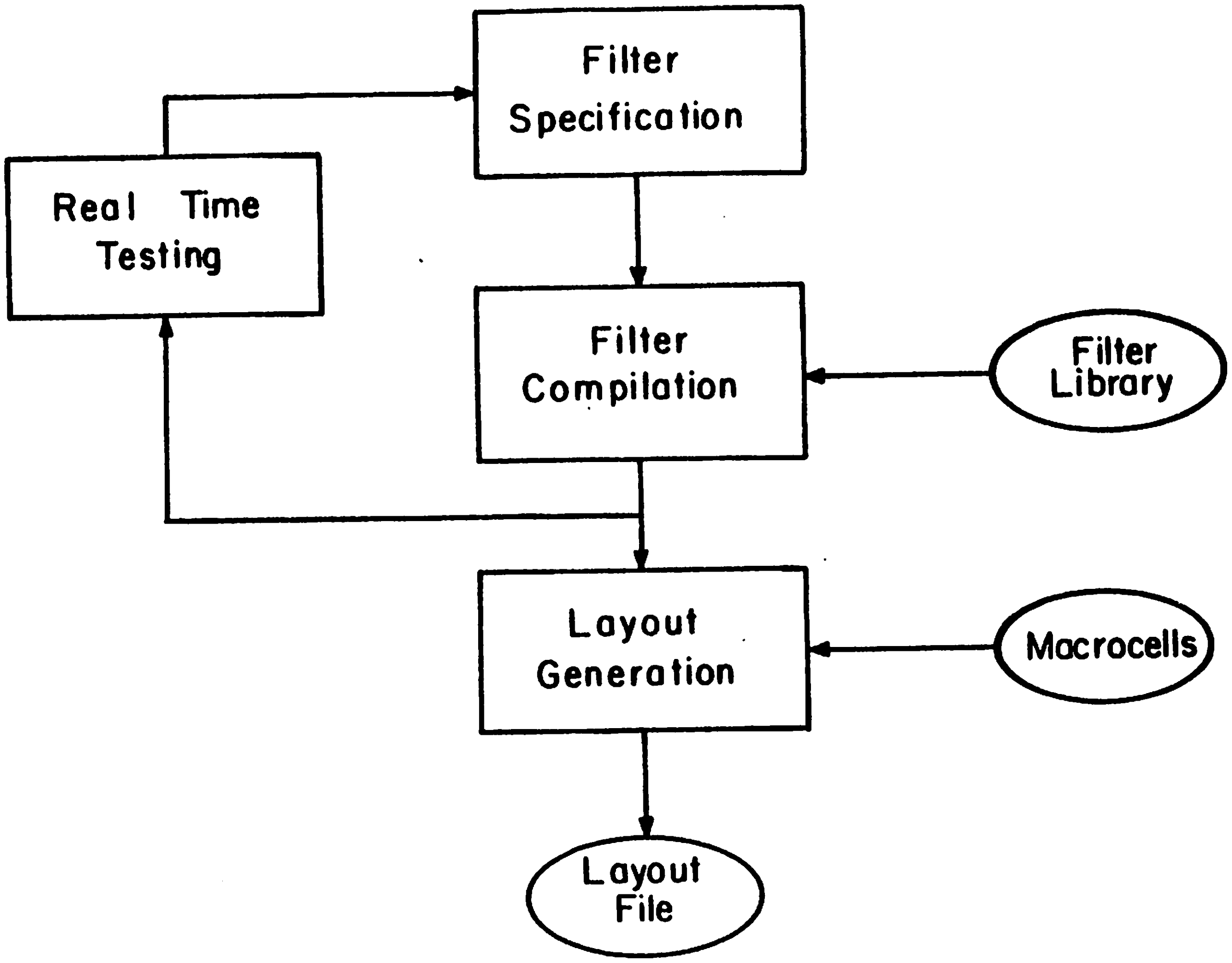


Figure 1. Filter Bank Design Procedure

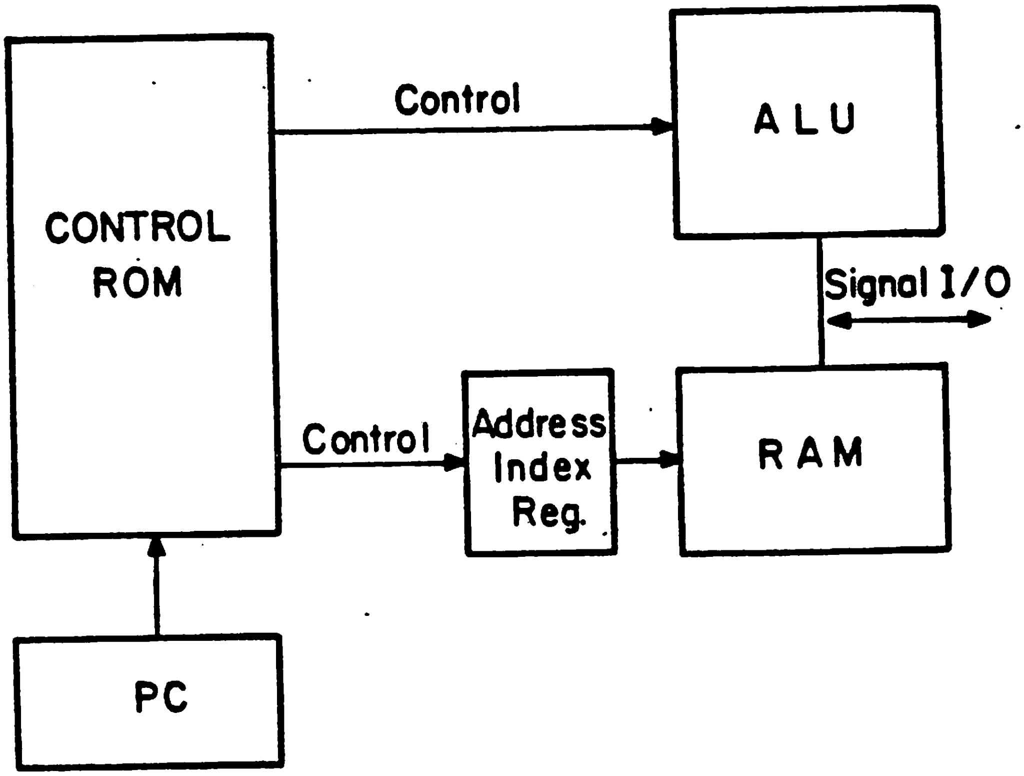


Figure 2. Hardware Architecture

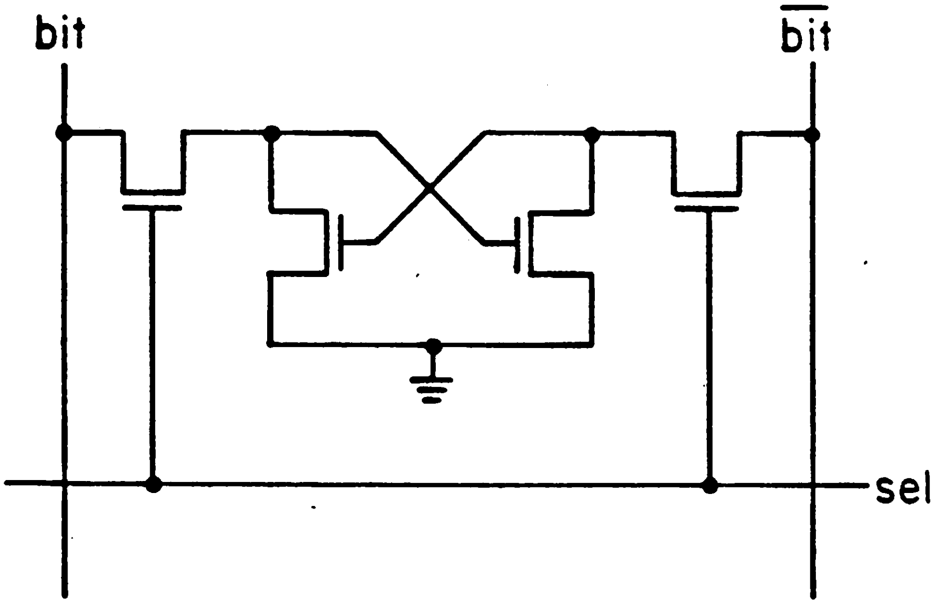


Figure 4. 4 transistor RAM cell

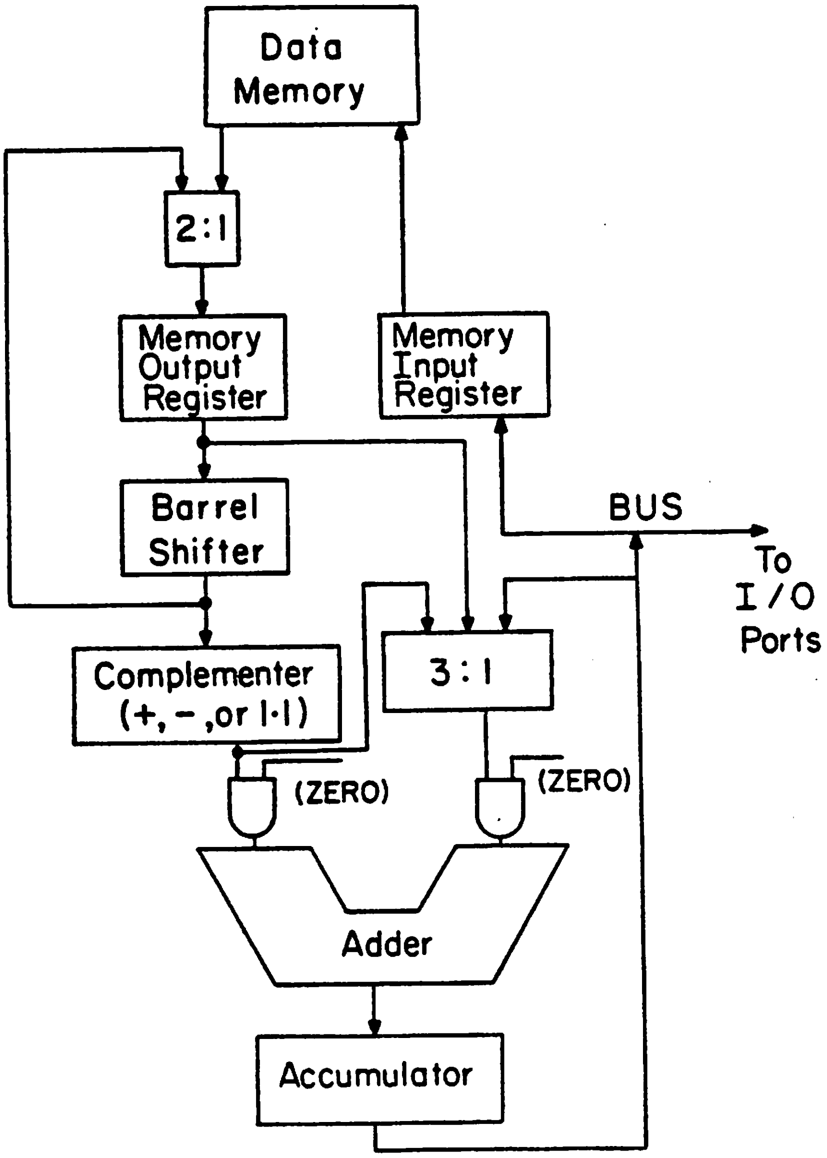


Figure 3. Data Path Block Diagram

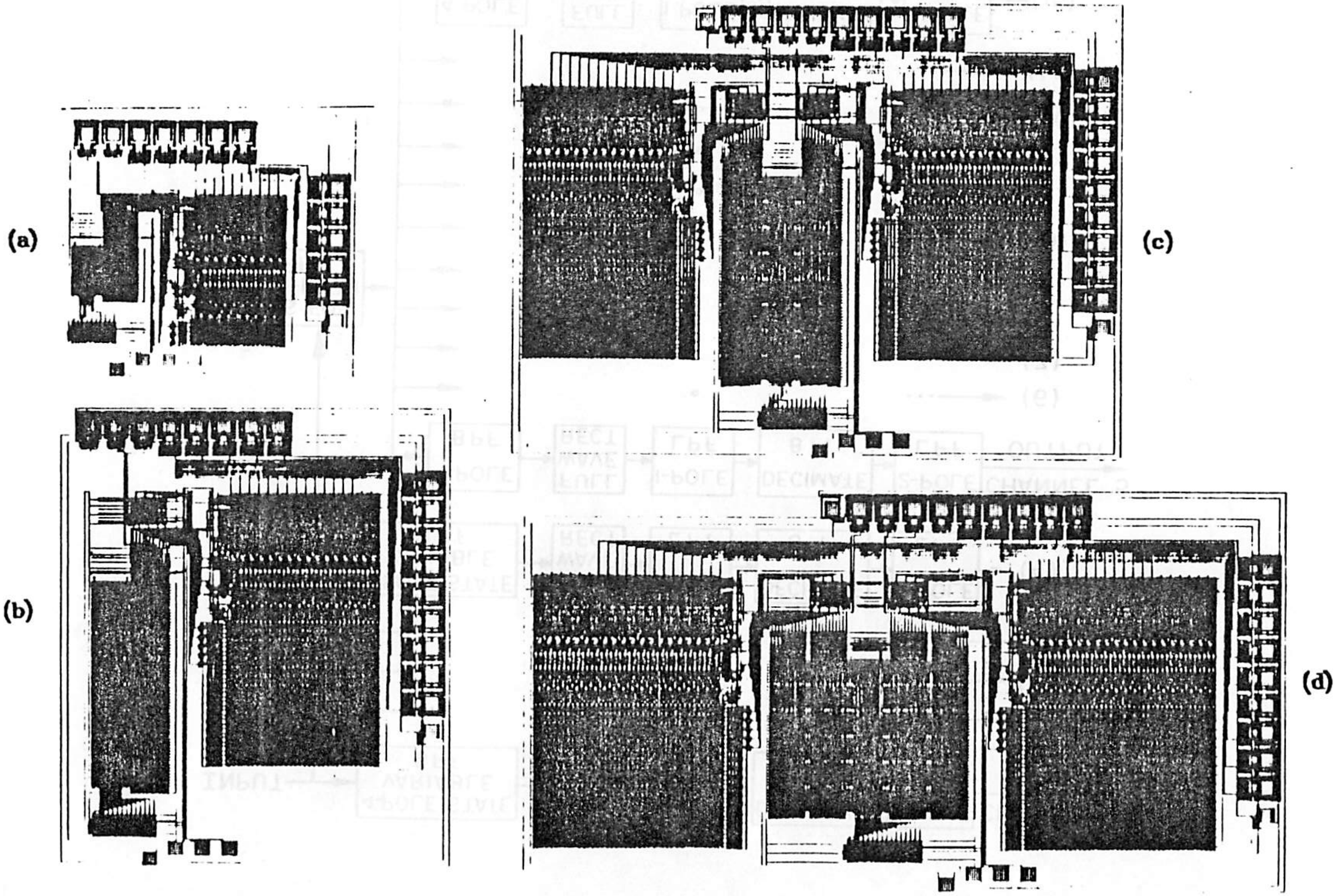


Figure 5. Layout Generator Output (see table 1)

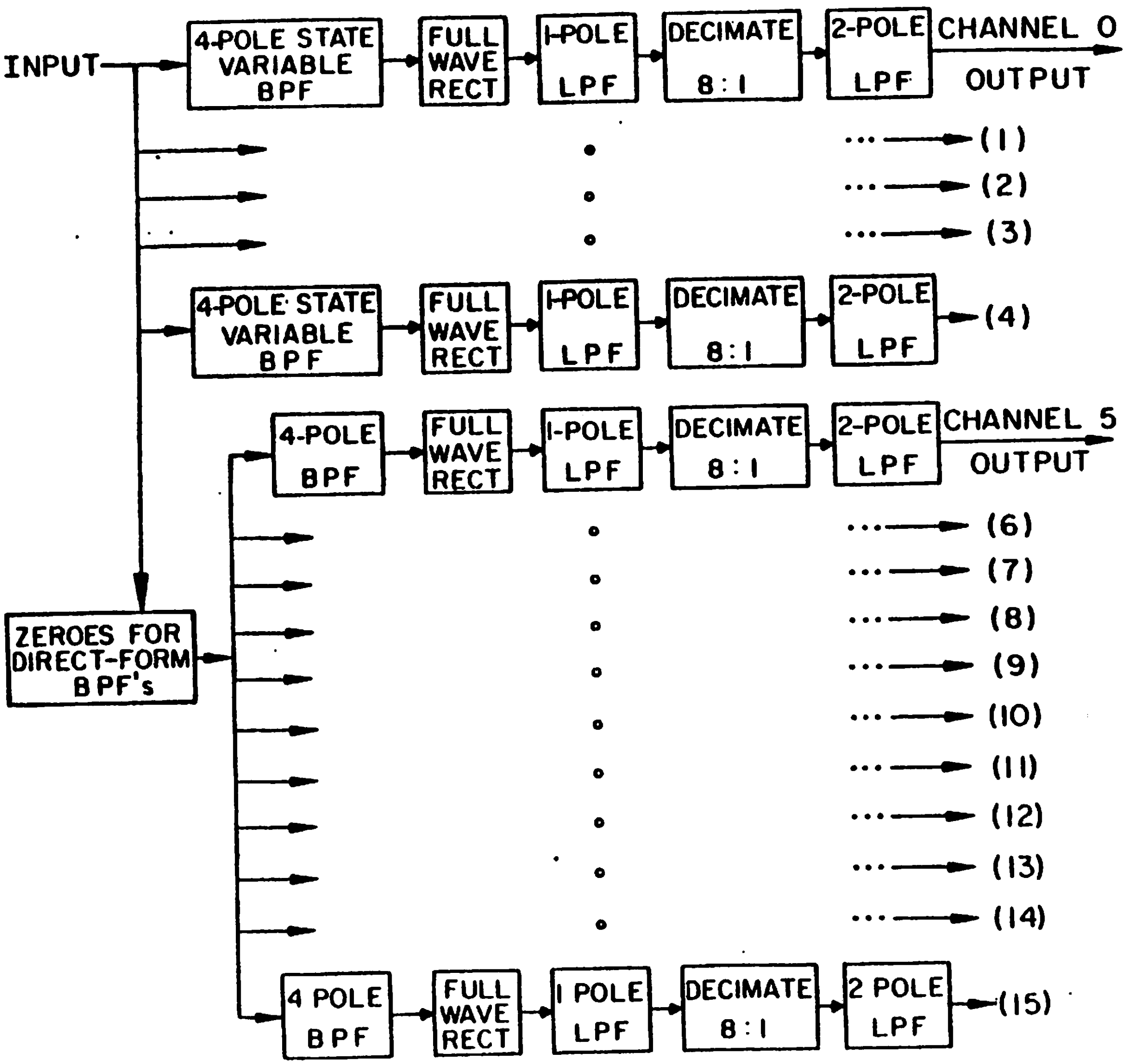
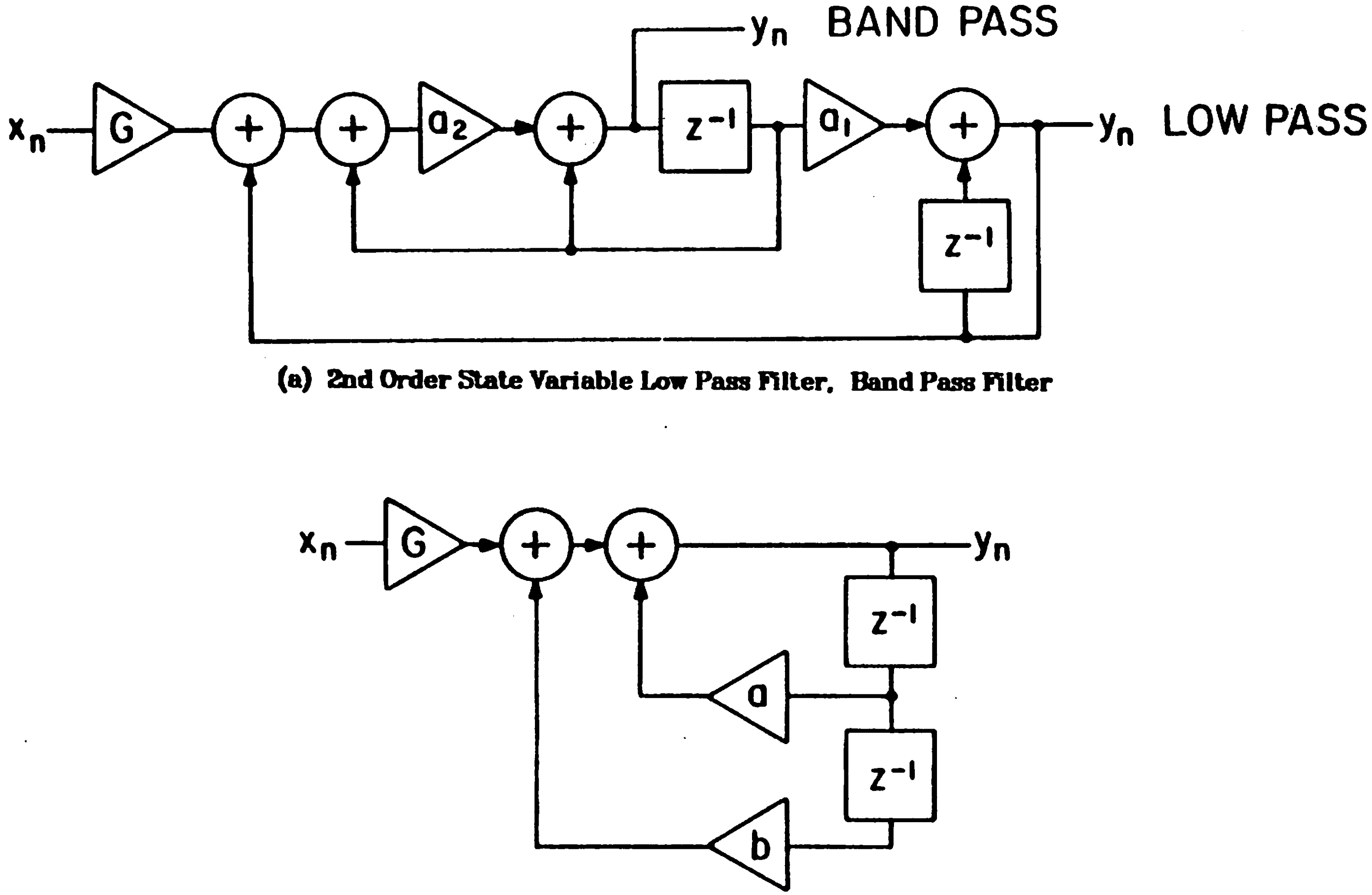


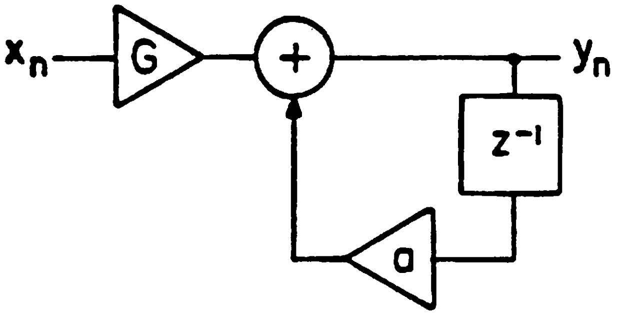
Figure 6. Speech Recognition System Filter Bank Organization



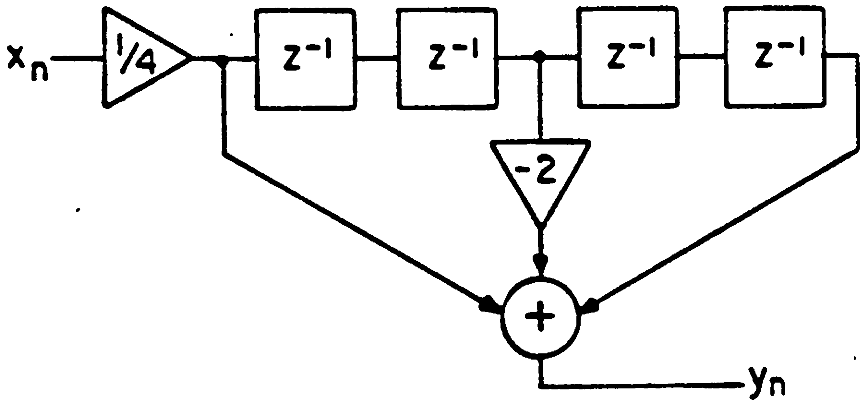
(a) 2nd Order State Variable Low Pass Filter, Band Pass Filter

(b) 2nd Order Direct Form Poles

Figure 7. Filter Structures used in the Speech Recognition Chip



(c) Single Pole Low Pass Filter



(d) 4th Order Zeroes for Direct Form Band Pass Filters

Figure 7. Filter Structures used in the Speech Recognition Chip

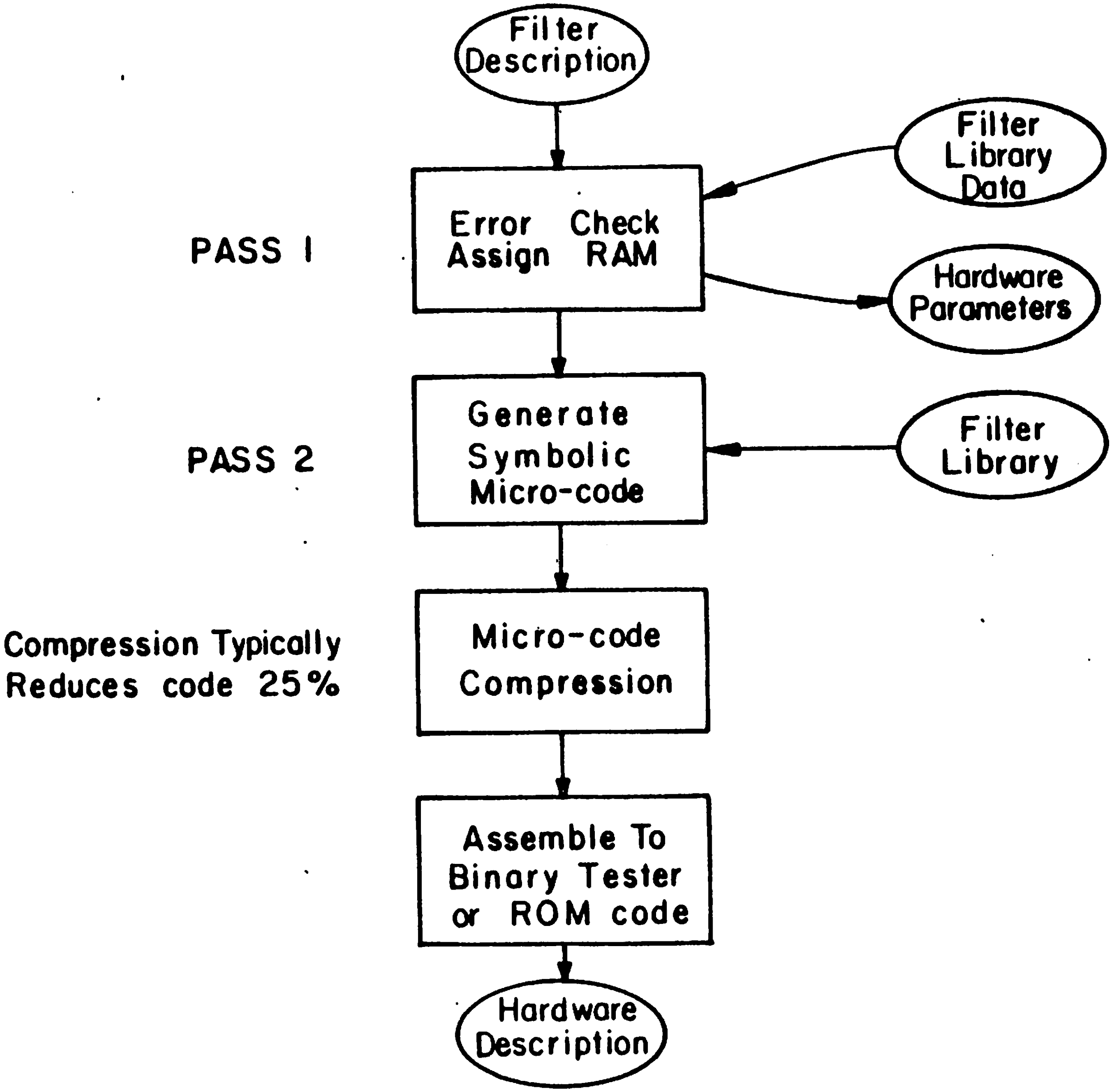


Figure 8. Filter Compiler Operation

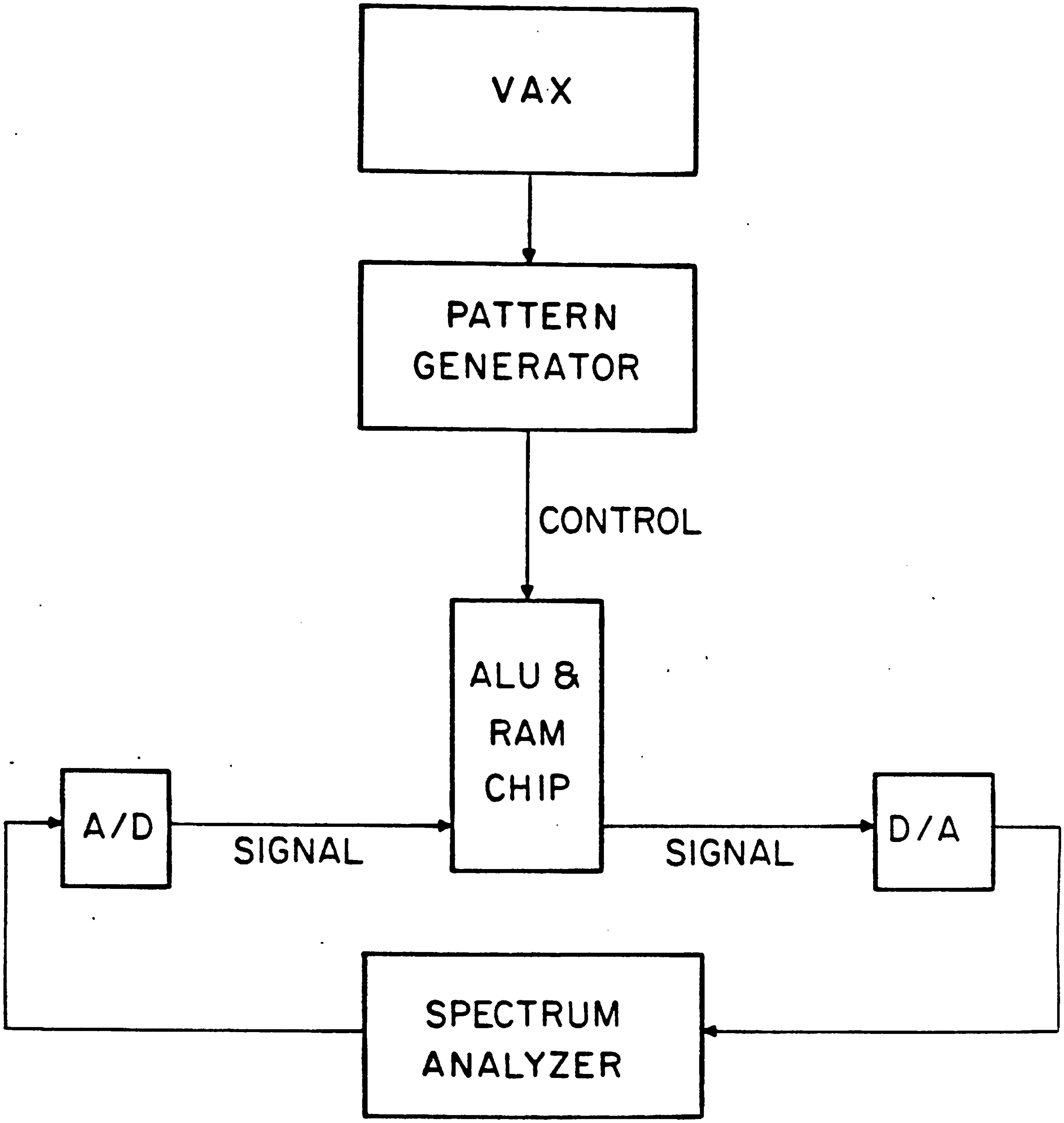


Figure 9. Test System for Verifying Algorithms

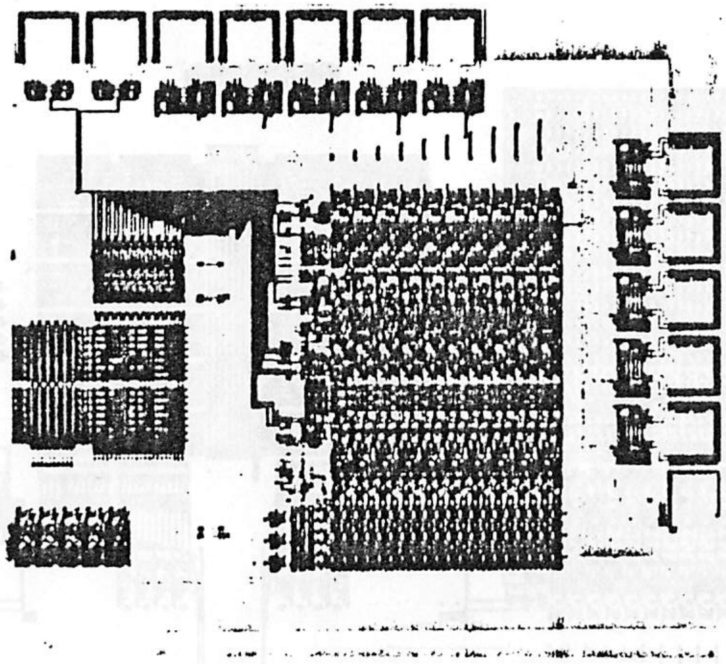


Figure 10a. 4 pole single BPF chip Die Photo

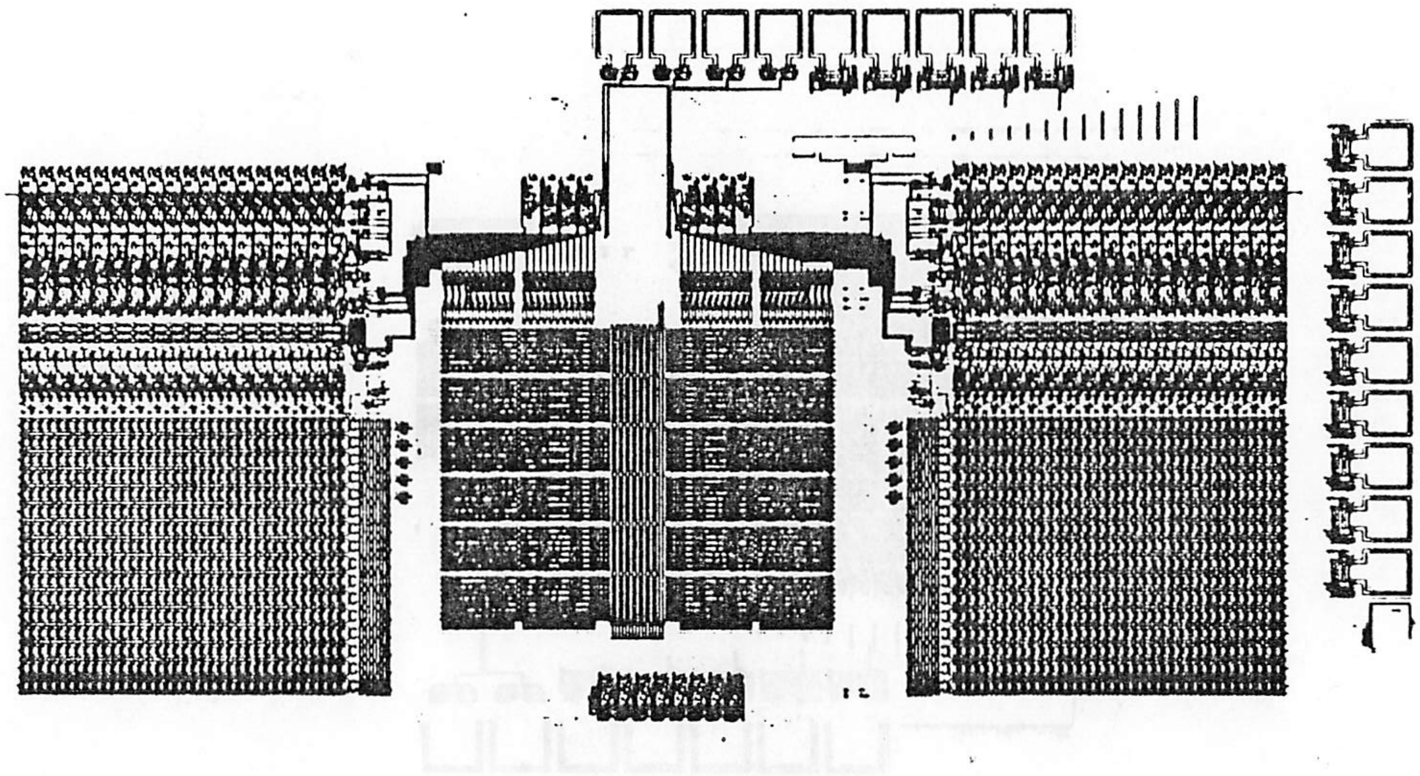


Figure 10b. Speech Recognition chip Die Photo

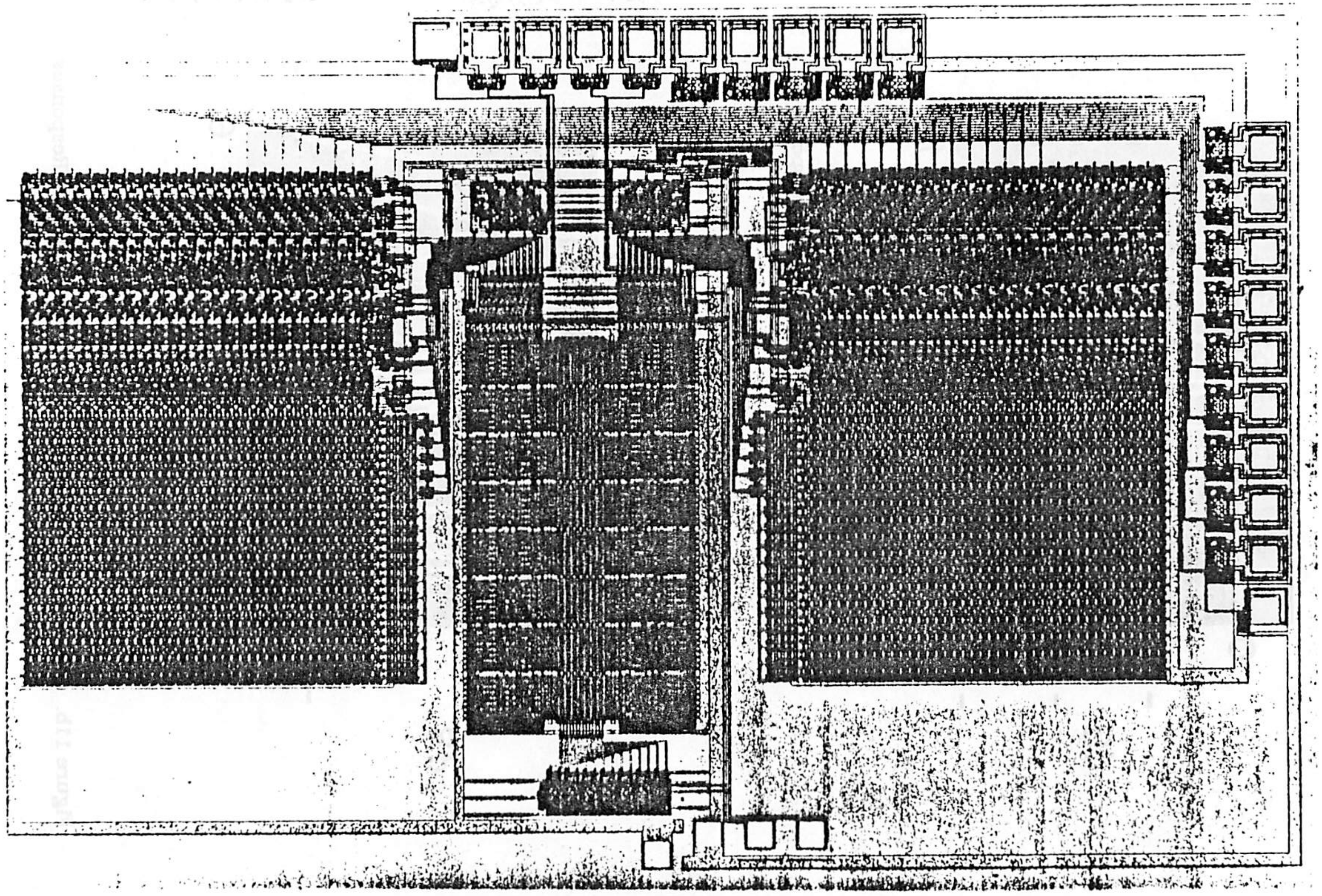


Figure 10c. Stereo Spectrum Analyzer chip Die Photo

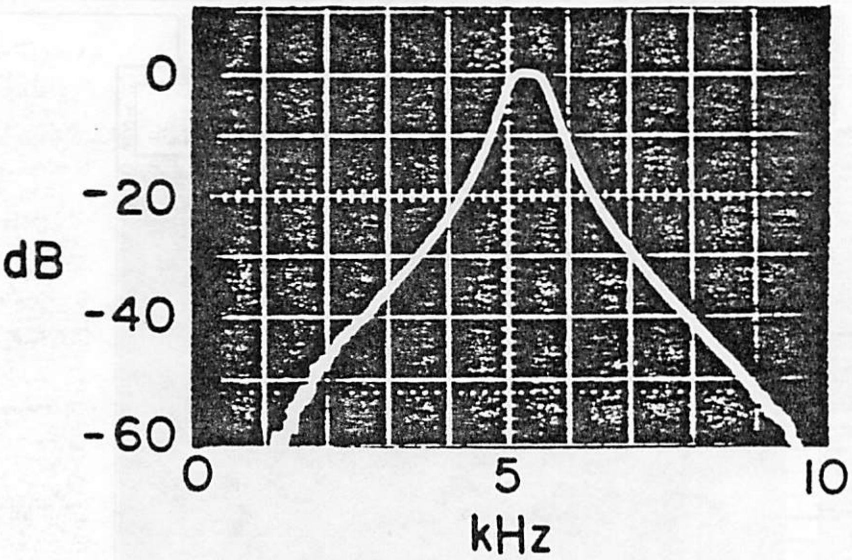


Figure 11a. 4 pole single BPF Frequency Response

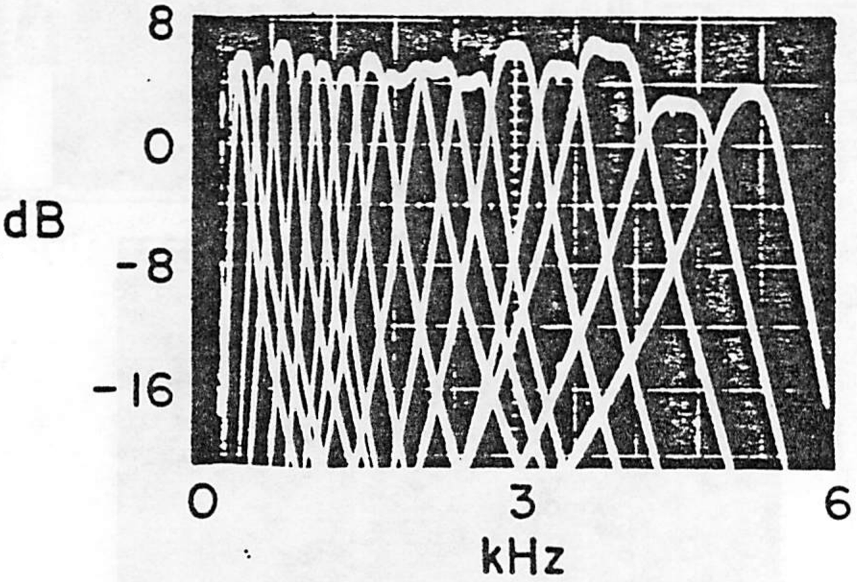


Figure 11b. 16 channel Speech Recognition chip Frequency Responses

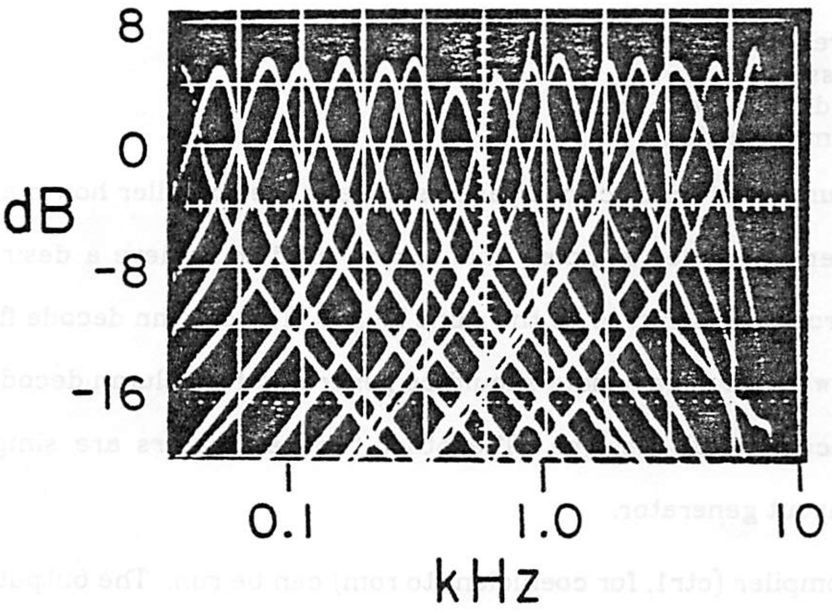


Figure 11c. 16 channel 1/2 octave spectrum analyzer Frequency Responses

APPENDIX A USING THE FILTER COMPILER

Before the filter compiler can be used to generate the hardware descriptions, the user must do several things. The filter descriptions must be written into an input file ('filterdata') according to the format described later. If sections are used that are not currently in the filter library, the library must be expanded. Also the file 'filtparm' must be set up with the appropriate information. Example 'filtparm' file:

```
128    desired instructions
1     processor
16     bit wide data path
false    column decoded
```

The desired number of instructions parameter tells the compiler how many instruction the user wants per sample. This can be used to achieve a desired clock rate or to force both banks to be the same length. The column decode flag tells the compiler whether binary data should be generated for column decoded or non-column decoded ROM designs. The other two parameters are simply passed on to the layout generator.

Finally, the compiler (ctr1, for coefficient to rom) can be run. The output is written to a file (romout) for use by the layout generator or the tester program.

The filter sections must be divided manually among the two processors if two are required. This is probably best done by trial and error. The amount each processor can do is determined by the sample rate, the maximum system clock rate (see App E) and limitations on the sizes of various circuit blocks (see App B). The compiler can be used to determine the total number of cycles required for the entire bank and if this is too many for one processor, the job can be split in two parts.

If the bank has been split into two parts, the program is simply run twice. As the program doesn't interact between the two runs, it is up to the user to ensure that both processors do not try to output at the same time. If it is desired that both processors operate on the same off chip input data, then both processors should input at the same time. This is easily accomplished by making the input channels for both banks the same. The compiler displays the cycle number that the processor is inputing data, outputting data and causing the index register to count. From this data, the user can assure that inputs and the counting of the index register for both processors occur at the same time, while outputs for the two processors occur at different times. This is usually not a problem for few outputs but may be a problem for many outputs. The user can adjust coefficients to add cycles and hence move the output.

The compiler outputs many messages when running, most of which just indicate what it is doing. If an error is found, it is printed and execution continues. Hardware parameters such as RAM length and decimation ratio are printed which the user needs to input into the layout generator.

INPUT FILE FORMAT

A. FILTER BANK FORMAT

1. {Input Channels}
   These are channels that fetch input data from outside the chip. Either the raw input or the channel output can be accessed by standard channels.

2. {Standard Channels}
   These are channels that process in parallel.

3. [Decimation Channel]
   This channel operates on the outputs of all standard channels above after decimation by m.

4. <Standard Channels>
   Standard channels that are not decimated.

5. E - end of file marker

B. CHANNEL FORMAT

1. Channel type identifier
   S - standard
   I - Input
   D - Decimation
2. {section}

C. SECTION FORMAT

(2 letter section identifier) [options] coefficients
options:
O - send section output off chip
R - rectify section input
o - take input from channel output (channel 1 default)
n - take input from channel n

coefficients:
are separated by commas and enclosed in parenthesis

D. COEFFICIENT FORMAT

Canonical signed digit format
[-]<binary digit>.<CSD>

a binary digit is either 0,1
a CSD is 0,1,-

The optional minus sign at the beginning negates the remainder of the coefficient unlike the CSD value '-'.

Example: -1.0010- = -(1+1/8-1/32)

Notation:
quantities in {} must be inserted at least once and can be repeated
quantities in [] are optional
quantities in <> are optional and can be repeated

Notes:

1. Comment lines with 'C' in the first column can be put anywhere except where coefficients are expected.

2. Spaces are ignored.

3. The letters 'E', 'S', 'I', 'D' should be treated as reserved and may cause trouble if used as part of a section identifier.

INPUT FILE EXAMPLE EVEN CHANNELS OF SPEECH RECOGNITION CHIP

```
C
C filter bank1 – even channels
C
C Gains corrected 10/7/83
C
I
ID
S
C channel 0
BS (.100-,-.00001000-,.000011)
BS (1,-.0001001,.00100-)
L1 R (1.000000-,.001)
S
C channel 2
BS (.0100-,-.00001101,.1001)
BS (10., -.0001,.1101)
L1 R (1.000000-,.001)
S
C channel 4
BS (.00100-,-.000100-,10.01)
BS (10., -.0001,10.11)
L1 R (1.000000-,.0100)
S
C channel 6
BD o (-1.000-000-,1.10101,.0001)
BD (-1.000-00-,1.100101,.0001)
L1 R (1.000000-,.0001)
S
C channel 8
BD o (-1.00-0001,1.001,.001)
BD (-1.00-001,1.011,.001)
L1 R (1.000000-,.0001)
S
C channel 10
BD o (-1.00-001,.101,.00011)
BD (-1.00-001,1.00-,.001)
L1 R (1.000000-,.0001)
S
C channel 12
BD o (-1.00-,-.00101,.001)
BD (-1.00-,.001,.001)
L1 R (1.000000-,.0001)
S
C channel 14
BD o (-1.00–,-1.001,.001)
BD (-1.0-001,-.11,.01)
L1 R  (1.000000-,.000101)
D
L2 O (-1,-.00011,.00011)
E
```

Notes:

Notes:

FILTER LIBRARY

The filter library describes the the way that the compiler should implement each section and the requirements for each section. Diagrams for the sections that are currently implemented are given in figure A1. The order that the coefficients must follow in the input file is given in table A2.

The filter library consists of the file 'filttypes' which contains data on the valid filter sections and a macro file for each filter section. Each macro file has the name which begins with its associated two letter identifier and ends with

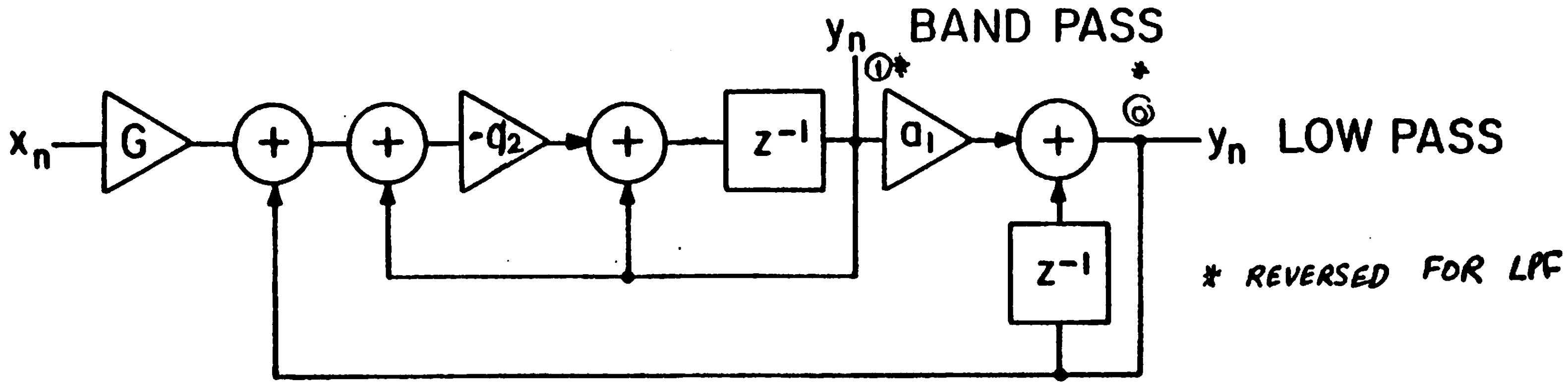
(2 character section identifier) (' ') (number of coefficients used)
(' ') (number of memory locations used) (' ') [comments]

Table A1 - Current 'filttypes'

BD	3	2	Direct	2nd order	section (poles only)
BS	3	2	State	2nd order	BPF
L1	2	1	Direct	1st order	LPF
L2	3	2	State	2nd order	LPF
ID	0	6	Direct	4th order	BPF zeros (input section)
IS	0	1	Input	0 order	
IT	0	1	Input	0 order	output abs value for test
I1	0	4	Direct	2nd order	zeros for DBPF (input section)
iz	3	4	Direct	2nd order	arbitrary zeros (input section)
pz	2	3	Direct	1st order	pole zero combination (input)
dz	0	5	Direct	4th order	BPF zeros (no input)

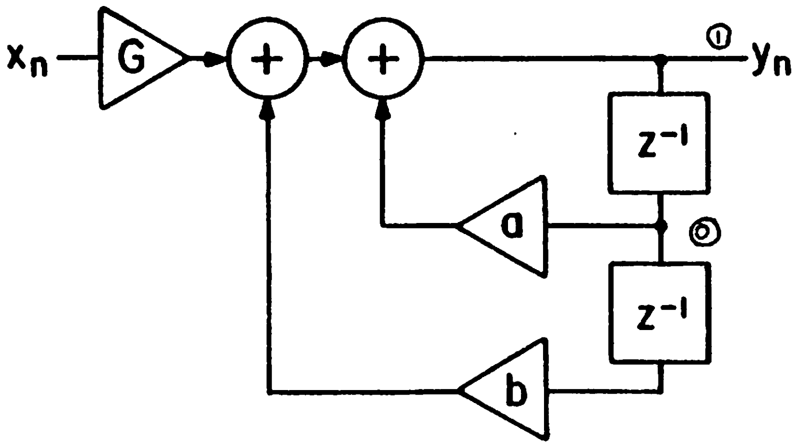
Table A2 - coefficient ordering

Section	coeff1	coeff2	coeff3
BD	b	a	G
BS	G	-a2	a1
L1	a	G	
L2	G	-a2	a1
ID			
IS			
IT			
I1			
iz	a	c	b
pz	a	b	
dz			



BS 2nd order state BPF (no zero at z=-1) $H_{BP}(z) = \dfrac{-Ga_2z^{-1}(1-z^{-1})}{z^{-2}(1-a_2)-z^{-1}(2-a_2-a_2a_1)+1}$

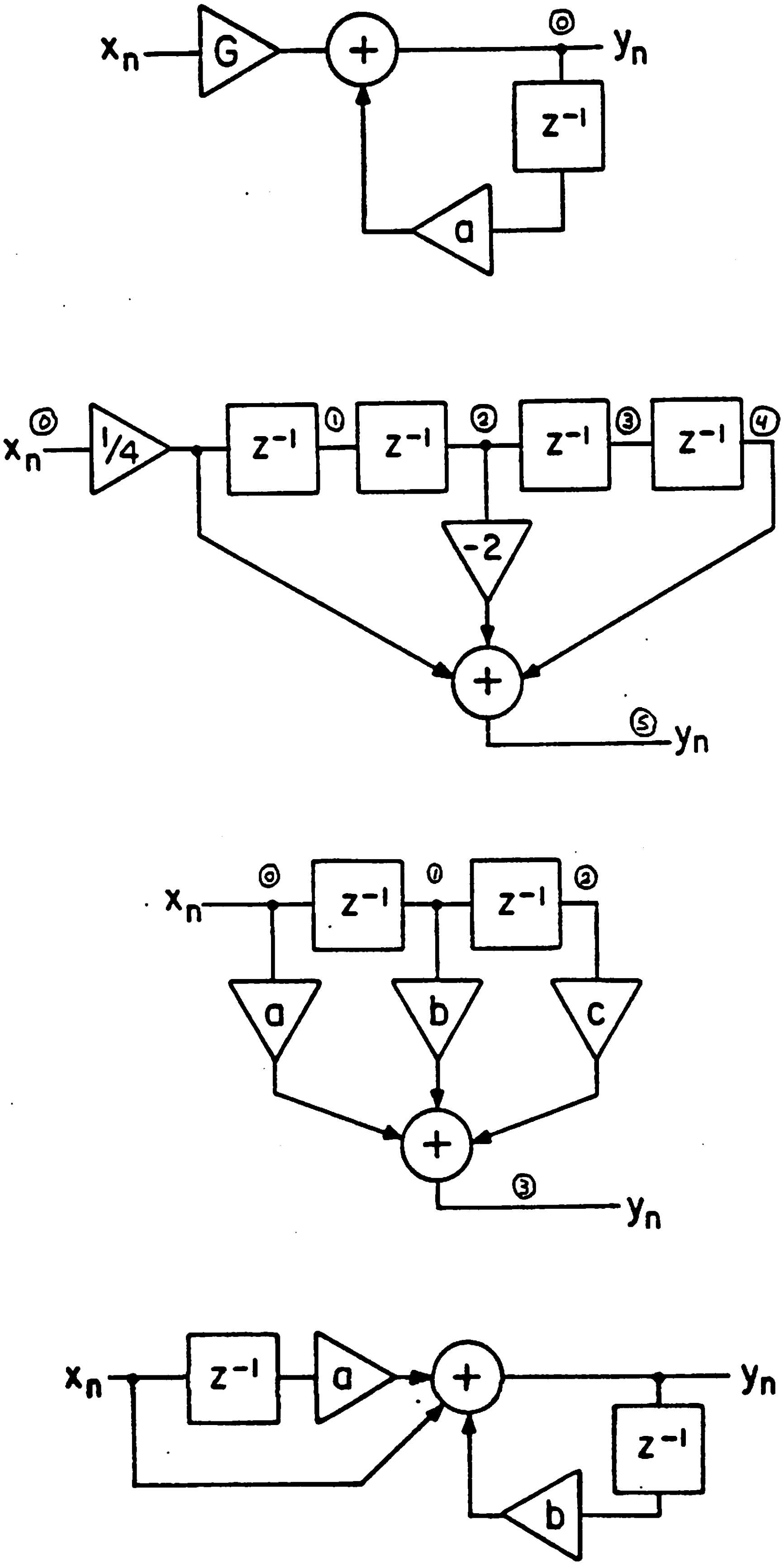
L2 2nd order state LPF (no zero at z=-1) $H_{LP}(z) = \dfrac{-Ga_1a_2z^{-1}}{z^{-2}(1-a_2)-z^{-1}(2-a_2-a_2a_1)+1}$



BD 2nd order direct form poles

$$H(z) = \frac{G}{1-az^{-1}-bz^{-2}}$$

FIGURE A1. FILTER SECTIONS



L1 1st order LPF

$$H(z) = \frac{G}{1 - az^{-1}}$$

ID 4th order zeros for DBPF

$$H(z) = (1 - z^{-2})^2/4$$

iz 2nd order zeros

$$H(z) = a + bz^{-1} + cz^{-2}$$

I1 2nd order zeros for DBP

$$H(z) = (1 - z^{-2})/2$$

pz pole zero pair

$$H(z) = \frac{1 + az^{-1}}{1 - bz^{-1}}$$

FIGURE A1. FILTER SECTIONS

Macros

The macro files include the micro code for the section and indicate where coefficients and options from the input file should be inserted. The symbolic micro code specifies the state of the control lines for that cycle according to the format:

Micro code format:

Col 1: Memory operation (r,w,x)
r=read
w=write
x=dont care (read)

Col 3,4: Relative memory address (0-99,**,xx)
00-99 : add to offset to get true address
** : section input, use address of input source
xx : dont care (00)

The section output must be given the highest address.
Address should be sequential starting from 0.
Sections which read inputs from off chip should store the input in add 0.

Col 6: Write latch operation (l,h,x)
l=latch data on membus
h=hold data in latch
x=dont care (hold)

Col 8: Barrel shifter input source (m,s,x)
m=memory
s=shifter output
x=dont care (shifter)

Col 10: Number of shifts for barrel shifter (0-5,x,*)
0,1,2,3,4,5 : shift designated number
* : set from coefficient value
x : dont care (shift 0)

Col 12: Adder a input operation (a,+,-,z,x,*)
a: absolute value
+: true
-: negate
z: zero
x: dont care (zero)
*: absolute value if rectify is specified

Col 14: Adder b input source (a,m,c,z,x)
a: accumulator
m: memory
c: 1's complementor output
z: zero
x: dont care (zero)

Col 16: I/O operation (i,O,a,x,*)
i: enable parallel input, set input strobe
O: enable parallel output, put acc on membus, set output strobe
a: put acc on membus, no output
x: dont care (acc on membus, no output)
*: parallel output if output option set

TIMING CONSIDERATIONS IN WRITING MICRO CODE

Several points must be kept in mind when writing the code:

1. Micro code is written in the order in which it comes out of the ROM. The order of actual operations may be different.

2. The results of the adder operation described in one cycle is valid the next (at the accumulator output).

3. The write latch and barrel shifter input controls are delayed one cycle and should be initiated one cycle before the desired operation.

4. Data from a memory read appears at the barrel shifter input mux the cycle after the read operation. Similarly, data stored in the write latch appears at the barrel shifter input mux the cycle after the write operation is performed.

5. Parallel input data is latched the phase 1 before xmitin2 goes low and is gated onto the membus when xmitin2 is low.

EXAMPLE:

```
1  r  00  x  m  x  x  x  x
2  r  **  x  m  *  +  z  x
3  x  xx  l  x  *  *  a  x
4  w  00  x  x  x  x  x  *
```

cycle 1: read data from mem(0), set barrel shifter
mux for memory.

cycle 2: read output of last section, acc=mem(0)*coeff

cycle 3: set write latch to get results of computation (1 cycle early),
acc=acc+coeff*abs(input) (if rectification option is set)
acc=acc+coeff*input (if rectification option not set)

cycle 4: membus=acc, (write latch operation takes place),
mem(0)=data in write latch (acc),
turn on I/O buffers if output option is set

Listings of macros in library

BD num coeff: 3 numloc: 2
Direct 2nd order section (poles only)

```
r  01  x  m  x  x  x  x
r  00  l  m  0  +  z  x
w  00  h  m  *  +  Z  a
r  **  x  m  *  +  a  x
x  xx  l  x  *  *  a  x
w  01  x  x  x  x  x  *
E
```

BS num coeff: 3 numloc: 2
State 2nd order BPF

```
r  **  x  m  x  x  x  x
r  01  x  m  *  *  z  x
r  00  h  m  0  +  a  a
r  01  l  m  0  +  a  x
w  01  x  m  0  +  z  a
r  00  l  m  *  +  a  x
w  01  x  m  0  +  z  *
x  xx  l  x  *  +  a  x
w  00  x  x  x  x  x  a
E
```

L1 num coeff: 2 numloc: 1
Direct 1st order LPF

```
r  00  x  m  x  x  x  x
r  **  x  m  *  +  z  x
x  xx  l  x  *  *  a  x
w  00  x  x  x  x  x  *
E
```

L2 num coeff: 3 numloc: 2
State 2nd order LPF

```
r  **  x  m  x  x  x  x
r  00  x  m  *  *  z  x
r  01  h  m  0  +  a  a
r  00  l  m  0  +  a  x
w  00  x  m  0  +  z  a
r  01  l  m  *  +  a  x
w  00  x  m  0  +  z  a
x  xx  l  x  *  +  a  x
w  01  x  x  x  x  x  *
E
```

```
ID num coeff:      0 numloc:      6
Direct 4th order  BPF zeros (input section)

r   02   l   m   x   x   x   i
r   04   h   m   1   -   z   i
w   00   h   m   2   +   a   x
r   03   l   m   2   +   a   x
w   05   l   x   0   +   z   *
r   02   h   m   x   x   x   a
w   04   l   x   0   +   z   x
r   01   h   m   x   x   x   a
w   03   l   x   0   +   z   x
r   00   h   m   x   x   x   a
w   02   l   x   0   +   z   x
w   01   x   m   x   x   x   a
E
```

```
IS  num coeff:      0 numloc:      1
Input  0   order

x   xx   l   x   x   x   x   i
r   00   h   x   x   x   x   i
w   00   h   x   x   x   x   x
E
```

```
IT  num coeff:      0 numloc:      1
Input  0   order  output abs value for test

x   xx   l   x   x   x   x   i
r   00   h   x   x   x   x   i
w   00   h   m   x   x   x   x
x   xx   x   x   0   a   z   x
x   xx   x   x   x   x   x   *
E
```

```
I1  num coeff:      0 numloc:      4
Direct 2nd order zeros for DBPF (input section)

r   01   l   m   x   x   x   i
w   00   l   m   0   +   z   i
r   02   h   m   1   +   z   x
w   02   x   m   1   -   a   x
r   00   l   m   x   z   a   x
w   03   l   x   0   +   z   *
w   01   x   x   x   x   x   a
E
```

iz num coeff: 3 numloc: 4
Direct 2nd order arbitrary zeros (input section)

```
r   01  l  m  x  x  x  i
w   00  l  m  0  +  z  i
r   02  h  m  *  +  z  x
w   02  x  m  *  +  a  x
r   00  l  m  *  +  a  x
w   03  l  x  0  +  z  *
w   01  x  x  x  x  x  a
E
```

pz num coeff: 2 numloc: 3
Direct 1st order pole zero combination (input)

```
r   01  l  m  x  x  x  i
r   02  h  m  *  +  z  i
w   00  h  m  *  +  a  x
w   01  l  x  0  +  a  x
w   02  x  x  x  x  x  *
E
```

dz num coeff: 0 numloc: 5
Direct 4th order BPF zeros (no input)

```
r   01  x  m  x  x  x  x
r   03  x  m  1  -  z  x
r   **  x  m  2  +  a  x
r   02  l  m  2  +  a  x
w   04  l  x  0  +  z  *
r   01  h  m  x  x  x  a
w   03  l  x  0  +  z  x
r   00  h  m  x  x  x  a
w   02  l  x  0  +  z  x
r   **  h  m  x  x  x  a
w   01  l  x  0  +  z  x
w   00  x  m  x  x  x  a
E
```

Notes:

APPENDIX B USING THE LAYOUT GENERATOR

The files 'bank1rom' and 'bank2rom' must contain the ROM data for the respective processors. The format (see below) of these rom files is the same as that of 'romout' created by the filter compiler.

The layout generator writes four files, all in KIC format. The file 'filter' contains the complete filter description, while 'controller' and 'procram' contain the controller and data path descriptions respectively. The file, 'indexreg', contains the index register.

To simplify the development of the layout generator, several restrictions have been placed on the range over which the hardware parameters can vary.

Restrictions:

1. The RAM size must be a multiple of 4 between 8 and 128 words.
   Max RAM size with decimation is 92 words.
2. The data path width must be a multiple of 2 greater than 6.
3. The ROM must contain a multiple of 32 words not greater than 256.
4. For two processor designs:
   The two data paths must be identical
   The two ROMs must have the same length.
   The decimation ratio (<=8) must be the same for each processor.

The format of the rom file is:

line 1. number of words of ROM
line 2. ROM width
line 3. number of desired instructions
line 4. RAM length
line 5. decimation ratio
line 6. number of data paths
line 7. data path width
line 8. column decoded (true/false)
line 9. micro instruction executed first
line 10. 2nd micro instruction
...
line n. last micro instruction

There must be a multiple of 32 micro instructions (16 for non-column decoded layouts).

Format for each line of the rom file starting with column 1:

1. RAM address/ index register control
   (without decimation: n bits, msb first)
   (with decimation: always 7 bits)
   format with decimation:

1 1 0 a a a a	select index register for msb's, ROM for lsb's
1 1 1 x x x x	increment index register
0 a a a x x x	3 bit address from ROM
0 a a a a x x	4 bit address from ROM
0 a a a a a x	5 bit address from ROM
0 a a a a a a	6 bit address from ROM
a a a a a a a	7 bit address from ROM (can not start 11)

   a= RAM address bits, msb first
   x= unused

2. memwrite
   memory write control 1=write, 0=read
3. wrlatch
   controls loading of write latch 1=latch, 0=hold
   delayed 1 cycle
4. shiftsrc
   controls barrel shifter input mux 1=shifter, 0=memory
   delayed 1 cycle
5. 5-shiftnum (3 bits msb first)
   number of barrel shifter shifts (subtracted from 5)
6. inv1
7. inv2
   controls complementor

inv1=0, inv2=0	true
inv1=x, inv2=1	invert
inv1=1, inv2=0	absolute value

8. bsel1
9. bsel2
   controls adder b input mux

bsel1=0, bsel2=0	zero b
bsel1=0, bsel2=1	1's complementor output
bsel1=1, bsel2=0	memory
bsel1=1, bsel2=1	floating (to select acc)

10. zeroa*
    Zero adder A input if zeroa*=0
11. xmitacc*
    transmit acc to membus if xmitacc*=0
12. accb*
    enable acc onto b input if accb*=0
13. xmitin2*
    transmit parallel input to membus if xmitin2*=0
14. iobusen
    enable parallel output when iobusen=1

The order of binary data in the ROM file is the same as that of the control lines entering the data path starting from the RAM end. Exceptions are the barrel shifter shift number which is reversed on the data path (i.e. lsb first) and the RAM address lines which consist only of the necessary data (no unused lines in the case of decimation). Several signals have been tied high or low at the data path and need not be specified.

Notes:

APPENDIX C USING THE TESTER

The pattern generator part of the tester set up was developed at UCB by Jim Beck. In order to use this, the 64 pin test head must be connected to the main tester unit. A special patch connector is used which contains the circuit shown in figure C1. As the tester has no capabilities to handle the decimation and the index register is not included on the data path chip, a switch bank is used to simulate the decimation. The user can then check each channel by setting the switches appropriately.

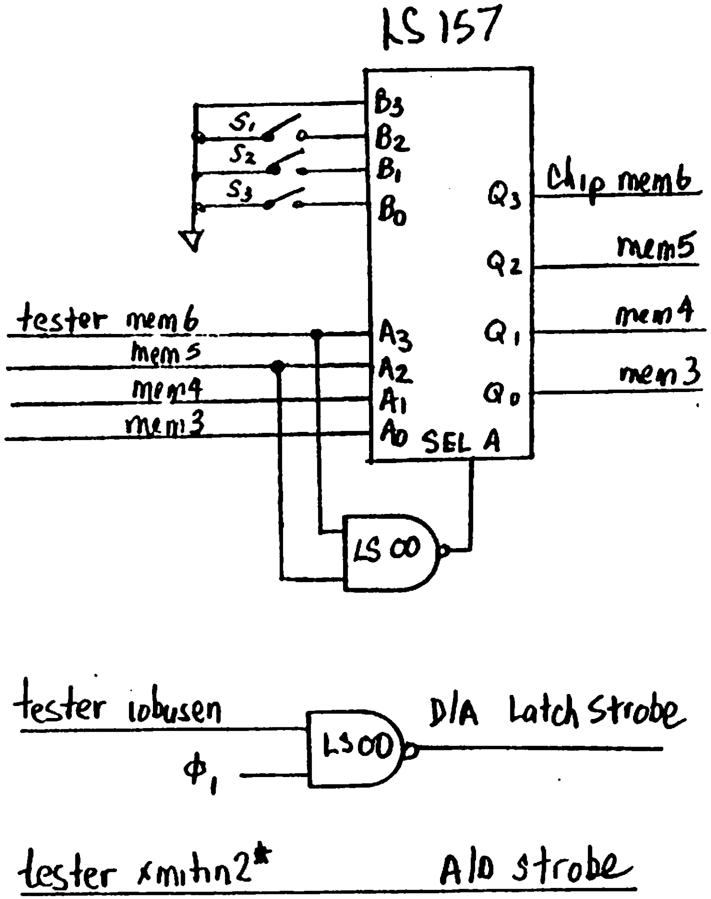
After the hardware is set up, the filter specifications should be compiled with the filter compiler. The ROM data is converted to tester format in the file 'testout' by running 'romtotest'. Then the programs 'tas' and 'tdown' (see the tester manual) can be used to start the tester. If two data paths are being used, the filters of each data path are tested separately.

This shell script will test the filters specified in 'filterdata':

```
ctr1
romtotest
tas testout
tdown
```

The data path circuits which should be used are proc.july, procfast.july (M37YPH1, M37YPG3). The circuits have the following pin out and connection to the tester:

IC	tester	function	IC	tester	function
pin	connection		pin	connection	
1	GND	substrate	33	2C	mem4
2	11	d17	34	2B	mem3
3	10	d16	35	2A	mem2
4	F	d15	36		
5	E	d14	37		
6			38		
7			39		
8			40		
9			41	29	mem1
10	D	d13	42	28	mem0
11	C	d12	43	27	memwrite
12	B	d11	44	23	wrlatch
13	A	d10	45	22	shiftscr
14	9	d9	46	26	sh2
15	8	d8	47	25	sh1
16	7	d8	48	24	sh0
17	6	d6	49	21	inv1
18	5	d5	50	20	inv2
19	4	d4	51	1F	bsel1
20	3	d3	52	1E	bsel2
21	2	d2	53	1D	zeroa*
22	1	d1	54	1B	xmitacc*
23	0	d0	55	1A	accb*
24			56		
25			57		
26			58		
27			59	16	xmitin2*
28			60	15	iobusen
29			61	clock1	ph1
30	GND	GND	62	clock0	ph2
31	2E	mem6	63	Vdd	Vdd
32	2D	mem5	64	14	paden



S1	S2	S3	select channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

1 = open
0 = closed

FIGURE C1. ADDITIONAL TESTER CIRCUITS

Notes:

APPENDIX D1. SPEECH RECOGNITION FILTER BANK

1. Filter compiler input files
2. Chip pinouts
3. Timing

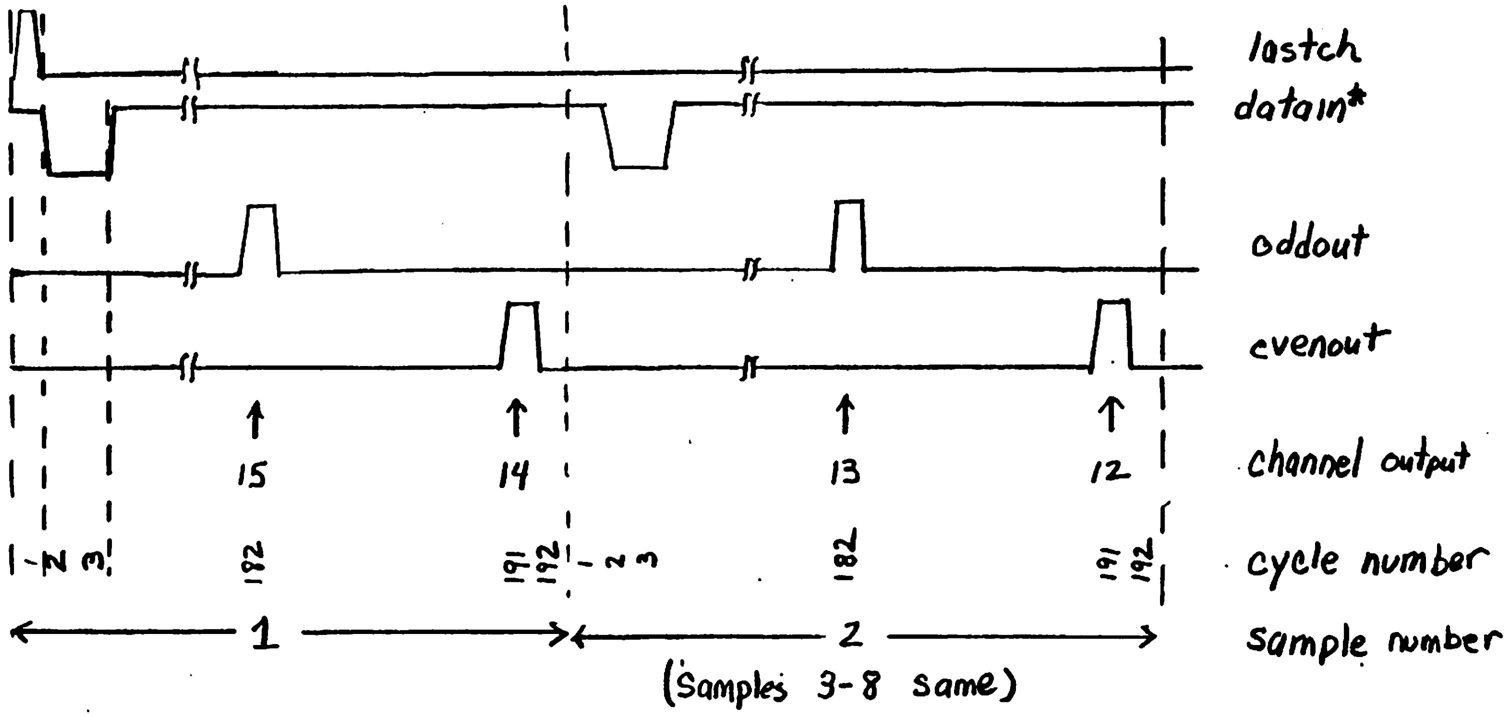
```
C
C filter bank1 — even channels
C
C Gains corrected 10/7/83
C
I
ID
S
C channel 0
BS (.100-,-.00001000-,.000011)
BS (1,-.0001001,.00100-)
L1 R (1.000000-,.001)
S
C channel 2
BS (.0100-,-.00001101,.1001)
BS (10., -.0001,.1101)
L1 R (1.000000-,.001)
S
C channel 4
BS (.00100-,-.000100-,10.01)
BS (10., -.0001,10.11)
L1 R (1.000000-,.0100)
S
C channel 6
BD o (-1.000-000-,1.10101,.0001)
BD (-1.000-00-,1.100101,.0001)
L1 R (1.000000-,.0001)
S
C channel 8
BD o (-1.00-0001,1.001,.001)
BD (-1.00-001,1.011,.001)
L1 R (1.000000-,.0001)
S
C channel 10
BD o (-1.00-001,.101,.00011)
BD (-1.00-001,1.00-,.001)
L1 R (1.000000-,.0001)
S
C channel 12
BD o (-1.00-,-.00101,.001)
BD (-1.00-,.001,.001)
L1 R (1.000000-,.0001)
S
C channel 14
BD o (-1.00--,-1.001,.001)
BD (-1.0-001,-.11,.01)
L1 R  (1.000000-,.000101)
D
L2 O (-1,-.00011,.00011)
E
```

```
C
C Filter Bank2 — odd channels
C
C
C Gains corrected 10/7/83
C
Input
ID
Standard
C channel 1
BS (.100-,-.00001011,.00101)
BS (1,-.0001001,.01001)
L1 R (1.000000-,.001)
S
C channel 3
BS (.0010-,-.00010001,1.1001)
BS (10., -.000100-,1.01)
L1 R (1.000000-,.01)
S
C channel 5
BD o (-1.000-001,10.0-,.000011)
BD (-1.000-,1.1011,.0001)
L1 R (1.000000-,.0001)
S
C channel 7
BD o (-1.00-001,1.011,.0001)
BD (-1.000--,1.1,.001)
L1 R (1.000000-,.0001)
S
C channel 9
BD o (-1.00-0001,1.00-,.00100-)
BD (-1.00-001,1.001,.001)
L1 R (1.000000-,.0001)
S
C channel 11
BD o (-1.00-,.01,.00100-)
BD (-1.00-,.1,.001)
L1 R (1.000000-,.0001)
S
C channel 13
BD o (-1.00-0-,-.101,.00100-)
BD (-1.00-0-,-.01,.01)
L1 R (1.000000-,.0001)
S
C channel 15
BD o (-1.00--,-1.1,.01)
BD (-1.0-,-1.01,.001)
L1 R (1.000000-,.0001)
D
L2 O (-1,-.00011,.00011)
E
```

Filter bank chip pin-out.

1	gnd	21	
2	d12	22	
3	d11	23	
4	d10	24	
5	d9	25	
6	d8	26	
7	d7	27	
8	d6	28	
9	d5	29	
10	d4	30	GND
11	datain*	31	Vdd
12	evenout	32	phase 2 clk
13	oddout	33	phase 1 clk
14	lastch	34	
15	GND	35	paden
16		36	d17 (msb)
17		37	d16
18		38	d15
19		39	d14
20		40	d13

For non column decoded versions, the output strobes are inverted and pin 15 is clear.



$f_{sample} = 14$ KHz $\quad f_{CLK} = 2.688$ MHz

FILTER BANK TIMING (see appendix E for details)

APPENDIX D2 Consumer stereo spectrum analyzer

1. 3 dB frequency files (datain) for both banks
2. Filter compiler input files for both banks
3. Chip pinout

Bank 1 'datain' file

20000	12
4680	6550
2340	3280
1170	1640
590	830
292	410
146	207
70	100
36	50
0	0

Bank 2 'datain' file

20000	15
6550	9000
3280	4680
1640	2340
830	1170
420	590
207	292
100	146
50	70
0	0

```
C
C Hi-Fi spectrum analyzer filter input file
C
I
I1
S
C
C  filter number        1 f1=     4680 f2=     6550
C
BD o  (-.1,-.10-,.01)
L1 R (1.000000-,.00001)
S
C
C  filter number        2 f1=     2340 f2=     3280
C
BD o  (-1.0-,1.001,.001)
L1 R (1.000000-,.00001)
S
C
C  filter number        3 f1=     1170 f2=     1640
C
BD o  (-1.00-00-,1.110-,.0001)
L1 R (1.000000-,.00001)
S
C
C  filter number        4 f1=      590 f2=      830
C
BD o  (-1.000-00-,10.00-,.00001)
L1 R (1.000000-,.00001)
S
C
C  filter number        5 f1=      292 f2=      410
C
BS  (.1,-.00001001,.0101)
L1 R (1.000000-,.00001)
S
C
C  filter number        6 f1=      146 f2=      207
C
BS  (.1,-.00000101,.00101)
L1 R (1.000000-,.00001)
S
C
C  filter number        7 f1=       70 f2=      100
C
BS  (.1,-.0000001,.000101)
L1 R (1.000000-,.00001)
S
C
C  filter number        8 f1=       36 f2=       50
C
BS  (.1,-.00000001,.0000101)
L1 R (1.000000-,.00001)
D
```

```
L2 0  (-1,-.00011,.00011)
E
```

```
C
C Hi-Fi spectrum analyzer filter input file
C
I
I1
S
C
C  filter number       1 f1=     6550 f2=      9000
C
BD o  (-.1001,-1.10-,.01)
L1 R (1.000000-,.00001)
S
C
C  filter number       2 f1=     3280 f2=      4680
C
BD o  (-.1,.1,.01)
L1 R (1.000000-,.00001)
S
C
C  filter number       3 f1=     1640 f2=      2340
C
BD o  (-1.0-,1.100-,.001)
L1 R (1.000000-,.00001)
S
C
C  filter number       4 f1=      830 f2=      1170
C
BD o  (-1.00-01,10.0-01,.00010-)
L1 R (1.000000-,.00001)
S
C
C  filter number       5 f1=      420 f2=       590
C
BD o  (-1.000-001,10.000-0-,.000010-)
L1 R (1.000000-,.00001)
S
C
C  filter number       6 f1=      207 f2=       292
C
BS  (.1,-.0000100-,.01)
L1 R (1.000000-,.00001)
S
C
C  filter number       7 f1=      100 f2=       146
C
BS  (.1,-.000001,.0010-)
L1 R (1.000000-,.00001)
S
C
C  filter number       8 f1=       50 f2=        70
C
BS  (.1,-.00000010-,.0001)
L1 R (1.000000-,.00001)
D
```

```
L2 0  (-1,-.00011,.00011)
E
```

Chip pinout

1	gnd	21	
2	d12	22	
3	d11	23	
4	d10	24	
5	d9	25	
6	d8	26	
7	d7	27	
8	d6	28	
9	d5	29	
10	d4	30	GND
11	datain*	31	Vdd
12	evenout	32	phase 2 clk
13	oddout	33	phase 1 clk
14	lastch	34	
15	(GND)	35	paden
16		36	d17 (msb)
17		37	d16
18		38	d15
19		39	d14
20		40	d13

For non-column decoded versions, the output strobes are inverted and pin 15 is clear.

APPENDIX D3 - Single filter chip

Filter compiler input file

```
I
ID
S
BD o (-1.00-,.01,.001)
BD O (-1.00-,.1,.01)
E
```

pinout

1	GND	21	d2
2		22	d1
3		23	d0
4		24	datain*
5		25	dataout*
6	paden	26	
7	d9 (msb)	27	
8	d8	28	
9	d7	29	
10	d6	30	
11	d5	31	
12		32	
13		33	
14		34	
15		35	
16		36	
17		37	GND
18		38	Vdd
19	d4	39	ph2
20	d3	40	ph1

Notes:

Notes:

APPENDIX E Chip I/O timing

clocks

The chips require a 2 phase non-overlapping clock. The clock separation should be at least 30 nS. To ensure proper operation, the clocks should pull up to Vdd. The simple 2/8 duty cycle clock generator in figure E1 can be used but restricts the maximum clock rate of the circuits to 67% that obtained with a 3/8 duty cycle clock.

The circuit clock rate is equal to the sample rate (the rate at which the inputs are sampled) multiplied by the number of cycles per sample. For example, the speech recognition chip has 192 cycles per sample and a 14 KHz sample rate. Therefore, a 2.688 MHz clock is required.

The circuits operate with a single +5 V supply and all inputs and outputs are TTL compatible (except the clocks). However, it can be seen that a significant reduction is supply current occurs when for Vbb < 0. This does limit the maximum clock rate that the circuits can be run at.

parallel i/o

The input and output data is in twos complement format and passes through the chip's parallel buss. The data is inverted once when input and again when output. Figure E2 shows the timing for data output. The data on the i/o lines and the output strobe(s) change after the rising edge of clock phase 2. The data is valid after the rising edge of the phase 1 clock when the output strobe is true.

The timing for data input is shown in figure E3. In the normal input mode, the input strobe (datain*) goes low for two cycles each sample. The data is latched during the first phase 1 after the strobe goes low. The strobe also gates the signal onto the internal membus during the second phase 2. In this mode, the input strobe can be used to enable the A/D converter tri-state outputs.

The I/O pads are controlled by the 'paden' (pad·enable) signal. When this signal is high, the pads are in the output mode. When it is low, the pads are in the input mode. Therefore, paden should be high when either output strobe is true and low when the input strobe is true. Usually, paden is connected to the output strobe (or the OR of the output strobes if there are two).

synchronization with decimation

When decimation is used and the post decimation filter sends its output off chip, a reference is required. The lastch signal indicates when the index counter counts down to its minimum value (ie the post decimation filter just output the value for the first channel that is decimated). When the counter is at its minimum value, lastch will go high for the first cycle of the sample and changes on phase 2. When two processors are used with decimation both counters are synchronized so that only a single lastch signal is needed. This timing is illustrated in figure E4.

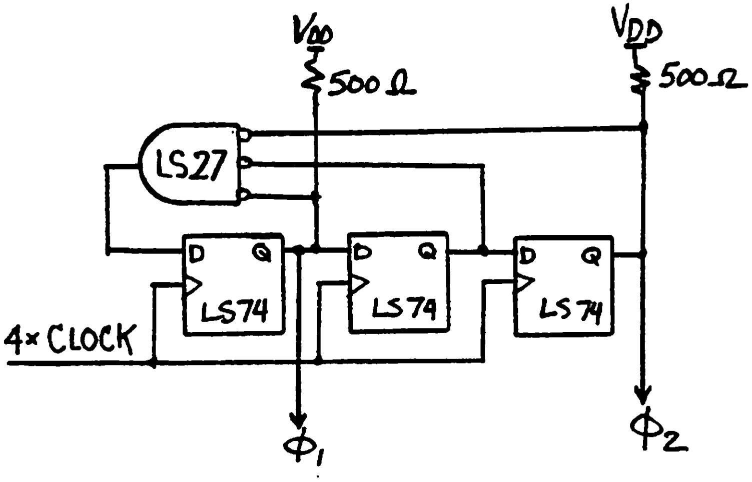


Figure E1 Simple Clock Circuit

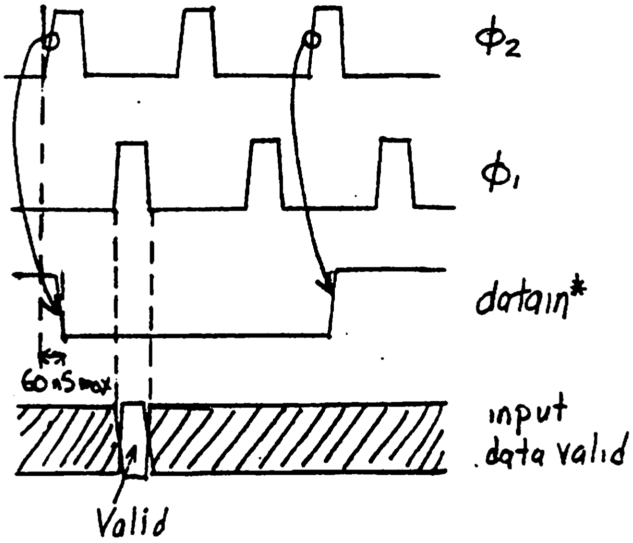


FIGURE E3. Parallel Input Timing

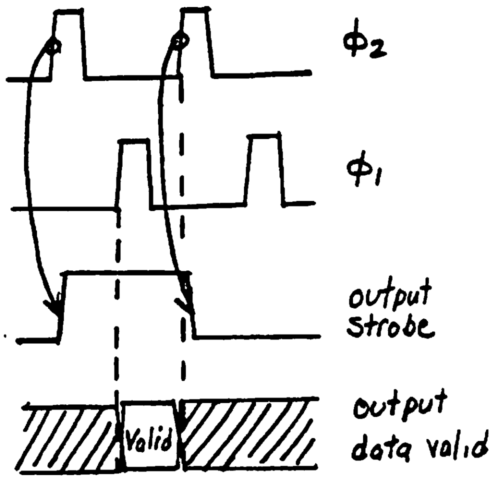


FIGURE E2. Parallel Output Timing

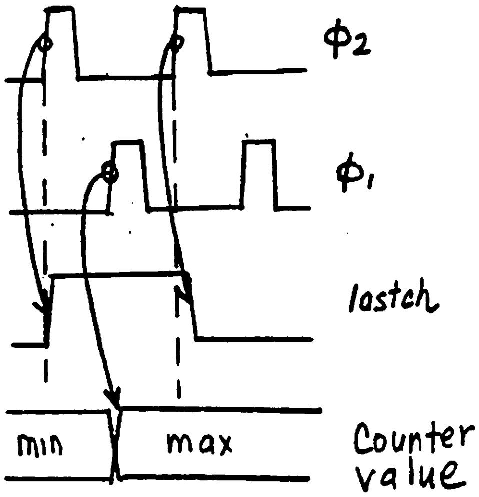


FIGURE E4. Lastch Timing

Notes:

APPENDIX F THE CIRCUIT CELLS

The primary circuit cells of interest have their schematics and plots shown in the figures. The hierarchy listed below indicates how the cells are made of sub-cells.

There are currently two sets of circuit cells. They both follow Mead and Conway layout rules (also rules for MOSIS 4u NMOS) but follow different ratio rules. The cells in the directory 'chip.jan' were designed with the more conservative K=4 (K refers to the ratio of enhancement W/L to depletion W/L) rules, while the cells in 'chipfast.jan' were designed with K=3 rules in critical places. This was done because the processing from various MOSIS vendors varies significantly.

To generate a K=4 design after the generating the layout, the '.KIC' file should include chip.jan in its path before chipfast.jan. For K=3 chips, chipfast.jan should be first in the path. Running 'kictocif' will then generate a CIF file with the desired cells.

Hierarchy of KIC files:

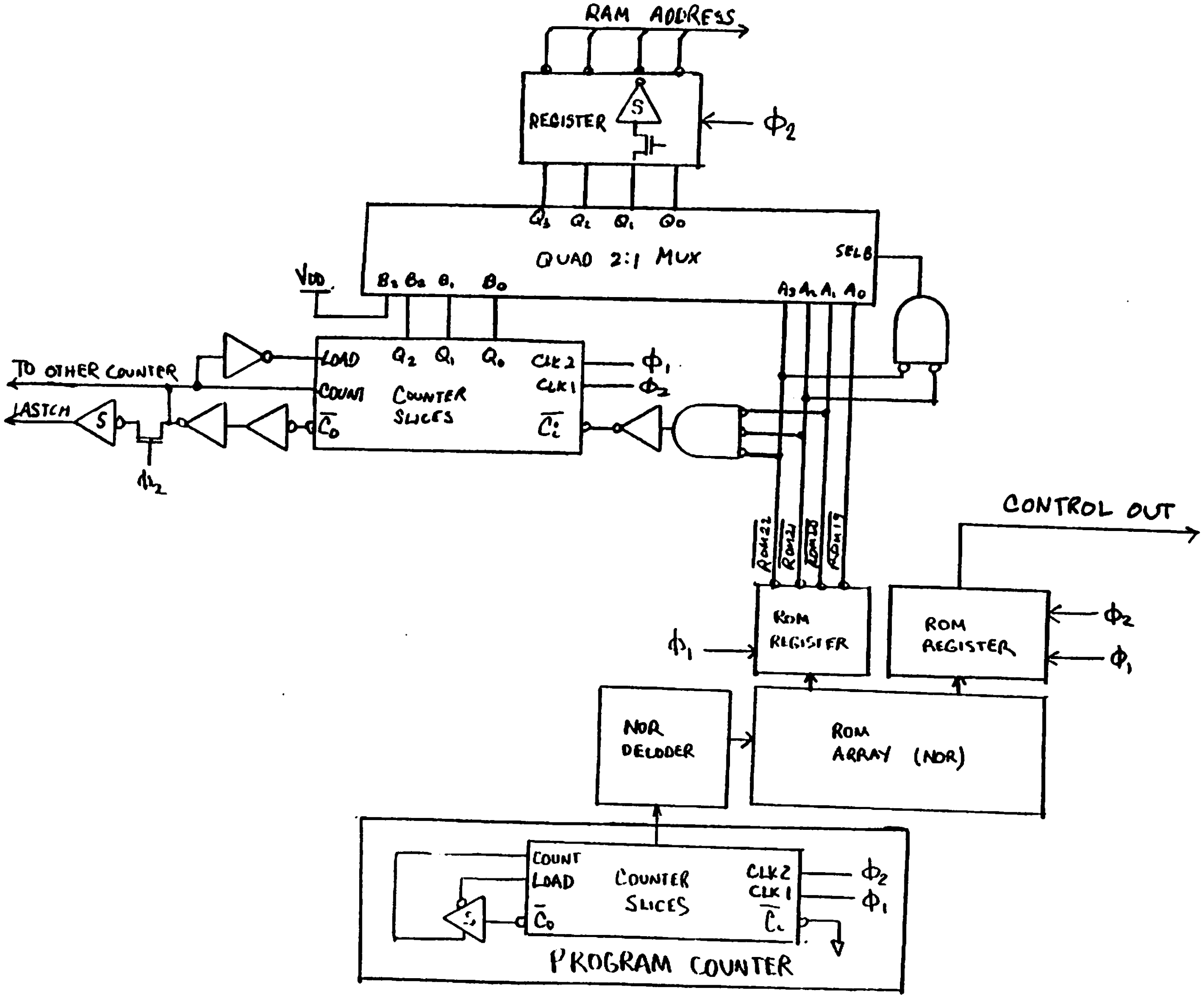
- filter (complete filter)
  - procram (data path) (see below)
  - controller (controller) (see below)
  - datacon.1 (.2)
  - vddcon.1 (.2,.i1,.i2,.3) (Vdd, clock connection)
  - PadIOData (I/O pad)
    - PadBlank
    - PadDriver
  - PadIn (non buffered pad)
  - PadInBuffered (buffered input pad)
  - PadOut (Buffered output pad)
  - PadClk.2 (clock pad)
  - PadVdd (Vdd pad)
  - PadGND (GND pad)
  - PadIO5 (5 data pads always included)
  - padsupply (Vdd, GNd, clock pads)
  - padstr.1 (.2,.i1) (strobe pads for 1 or 2 processor)
  - rout.3 (.4,.5,.6,.7) (routing cells between controller and data path no indexing)
  - rout.i3 (.i4,.i5,.i6,.i7) (routing cells for indexing)

- controller
  - pc.8 (.7,.6,.5,.4) (program counter with n bits)
    - pc.end (control logic for pc)
    - pcprog.0 (cell to program counter to load 0)
    - pcprog.1 (cell to program counter to load 1)
    - counterslice (counter cell)
    - bufferslice (output drivers of pc)
  - deccell0 (ROM decoder cell '0')
  - deccell1 (ROM decoder cell '1')
  - deccell2 (ROM decoder cell that passes column address)
  - decoder.5 (ROM decoder gnd connection)
  - romc0 (ROM cell '0')
  - romc1 (ROM cell '1')
  - romc2 (ROM array ground connection)

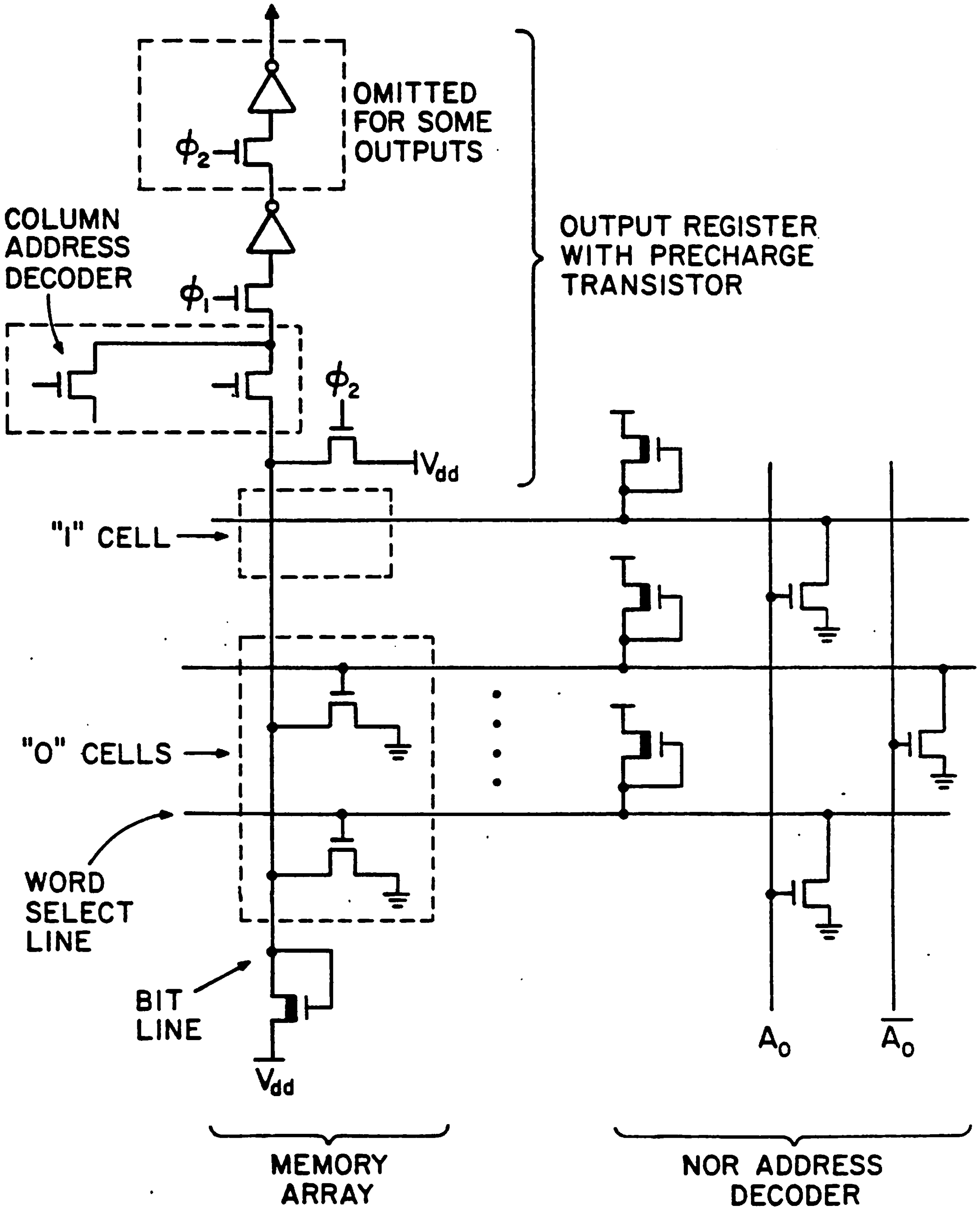
```
  romreg.2 (ROM output register, latches data on ph1)
  romreg.1 (ROM output register, latches data on ph2)
  romreg.3 (.4,.5,.6,.8)
  romregtop.i (routing from ROM registers to index reg and data path
          for decimation)
  romregtop.7 (.6,.5,.4,.3) (routing from ROM register to data
                  path for n RAM address lines)
  pullup.1 (.2,.3,.4)
  romedge.2 (.3,.5,.6,.8,.9,.10)
  indextop.m (wiring to connect left and right index regs)
  indextop.r (.l) (index registers and routing for right and left)
    indexreg (complete index register)
      indexreg.5  (index register with only 1 counter and other circuits)
        indexreg.1 (.2) (muxs, and misc control)
      counterslice (counter)
      pcprog.0 (.1) (counter load program cells)

procram
  dec41 (RAM decoder cell '1')
  dec40 (RAM decoder cell '0')
  ram4c2 (4t RAM cell mirrored)
    ram4c (RAM cell)
  ramdec.1 (.2,.3,.4,.5)  (connects address drivers to decoder rows)
  ramdriver (address bit drivers)
  ramdecgnd (gnd return line for RAM decoder)
  ramgnd.1 (.2)
  ramedge.1
  mem4endrc (RAM select control)
  auslicegnd (ground connections for data path)
  Auslice.msb (msb slice of complete arithmetic unit)
    auslice.msb (1/2 au slice for msb)
      naddce (even adder cell)
      naddco.msb (odd adder cell)
      nsataccum.msb (saturaing accumulator)
      nioblock.msb (parallel and serial I/O)
        nioportc.msb (parallel I/O)
        ninportc.msb (serial input)
        noutportc.msb (serial output)
    bshSlice6.msb (1/2 au slice for msb )
      nauinput.msb (A,B adder input muxs)
      bshtopinput (barrel shifter input mux)
      senselatchbs (sense amp)
      senselatdriv (control for sense amp)
      dec3to6 (barell shifter decoder)
        bshdeccell0 (1,2,3) (decoder cells)
        bshdecload
        bshdecend
        bshdecloadend
        bshdecinv
      bshdriv3to6 (control for 6 shift b.s.)
      bshslice6 (6 bit barrel shifter slice) (see below)
      bshtopsl6 (msb barrel shifter- just straight through)
        bshtopcell1 (basic cell)
        bshendtop1 (end cell)
```

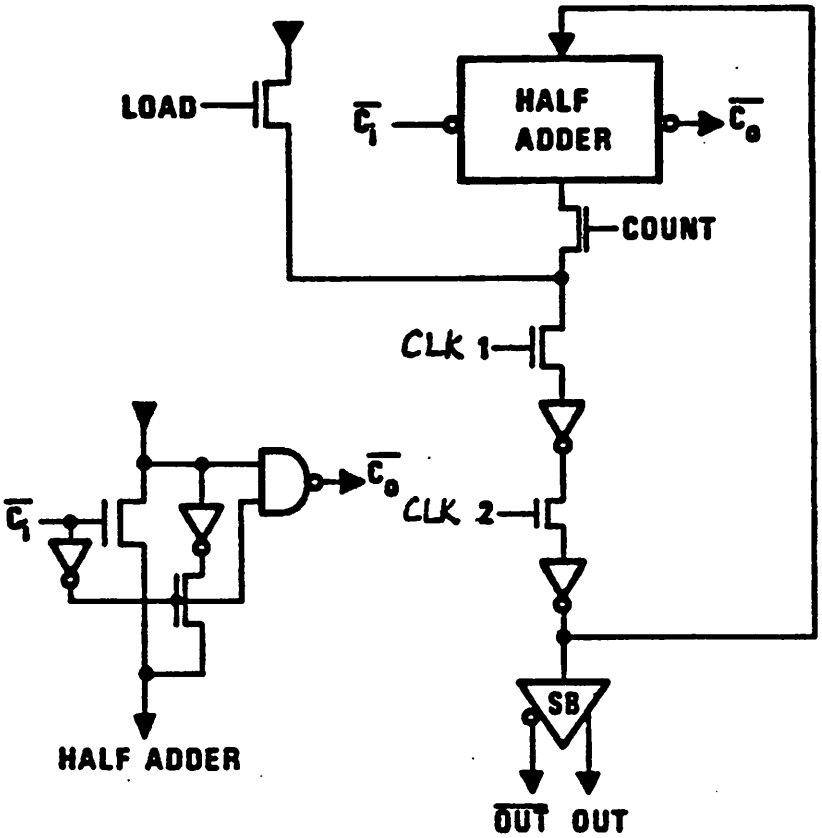
```
Auslice  (slice of arithmetic unit)
  auslice (1/2 au passed barrel shifter)
    nmultc (adder, accumulator)
      naddce (even adder cell)
      naddco (odd adder cell)
      nsataccum (saturaing accumulator)
    nioblock (parallel and serial I/O)
      nioportc (parallel I/O)
      ninportc (serial input)
      noutportc (serial output)
 bshSlice6 (1/2 au including 6 shift barrel shifter)
    senselatchbbs (sense amp)
    bshslice6 (6 bit barrel shifter, input mux, adder muxs)
      bshinput (input mux circuits)
      nauinput (A,B adder input muxs)
      bshsl6 (6 bit barrel shifter)
        bshcell1 (2 bit barrel shifter cell)
        bshend1 (end cell)
```



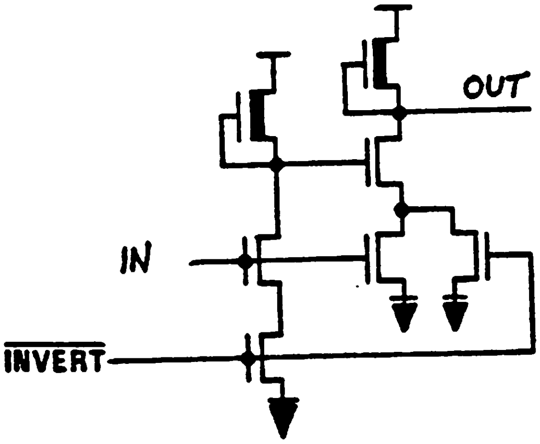
CONTROLLER WITH INDEX REGISTER



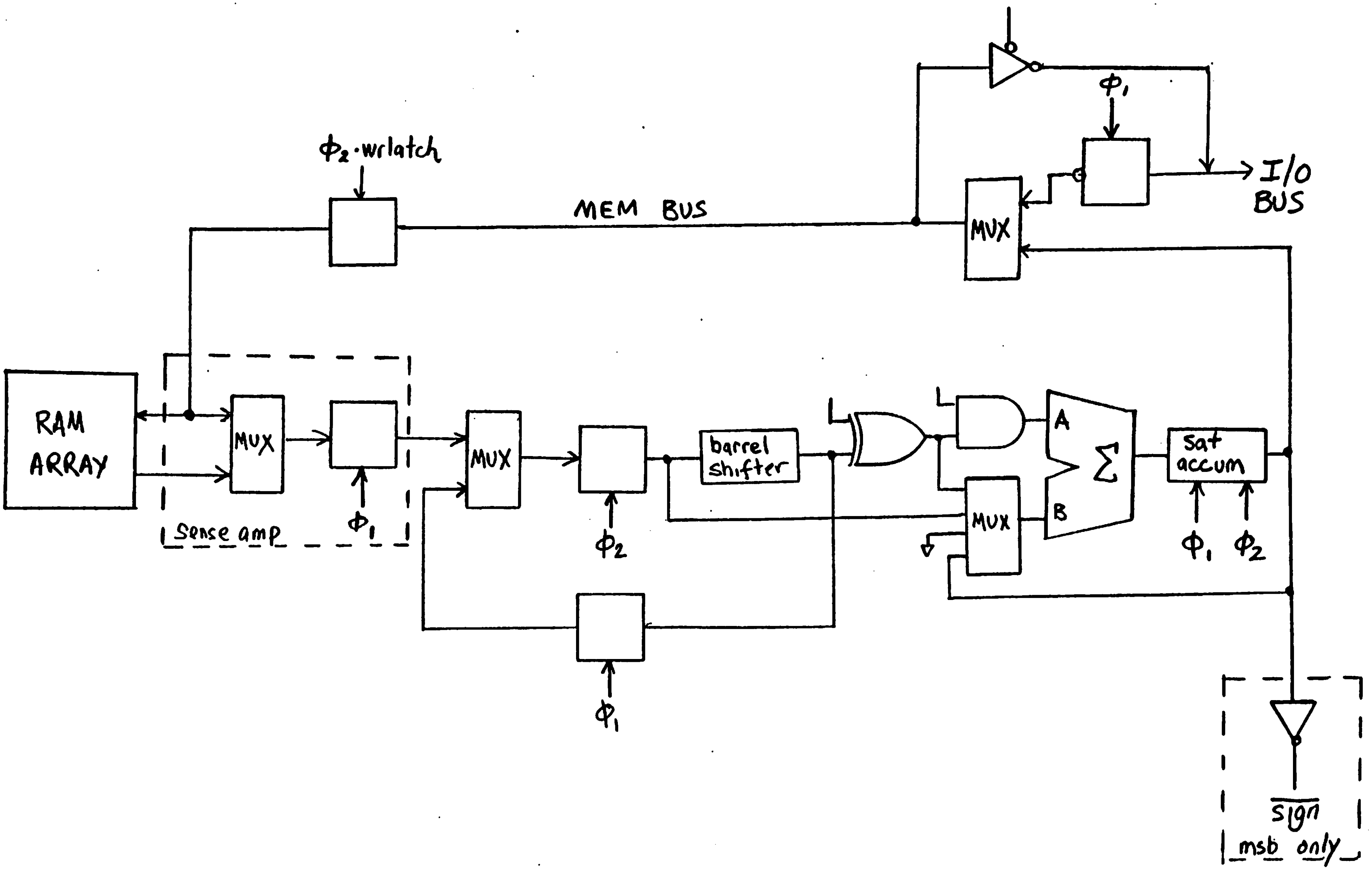
ROM AND OUTPUT REGISTERS



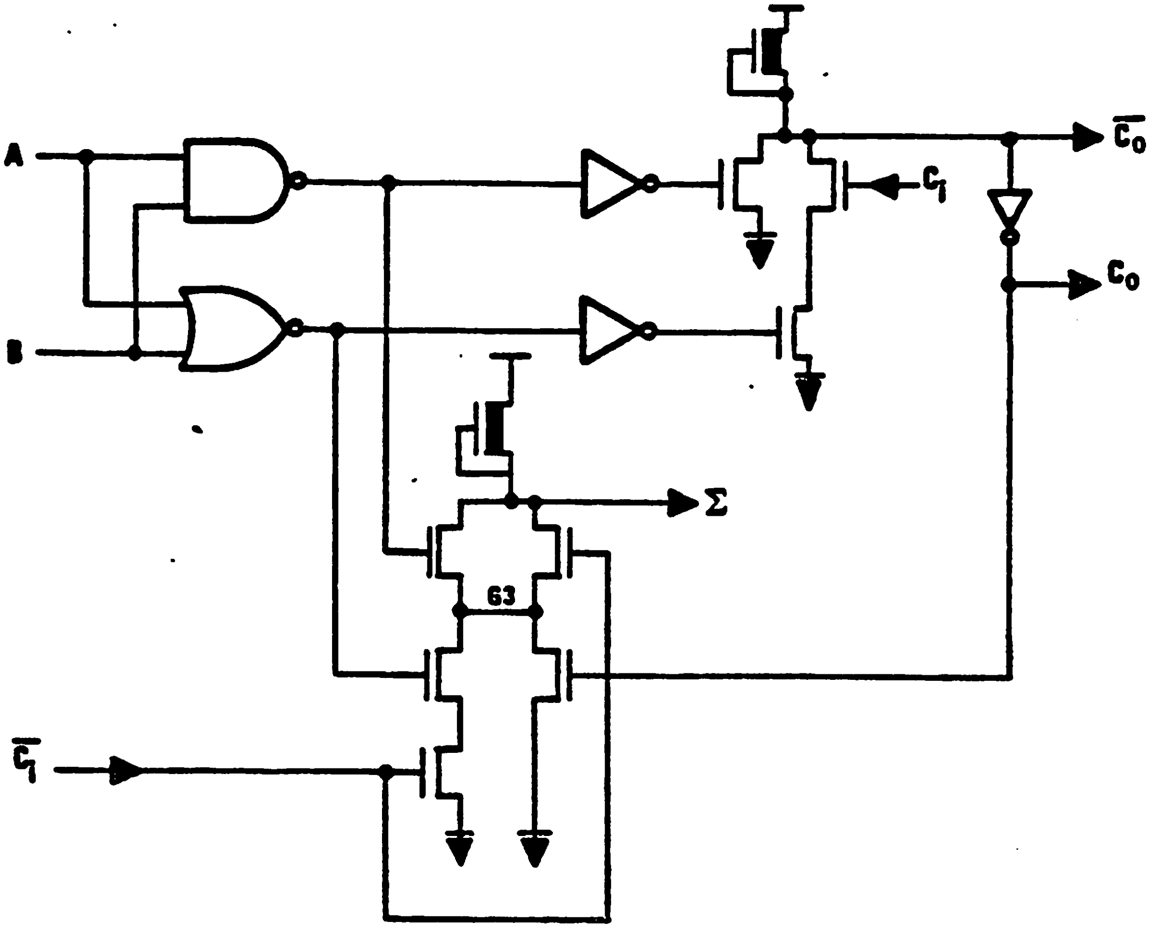
COUNTER CELL (counterslice)



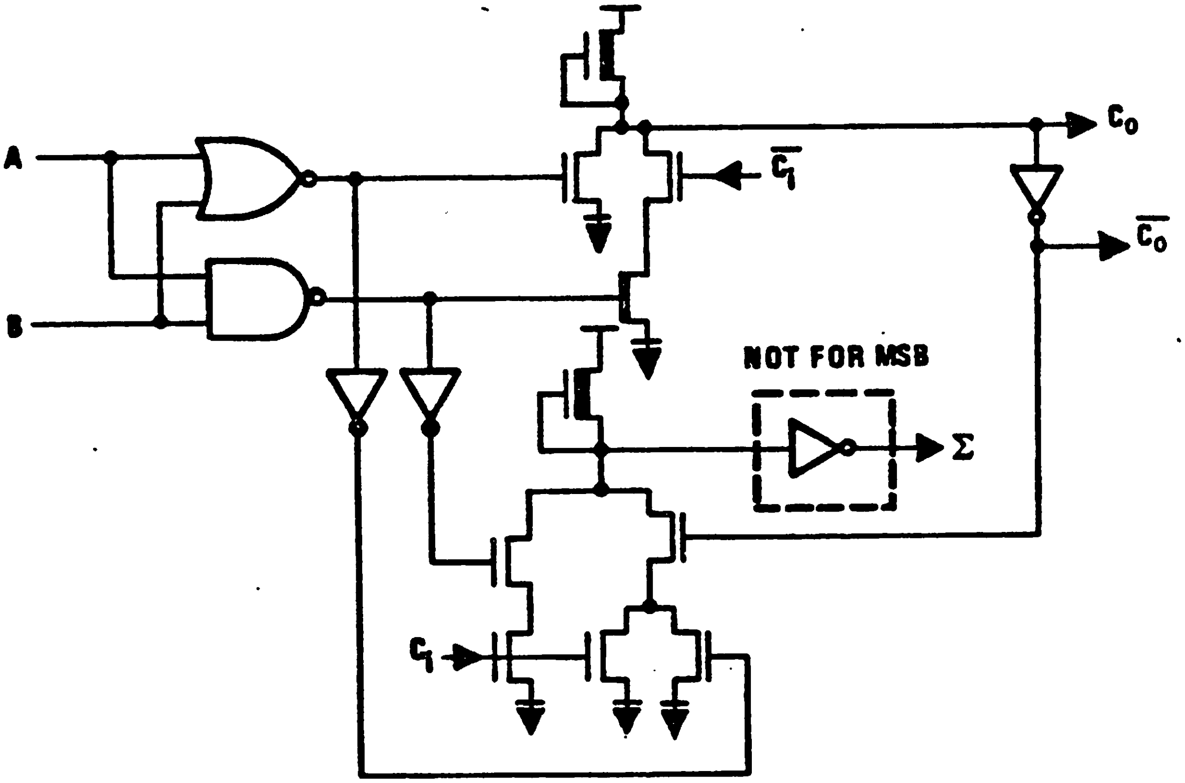
EXOR CIRCUIT



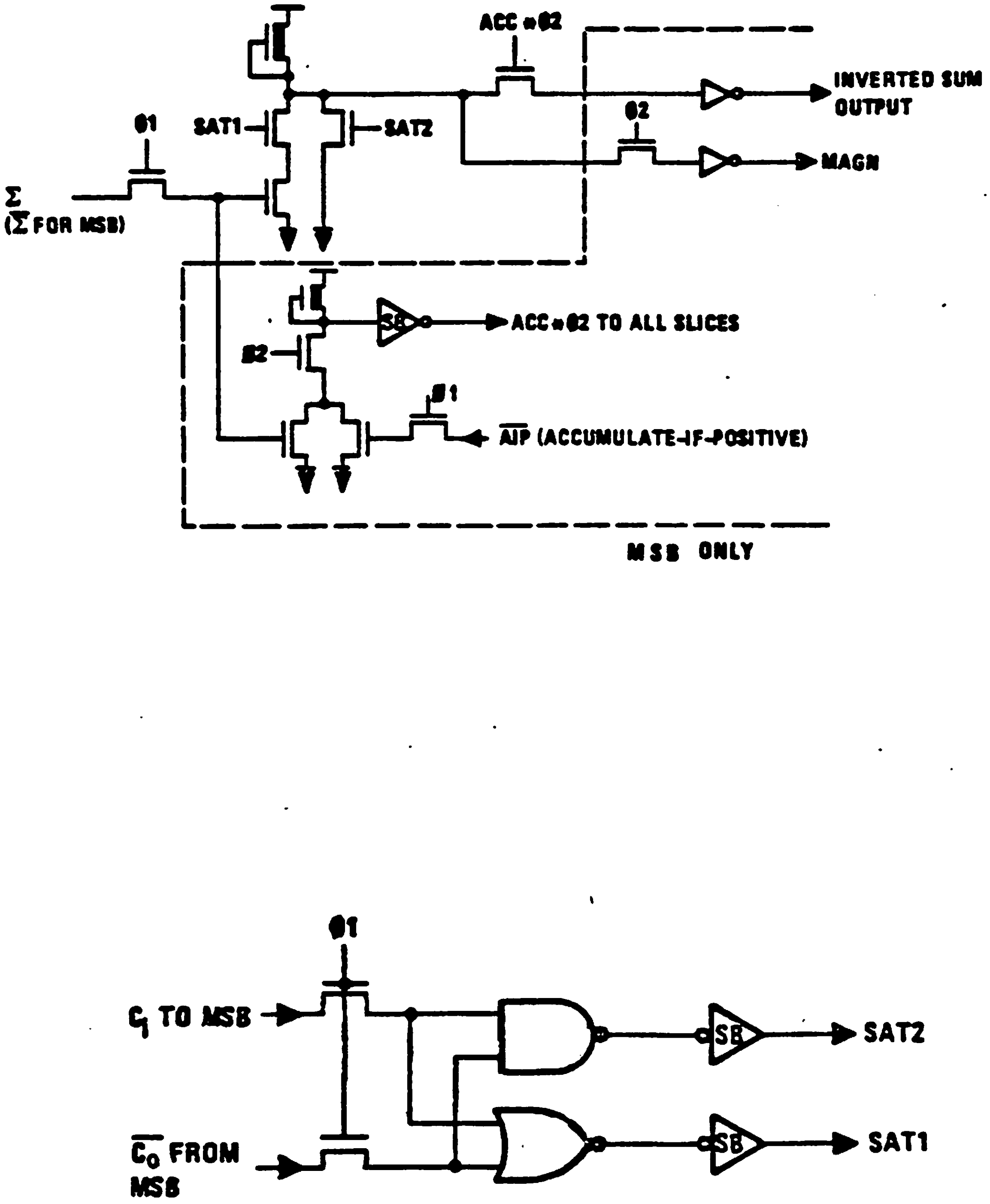
DATA PATH BLOCK DIAGRAM



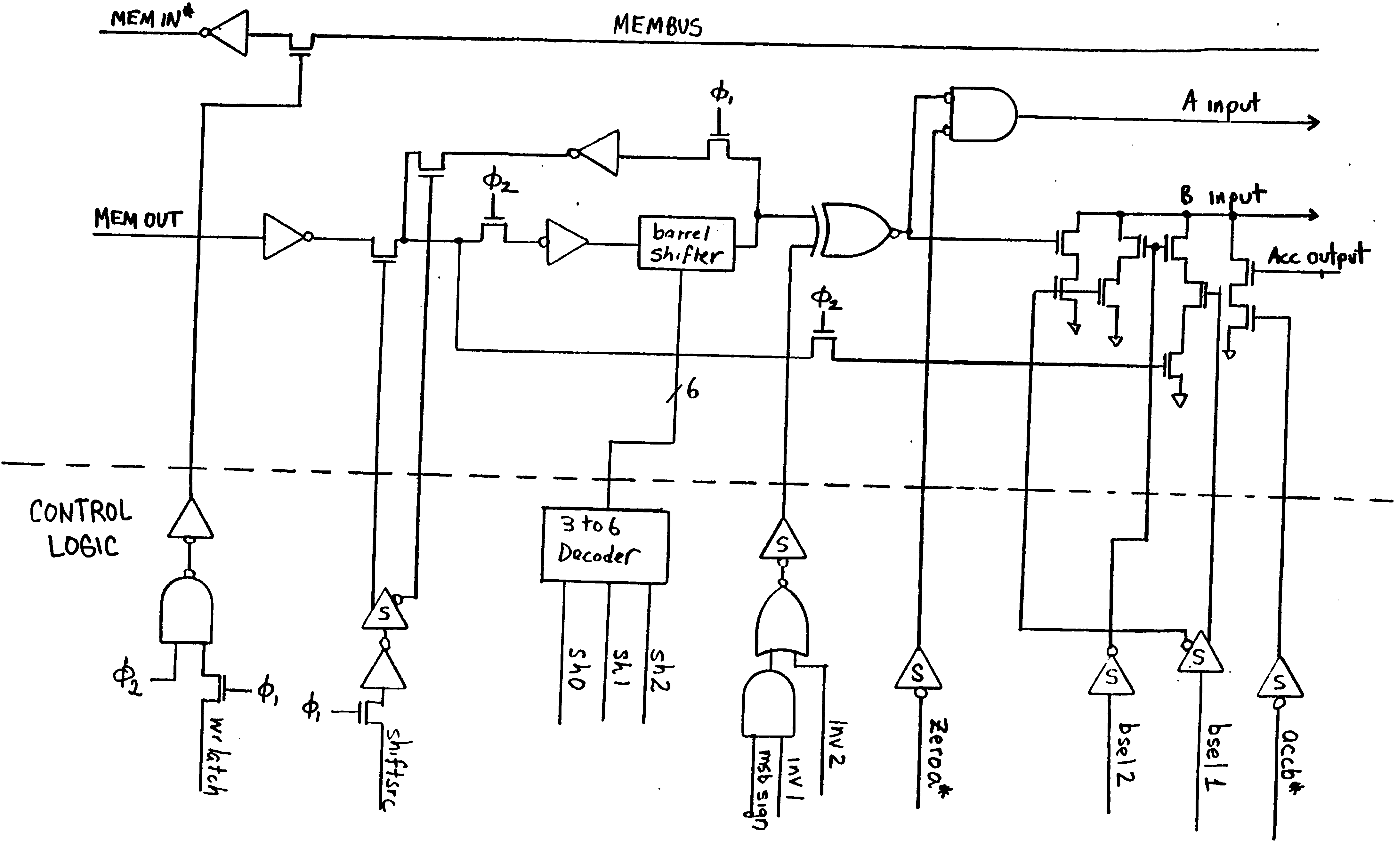
EVEN ADDER CELL (naddce)



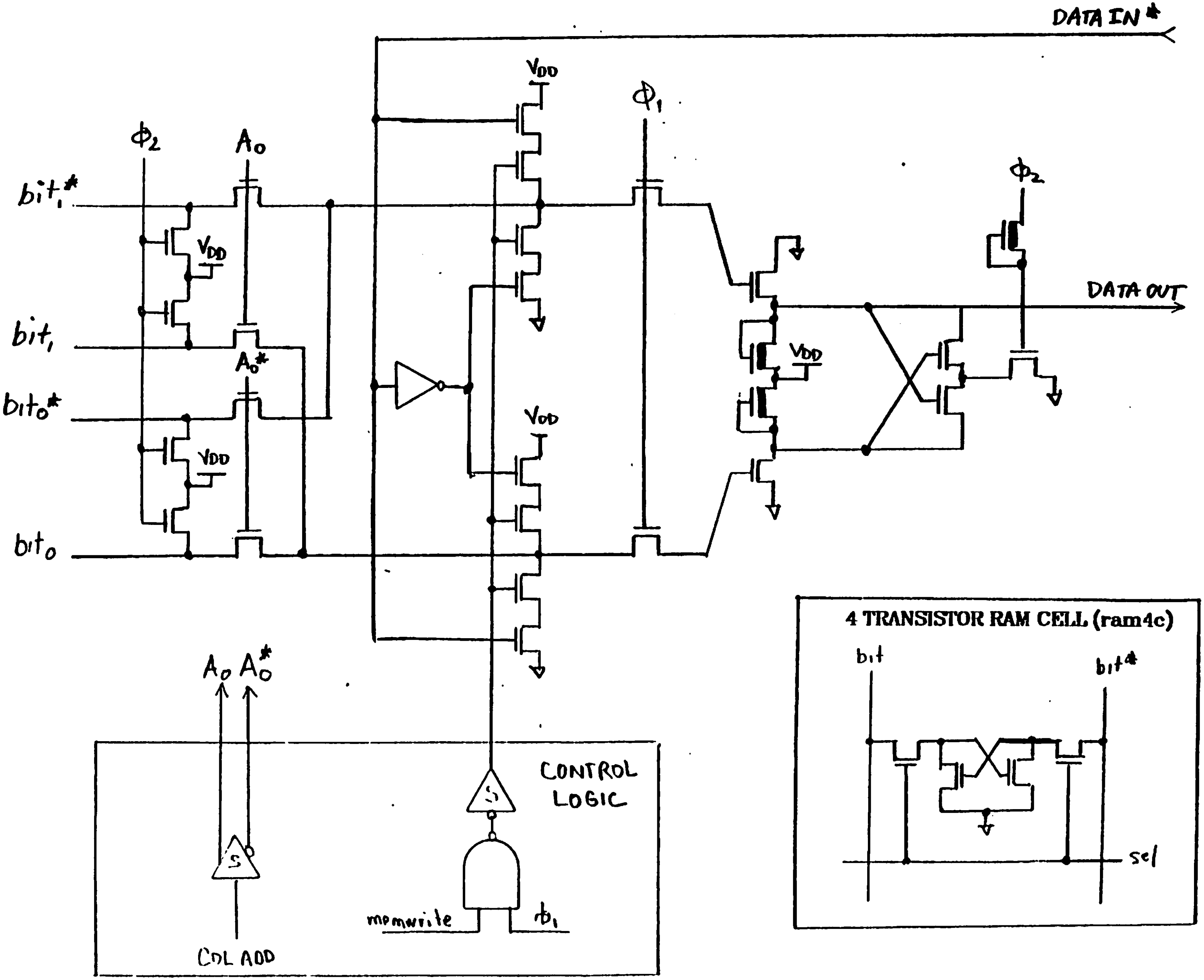
ODD ADDER CELL (naddco)

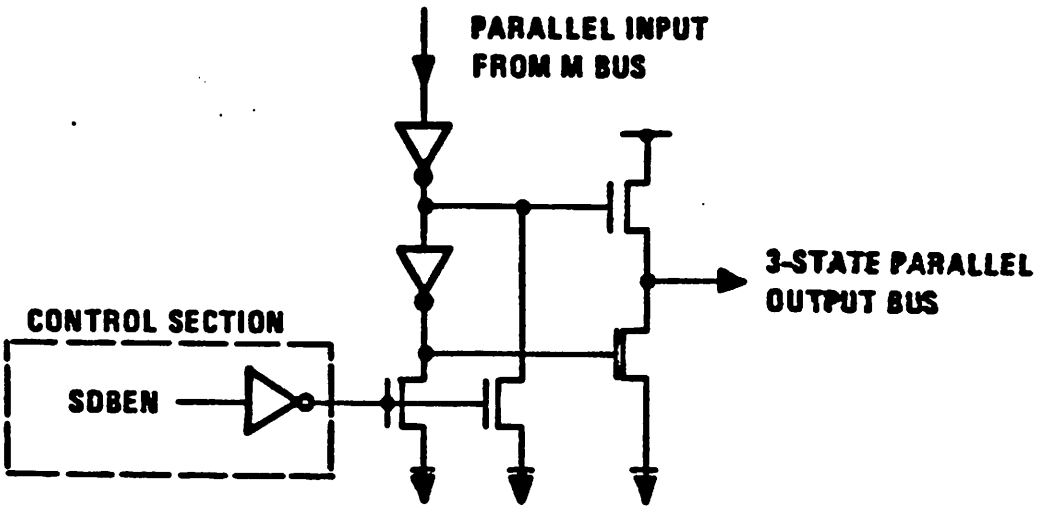


SATURATION LOGIC

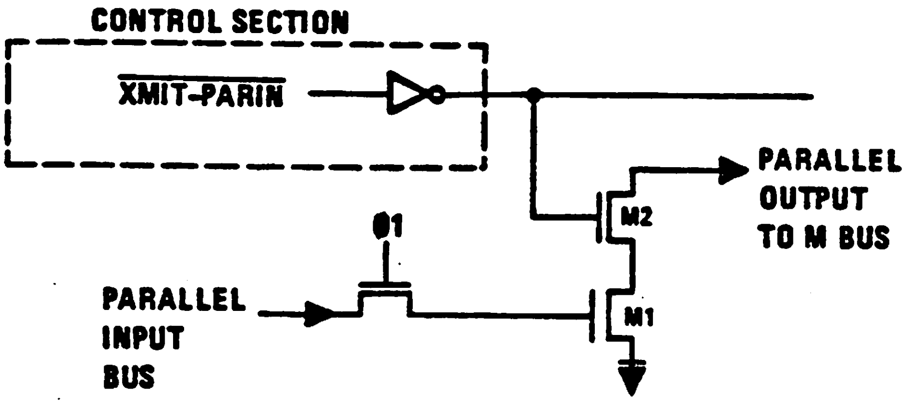


DATA PATH BETWEEN SENSE AMP AND ADDER (Bshslice6)

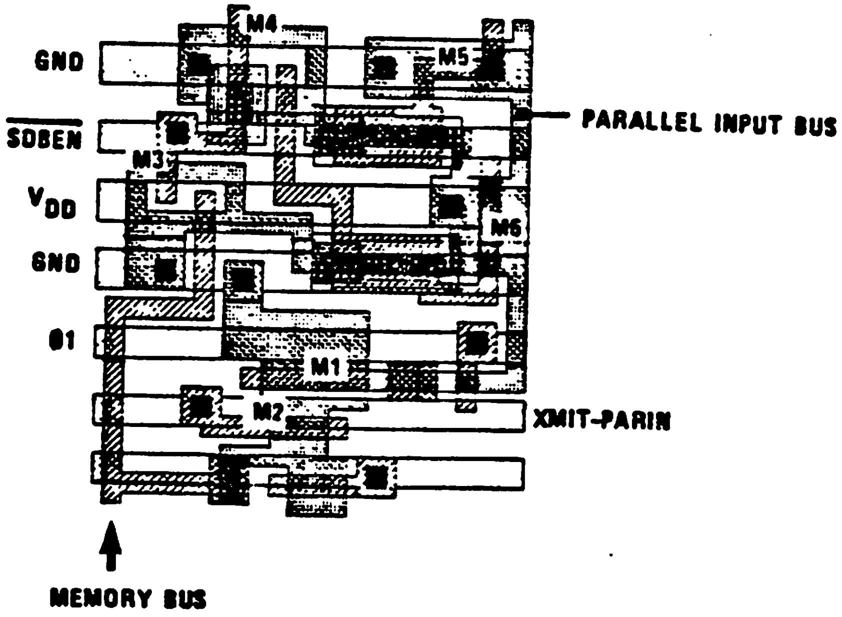

DATA IN*
V_DD
Φ1
Φ2
A0
bit1*
bit1
A0*
V_DD
bit0*
bit0
V_DD
Φ2
DATA OUT
V_DD
V_DD
A0 A0*
S
CONTROL LOGIC
S
memwrite
φ1
COL ADD
4 TRANSISTOR RAM CELL (ram4c)
bit
bit*
sel



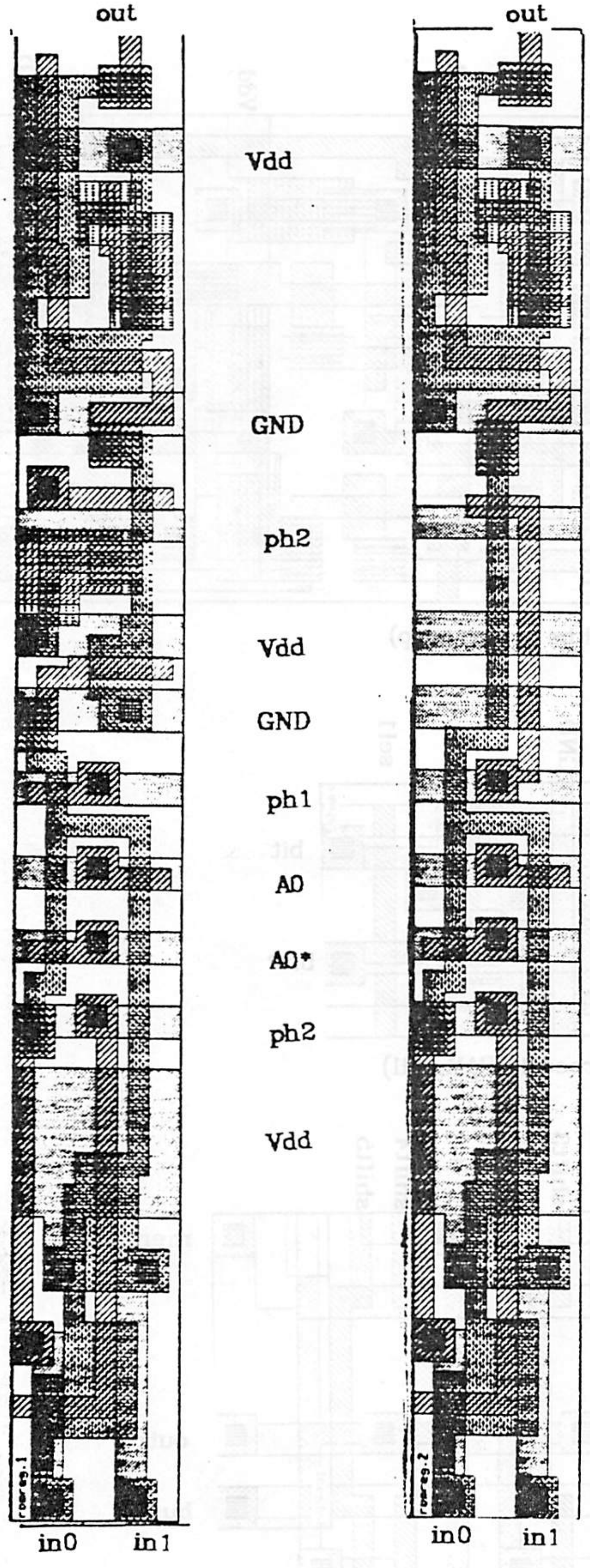
PARALLEL OUTPUT CIRCUIT



PARALLEL INPUT CIRCUIT



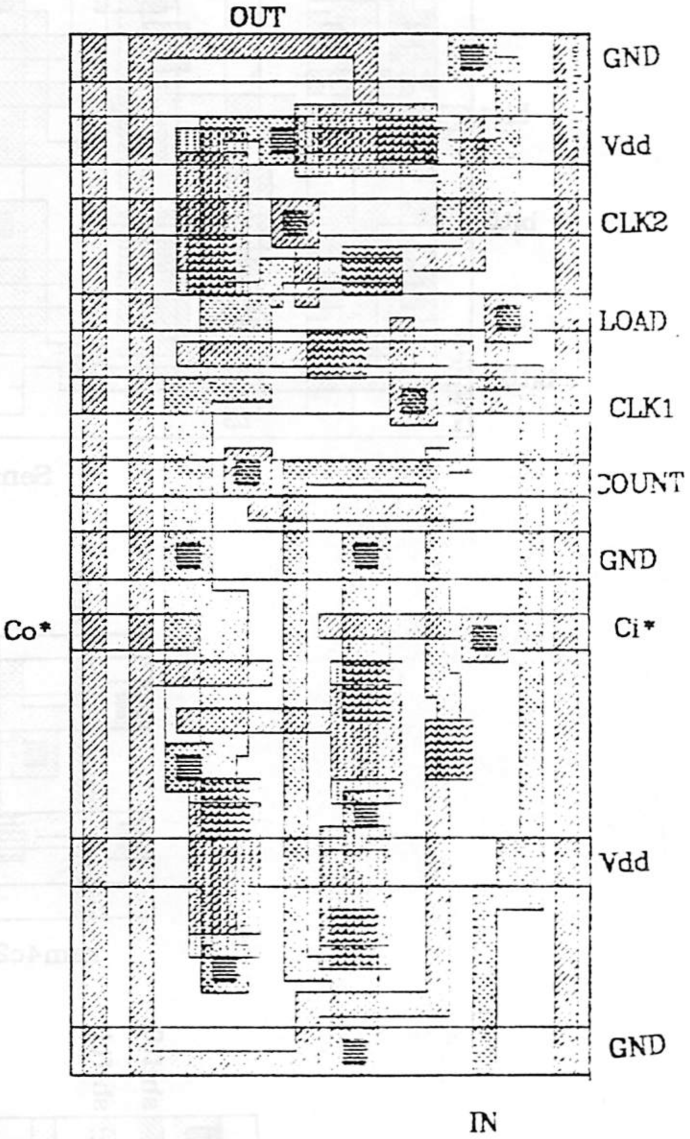
PARALLEL I/O PLOT



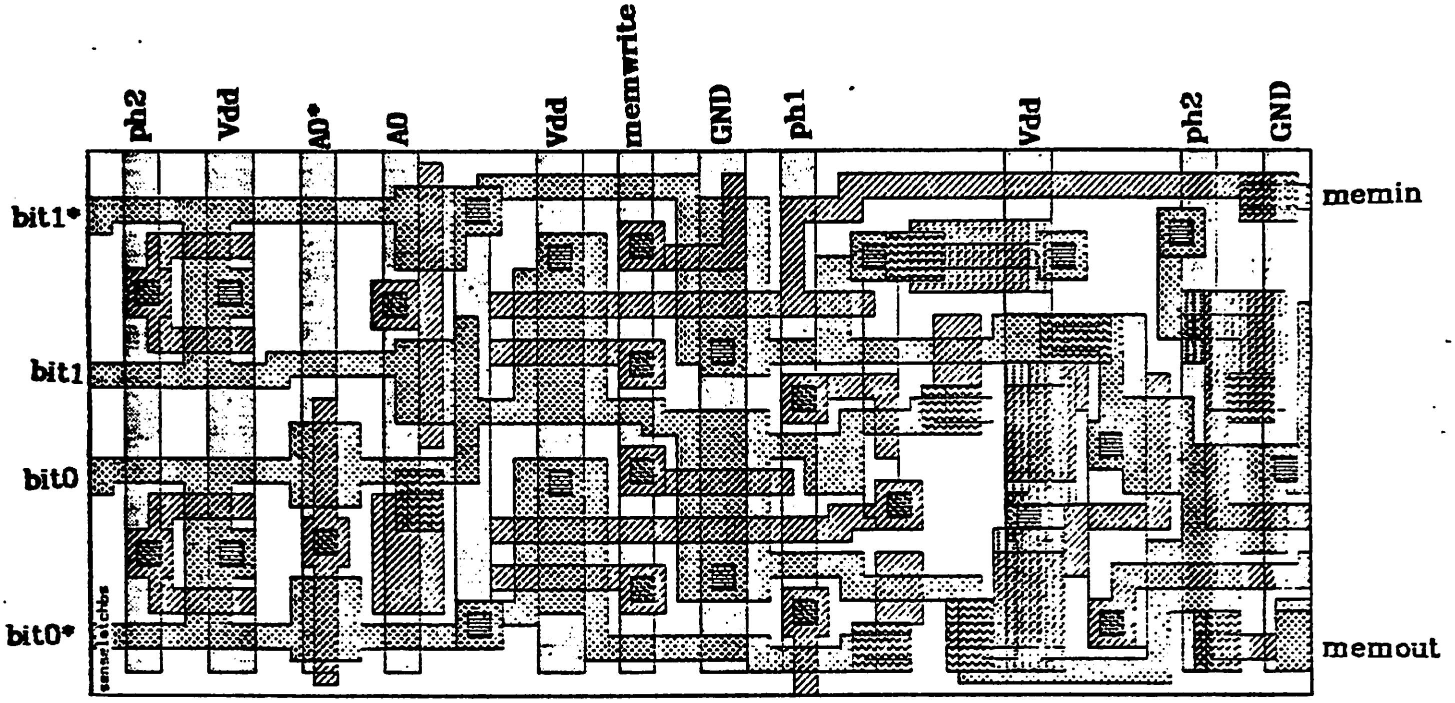
(romreg.1)

(romreg.2)

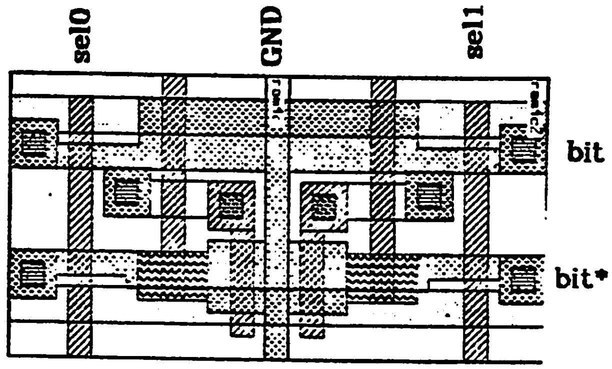
ROM OUTPUT REGISTER



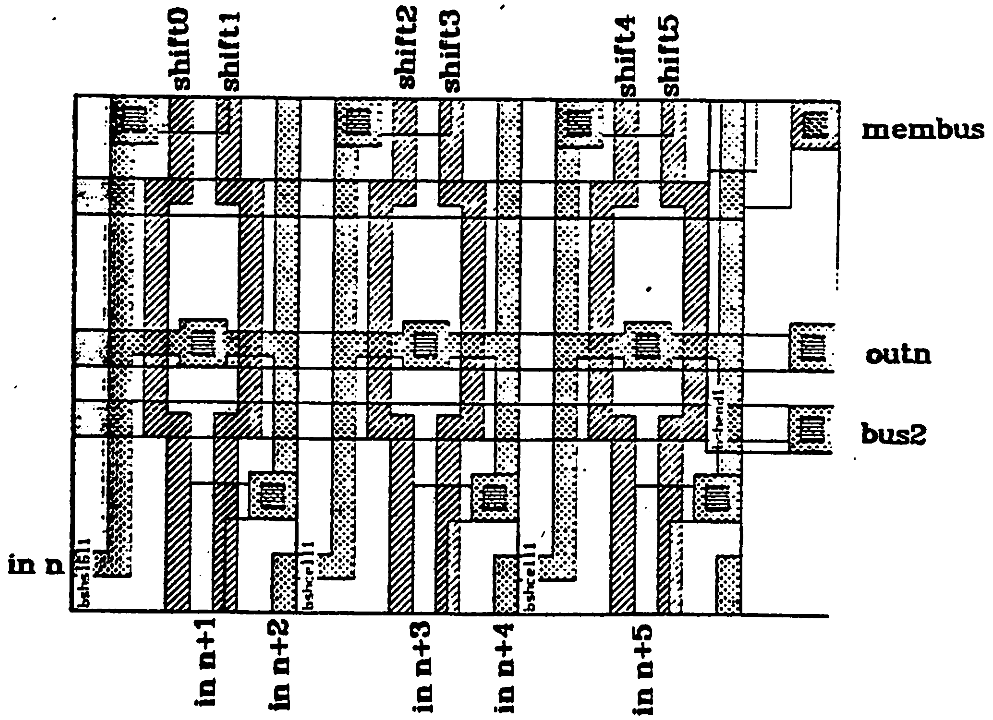
COUNTER CELL (counterslice)



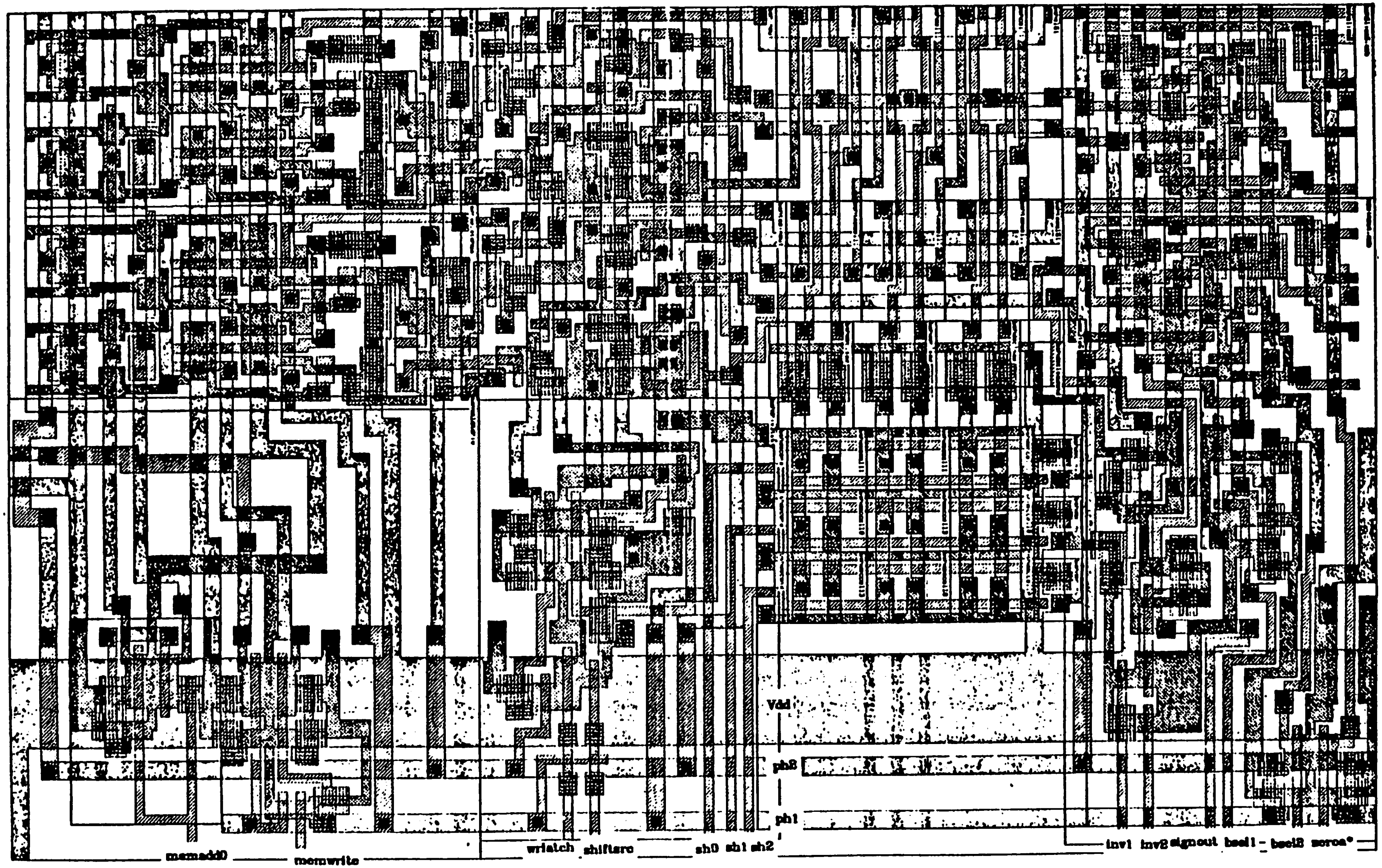
Senselatchbs (sense amp)



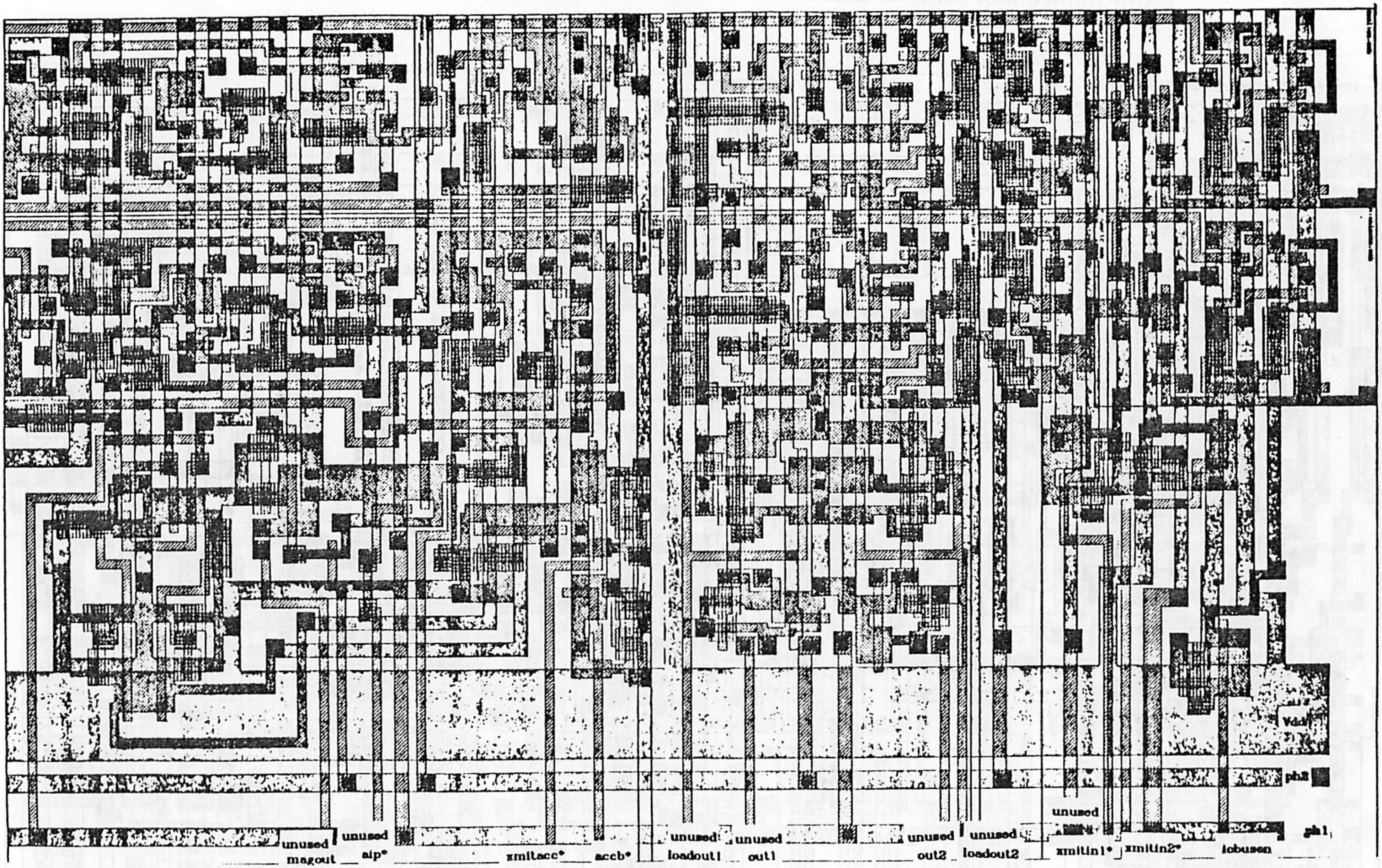
ram4c2 (mirroed 4T RAM cell)



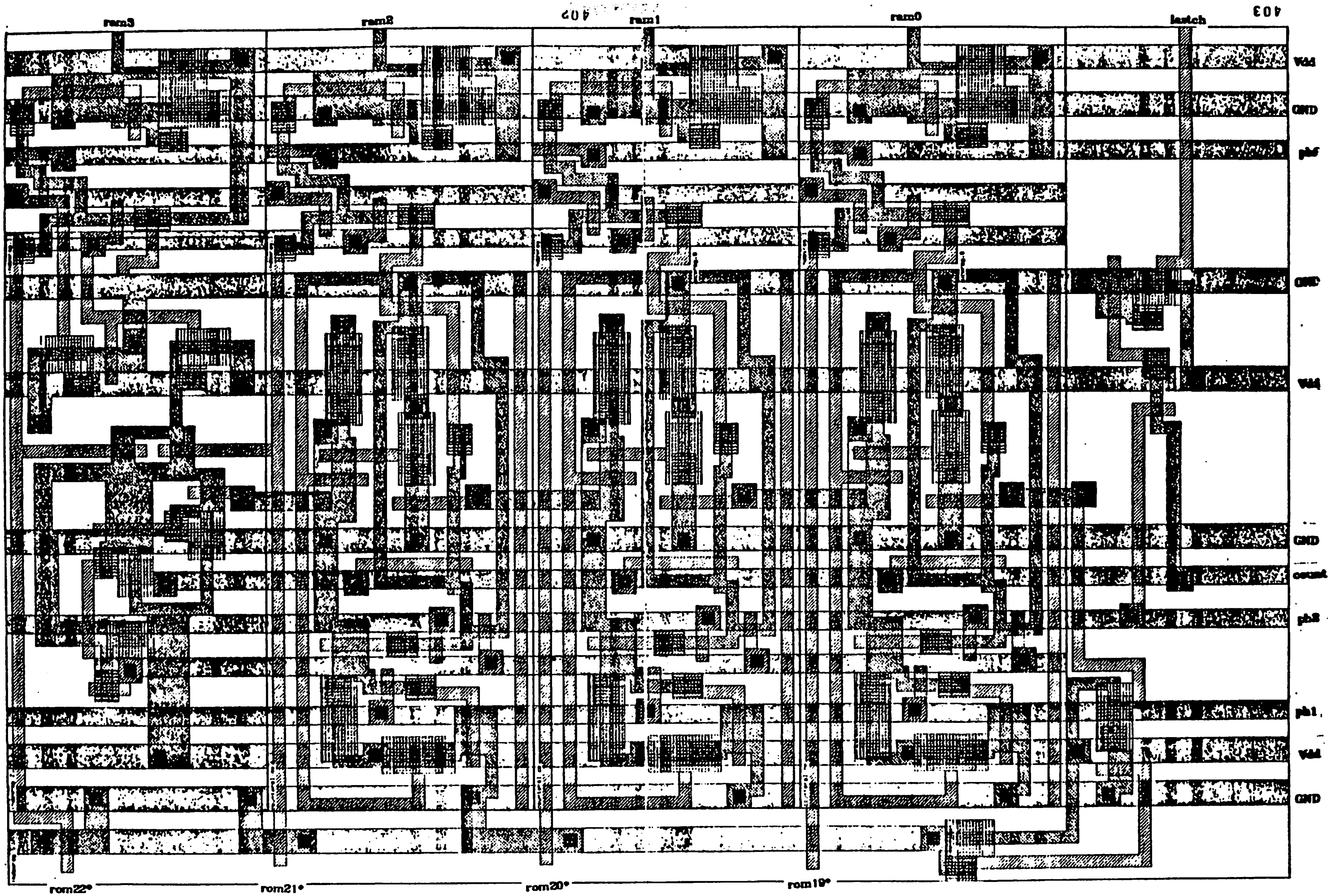
bshs16 (6 bit wide barrel shifter)



DATA PATH (SENSE AMP, BARREL SHIFTER, ADDER INPUT MUXS)



DATA PATH (ADDER, ACCUMULATOR AND I/O)



INDEX REGISTER (indexreg)

Notes:

APPENDIX G FIFO BUFFER

In order to ease testing and interfacing the filter bank chips to the outside world, a fifo chip was designed. This circuit is really a circular buffer, as writing is not inhibited when the buffer fills. See figure G1 for schematics.

The circuit is a 3 transistor memory with separate read and write ports. Writing and reading are totally independent and are controlled by shift registers that point to current row being written or read. Timing is shown in figure G2.

When the wclear signal goes high, a '0' is entered into the write shift register (reseting the write pointer). Every time wshift goes high, the data at the input lines is written into the row pointed at by the write pointer and the write pointer is advanced. The wclear line should go low only after the write pointer is shifted past the last row or multiple rows will be written.

When reading, the read pointer is automatically reset after the entire memory has been read. The pointer needs, however, to be reset upon power up by holding rclear* low for at least one cycle. Each time the rshift* line goes low, the read pointer advances and new data appears at the output lines. This data will remain until the rshift* line goes high and low again or new data is written into the row being read. The outputs are tristate and float if paden is low.

The write control circuitry was designed specifically for the filter bank. The lastch signal provides the requirements for the wclear signal. Wshift can be obtained from the exclusive-or of the two output strobes. If it is desired to inhibit writing, wclear can be pulled to ground.

For testing, two reading modes are useful. In the first all channels are read in sequence and the rshift* line is connected to datain* on the filter bank. Figure G3 shows the connection between the fifo and the filter bank chip used in this test mode. The sync* signal goes low after the last row has been read and can be used to synchronize an oscilloscope if the outputs are converted to analog for display. To look at a single channel, rshift* can be connected to a debounced switch and controlled manually.

As with any dynamic circuits, certain timing constraints must be met. The shift registers are self refreshing so that for normal clock rates (>1 KHz) there are no constraints on the rshift* timing. If the circuit is used in a fifo mode with writing inhibited, care must be taken not to exceed the refresh time before the data is read. At room temperature, tests have shown that the refresh time is greater than four seconds. The maximum clock rate for a 16 word, 12 bit memory is greater than 7.5 MHz as this is as fast as they could be tested.

FIFO PINOUT

1	GND	21	i9
2	o2 [o3]	22	i10
3	o1 [o2]	23	i11 (msb)
4	o0 [o1]	24	wclear
5	[o0]	25	
6		26	GND
7	paden	27	Vdd
8	rclear*	28	ph2
9	rshift*	29	ph1
10	sync*	30	wshift
11	i0 (lsb)	31	o11 (msb)
12	i1	32	o10
13	i2	33	o9
14	i3	34	o8
15	i4	35	o7
16		36	
17	i5	37	o6
18	i6	38	o5 [o6]
19	i7	39	o4 [o5]
20	i8	40	o3 [o4]

[] Differences for fifos with FAB ID's before M43AJQ1

FIFO CELLS

```
bufferchip (complete chip 16 word, 12 bit)
 {pads}
 bufferblock (the fifo itself)
  readriv (read pointer control)
   readend (control for shift registers)
   shiftcell1 (the actual shift register)
   readtop (other control)
  writedriv (write pointer control)
   writeend (control for shift reg)
   shiftcell1 (shift register)
   shiftcell (write select gating logic)
  outputreg (output register)
  3tgnd (gnd connection)
  memarray (the array of memory cells)
   3tcell4 (4 cells mirrored both directions)
    3tcell (single cell)
```

FIFO SIMULATIONS

```
P
P Simulation of Filter Bank FIFO  10/3/83
P
K ph2 0100 ph1 0001
w 011 010 09 08 07 06 05 04 03 02 01 00 Sync
V Wclear 10000000000000000001000000000000000000000000000000000
V Wshift 00111111111111111111001010101010101010101010101010100
V I11    11111111111111111111000000000000000000000000000000000
V I10    00000000000000000000000000000000000000000000000000000
V I9     11111111111111111111000000000000000000000000000000000
V I8     00000000000000000000000000000000000000000000000000000
V I7     11111111111111111111000000000000000000000000000000000
V I6     00000000000000000000000000000000000000000000000000000
V I5     00000000000000000000000000000000000000000000000000000
V I4     00000000000000000000000000000000000000000000000000000
V I3     00000000000111111110000000000000000000000000000000000
V I2     00000001111000011110000000000000000000000000000000000
V I1     00000110011001100110000000000000000000000000000000000
V I0     00001010101010101010000000000000000000000000000000000
V Rclear 10111111111111111111111111111111111111111111111111111
V Rshift 11111111111111111110101010101010101010101010101010101
P
P write and read
P
R
```

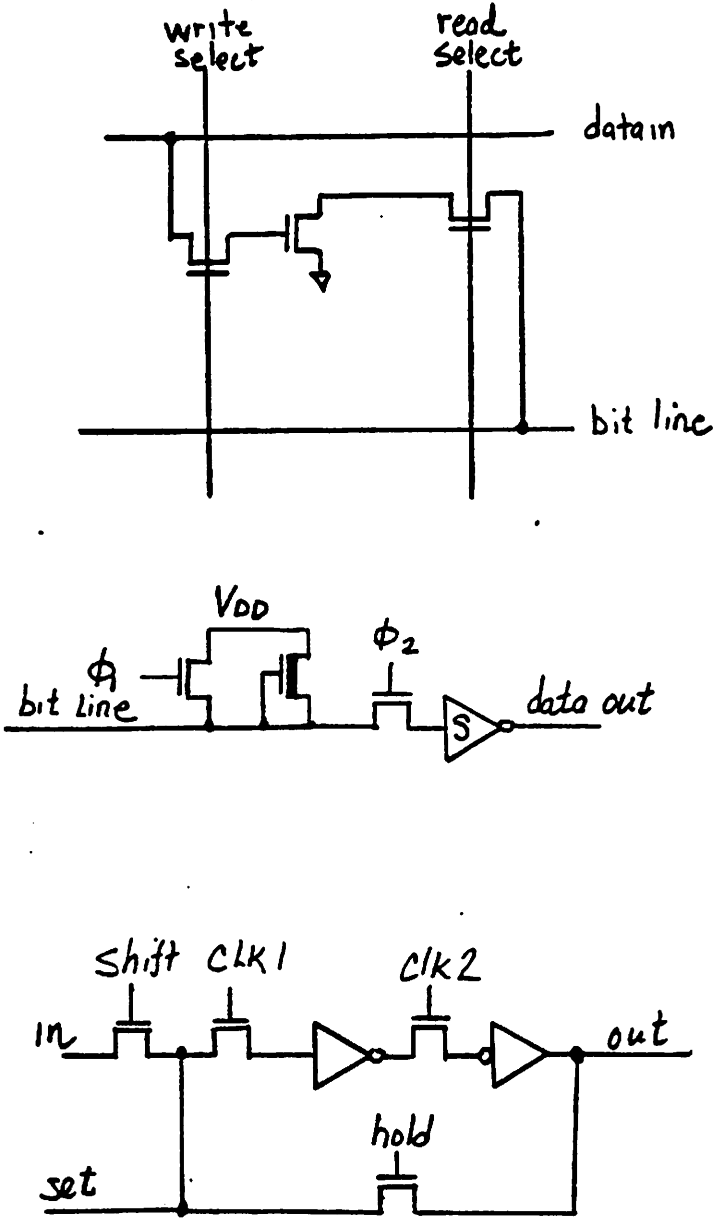
SIMULATION OUTPUT

905 transistors, 657 nodes (130 pulled up)

Simulation of Filter Bank FIFO 10/3/83

write and read

```
>XXD111111111111111111101010101010101010101010101010:011
>XXD000000000000000000000000000000000000000000000000:010
>XXD111111111111111111101010101010101010101010101010:09
>XXD000000000000000000000000000000000000000000000000:08
>XXD111111111111111111101010101010101010101010101010:07
>XXD000000000000000000000000000000000000000000000000:06
>XXD000000000000000000000000000000000000000000000000:05
>XXD000000000000000000000000000000000000000000000000:04
>XXD000000000000000000000000000000000010101010101010:03
>XXD000000000000000000000000001010101000000000101010:02
>XXD000000000000000000000010100000101000001010000010:01
>XXD000000000000000000001000100010001000100010001000:00
>XD1111111111111111111111111111111111111111111111001:Sync
```

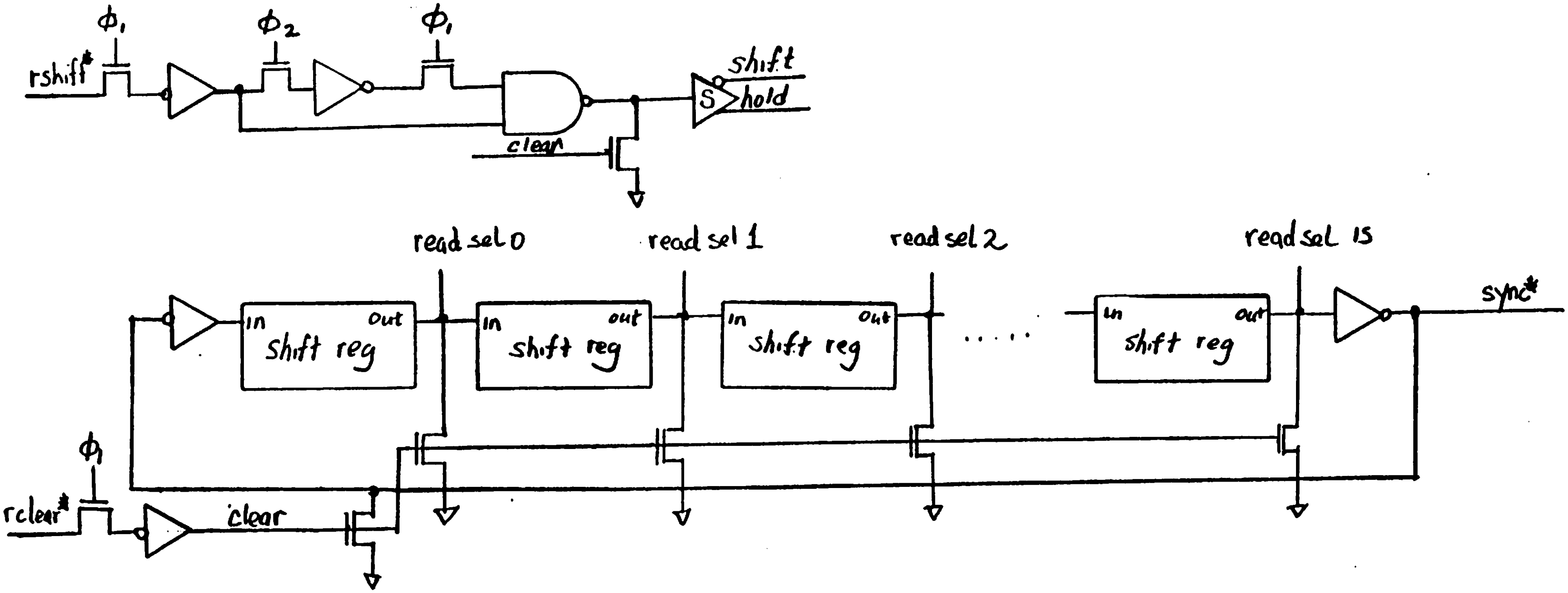


Stcell (3 transistor RAM cell)

outputreg (output register)

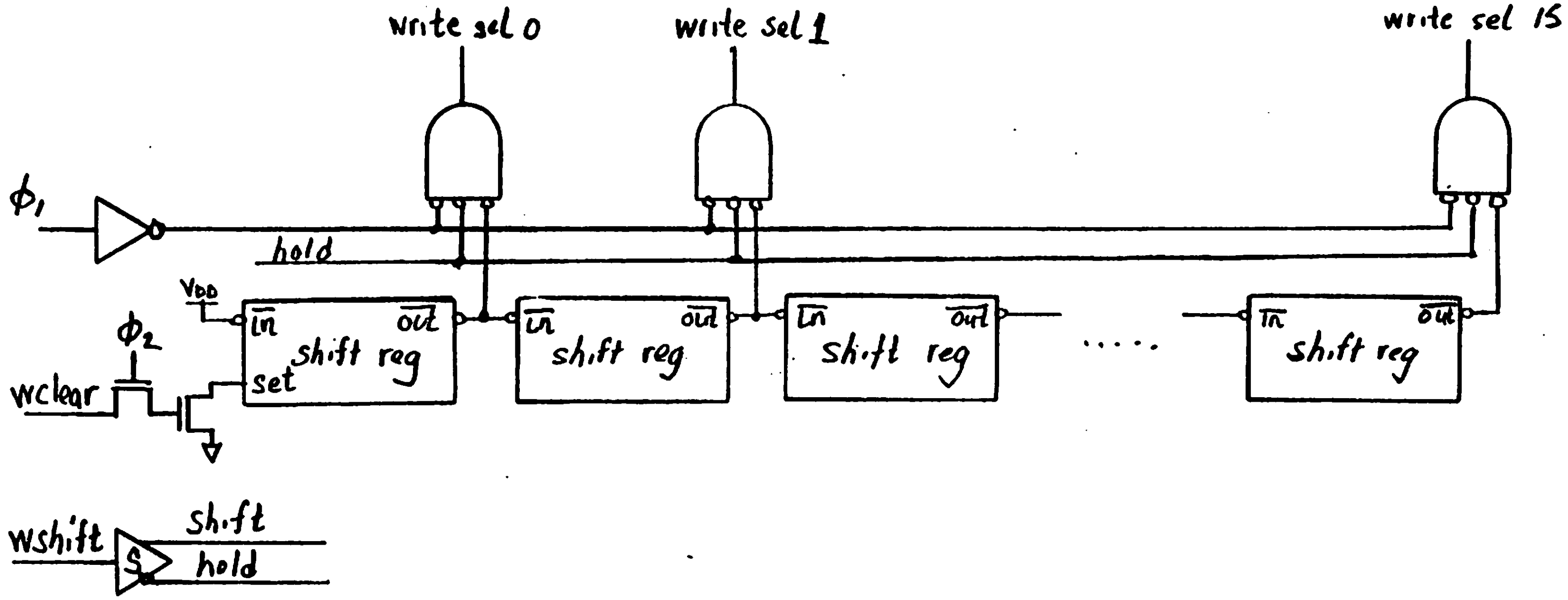
shiftcell1 (shift register)

Figure G1 FIFO Circuits



readriv (clk1=ph2, clk2=ph1)

Figure G1 FIFO Circuits



writedriv (clk1=ph1, clk2=ph2)

Figure G1 FIFO Circuits

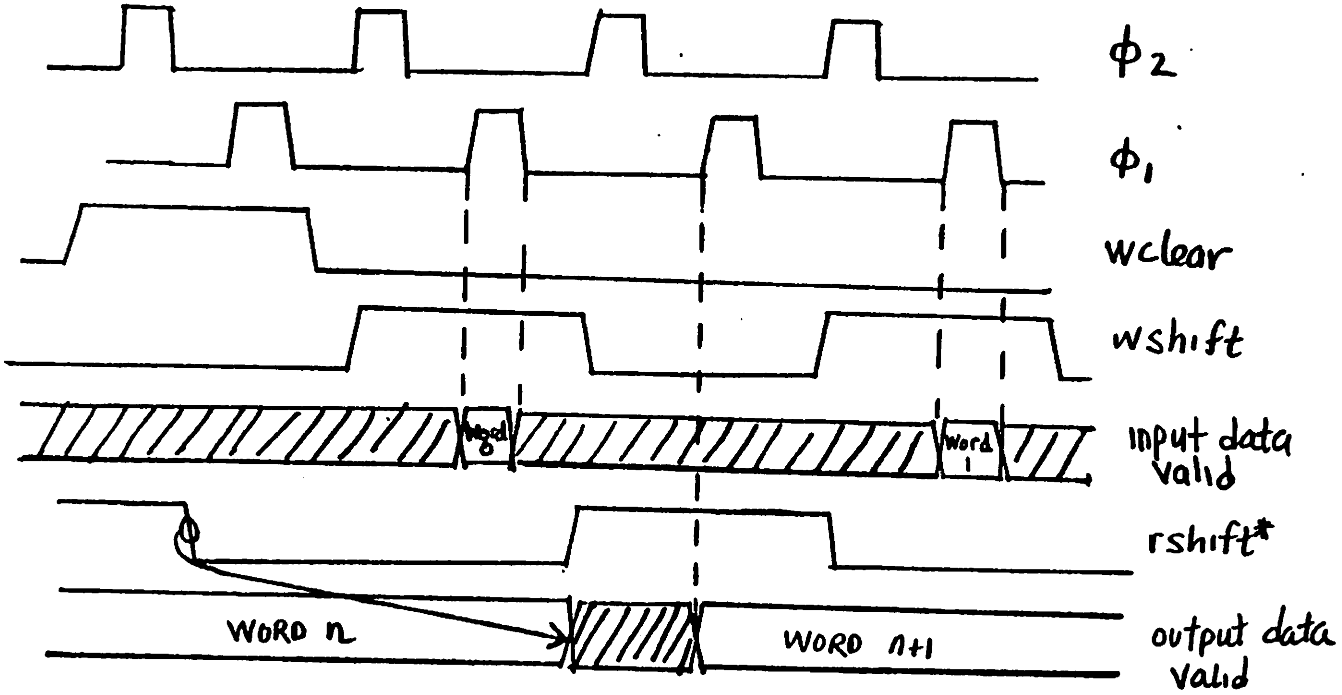


Figure G2 FIFO Timing

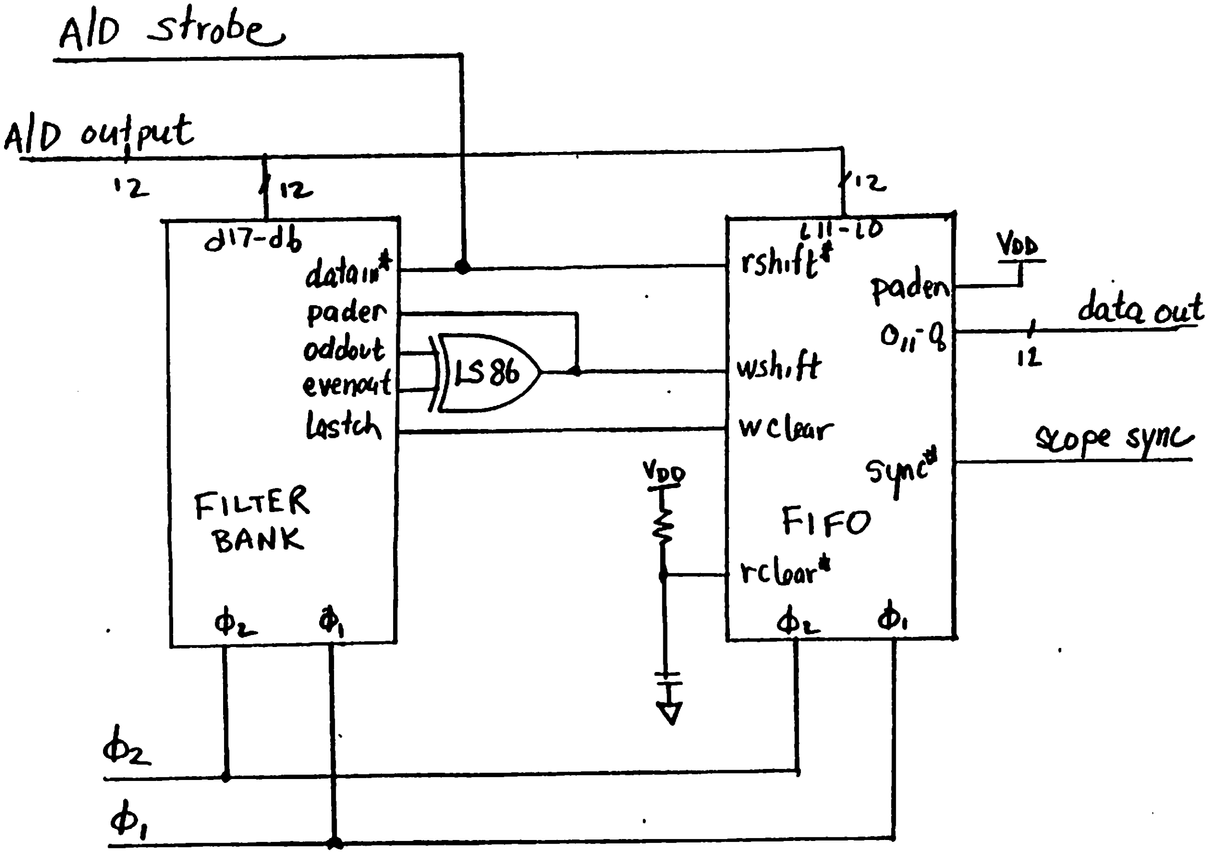
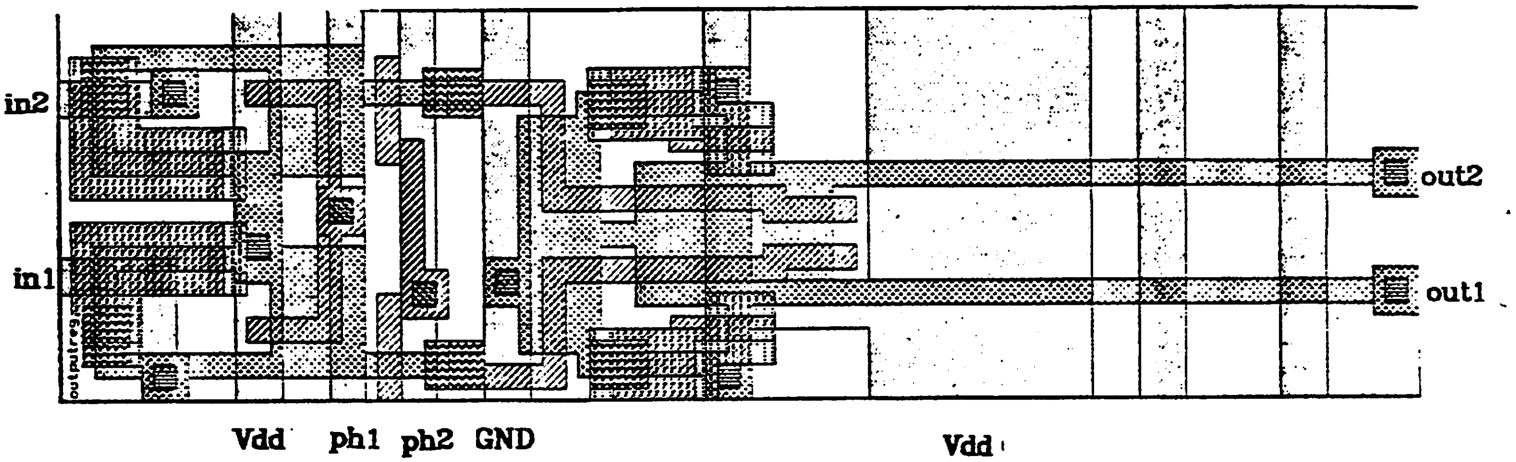
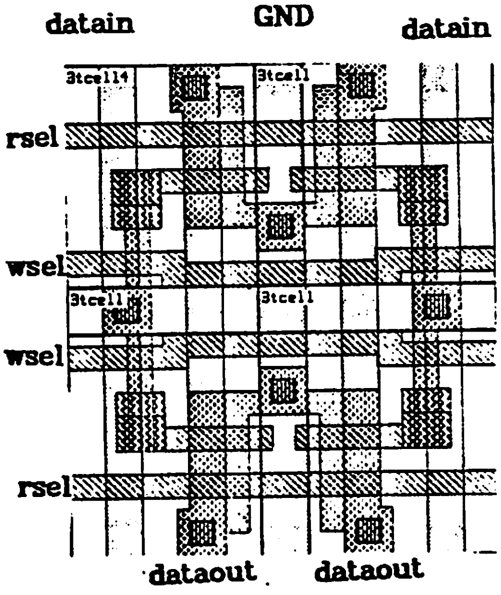


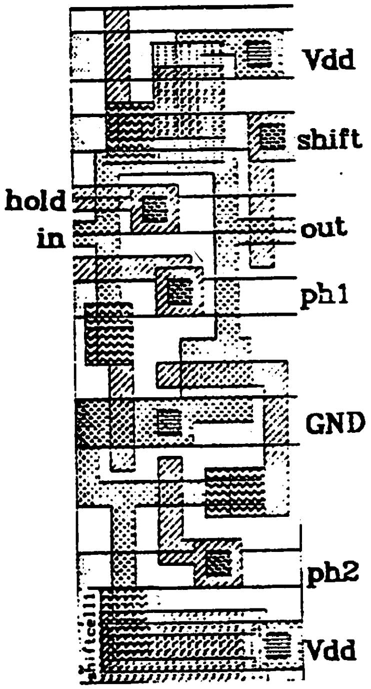
Figure G3 FIFO, Filter Bank Interface



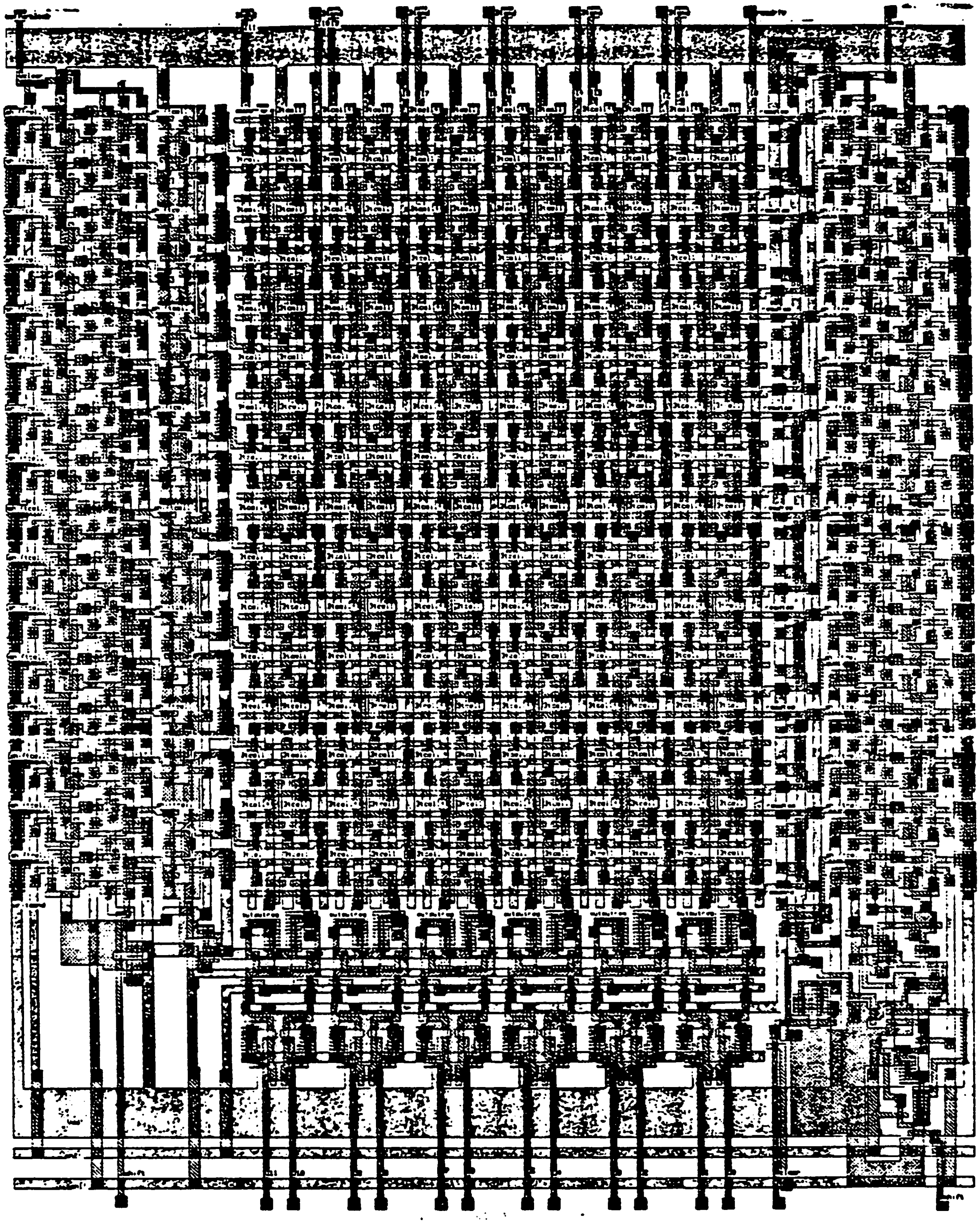
outputreg



3tcell4



shiftcell1



COMPLETE FIFO WITHOUT PADS

Notes:

APPENDIX H. SPICE SIMULATIONS

The following SPICE level 2 parameters were derived from measured curve tracer data. Simulations using these parameters gave very good estimates of the actual propagation delays. It should be noted that for the MOSIS 4 micron NMOS runs, different vendors provided circuits with vastly different characteristics. Propagation delays varied by almost a factor of two between different processes.

parameter	SPICE II	SPICE II	SPICE II	SPICE II	units
	slow	slow	fast	fast	
	ENHANC	DEPL	ENHANC	DEP	
vto	0.6	-2.5	0.48	-2.9	V
cjo	1.3e-4	1.6e-4	1.5e-4	1.5e-4	F/m^2
gamma	0.4	0.5	0.5	0.56	V^.5
lambda	0.01	0.015	0.02	0.025	1/V
vmax	4.0e4	3.0e4	4.4e4	3.3e4	m/sec
ucrit	2.6e5	2.5e5	**	**	V/cm
uexp	0.23	0.23	**	**	
uo	350	366	550	590	cm^2/V/sec
kp	17.2	18.0	25	27	uA/V^2

** not measured because it has such a small effect

Other parameters used:
gate tox: 750 A
poly tox: 7000 A
metal tox: 14000 A
ld: .5 u
cjsw: 3.5e-10 F/m

It was found that using only the level 1 parameters still gave good results with a decrease in simulation time.

The main circuits simulated were those that were added for the filter bank. This includes the critical path through the data path up to the adder, the column decoded ROM and the fifo. The half of the data path passed the adder input multiplexors was already designed and simulated. The RAM array and sense amplifiers were also simulated previously.

All spice simulations used parameters between the fast and slow values.

COLUMN DECODED ROM SIMULATIONS

192 word by 22 bit by 2 processor ROM

delay from PC change to decoder line low (2.5 V): 50 nSec
delay from PC change to bit line low (2.5 V): 93 nSec
delay from PC change to bit line low (0.5 V): 112 nSec

delay from PC to row deselect (2.5 V) : 23 nSec
delay from PC to row deselect (0.5 V) : 43 nSec

CRITICAL PATH SIMULATION OF DATA PATH FROM MEMORY OUTPUT TO ADDER INPUT

22 bit wide data path, 6 bit barrel shifter, K=4
delay from ROM to A input pulldown (2.5 V) : 185 nSec
delay from ROM to A input pullup (2.5 V) : 130 nSec
delay from ROM to barrel shifter select pullup (2.5 V) : 110 nSec
delay froM ROM to barrel shifter select pulldown (2.5 V) : 30 nSec

22 bit wide data path, 6 bit barrel shifter, K=3
delay from ROM to A input pulldown (2.5 V) : 142 nSec
delay from ROM to A input pullup (2.5 V) : 110 nSec
delay from ROM to barrel shifter select pullup (2.5 V) : 75 nSec
delay from ROM to barrel shifter select pulldown (2.5 V) : 30 nSec

FIFO SIMULATION 16 word by 12 bit

read select pullup time from ph1 (2.5 V): 34 nSec
output pullup time from ph2 (2.5 V): 19 nSec
bit line pulldown time from ph1 (2.5 V): 31 nSec
storage node pullup time from ph1 (2.5 V): 18 nSec

SIMULATION OF INVERTER AND SUPER BUFFER PERFORMANCE

It is often possible to obtain quick estimates of the performance of simple circuits using only a few benchmarks: Many paths in the circuit can be reduced to a buffer driving a load capacitance, so that several different kinds of buffers driving a capacitive load have been simulated. Although each buffer is simulated with a single W/L ratio, load capacitance and Kp, changes in these parameters will result in a simple scaling of the delay time. This is because:

$$td= (Cload)*(dV)/(Icharge)$$
$$= Cload*dV/(Kp*W/L*f(Vds,Vgs,Vt))$$

Therefore, one can use the value in the table and scale it for the appropriate circuit.

All simulations were done with:

Cload=2 pF
Kp= 25 uA/V^2 (enhancement and depletion)
W/L= 4u/4u (super-buffer output depletion device)
W/L= 4u/8u (super-buffer driver depletion device)
W/L= 4u/8u (inverter depletion device)
K ratio= 4 for no pass xter (npx), 8 with pass xster (wpx)

Delays for various buffer types

circuit	Tplh (2.5V)	Tphl (2.5 V)	Tplh (4.0 V)	Tphl (0.5V)
non-inv S.B. npx	22 nS	10 nS	42 nS	15 nS
non-inv S.B. wpx	25 nS	10 nS	60 nS	17 nS
inv S.B. npx	23 nS	8 nS	45 nS	12 nS
inv S.B. wpx	23 nS	9 nS	45 nS	12 nS
inverter npx	150 nS	8 nS	200 nS (3.0V)	12 nS

APPENDIX I Simulations from the layout

In order to check that the layout generator was generating correct circuits, a switch level simulator (MOSSIM) was employed. Before simulation, the circuit must be extracted with MEXTRA.

The circuit was checked in blocks. The controller was simulated to check that the ROM output the correct data each cycle. The index register action was verified separately. Each data path was checked for proper arithmetic operation and connection to the controller and pads. The RAM can not be simulated because is uses ratioed enhancement devices.

Due to strange things in MOSSIM, all enhancement devices with gates tied to either supply must be changed so that the gates are tied to user defined signals. The signals are then set high or low in the input file.

Simulation files for the controller and the even channel processor are included. These simulations were done on the speech recognition filter bank.

CONTROLLER SIMULATION INPUT FILE

```
P
P  filter bank controller simulation (pc, decoder, rom, index regs)
P  filtercd16.oct 10/7/83  with column decode,no clear, correct index reg
P
h t2
l t1
w omem5 omem4 omem3 omem2 omem1 omem0
w omemwrite owrlatch oshiftsrc osh2 osh1 osh0 oinv1 oinv2
w obsel1 obsel2 ozeroa oxmitacc oaccb oxmitin2 oiobusen
w emem5 emem4 emem3 emem2 emem1 emem0
w ememwrite ewrlatch eshiftsrc esh2 esh1 esh0 einv1 einv2
w ebsel1 ebsel2 ezeroa exmitacc eaccb exmitin2 eiobusen
w lastch evenout oddout datain
K ph2 0100 ph1 0001
l cout  cout1
P
P clear counter
P
R 6
x cout
P
P clear pc, index registers
P
R 1
P
P let pc run
P
R  64
P
x cout1
R  64
P
R  64
P
R 1
P
P test index reg
P
l cout
R 10
```

CONTROLLER SIMULATION OUTPUT

```
20460 transistors, 8438 nodes (1844 pulled up)

filter bank controller simulation (pc, decoder, rom, index regs)
filtercd16.oct 10/7/83 with column decode,no clear, correct index reg


clear counter

>XXXXXX:omem5
>XXXXXX:omem4
>XXXXXX:omem3
>XXX000:omem2
>XXX000:omem1
>XXX000:omem0
>XXX000:omemwrite
>XXX000:owrlatch
>XXX111:oshiftsrc
>XXX111:osh2
>XXX000:osh1
>XXX111:osh0
>XXX000:oinv1
>XXX000:oinv2
>XXX000:obsel1
>XXX000:obsel2
>XXX000:ozeroa
>XXX000:oxmitacc
>XXX111:oaccb
>XXX111:oxmitin2
>XXX000:oiobusen
>XXXX11:emem5
>XXXX11:emem4
>XXXX11:emem3
>XXX000:emem2
>XXX000:emem1
>XXX000:emem0
>XXX000:ememwrite
>XXX000:ewrlatch
>XXX111:eshiftsrc
>XXX111:esh2
>XXX000:esh1
>XXX111:esh0
>XXX000:einv1
>XXX000:einv2
>XXX000:ebsel1
>XXX000:ebsel2
>XXX000:ezeroa
>XXX000:exmitacc
>XXX111:eaccb
>XXX111:exmitin2
>XXX000:eiobusen
>X11111:lastch
```

```
>XXX000:evenout
>XXX000:oddout
>XXX111:datain
```

clear pc, index registers

```
>X:omem5
>X:omem4
>X:omem3
>0:omem2
>0:omem1
>0:omem0
>0:omemwrite
>0:owrlatch
>1:oshiftsrc
>1:osh2
>0:osh1
>1:osh0
>0:oinv1
>0:oinv2
>0:obsel1
>0:obsel2
>0:ozeroa
>0:oxmitacc
>1:oaccb
>1:oxmitin2
>0:oiobusen
>1:ememS
>1:emem4
>1:emem3
>0:emem2
>0:emem1
>0:emem0
>0:ememwrite
>0:ewrlatch
>1:eshiftsrc
>1:esh2
>0:esh1
>1:esh0
>0:einv1
>0:einv2
>0:ebsel1
>0:ebsel2
>0:ezeroa
>0:exmitacc
>1:eaccb
>1:exmitin2
>0:eiobusen
>1:lastch
>0:evenout
>0:oddout
>1:datain
```

let pc run

```
>X000000000000000000000000000000000000000000000000000000000000000:omem5
>X000000000000000000000000000000000000001111011001111110110011101:omem4
>X000010000000001111001101111110110011000000000000000000000100110:omem3
>0110101111011001111001101111110110011001011001001011110110011001:omem2
>0011110101100100000000000011110110011100100010000110110010001101:omem1
>0111011100110101011000101010110010001100100010000101000100010101:omem0
>0001010101011000001000100100010010010010001001000100010010010010:omemwrite
>0100110101010000010001001000100100100100010010001000100100100101:owrlatch
>1000010101010010000110010100001001010010000100110100001001010010:oshiftsrc
>1110011111111110111001100111110010000001110011101111110011000011:osh2
>0001100000000010000100110000001011011111000000001000000100101100:osh1
>1101111111111001110101011111101100100111111111001111110111011111:osh0
>0000000000000000000000000000000000000100000000000000000000000100:oinv1
>0010000000000001000111000000001100001001000110000000010000001000:oinv2
>0001100000000001110111011001101101111011101101110111011011111100:obsel1
>0001100000000001110111011001101101101101110110111011101101101100:obsel2
>0011110101010011111111111011111111101111111111111111111111101101:ozeroa
>0110000000000000000000000000000000000000000000000000000000000000:oxmitacc
>1110011111111110001000100110010010010010001001000100010010010011:oaccb
>1001111111111111111111111111111111111111111111111111111111111111:oxmitin2
>0000000000000000000000000000000000000000000000000000000000000000:oiobusen
>1000000000000000000000000000000000000000000000000000000000000000:emem5
>1000000000000000000000000000000000000011110011011111111000110011:emem4
>1000010000000001111011011111101100110000000000000000000001000110:emem3
>0110101111011001111011011111101100110010110001010111111000110001:emem2
>0011110101100100000000000111101100111001000010001101101000011011:emem1
>0111011100110101011001010101100100011001000010001010010000101011:emem0
>0001010101011000001001001000100100100100010001001000101000100100:ememwrite
>0100110101010000010010010001001001001000100010010001010001001000:ewrlatch
>1000010101010010000100101000010010100100001100101000000110100100:eshiftsrc
>1110011111111110111001011111100100000011101011011111101110000001:esh2
>0001100000000010000000000000010110111100000100100000000010111110:esh1
>1101111111111001110110011111101000101011100110011111111001010001:esh0
>0000000000000000000000000000000000000100000000000000000000001000:einv1
>0010000000000010001000000000110010100100011100000000100000010010:einv2
>0001100000000011101101100110110111110111011101101110101111111011:ebsel1
>0001100000000001110110110011011011011011101110110111010111011011:ebsel2
>0011110101010011111111111011111111101111111111111111111111011111:ezeroa
>0110000000000000000000000000000000000000000000000000000000000000:exmitacc
>1110011111111110001001001100100100100100010001001000101000100100:eaccb
>1001111111111111111111111111111111111111111111111111111111111111:exmitin2
>0000000000000000000000000000000000000000000000000000000000000000:eiobusen
>1111111111111111111111111111111111111111111111111111111111111111:lastch
>0000000000000000000000000000000000000000000000000000000000000000:evenout
>0000000000000000000000000000000000000000000000000000000000000000:oddout
>1001111111111111111111111111111111111111111111111111111111111111:datain

>0000000000000000000000001010010011101001001110011111111101001110:omem5
>0010001110100111111100100010000110001000000000000000000000000000:omem4
>0000101110100101110100101010000110001000010100000001110101101110:omem3
>0010001010100001110100100110000101110100100011001100010101001110:omem2
>0010100100000100010100101011001100010000010111001101100000101010:omem1
>0010101100000101110000001011001100001001101010010011100000101100:omem0
>0010000100100001001000100010001001001000100010001001001001000100:omemwrite
```

```
>0000001010000001001010000001010000100101000000101000010010100001010:owrlatch
>1101010101011001001011010010110001001011001010110001001010010101:oshiftsrc
>10010011110101000011100010111011000011100000111000000001110000111:osh2
>001010000000010001000011100000000010100001011100011110100001111000:osh1
>1101100111101010111110010111110000111110100011100000111110100111:osh0
>000000000000000010000000000000000000010000000000000000001000000000:oinv1
>1010100001000000100010100000100000100010110100101000100010001001:oinv2
>1111111001111111111001111110011111111100111111001111111110001111000:obsel1
>1111111001111110110011111100111110110011111100111110110001111000:obsel2
>1111111011111110110111111101111110110111111101111110110111111011:ozeroa
>000000000000000000000000000000000000000000000000000000000000000000:oxmitacc
>00000001100000010011000000011000001001100000001100000010011100000111:oaccb
>11111111111111111111111111111111111111111111111111111111111111111:oxmitin2
>000000000000000000000000000000000000000000000000000000000000000000:oiobusen
>000000000000000000000000000000001010010000111010010111001011111110:emem5
>11011001111111100011111100100001000000110001000000000000000000000:emem4
>1100100101111111000011101001001010000000110001000011000000000111010:emem3
>000100001111111100001110100100011000000010111010010011001010001010:emem2
>1101100110000000000000010100100101100100100010000001111001010110000:emem1
>110110011101101000001100000000101100100100000010011101001000111000:emem0
>0100100010001010000100100010000100010000100100010010001000100100:ememwrite
>1001000100010100001001010000000101000000010010100001010000001001010:ewrlatch
>00100110100000001110100101101100101101100100101100010110100100101:eshiftsrc
>110011101111101111110001110010001110010000000111000011100010000111:esh2
>00010001000000000000001100000001100000010110010000101100011101010000:esh1
>1110111111111111110001111111011111111000110111111010011100100011111:esh0
>00000000000000000000001000000000000000000000001000000000000000000001000:einv1
>001000000000001000000010001000000000010000000010001010000101000010001:einv2
>1011011101110101111111100111111100111111111110011111100111111111000:ebsel1
>10110111011101011110110011111110011111110110011111100111111011000:ebsel2
>1111111111111111111101101111111110111111111011011111110111111111011011:ezeroa
>000000000000000000000000000000000000000000000000000000000000000000:exmitacc
>010010001000101000010011000000001100000000100110000001100000000100111:eaccb
>11111111111111111111111111111111111111111111111111111111111111111:exmitin2
>000000000000000000000000000000000000000000000000000000000000000000:eiobusen
>100000000000000000000000000000000000000000000000000000000000000000:lastch
>000000000000000000000000000000000000000000000000000000000000000000:evenout
>000000000000000000000000000000000000000000000000000000000000000000:oddout
>11111111111111111111111111111111111111111111111111111111111111111:datain

>1111111100100011100111111110010011101011111111111011001000000000000:omem5
>000001010010001110011111111001001110101111111111101100100000000000:omem4
>111110100000100000000000010000001111010111111111110110010000000000:omem3
>110111000000001110011011001001001010100011000000000000000000000000:omem2
>1001100100101010000000011101001010100001000000100010001000000000000:omem1
>010110010010100010010000101001011100001011001011001000000000000000:omem0
>1001001000100001000100100100010001001000100100010010010000000000:omemwrite
>0010010100000001010000100101000000101000010010001001001000000000000:owrlatch
>00010010110101010110001001011010010101001001000010011111111111111:oshiftsrc
>01000011100100011100000000111011101110100000111110110111111111111:osh2
>101010000110111000111101000010001000101101000000000000000000000000:osh1
>00001111101010011101110111111001011111110011111111011011111111111:osh0
>000001000000000000000000010000000000000000000100000000000000000000000:oinv1
>0000100010011010010010010001001100010110010010001100000000000000000:oinv2
```

```
>1111110001111110001111111000111110001111111001101101100000000000:obsel1
>1110110001111110001111011000111110001111011001101101100000000000:obsel2
>1110110111111110111111011011111110111111011011111111100000000000:ozeroa
>0000000000000000000000000000000000000000000000000000000000000000:oxmitacc
>0001001110000000111000010011100000111000010011001001001111111111:oaccb
>1111111111111111111111111111111111111111111111111111111111111111:oxmitin2
>0000000000000000000000000000000000000000000000000010000000000:oiobusen
>0100011100111111101001110111111110010011100101111011111110110010:emem5
>0000000000000001010100111011111111001001110010111101111110110010:emem4
>0101011100111110100001000000000010000001111001011110111111110110010:emem3
>0100011100110111000000111011011001001001010010001100000000000000:emem2
>0001010100100110010101100000011101001010100000100000001000100010:emem1
>0001011000010110010101001010000101001011100000101100010110010000:emem0
>0100001000100100100100010010010010001000100010001000100010010010:ememwrite
>0000010100001001010000101000100101000001010000010001000100100100:ewrlatch
>1010101011000100101010010100010010110100101101001010100001001111:eshiftsrc
>0010011110000000111000011100000011101100111001100000111110110111:esh2
>1101000001111010000111100011101000010011000110010101000000000000:esh1
>0001101110000011111001011100001111100100111100010111111111011011:esh0
>0000000000000001000000000000000100000000000000000011000000000000:einv1
>0100000010110010001011000100001000100110001011100100010001100000:einv2
>1111110011111111000111100011111100011111000111111111001101101100:ebsel1
>1111110011111011000111100011101100011111000111110111001101101100:ebsel2
>1111110111111011011111110111110110111111101111111011101111111100:ezeroa
>0000000000000000000000000000000000000000000000000000000000000000:exmitacc
>0000001100000100111000011100010011100000111000001000110010010011:eaccb
>1111111111111111111111111111111111111111111111111111111111111111:exmitin2
>0000000000000000000000000000000000000000000000000000000000000010:eiobusen
>0000000000000000000000000000000000000000000000000000000000000000:lastch
>0000000000000000000000000000000000000000000000000000000000000010:evenout
>0000000000000000000000000000000000000000000000000010000000000:oddout
>1111111111111111111111111111111111111111111111111111111111111111:datain

>1:omem5
>1:omem4
>1:omem3
>0:omem2
>0:omem1
>0:omem0
>0:omemwrite
>0:owrlatch
>1:oshiftsrc
>1:osh2
>0:osh1
>1:osh0
>0:oinv1
>0:oinv2
>0:obsel1
>0:obsel2
>0:ozeroa
>0:oxmitacc
>1:oaccb
>1:oxmitin2
>0:oiobusen
```

```
>1:emem5
>1:emem4
>1:emem3
>0:emem2
>0:emem1
>0:emem0
>0:ememwrite
>0:ewrlatch
>1:eshiftsrc
>1:esh2
>0:esh1
>1:esh0
>0:einv1
>0:einv2
>0:ebsel1
>0:ebsel2
>0:ezeroa
>0:exmitacc
>1:eaccb
>1:exmitin2
>0:eiobusen
>0:lastch
>0:evenout
>0:oddout
>1:datain
```

test index reg

```
>0011100001:omem5
>0010011001:omem4
>0001010101:omem3
>1100000000:omem2
>0100000000:omem1
>1100000000:omem0
>0000000000:omemwrite
>1000000000:owrlatch
>0011111111:oshiftsrc
>1111111111:osh2
>0000000000:osh1
>1011111111:osh0
>0000000000:oinv1
>0100000000:oinv2
>0000000000:obsel1
>0000000000:obsel2
>0100000000:ozeroa
>1100000000:oxmitacc
>1111111111:oaccb
>0011111111:oxmitin2
>0000000000:oiobusen
>0011100001:emem5
>0010011001:emem4
>0001010101:emem3
>1100000000:emem2
>0100000000:emem1
```

```
>1100000000:ememO
>0000000000:ememwrite
>1000000000:ewrlatch
>0011111111:eshiftsrc
>1111111111:esh2
>0000000000:esh1
>1011111111:esh0
>0000000000:einv1
>0100000000:einv2
>0000000000:ebsel1
>0000000000:ebsel2
>0100000000:ezeroa
>1100000000:exmitacc
>1111111111:eaccb
>0011111111:exmitin2
>0000000000:eiobusen
>0000000010:lastch
>0000000000:evenout
>0000000000:oddout
>0011111111:datain
```

DATA PATH SIMULATION INPUT FILE

```
P
P filter.july proc mossim simulation 6/29/83
P filternew.july proc simulation 7/8/83
P filternew.july proc simulation 9/11/83
P Simulation of processor for even channels
P
l t1
h t2
w d17 d16 d15 d14 d13 d12 d11 d10 d9 d8 d7 d6 d5 d4 d3 d2
l emem0 emem1 emem2 emem3 emem4 emem5 emem6 eaccb esh0 esh1 esh2
l eshiftsrc ezeroa ebsel1 ebsel2 oiobusen
h d17 d15 d13 d11 d9 d7 d5 d3 einv1 einv2 exmitacc ewrlatch
l d16 d14 d12 d10 d8 d6 d4 d2 exmitin2 eiobusen paden ememwrite
K ph2 0100 ph1 0001
R 3
P
P output zero
P
h paden eiobusen ememwrite exmitin2
l exmitacc eshiftsrc ewrlatch einv1 einv2
x d17 d16 d15 d14 d13 d12 d11 d10 d9 d8 d7 d6 d5 d4 d3 d2
R 3
P
P output data
h esh0 esh2 ezeroa eaccb
l esh1
R 3
P
P output data shift 1
P
l esh0
R 3
P
P output data shift 2
l esh2
h esh0 esh1
R 3
P
P output data shift3
l esh0
R 3
P
P output data shift 4
h esh0
l esh1
R 3
P
P output data shift 5
P
l esh0
R 3
```

```
P
P recirculate shift by 1
P
h esh2
h eshiftsrc
l esh1 esh0
R 3
P
P output inverted data
P
l eshiftsrc
h einv2
R 3
P
P zero a , acc b
P
h ebsel1 ebsel2
l ezeroa eaccb
R 3
```

DATA PATH SIMULATION OUTPUT

```
20460 transistors, 8438 nodes (1844 pulled up)

filter.july proc mossim simulation 6/29/83
filternew.july proc simulation 7/8/83
filtergen16.sep proc simulation 9/11/83
Simulation of processor for even channels

>111:d17
>000:d16
>111:d15
>000:d14
>111:d13
>000:d12
>111:d11
>000:d10
>111:d9
>000:d4
>111:d3
>000:d2
>111:d1
>000:d0

output zero

>111:d17
>111:d16
>111:d15
>111:d14
>111:d13
>111:d12
>111:d11
>111:d10
>111:d9
>111:d4
>111:d3
>111:d2
>111:d1
>111:d0

output data
>111:d17
>100:d16
>111:d15
>100:d14
>111:d13
>100:d12
>111:d11
>100:d10
>111:d9
>100:d4
>111:d3
```

```
>100:d2
>111:d1
>100:d0
```

output data shift 1

```
>111:d17
>011:d16
>100:d15
>011:d14
>100:d13
>011:d12
>100:d11
>011:d10
>100:d9
>011:d4
>100:d3
>011:d2
>100:d1
>011:d0
```

output data shift 2

```
>111:d17
>111:d16
>011:d15
>100:d14
>011:d13
>100:d12
>011:d11
>100:d10
>011:d9
>100:d4
>011:d3
>100:d2
>011:d1
>100:d0
```

output data shift3

```
>111:d17
>111:d16
>111:d15
>011:d14
>100:d13
>011:d12
>100:d11
>011:d10
>100:d9
>011:d4
>100:d3
>011:d2
>100:d1
>011:d0
```

output data shift 4

```
>111:d17
>111:d16
>111:d15
>111:d14
>011:d13
>100:d12
>011:d11
>100:d10
>011:d9
>100:d4
>011:d3
>100:d2
>011:d1
>100:d0
```

output data shift 5

```
>111:d17
>111:d16
>111:d15
>111:d14
>111:d13
>011:d12
>100:d11
>011:d10
>100:d9
>011:d4
>100:d3
>011:d2
>100:d1
>011:d0
```

recirculate shift by 1

```
>111:d17
>111:d16
>101:d15
>110:d14
>101:d13
>110:d12
>001:d11
>110:d10
>001:d9
>110:d4
>001:d3
>110:d2
>001:d1
>110:d0
```

output inverted data

```
>100:d17
>100:d16
>101:d15
```

```
>100:d14
>001:d13
>110:d12
>001:d11
>110:d10
>001:d9
>110:d4
>001:d3
>110:d2
>001:d1
>110:d0

zero a , acc b

>000:d17
>000:d16
>111:d15
>000:d14
>111:d13
>000:d12
>111:d11
>000:d10
>111:d9
>000:d4
>111:d3
>000:d2
>111:d1
>000:d0
```

Notes:

APPENDIX J. KNOWN PROBLEMS AND POSSIBLE UPGRADES

The currently known 'problems' include:

Layout Generator:

The compiler will put a 'ground' line through the I/O pads if a wide data path is specified with a small memory. This problem is easily detected by checking the cifplot. Then the ground line can be corrected.

Layout Generator:

When indexing is implemented it is possible that RAM will be wasted. This occurs because the index register is very simple and RAM addresses are always sequential. By changing the layout generator to create a RAM and decoder with only used locations, RAM could be saved.

Filter Compiler:

Handles each half of the filter bank separately (when two data paths are used). This forces the user to make sure that the two programs have the same length (the programs are always made a multiple of 32 so that the address lengths must be in the same 32 word block) and that the two data paths dont try to output at the same time. The compiler could be modified fairly easily to perform these functions.

Layout Generator (non-column decoded ROM):

This report assumes that only circuits with column decoded ROMs will be used. However, there is a version of the layout generator that creates circuits with out column decoded ROMs. It however, has not been upgraded all the way. The index register still has slight timing problems that prevent it from working as fast as that in the column decoded circuit. The index register for non-column decoded circuits also requires a clear signal (active high) to synchronize the two index registers (for two data paths) once on power up. The index register for column decoded circuits could be used in the non-column decoded ones. The strobes for non-column decoded designs are active low, whereas they are active high in column decoded designs.