REPORT SSD-TR-90-11

4

Experimental High Speed/Power Ratio ASIC Designs Using Residue Numbers

Prepared by

A. SIMONEAU, J. PIZARRO, and A. PARKER
Computer Science Laboratory
Laboratory Operations
The Aerospace Corporation
El Segundo, CA 90245

15 March 1990

Prepared for

SPACE SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Base
P.O. Box 92960
Los Angeles, CA 90009-2960

This report was submitted by The Aerospace Corporation, El Segundo, CA 90245, under Contract No. F04701-88-C-0089 with the Space Systems Division, P.O. Box 92960, Los Angeles, CA 90009-2960. It was reviewed and approved for The Aerospace Corporation by A. J. Schiewe, Acting Director, Computer Sciences Laboratory.

Lt Stephen A. Way was the project officer for the Mission-Oriented Investigation and Experimentation (MOIE) program.

[illegible]

STEPHEN A. WAY, LT, USAF
MOIE Project Officer
AFSTC/WCO OL-AB

Raymond M Leong

RAYMOND M. LEONG, MAJ, USAF
MOIE Program Manager
AFSTC/WCO OL-AB

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-0086(6920-03)-2	5. MONITORING ORGANIZATION REPORT NUMBER(S) SSD-TR-90-11
6a. NAME OF PERFORMING ORGANIZATION The Aerospace Corporation Laboratory Operations 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Space Systems Division Air Force Systems Command
6c. ADDRESS (City, State, and ZIP Code) 2350 E. El Segundo Blvd. El Segundo, CA 90245	7b. ADDRESS (City, State, and ZIP Code) Los Angeles Air Force Base P.O. Box 92960 Los Angeles, CA 90009-2960
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F04701-88-C-0089
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Experimental High Speed/Power Ratio ASIC Designs Using Residue Numbers

12. PERSONAL AUTHOR(S)
Simoneau Arthur R.; Pizarro, Jorge T.; and Parker, Alice C.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
	FROM ______ TO ______	1990 March 15	27

16. SUPPLEMENTARY NOTATION
The work reported was performed during the period October 1985 through September 1987

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
ASICs; Number conversion; High-speed transforms; Power/speed tradeoffs; Low-power implementations; Residue arithmetic (cont'd)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Residue numbers have traditionally been popular for signal processing applications and have recently received renewed interest, for a number of reasons. The inherent parallelism of the residue number system enables the throughput of a system to be increased while keeping the cost reasonable. With the advent of VLSI circuits, the increased size and complexity of parallel residue number circuits may no longer be prohibitive.

A method for implementing onboard signal processing algorithms using CMOS Application Specific Integrated Circuits (ASIC) is presented in this report. This method uses residue arithmetic in a one-out-of-n representation. The representation and implementation methods provide low-power circuits that retain their high-performance capabilities.

Details of the adder and multiplier circuits are given, along with an overview of the input/output conversion logic. A specific example algorithm that performs a matrix

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. ABSTRACT (Continued)

multiplication has been chosen and implemented. Test chips of the adder demonstrate that the add function performs within our design constraints. Test chips of the algorithm for a particular residue have demonstrated that the arithmetic portions of the chip functioned as expected.

Some preliminary results are included at the end of this report with some caveats regarding the particular implementation that we chose. [illegible]

18. SUBJECT TERMS (Continued)

Signal processing
VLSI

CONTENTS

FIGURES

TABLE

PREFACE

We are grateful to Hilarie Orman, Steve Kelem and Lou Gallenson for their many contributions as a members of the development team, and to Steve Crocker for his insights and encouragement in initiating and fostering this work.

I. INTRODUCTION

The straightforward approach of performing a signal processing calculation on a sequential Von Neumann machine does not provide the data rates that are required in some advanced systems. In order to increase throughput, it is necessary to find tasks or subtasks that can be evaluated concurrently and then recombined quickly.

When the signal processing is to be done on board a satellite, there are the additional constraints that the necessary hardware has to be small, power-efficient, stable, and reliable. The power requirements in particular call for a different solution to the problem than would suffice for a stationary ground-based system. Static logic is recommended to provide radiation hardness, reducing the cost advantages of heavily pipelined bit-serial implementations.

The implementation of on-board signal processing circuits to meet speed, power, size, and reliability constraints is difficult with conventional circuits. In general, implementations that are easily partitionable onto multiple chips, which exhibit regular layouts (for area efficiency), and which use design styles purported to be high performance (*e.g.*, parallel and pipelined) are desirable. However, such implementations might still contain standard binary-encoded numbers and binary arithmetic functions. Standard binary multiplication implementations are either large or slow, and the addition and multiplication circuit performances are limited by carry propagation delays. These delays can be reduced by designing the adders with circuits such as carry-look-ahead and by using similar methods for the multipliers, but the additional circuitry increases the power consumption.

It has been known for centuries that residues (remainders) obtained by dividing a number by a set of relatively prime numbers can be used to uniquely represent numbers within a given range, as described by Szabo and Tanaka[1]. Calculations to be performed on the original number can be performed on the residues in parallel, using the algebra of rings. The results of these independent calculations can be combined to give a single binary result: the same one that the usual binary arithmetic would produce. Calculations with residue numbers are valid provided no operation yields a result that would exceed the defined range. Overflow cannot be detected as easily in residue codes as in weighted binary codes; thus, for economy, it is necessary use a large enough range to contain all possible results of the calculation. Many researchers have reported using residues for signal processing calculations (*e.g.*,[2] and[3]), and some have implemented VLSI circuits (*e.g.*,[4] and[5] .)

Using a residue representation for the numbers of interest allows computations using each of the different moduli to proceed independently and concurrently. Since each of the moduli is much smaller than the range of interest, its computations can be performed by table look-up, providing a regular structure that can be easily laid out.

Since aerospace applications often require circuits to be not only fast but also *low power*, the increased complexity of a residue number system would seem to offset any speed advantages with an increase in power consumption. However, at The Aerospace Corporation we have devised an approach to the residue number representation that exploits the speed possible with these circuits while keeping power consumption to a minimum.

The next section of this report describes our approach to implementation of on-board signal processing, followed by details of the input/output conversion, the adder implementation, the general multiplication scheme, and multiplication by a constant. Section VII describes details of the implementation. Finally, Section VIII summarizes the work and presents a comparison to binary arithmetic.

II. THE PROBLEM APPROACH

The approach that we are taking to implement on-board signal processing is to use a set of residue numbers to perform computations, with a "one-out-of-n"* representation for the various residues. The resulting circuits are highly efficient in speed and power.

A one-out-of-n coding means there is a unique line for each value to be represented. For example, representing a residue of 13 requires exactly 13 lines. This exponential increase in the number of bits used to represent a number has a payoff in reduced power consumption when we design residue arithmetic computational circuits.

The use of a single line for each number in a residue greatly simplifies the logic required to do additions and multiplications. It is no longer necessary to decode the number or examine the value of any line other than the single line that is high. This means that addition simply requires a circuit to detect which line is high in each of the addends and then table lookup or simple logic to determine which line is to be raised on the output. Our innovation is to implement these tables with circuits that are regular, fast (one gate delay), and correspondingly low-power. Our basic adder design uses a square matrix of lines, as given in Section IV. These adders require constant time for an operation, virtually independent of the size of the residue (excluding the effects of fanout on performance). This combination of residue and one-out-of-n representation for addition was previously applied to magnetic core technology, as described in Ref.[1].

Multiplication can be done with the same circuitry as addition by using logarithms. This is shown in detail in Section V. If the addition or multiplication is by a constant, then an even simpler method exists for computation: the input lines are simply rearranged to form the result. This can be hardwired, or some form of shifting network can be used to re-route in hardware. The fact that multiplication by, or addition of, a constant can be a simple wire routing problem means that operations with constants use little area and have negligible power and delay costs. Constant multiplications are also shown in Section V.

For the purposes of the study, an algorithm typical of those encountered in on-board signal processing was chosen. This algorithm computes a vector result of a set of linear equations. The algorithm will be presented in Section VI.

A fundamental decision regarding these circuits is selection of the set of moduli to be used in the design to achieve the desired precision and to minimize the area, power, and other constraints. For our example algorithm, 12 bits of accuracy on output are required. Since multiplications in the linear transform increase the dynamic range, we found we had to carry 30 bits of precision internally. We chose the residues 4, 5, 7, 9, 11, 13, 17, 19, and 23 to minimize the area. Our adder circuit area depends roughly on the square of the largest modulus, so our choice of moduli represents an optimization of minimal area and minimal active lines for the range we are using. The overall data flow for this algorithm is represented in Figure 1.

A more-detailed flow diagram of the overall scheme is shown in Figure 2, and a timing diagram is shown in

* Also known as "one-line-high."

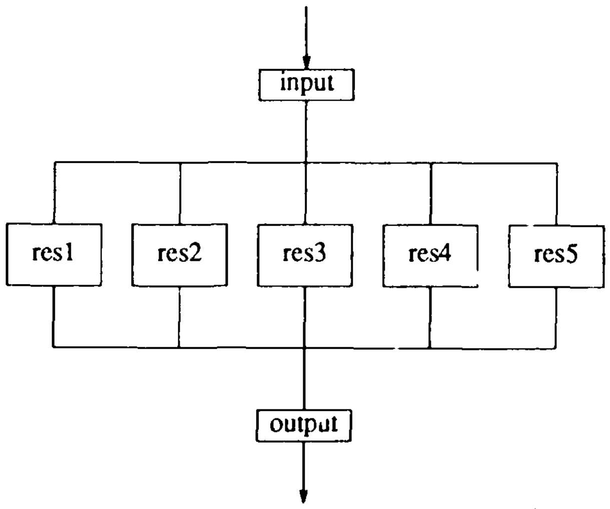


Figure 1. Overall Data Flow

Figure 3. Data inputs are provided to the chip serially, one 12-bit value per clock phase. There are nine data inputs, and r' ıe outputs. These inputs are then converted, input by input, into residue representations and latched. This conversion process is overlapped in time with the computation of the equations on previously input data and the output of a still earlier computation. The computed data are then latched and output while new data are being input. Data values are expected to arrive at the chip every 100 nanoseconds with results to be output after a 900 nanosecond delay.

The selection of an algorithm and speed requirements was somewhat arbitrary. Faster performance could certainly be gained by inputting more than a single value per 100 nanoseconds and pipelining the computations. The algorithm was written to minimize costly multiplications (as if the design were to be encoded in normal binary form) in favor of additions.

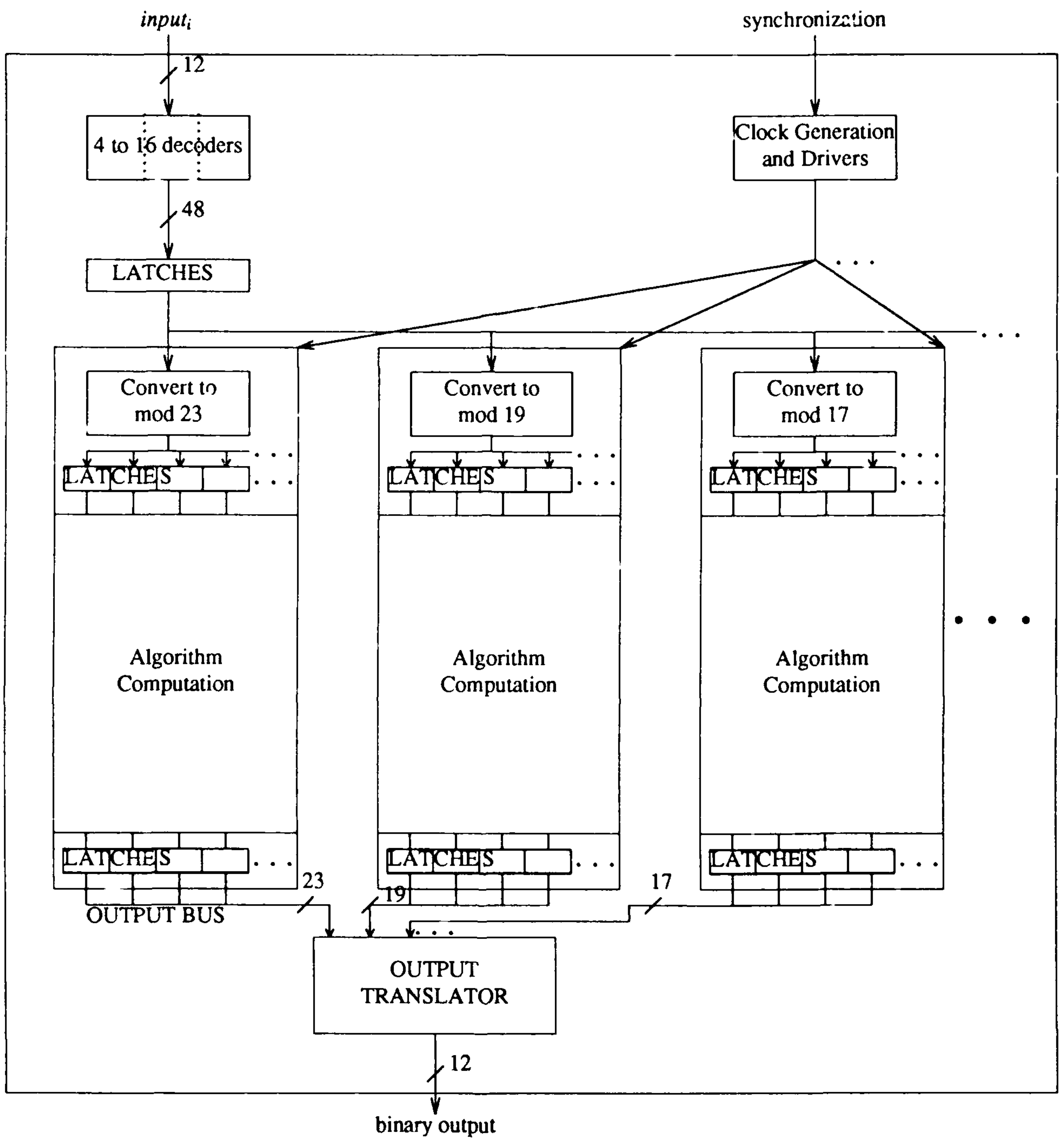
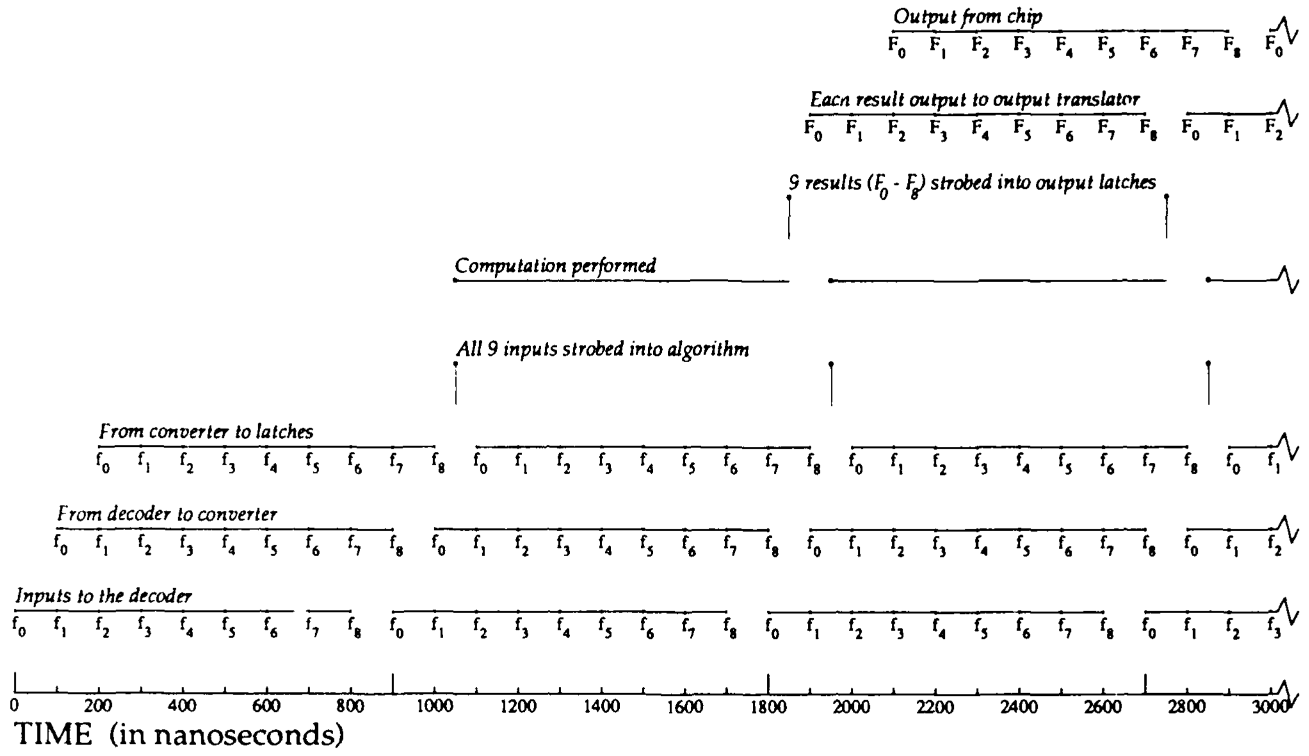


Figure 2. Details of the Overall Data Flow



Output from chip
F_0 F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_0
Each result output to output translator
9 results (F_0 - F_8) strobed into output latches
Computation performed
All 9 inputs strobed into algorithm
From converter to latches
From decoder to converter
Inputs to the decoder
f_0 f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8
0 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800 3000
TIME (in nanoseconds)

Figure 3. Timing Diagram for the Overall Operation

III. INPUT AND OUTPUT CONVERSION

Using the one-out-of-n representation greatly speeds and simplifies the input and output conversions. The weighted binary input can be broken into 4-bit half-bytes, and each of these can be fully decoded to 16 one-out-of-n lines. These are combined using constant multiplications and residue/one-out-of-n adders to form the desired values for each modulus. Since each residue can use the results from one set of decoders, all of the moduli can be translated from the input in parallel. This is illustrated in Figure 4 for the conversion of 12-bit binary numbers to one-out-of-7 representation.

The decoding begins by decomposing the input into 3 half-bytes. Each half-byte is decoded into 16 bits, representing one-out-of-n encoding. Each encoded number is then reduced to 7 bits, which represents the number modulo 7. The mapping from 1-out-of-n to 1-out-of-m where $m < n$ is accomplished by:

$$i > m - 1,\ i \rightarrow i \bmod m.$$

This is implemented by OR'ing the lines above $m - 1$ with their counterparts less than m.

For combining the half-bytes, the equation used is:

$$\text{Input} = x_1 2^{16} + x_2 2^4 + x_3 \equiv x_1 a + x_2 b + x_3 \bmod m$$

where the x's are the half-bytes and

$$2^{16} \equiv a \bmod m$$
$$2^4 \equiv b \bmod m$$

In the illustration, the two higher-order half-bytes are multiplied by 256 and 16 (4 and 2 mod 7), as described in Section V.B, and the two products are added together using a 7-bit modulo 7 residue adder. This result is then added to the low-order 4 bits to give the final result. These adders will be described in detail in Section IV. The entire input conversion process is shown in Figure 5.

Translation back into weighted binary is a harder problem that incurs a cost dependent on the number of moduli used and the desired output precision. The conversion to a weighted mixed radix code (MRC)[1] can be accomplished using the same general purpose adders and constant multipliers used in the rest of the circuit. The conversion of this code to binary requires a series of binary additions. Only the final binary add need be full precision; other additions vary in length according to the weight given the values in MRC.

Since the cost of conversion is fixed, given the number of moduli for the computation, and is independent of the computation performed, it can be amortized over the computations that compute the output function. As a consequence, evaluating a long expression in residue arithmetic is more cost effective than a short one if the output must be converted to binary. For the given algorithm, output conversion is much faster than the computation of the algorithm as shown in Figure 3. Thus, since the computation is overlapped in time with the output conversion, the throughput of the overall computation is not affected by the output conversion.

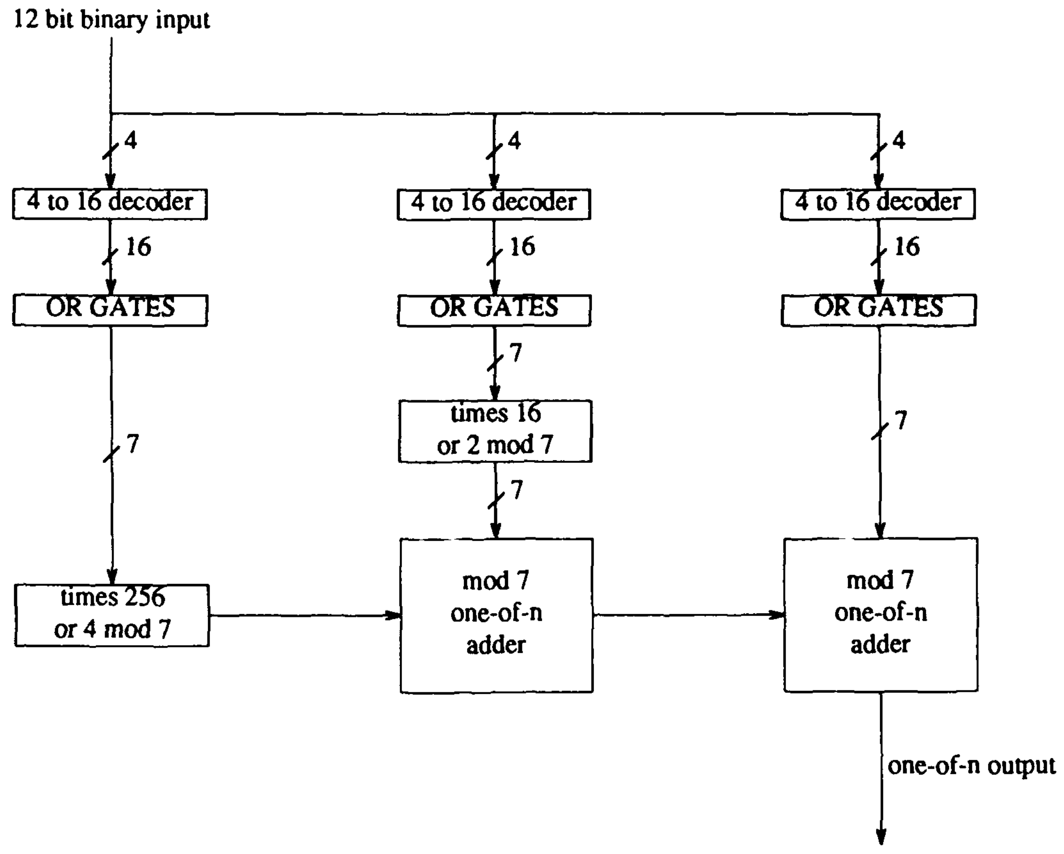
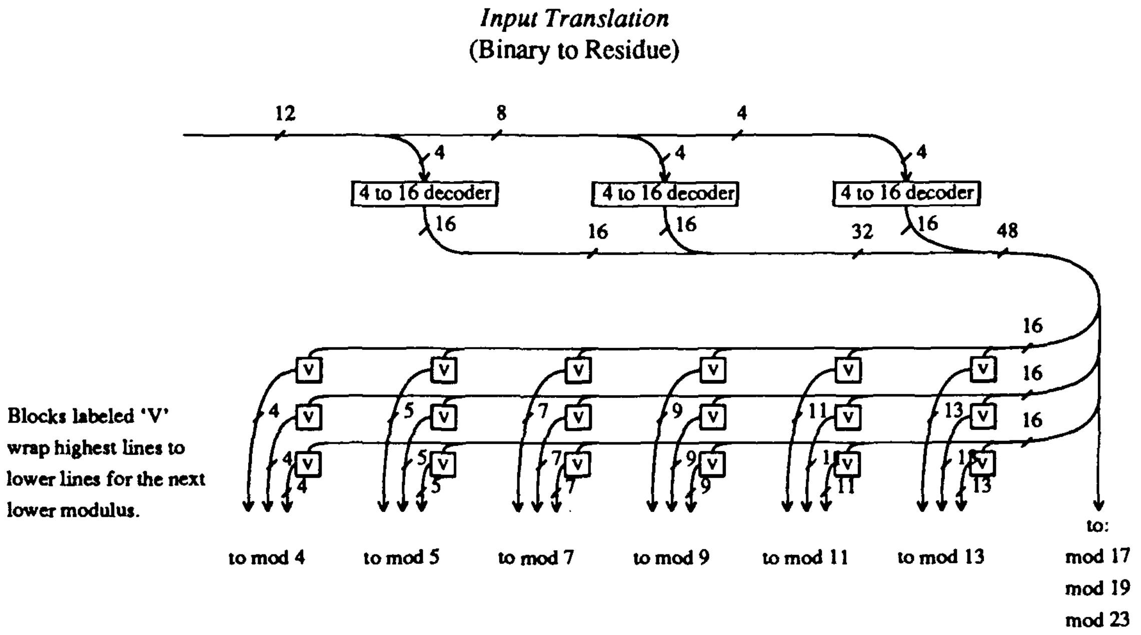


Figure 4. Example Illustrating Conversion From Binary to a One-out-of-n Residue Representation



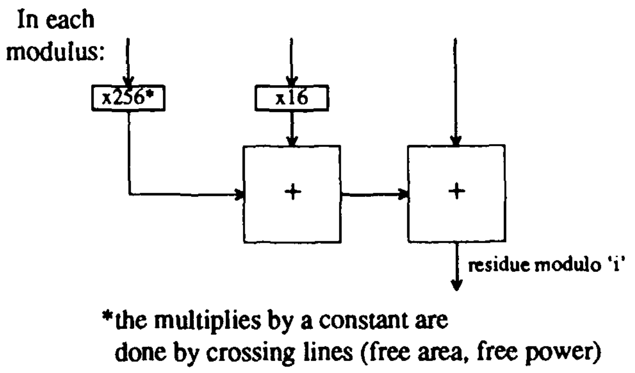

In each modulus:
x256*
x16
+
+
residue modulo 'i'
*the multiplies by a constant are done by crossing lines (free area, free power)

Figure 5. Input Conversion for the Example Algorithm

IV. ADDER DESIGN

The design of the adder can be illustrated by the example of a one-out-of-n representation for $n = 7$. Let us call the two seven-bit inputs *in1* and *in2* with the output being called *out*.

out[0] represents the *0th* line of *out* and *out[6]* represents the *6th* line of *out*. Thus, in Boolean form,

$$\begin{gathered} out[0] = \\ in1[0] * in2[0] + in1[1] * in2[6] + \\ in1[2] * in2[5] + in1[3] * in2[4] + \\ in1[4] * in2[3] + in1[5] * in2[2] + \\ in1[6] * in2[1] \end{gathered}$$

If we were to design such circuitry using logic gates (e.g., NAND gates) this design would be expensive in time, power and area. But, because *in1* will have one and only one line high, we can replace a relatively expensive NAND gate with a simple transmission gate. With transmission gates replacing the NAND gates, only one term will be driven, and the other n-1 terms will be tri-stated or high-impedance. This allows us to WIRE-OR the AND terms together along each output line. The result is a simple, regular structure for the residue adders (Figures 6, 7, and 8). The delay is roughly the time to drive the single transmission gate and propagate the charge to the edge of the adder. Figure 6 shows a straightforward implementation of the adder using AND and OR gates. Figure 7 shows a transmission gate implementation, and Figure 8 shows the layout.

In our NMOS prototype circuits, the adders used only single n-type pass transistor for the boolean AND function. In designing CMOS circuits, we found we could conserve power and area by retaining this design, rather than using the pass transistor pair. This nonstandard implementation of the AND function has been verified in simulation and by prototyping in CMOS.

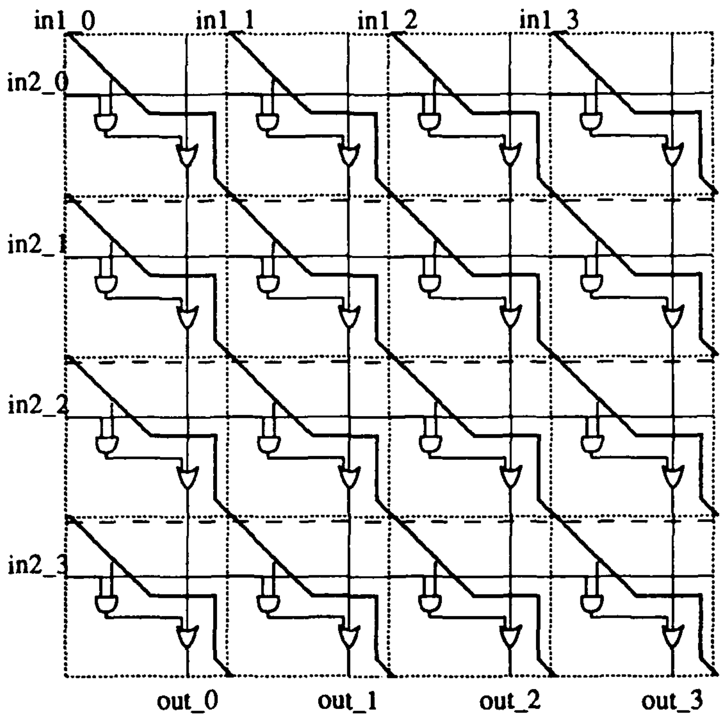


Figure 6. And/or Implementation of the Residue 5 Adder

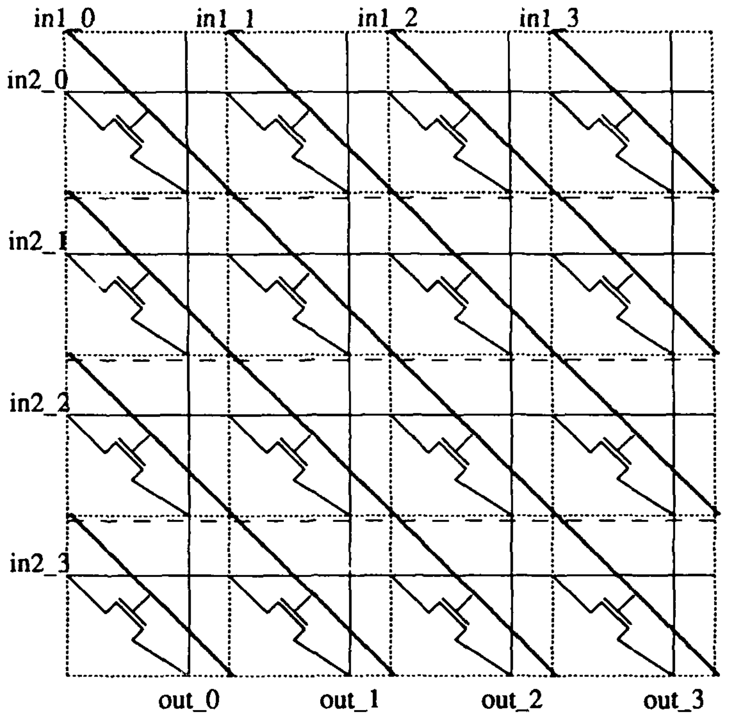


Figure 7. Transmission Gate/Wired-or Implementation of the Residue 5 Adder

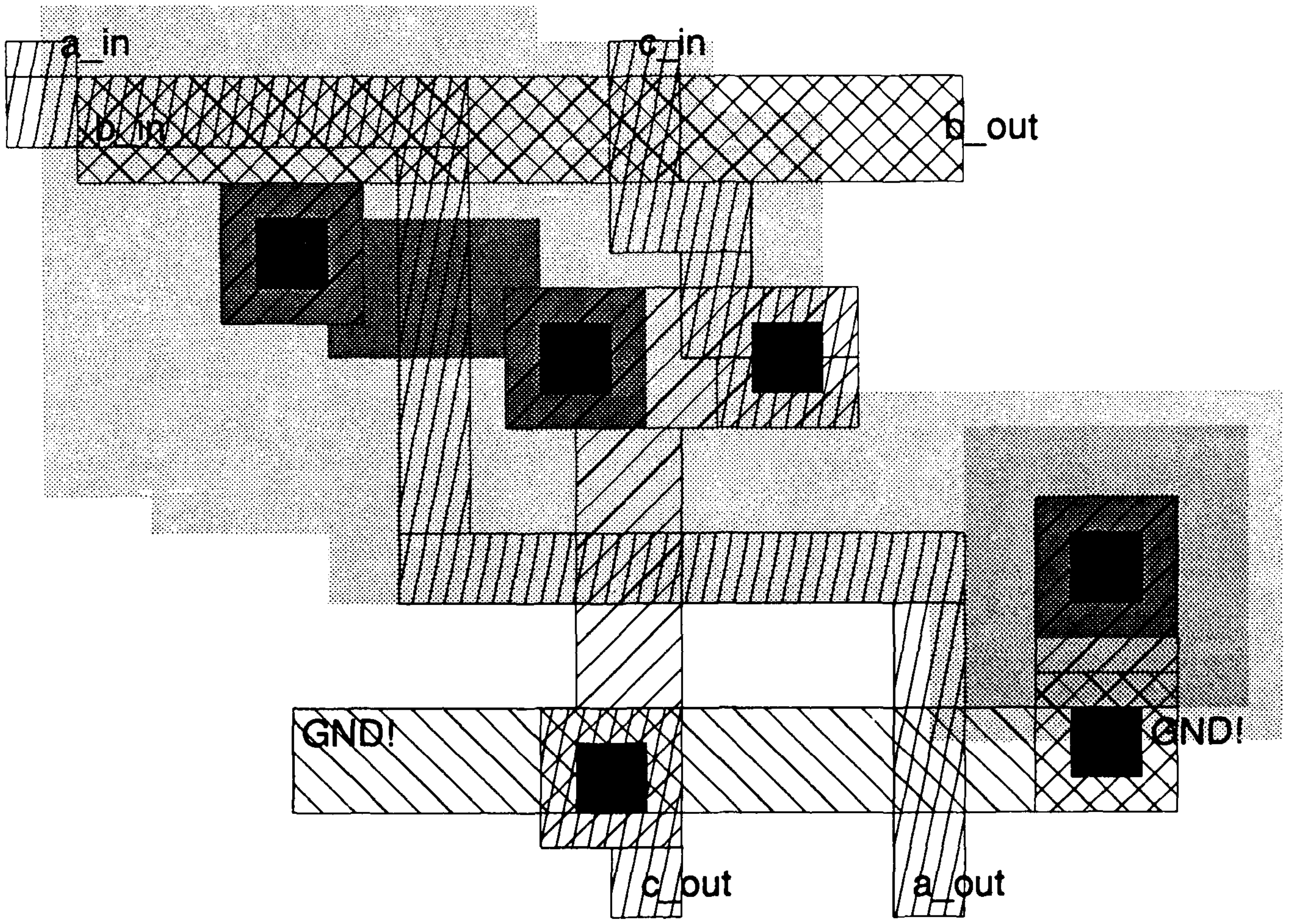


Figure 8. Transmission Gate Adder Cell in CMOS

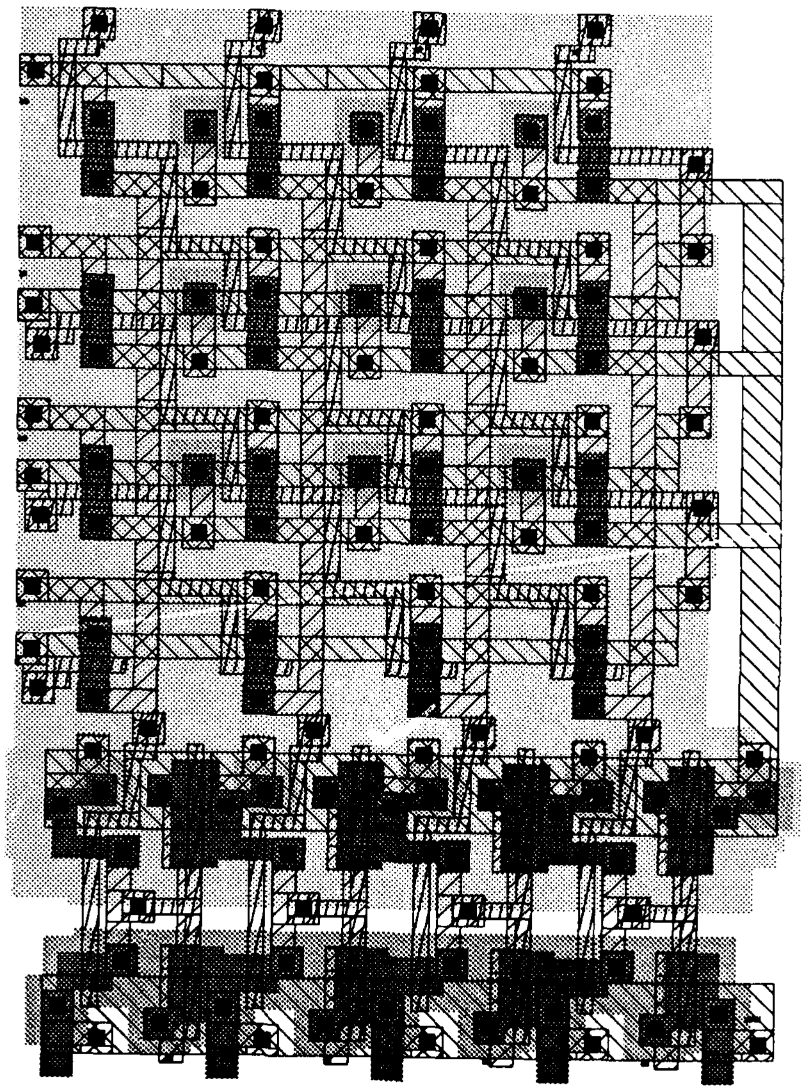


Figure 9. CMOS 4 X 4 Residue/One-Out-of-n Adder

V. MULTIPLIER DESIGN

A. MULTIPLICATION OF TWO VARIABLES

Multiplication in residue systems requires only a small amount of information for each residue. For the one-out-of-n representation, we can use our adders for multiplication by finding a logarithmic base, or generator, for prime moduli.

The set of residues modulo n that are relatively prime to n form a group under the operation of multiplication. In such groups there sometimes exists an element called a generator. The powers of the generating element will generate every residue in the multiplicative group. For a prime modulus there is always a generator for its multiplicative group*. A prime modulus p has a multiplicative group of size $p-1$, since 0 is not relatively prime to p.

For example: for the modulus 5 the number 2 is a generator, as is shown in the following table:

$$\begin{aligned}2^1 &\equiv 2 \bmod 5\\ 2^2 &\equiv 4 \bmod 5\\ 2^3 &\equiv 3 \bmod 5\\ 2^4 &\equiv 1 \bmod 5\end{aligned}$$

Therefore, the product 4 * 3 mod 5 is equivalent to $2^2 * 2^3$ mod 5. Because the group is cyclic, multiplying two numbers of the same base is the same as adding the exponents. The addition of the exponents is done mod 4, since 4 is the group size.[7] Therefore,

$$\begin{aligned}2^2 * 2^3 &\equiv 2^{2+3} \bmod 5\\ &\equiv 2^1 \bmod 5 \text{ since } 2 + 3 \equiv 1 \bmod (5 - 1)\\ &\equiv 2 \bmod 5\end{aligned}$$

This means that we can implement multiplication by permuting the inputs according to the generator mapping, performing a modular add, and then permute the lines from the generator mapping back to the one-out-of-n representation.

As noted earlier, the multiplicative group does not include the residue zero. Therefore, we need an OR gate to detect if either of the inputs is zero, and, if so, set the output to zero. Since the adder is built out of transmission gates with the expectation that exactly one line of each of the inputs will be high, it is also necessary that we place a small MUX on the output of the adder so that stored charge (from an undriven pass transistor) will not show up as an erroneous value. Figure 10 shows a general multiplier using mod 5 as an example.

* For a proof see[6].

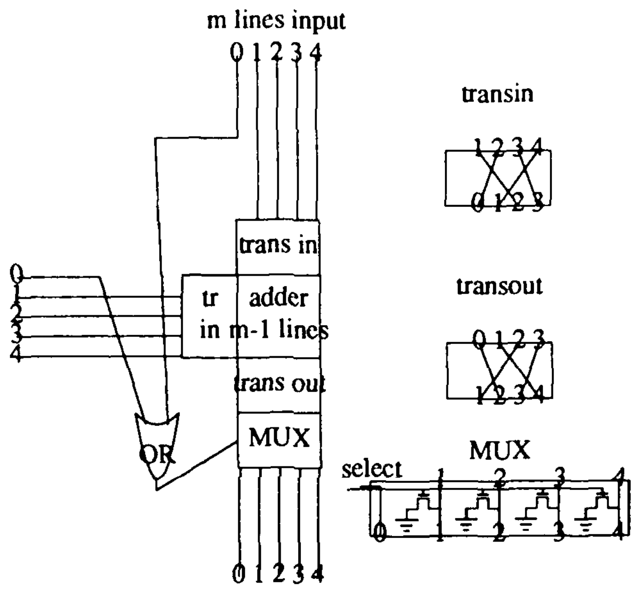


Figure 10. General Multiplier Modulo 5

B. MULTIPLICATION BY A CONSTANT

Multinlication by a constant is simple, requiring no active elements, and therefore has minimal extra power or delay. If the modulus and the constant are relatively prime, then to achieve the multiplication the lines are simply permuted. If they are not relatively prime, then the multiplication is a function which has a smaller output set than its input set. This is implemented either with wired-or's, or, in the case of multiplication by zero, elimination of the operation altogether.

Below there are two examples of constant multiplies for modulus 7, one by 5 (Figure 11) and the other by -1 (Figure 12).

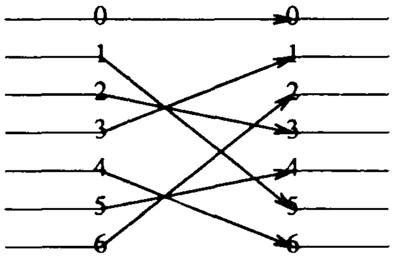


Figure 11. Multiply by 5 Modulo 7

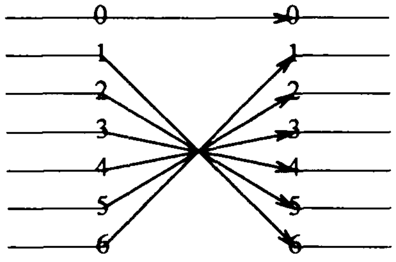


Figure 12. Multiply by 6 Modulo 7

VI. THE EXAMPLE ALGORITHM

The adder and multiplier designs for each of the moduli are similar, with variations in size and aspect ratio. Furthermore, for each of the moduli, a given algorithm requires an identical configuration of additions and multiplications. This isomorphism allows us to floorplan and wire one modulus and then easily extend the results to all moduli by scaling the design. Using the adders and multipliers described here, designed in CMOS, we have been able to implement a linear transform of interest. This transform is:

$$\mathbf{F} = \mathbf{A}\,\mathbf{f}$$

where

- F is the resultant vector (9 words at 12 bits)
- A is a 9 x 9 matrix of constants (2 to 12 bits of significance)
- f is the input vector (9 words at 12 bits)

Each term in this matrix product can be decomposed into a series of additions and multiplications by constants. The number of additions can be reduced by sharing common subexpressions. Then the equations can then be translated directly into residue adders with multiplications between addition stages. In the example design, the equation was reduced to 34 additions and 17 multiplications by constants. Each operation is implemented by a dedicated circuit for each modulus. Therefore, we will require 34 adders for each modulus. We are using nine moduli, therefore the algorithm requires 9*34 or 306 adders. Likewise, the algorithm requires 9*17=153 multipliers.

As described in the introduction, the input words arrive serially, bit-parallel, and are output the same way. On a single chip we have been able to receive the binary encoded words, partition them into the various residues, translate them into the one-out-of-n representation, perform the residue 11 calculations, and re-encode the residue 11 result into binary. We plan to use a separate chip to recombine the residues into a single binary word to be delivered to the next processing stage.

VII. IMPLEMENTATION

The algorithm was locally floorplanned for all the moduli. Presently, the same floorplan has been used for all moduli. Howev ince each modulus' floorplan must fit into a global floorplan, actual aspect ratios of the local floorplans has not been determined. Another reason for customization of local floorplans results from some of the constant multipliers being exact multiples of the modulus (congruent to zero mod n). In this case the input to the multiplier need not be calculated, nor does anything directly depending on the multiplier output; these components can be removed from the circuit. It is possible to introduce pipelining into the design at this stage. We were able to meet our particular performance constraints without it, but in general a significant speed improvement could be expected if there are several addition stages.

Figure 13 shows a local floorplan with adder cells and routing. Multiplications by constants are indicated by the labels "negate," "*2," and "m." The busses are indicated as single channels, although when implemented each is *n* wires wide, where *n* is the modulus.

After the local floorplans were designed, then the various residues were laid out and the interconnections were done. Various ancillary circuits had to be designed and routed appropriately. These include

- Input decoding
- State machines to load the various registers and to shift the data into the combinational algorithm
- State machines to load the output of the combinational part into storage registers and to move the data from the storage registers onto the output bus, and
- Circuitry to encode the one-out-of-n representation into a more compact form for transmission to the chip that will recombine the residues into standard binary.

A. DESIGN SIMULATION

The fidelity of the implementation to the original equations has been checked at the switch level by comparing simulation results derived from the chip representation to results from a bit-level simulator that is based on the equations. This was especially helpful in detecting routing errors in the multipliers and input converters.

The adder design was verified at the circuit level using SPICE.

B. TEST CHIPS

Two test chips have been fabricated: a modulus 11 adder, and the full algorithm with input and output conversion for modulus 11 computations. The adder test chips performed as predicted. Parts of the algorithm chip performed according to predictions. However, fabrication problems seemed to cause parts of the circuit to fail. A fully working chip is expected on the next iteration.

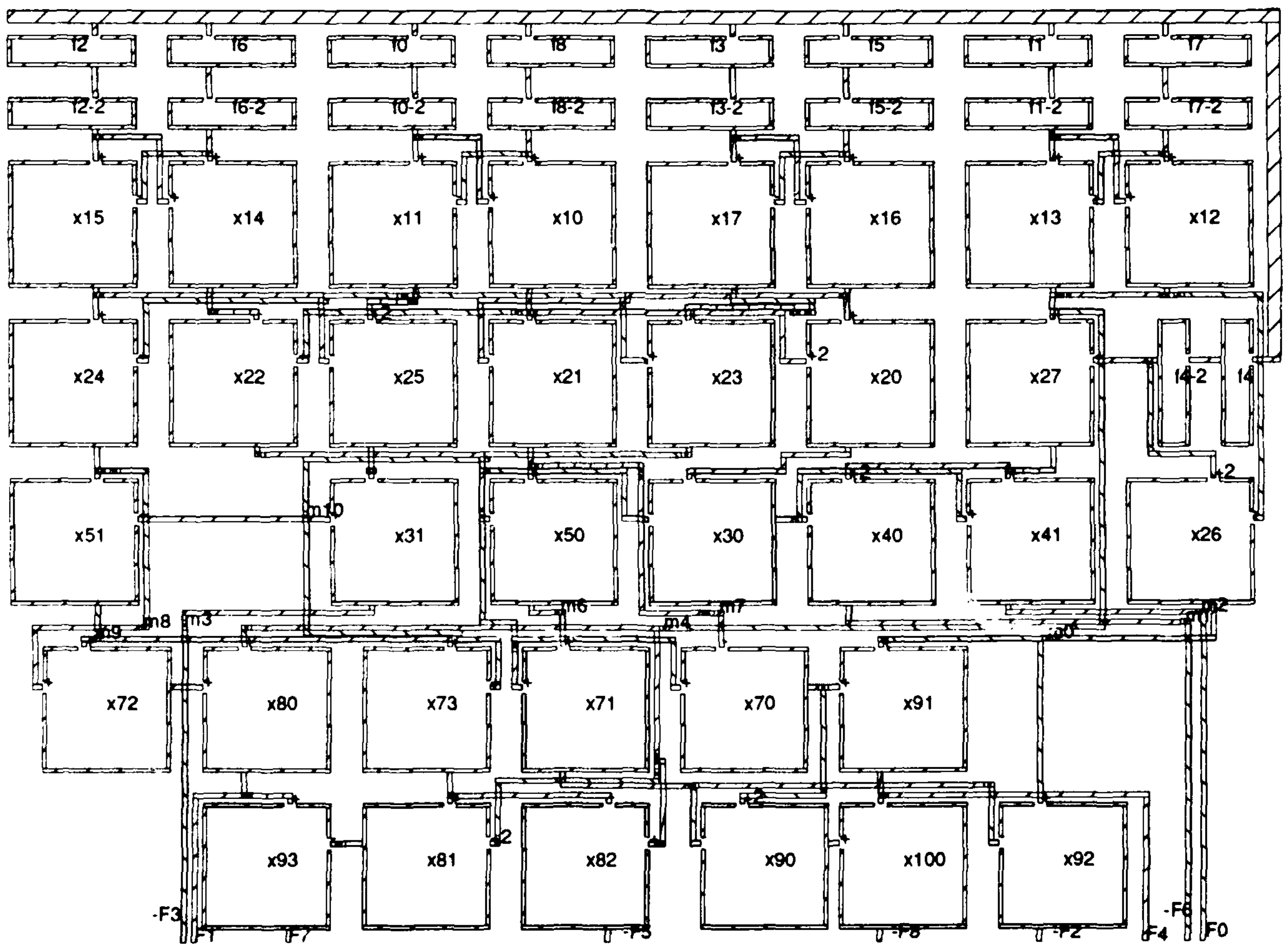


Figure 13. Local Floorplan

VIII. SUMMARY

Our initial results indicate that our particular design is able to operate well within our required specification of 10 MHz clock rate and 1 W power consumption in 3μ CMOS. The concept appears to be general enough that this methodology can be applied to a large family of signal processing and other arithmetic applications. The regular structure of the computational components facilitates the design of CAD tools for automating layout, placement, and routing. For computations of under 100 operations and about 16 bits of precision, we believe that this design method will be rapid and provide a high degree of confidence in the accuracy of the produced designs.

A. COMPARISON TO BINARY ARITHMETIC

An extensive comparison of the residue implementation to binary arithmetic is underway. Early estimates indicate that the residue implementation is somewhat larger than the binary implementation, with lower power consumption and higher performance.

Binary adders using both ripple-carry and carry-look-ahead circuits were designed for comparison purposes. These adders were implemented using pass transistors wherever possible, so that the circuitry was similar to the residue implementation. A constant binary multiplier was also designed, using an array of these adders. The output of each adder in the multipliers was buffered with an inverter pair. Ripple-carry adders were used in the multipliers, except for the last adder in each array, which was either ripple-carry, or CLA, as appropriate.

Early estimates of the comparison are summarized in Table 1. Area is given in units of λ^2, and the estimates for the binary implementation compare closely to published sizes for CMOS implementations. Power consumption is *estimated* by totalling the worst-case number of transistor gates that must be charged for a unit delay. Delays are given in inverter delays. For the delay calculations, a nominal fanout of 3 is assumed for each inverter. Some additional delay is expected due to interconnect and pass-transistor delays. These delays are being incorporated into SPICE simulations that will provide better estimates of the circuit performance. The area, delay and power contributions by the buffers are included in the adder and multiplier computations.

The first set of residue implementation computations are for the residue 23 case. The entire set of residues is expected to be about 3.93 times as large as the residue 23 implementation, and to use nine times as much power since there will be nine residues, and only one line will change state in each residue for each cycle. The residue implementation has been computed for the lowest power implementation. Significant speedup could be obtained with more power consumption. A 10-fold increase in power could produce a tenth the delay.

Routing area is not included in the algorithm area. However, we expect routing area to be approximately 200% of functional area for both traditional and residue implementations. This overhead has been confirmed for the moduli that have been implemented to date.

Fault tolerance, comparisons to binary, output conversion techniques, and CAD tools that can be used to design such chips will be addressed in future publications.

Table 1. Preliminary Comparison of Binary and Residue Arithmetic

	Binary (Ripple-carry)	Binary (C.L.A.)	Residue (Mod 23)	Residue (all)
ADDER AND BUFFER				
Est. Area	100,000	200,000	300,000	1,000,000
Est. Power	160	288	3	27
Est. Delay	33	11	25	25
CONSTANT MULTIPLIER INCLUDING BUFFERS				
Est. Area	1,000,00	1,000,00	0 (multiplies are free)	
Est. Power	1,280	1,408	0	
Est. Delay	68	46	0	
THE ENTIRE EXAMPLE ALGORITHM				
Est. Area	15,000,000	21,000,000	10,000,000	37,000,000
Est. Power	19,520	25,280	102	918
Est. Delay	266	112	150	150

B. CAVEATS

As originally designed the adders were unidirectional with respect to the inputs, and the output was bidirectional (a pass transistor) but would be operated in such a way that it was unidirectional.

During the design process it was observed that if the buffers were moved from the inputs to the output of the adders then we would only need half as many buffers. Unfortunately, we didn't think the problem through well enough to realize the error of our logic. The problem is that one of the inputs is now bidirectional, and can be changed if there are ever two one's on the other input. This change to the bidirectional input can either be a transient change that is soon remedied when the unidirectional input stabilizes, or the change can produce a form of feedback (feed sideways actually) that could cause the circuit to oscillate indefinitely.

The only solution to this problem is to make all of the inputs unidirectional. This can be done by the following:

- Moving the buffers back to the inputs. Not good in that there are (four times the residue) number of inverters needed to make the two buffers, and the buffers are somewhat removed from the long lines that have to be driven.
- Putting buffers on the inputs and outputs. Now we need (six times the residue) number of inverters for the three buffers.
- Putting an inverter at the inputs and an inverter at the output. This only needs (three times the residue) number of inverters and allows for each of the inverters to do some of the driving where it is necessary.

Any of these solutions means that the design will grow in size and, of course, will require design modifications.

Another, unrelated, problem is that when inputs are changing one of two things will happen to the input to an adder. Looking at just one of the inputs to an adder we start with one line high, and all of the rest low. Since nothing happens simultaneously in the world the transition from one line being high to another line being high will either go through a state where two lines are high or through a state where all lines are low.

The all lines low will not cause a problem, it will just leave lines undriven for a short period of time. Although the situation where two lines are high is a problem. While both lines are high there will exist a path from Vdd to ground that will be going through four pass transistors. Since pass transistors have such a low resistance, this will cause a large power surge and will lead to reliability problems.

In looking at the automatically produced file that shows the extracted parasitics from the laid out design, it is observed that there is up to a 10% variance in the capacitance of an adder output bus. This variance could cause the two input lines to remain high for an appreciable amount of time.

There are two ways to alleviate this problem: One is to put some circuitry at the input (or output) of an adder to ensure that at most one line is ever able to go high at a time. This will guarantee that this problem will never happen, unfortunately it will cost in time, power, area, and redesign. A second solution would be to design the buffers so that they are not symmetrical in operation. We could design them so that they are much quicker and stronger in going to a low than to a high. This "solution" makes no guarantee that two lines will not be high at once, but it does make it much less probable. The good things about this solution is that it will sacrifice a small amount of time, but it will not consume any more power, area or much redesign.

REFERENCES

[1] Nicholas S. Szabo and Richard I. Tanaka.
Residue Arithmetic and its Applications to Computer Technology.
McGraw-Hill, New York, 1967.

[2] W. K. Jenkins and B. J. Leon.
The Use of Residue Number Systems in the Design of Finite Impulse Response Digital Filters.
IEEE Transactions CAS-24(4):191-200, April 1977.

[3] W. K. Jenkins.
Recent Advances in Residue Number Techniques for Recursive Digital Filtering.
IEEE Transactions ASSP-27(1):19-30, February 1979.

[4] I. S. Reed and T. K. Truong.
Convolutions over Residue Classes of Quadratic Integers.
IEEE Transactions on Information Theory IT-22(4), July 1976.

[5] J. J. Vaccaro and B. L. Johnson and C. L. Nowacki.
A Systolic Discrete Fourier Transform Using Residue Number Systems Over the Ring of Gaussian Integers.
Technical Report 5, MITRE, February 1986.

[6] Daniel Shanks.
Solved and Unsolved Problems in Number Theory.
Chelsea Publishing Company, New York, New York, 1978.

[7] Ivan Niven and Herbert S. Zuckerman.
An Introduction to the Theory of Numbers.
John Wiley and Sons, New York, 1962.

LABORATORY OPERATIONS

The Aerospace Corporation functions as an "architect-engineer" for national security projects, specializing in advanced military space systems. Providing research support, the corporation's Laboratory Operations conducts experimental and theoretical investigations that focus on the application of scientific and technical advances to such systems. Vital to the success of these investigations is the technical staff's wide-ranging expertise and its ability to stay current with new developments. This expertise is enhanced by a research program aimed at dealing with the many problems associated with rapidly evolving space systems. Contributing their capabilities to the research effort are these individual laboratories:

Aerophysics Laboratory: Launch vehicle and reentry fluid mechanics, heat transfer and flight dynamics; chemical and electric propulsion, propellant chemistry, chemical dynamics, environmental chemistry, trace detection; spacecraft structural mechanics, contamination, thermal and structural control; high temperature thermomechanics, gas kinetics and radiation; cw and pulsed chemical and excimer laser development, including chemical kinetics, spectroscopy, optical resonators, beam control, atmospheric propagation, laser effects and countermeasures.

Chemistry and Physics Laboratory: Atmospheric chemical reactions, atmospheric optics, light scattering, state-specific chemical reactions and radiative signatures of missile plumes, sensor out-of-field-of-view rejection, applied laser spectroscopy, laser chemistry, laser optoelectronics, solar cell physics, battery electrochemistry, space vacuum and radiation effects on materials, lubrication and surface phenomena, thermionic emission, photosensitive materials and detectors, atomic frequency standards, and environmental chemistry.

Electronics Research Laboratory: Microelectronics, solid-state device physics, compound semiconductors, radiation hardening; electro-optics, quantum electronics, solid-state lasers, optical propagation and communications; microwave semiconductor devices, microwave/millimeter wave measurements, diagnostics and radiometry, microwave/millimeter wave thermionic devices; atomic time and frequency standards; antennas, rf systems, electromagnetic propagation phenomena, space communication systems.

Materials Sciences Laboratory: Development of new materials: metals, alloys, ceramics, polymers and their composites, and new forms of carbon; nondestructive evaluation, component failure analysis and reliability; fracture mechanics and stress corrosion; analysis and evaluation of materials at cryogenic and elevated temperatures as well as in space and enemy-induced environments.

Space Sciences Laboratory: Magnetospheric, auroral and cosmic ray physics, wave-particle interactions, magnetospheric plasma waves; atmospheric and ionospheric physics, density and composition of the upper atmosphere, remote sensing using atmospheric radiation; solar physics, infrared astronomy, infrared signature analysis; effects of solar activity, magnetic storms and nuclear explosions on the earth's atmosphere, ionosphere and magnetosphere; effects of electromagnetic and particulate radiations on space systems; space instrumentation.