AN EFFICIENT ASYNCHRONOUS MULTIPLIER

RODNEY M. GOODMAN
Department of Electrical Engineering
(116-81)
California Institute of Technology
Pasadena, CA. 91125

ANTHONY J. McAULEY*
Bell Communications Research
(MRE-2L340)
Morristown, NJ. 07950
(* this work performed while employed at Caltech)

ABSTRACT

A new, efficient asynchronous serial-parallel multiplier architecture is presented. It offers significant advantages over conventional clocked versions, without some of the drawbacks normally associated with similar asynchronous techniques, such as excessive area. In the paper we show how a general asynchronous communication element can be designed, and illustrate this with the CMOS multiplier chip implementation. It will also be shown how the multiplier could form the basis for a faster and more robust implementation of the Rivest-Shamir-Adleman (RSA) Public Key Cryptosystem.

INTRODUCTION

This paper shows how two powerful techniques can be combined to enhance communication security. They are: Public Key Cryptography (PKC) and asynchronous or wavefront chip architectures (AC). Public Key Cryptosystems [1,2] have, in theory, revolutionized cryptography. Unfortunately, although PKCs offer many advantages over conventional cryptosystems (eg key distribution and authentication), such systems have not been widely implemented to date. The reason for this lies not in their security, a decade of intensive cryptanalysis has given a good measure of confidence, but in the lack of fast hardware implementations. In the paper we will briefly review the RSA PKC algorithm and its variants and show that the heart of the hardware required is a large operand multiplier and divider. We note that conventional methods cannot eliminate the limitations on clock speed imposed by the large operands required for security. Even systolic arrays [3,4] with their regular structure suffer from unacceptable clock synchronization problems in this application due to the large operand sizes. The methods we employ to design AC hardware are different from traditional approaches [5-9]. In the paper we introduce the idea of asynchronous design and illustrate this with the design of an asynchronous FIFO buffer. We generalize the basic cell used for the FIFO into more flexible elements which are then used in our new serial-parallel multiplier. We then describe a 3 micron CMOS implementation of the multiplier and show that low complexity, fast speeds and reasonable area can be achieved.

THE RSA PUBLIC KEY CRYPTOSYSTEM

Rivest, Shamir and Adleman (RSA) invented the most elegant PKC in 1978 [2]. This RSA-PKC is able to greatly simplify the key distribution problem, since each user

CH2603-9/88/0000/0593$01.00 © 1988 IEEE

need only have one pair of keys, no matter how many people are on the network. In conventional cryptosystems every pair of users wishing to communicate must have a unique key. A second advantage of the RSA is that it is easily able to provide the equivalent of written signatures. For the purposes of this paper, we address the problem of implementing the encryption and decryption transformations. The enciphering and deciphering process in the RSA cryptosystem are conceptually very simple [1]. They are both modular exponentiations using normal integer arithmetic. So, for example, the cryptogram C is produced by raising the message M to the power of e, where e is the public encryption key. Similarly the message is recovered by raising the cryptogram C to the power of d, where d is the secret decryption key. Both operations are reduced modulo n, where n is also public. Based on current cryptanalytic techniques, it is computationally infeasible to find M from C, e and n without d, if the operands were all 1024 bits in length. The two modular exponentiations could be performed in software with the program below:

```
C = 1
FOR i = 1 to x
BEGIN
        IF e(i) =1 THEN C = (C).(M) mod n
        M = (M).(M) mod n
END.
```

where x is the number of bits in the operands and e(i) is the i th bit of the exponent.

Although there are other slightly more optimal algorithms for computing the exponent, the binary method above is always close to the best and definitely the simplest. From this program we can see the computational heart is the two modular multiplications. Integer multiplication and division (for taking the modulus) are well understood, and easily implemented for operands up to 64 bits. However, for larger operands, such as the 1024 bits required by the RSA, standard discrete hardware techniques are not optimal.

SERIAL-PARALLEL MULTIPLIERS

For large operands, where a full parallel multiplier is impractical, the serial-parallel architecture offers the best asymptotic performance. Because of the widespread application of these multipliers there has recently been great interest in finding the best architecture [11-13]. The traditional approach has the multiplicand loaded in parallel and the multiplier serially. Normally, for large operands, some method of loading the multiplicand serially is included. More recent systolic designs eliminate global data lines and pipeline the multiplier from cell to cell resulting in faster performance. The global clock line thus remains as the speed limit to performance. We can eliminate this by using the wavefront asynchronous approach, and thus achieve high speed modular exponentiation.

ASYNCHRONOUS COMMUNICATION

In an asynchronous wavefront design each element handshakes with its nearest neighbors in order to ensure that data is passed correctly. The handshaking is done

with a ternary signal which communicates either true information (1), false information (0) or a spacer (X). A sufficient condition to keep information (1 or 0) separate from other pieces of information is to make sure every element in the chain does not accept new information (and map this into a new output) unless the elements which receive its output have a spacer (X), and all its inputs have data (1 or 0) Similarly, an element cannot change to an X unless the next element has data and all its inputs have spacers. We employ the standard two rail logic coding [5,8] for the three states assumed by an element. That is: 00 = acknowledge, reset or null; 01 = data value logical 0; 10 = data value logical 1; 11 = not allowed. Since the null state (00) can only change to data (01 or 10) and data (01 or 10) to null (00) we have a Gray coding, avoiding races where two signals have to change state at the same time.

THE C11 ELEMENT

The basic building block used for an asynchronous FIFO is the Muller C-element [5]. We call this a C11 element as shown in Figure 1 and table 1.

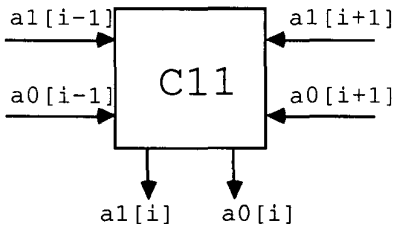

Figure 1 C11 Block Diagram

Table 1. C11 State Table

A[i-1] (a1,a0)	A[i+1] (a1,a0)	A[i] (a1,a0)
00	01	00
00	10	00
01	00	01
10	00	10
else		No change

A linear array of these elements forms a simple FIFO as shown in Figure 2.

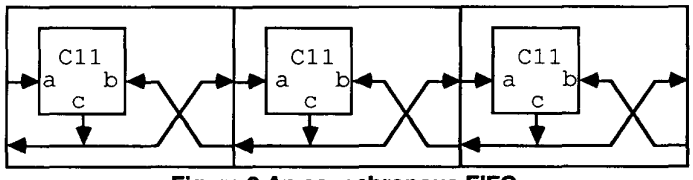

Figure 2 An asynchronous FIFO

A GENERALIZED 'CxyZ-ELEMENT' FOR ASYNCHRONOUS DESIGN

The distinguishing feature of the approach described in this paper is the use of more generalized C-elements. Using these elements leads to a faster and more area efficient design than could be achieved using just C11 elements. The C11 element simply passes data from its input to its output, ensuring there are no collisions. We now describe how we can perform functions on more than one input and drive more than one output element. Figure 3 shows our generalized C-element performing the function F, with x inputs and the output (O) going to y other C-elements.

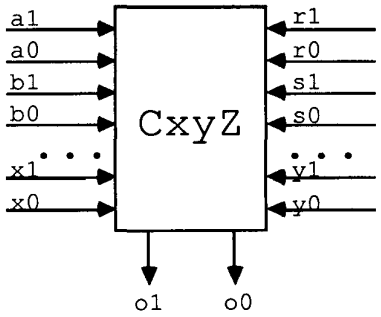

Figure 3 CxyZ Block Diagram

As for the C11 element two conditions must be obeyed for data to be output. First, all x inputs must have valid data and second, all y elements that the output feeds must have no data (that is have spacers or 'be reset'). Similarly, for the output to be equal to reset, all x inputs must be reset and all y elements must have data. The above is illustrated in table 2 for a C22F element.

Table 2 C22F State Table

A	B	R	S	O
00	00	01	01	00
00	00	10	10	00
00	00	10	01	00
00	00	01	10	00
10	10	00	00	F(00)
10	01	00	00	F(01)
01	10	00	00	F(10)
01	01	00	00	F(11)
		else		No change

When both inputs have data (eg A=10 and B=01), and the cells that have O as an input are reset (R=S=00), then the output can present new data using its functional mapping (F). That is: O=10 or 01 depending on f(01), where f(01) is the mapping F with B=logic

'1' (B=01) and A=logic '0' (A=10). When both inputs are reset (A=B=00), and both the cells it feeds have data (ie R and S are 01 or 10), then the output is reset (ie. O=00). Once we have designed a C11 cell it is then relatively easy to build more general CxyZ cells. Changing the mapping (Z) from the x inputs to the output is only slightly harder than when designing in single rail logic.

THE SERIAL-PARALLEL MULTIPLIER ARCHITECTURE

Figure 4 shows two stages of our asynchronous serial-parallel multiplier (ASPM) architecture. It is built from an AND gate and eight CxyZ elements: three C11's, two C12's, a C13, a C31S and a C13C. The AND gate is the standard two rail AND gate described by Martin [8]. The gate outputs a null (00) if either input is a null, a high (01) if both inputs are high, or a low (01) otherwise. The C12 and C13 are similar to the C11 described previously, with the input being passed to the output as in a clocked latch; but with extra logic to allow for the outputs being fed to 2 and 3 other C elements respectively. The C31S and C131C form the computational heart of the ASPM. The C31C generates the carry from its three inputs (a,b and c) according to the mapping: Output = (a AND b) OR (b AND c) OR (a AND c). The C31S element generates the sum from the same three inputs: Output = a EXCLUSIVE-OR b EXCLUSIVE-OR c. In both cases the elements can be designed using C11 elements, with an additional logic gate to perform the mapping.

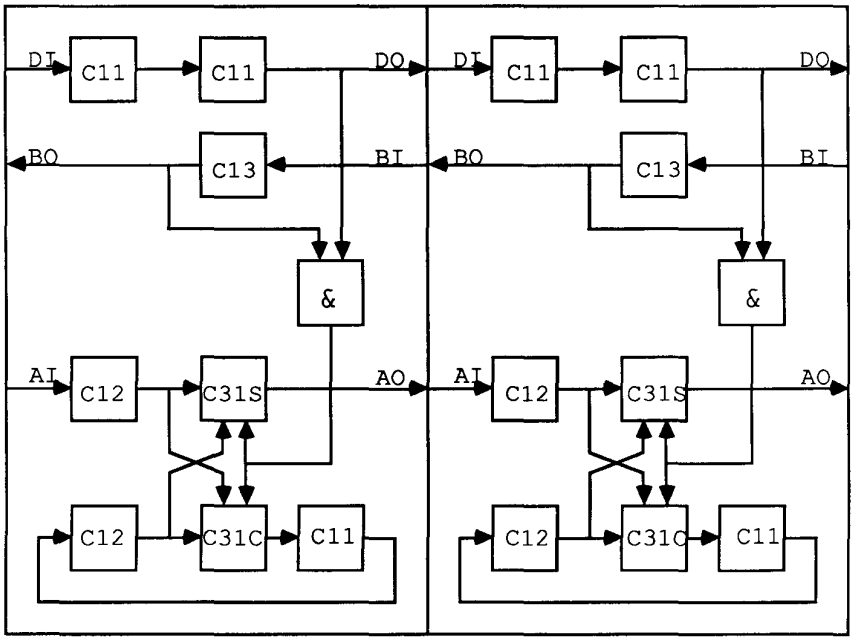

Figure 4 Two Stages of the Asynchronous Serial-Parallel Multiplier

THE ASYNCHRONOUS CMOS MULTIPLIER CHIP

Figure 5 shows a photograph of a sixteen stage ASPM chip, employing the architecture described in the previous section. This chip was built in order to demonstrate the speed and area that could be achieved using the generalized C-element. There was no attempt to optimize the speed of the chip, through dynamic logic or by using transmisssion gates; instead complementary CMOS gates with full logic swings were employed throughout. Nevertheless, the chip has been 'clocked' at 40MHz, which is twice the speed of a similarly designed systolic version. From our layout we estimate that it would be possible to build a 1024 stage multiplier (as required by the RSA encryption algorithm) in 1 micron CMOS with a 8mmx8mm die. One unexpected advantage of our approach was the design time - it was actually implemented faster than an equivalent clocked systolic version previously implemented by the authors. Though the ASPM appears quite complex, it only involves one major cell design, with a few modifications, and requires no careful clock distribution or clock buffer design.

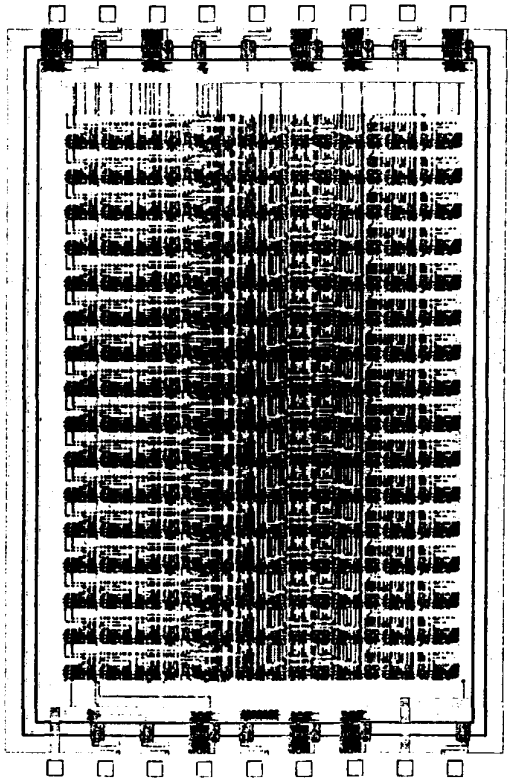

Figure 5 A 16 Stage CMOS Asynchronous Multiplier Chip

CONCLUSIONS

We have described a new asynchronous SPM. It is approximately 2 times faster than the equivalent systolic version at a cost of 4 times the area. Other less obvious benefits are its insensitivity to power spikes, greater reliability and suitability for automated design. We are currently investigating ways of further improving the inherent reliability of the asynchronous approach by the addition of fault tolerant circuitry [10]. The benefits of the asynchronous approach are particularly important for larger operands such as that required by the RSA-PKC. From our results with the 3 micron CMOS multiplier we estimate that a 1024 bit RSA-PKC built in 1 micron CMOS could run at over 100K bits/second.

REFERENCES

[1] D.E.Denning. "Cryptography and Data Security.", Addison Wesley, 1982.

[2] R.L. Rivest, A. Shamir & L. Adleman. " A Method for Obtaining Signatures and Public Key Cryptosystems." Communications of ACM, Vol. 21, No.2, pp. 120-126, Feb. 1978.

[3] H.T. Kung. "Why Systolic Architectures?" IEEE Computer, Vol.15, No.1, pp.97-107, Jan. 1982

[4] J. Fortes & B.W. Wah."Systolic Arrays - From Concept to Implementation." IEEE Computer, Vol.20, No.7, pp.12-17, July 1987.

[5] C.L. Seitz. "System Timing" - Chapter 7 in: Mead & Conway, "Introduction to VLSI Systems", Addison-Wesley, Reading MA, 1980.

[6] D.L. Dill & E.M. Clarke. "Automatic Verification of Asynchronous Circuit Using Temporal Logic.", Chapter Hill Conference on VLSI, 1985.

[7] A.L. Fisher & H.T. Kung. "Synchronizing Large VLSI Arrays." IEEE Trans. Computers, pp. 734-740, Aug. 1985

[8] A.J. Martin. "Compiling Communicating Processes into Delay-Insensitive VLSI Circuits.", Distributed Computing 1986.

[9] S.Y. Kung, S.C. Lo, S.N. Jean, J.N. Hwang. "Wavefront Array Processors - Concept to Implementation." IEEE Computer, Vol. 20, No.7, pp.18-33, July 1987.

[10] J.A. Abraham et al. "Fault tolerance Techniques for Systolic Arrays." IEEE Computer, Vol. 20, No. 7, pp. 18-33, July 1987.

[11] I-N Chen & R. Willoner "An O(n) parallel multiplier with bit sequential input and output" IEEE Tran. Computers, Vol.28, pp. 721-727, October 1979.

[12] R. Gnanasekarean "A fast serial-parallel binary multiplier" IEEE Tran. Computers Vol. 34, pp. 741-744, August 1985.

[13] I-Chen Wu "A fast 1-D serial parallel systolic multiplier" IEEE Tran. Computers, Vol.36, pp. 1243-1247, October 1987.