DEVELOPING AN RSA CHIP

Martin Kochanski
Business Simulations Ltd
Scriventon House
Speldhurst
Kent TN3 0TU
England.

Introduction.

FAP4 is a fast arithmetic processor designed specifically for modular operations, including exponentiation, on large integers. It is at present implemented as an array of 32-bit bit-slice processors, which may be interconnected without additional circuitry to obtain word lengths of up to 1023 bits. With 512-bit operands, exponentiation takes 133 milliseconds at worst (100 ms typically).

Architecture.

FAP4 is based on a new serial/parallel architecture for performing multiplication and modulo reduction together in one pass. It takes one clock cycle per bit of the multiplier, plus a small fixed overhead, to perform a single modular multiplication. While the overall serial one-pass nature of this architecture resembles Brickell's scheme [1], it is less complex, requires less circuitry to implement it, and is less sensitive to the exact details of implementation.

In implementing this architecture, a decision on partitioning had to be made: what should be done in hardware, and what by the host microprocessor?

At one extreme, a design such as Rivest's original one [3] does everything on-chip, even primality testing. The larger the chip, the

H.C. Williams (Ed.): Advances in Cryptology - CRYPTO '85, LNCS 218, pp. 350-357, 1986.

more difficult it is to make; and if high-level algorithms are built in to it, what happens if someone develops a better algorithm?

At the other extreme, one could implement as little as possible in hardware. The drawbacks of this can be seen in our own first-ever fast arithmetic processor (FAP1), whose central multiplier chip did a multiplication in 150ns and then waited over 1000ns for the host microprocessor to tell it what to do next!

We therefore decided that the complexity of operations performed by a fast integer arithmetic chip should be such that the host microprocessor knew what to do next by the time the chip had finished: chip time (which was expensive) would not be wasted, and excessive amounts of chip space (also expensive) would not be needed. In concrete terms, this meant that the chip should be able to do at least a single full-length modular multiplication. Even if this took only 30us, it would still leave enough time for the microprocessor to extract (say) the next bit of an exponent and formulate an appropriate command.

Implementation.

Implementing this architecture posed a problem. Typically, one would construct a small prototype using standard TTL chips before planning further designs - but FAP4 involved a "long, thin" design consisting of a few register bits and a little logic at each stage, and such a design does not lend itself to an efficient TTL implementation.

Accordingly, we decided to omit this prototyping phase and implement the architecture directly on semi-custom gate arrays. Because of the serial architecture, it was easy to partition the design into manageable slices, and 32 bits was selected as the slice size. Fujitsu's VH2600 series of 2.3 micron CMOS arrays was chosen.

The chip.

The FAP4 chips are packaged as 64-pin pin grid arrays. Pins are allocated as follows:

Address bus:	10 pins	┐
Data bus:	8 pins	├ Microprocessor
Control lines:	6 pins	│ interface.
READY signal:	1 pin	┘
Power:	4 pins	
Clock:	1 pin	
Serial interconnections:	14 pins	(7 between each pair of adjacent chips)
Identifiers:	6 pins	(tied to logic 1 or 0 to identify each chip's position in the array)
Internal control bus:	10 pins	
Controller slicing:	3 pins	(see below)

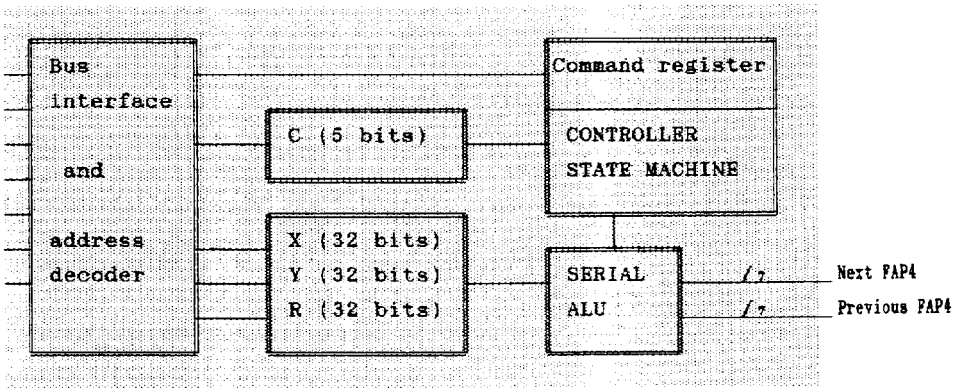
The shaded area shows the pins which exist only because of the partitioning into 32-bit slices. It will be seen that without this overhead, a single-chip implementation would require only 28 pins.

As well as a 32-bit arithmetic slice, each chip contains a 5-bit controller slice. In an array, two of these slices are used together to implement a 10-bit controller for the array, and the remaining controller slices are inactive. This approach allowed the use of a single type of chip throughout, rather than having separate arithmetic and controller chips. An array can thus be expanded to up to 1,024 bits without exceeding the controller's capacity; futher expansion requires a separate controller to be provided.

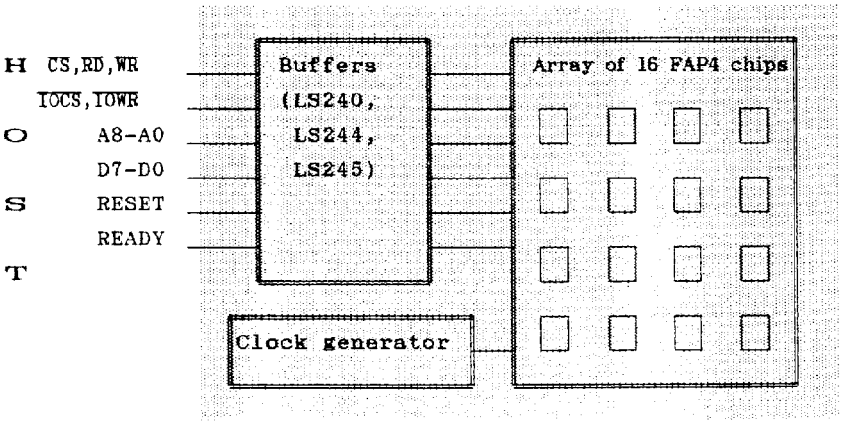
Each chip uses 2,400 2-input NAND equivalent cells, of which about 2,000 are used for the arithmetic slice - about 63 cells per bit.

A typical 512-bit (16-chip) array runs at 5MHz. A 512-bit array will be assumed in all the descriptions of operation and algorithms in this paper.

FAP4 chip block diagram.



FAP4 512 bit array block diagram.



Operation.

The FAP4 array has a standard byte-wide bus interface, and appears to a host microprocessor as a 512-byte memory space and a single output port.

The output port accesses the FAP4 array's *command* register, and is used to convey commands to the array. As soon as a command is written, the READY interface line goes low and the specified operation is performed. When the operation is completed, READY goes high, and the host can access the array's memory or write a new command to the command register. The READY signal can be interfaced to the host system in a variety of ways: directly to an input port, which the host then polls to test for completion of an operation; to an interrupt controller, so that the host is interrupted whenever the FAP4 array has completed an operation; or to a direct memory access (DMA) controller, so that a sequence of commands can be sent to the FAP4 array without any need for intervention by the host.

Possible commands are as follows (commands are shown in binary, most significant bit first):

```
00000000          Y := Y * Y
00000001          Y := Y * X
00000010          Y := (Y * Y) * X
00000011          Y := (Y * X) * X
```

where *a* * *b* means "multiply register *a* by the *C* most significant bits of register *b*, reducing the result modulo *R*".

The first two operations take approximately C+100 clock cycles; the last two take twice as long.

Memory is assigned to registers as follows (addresses are in hexadecimal relative to the base of the board's address space):

```
  00 to 7F:   Y register
  80 to FF:   X register
 100 to 17F:  R register (write-only)
 1F8 and 1FC: C register (write-only)
```

The X, Y, and R registers are up to 1024 bits (128 bytes) long. In an array of fewer than 32 chips, the X, Y, and R registers are correspondingly shorter, so that in a 16-chip (512-bit) array, register R will extend only from address 140 to 17F.

Register R defines the modulus to be used. If it is zero, then no modulo reduction takes place; otherwise, for correct operation, the two most

significant bits of R must be 01: in other words,

$$2^{510} \leq R \leq 2^{511}-1$$

If the modulus required is outside this range, it can be shifted to fall within it; the Y and X operands can then be shifted to correspond with this.

Registers X and Y define the operand values to be used; the result of each arithmetic operation is stored in register Y, from where it can be read by the host processor. For correct operation, the value of Y should always be less than that of R in modular operations.

Register C is 10 bits long, and is used to define the precision of the current operation, which may be less than the available capacity of the board: smaller values of C produce faster operations.

Operation.

A typical sequence of operations is:

1. Write operands to FAP4's memory.
2. Issue a command.
3. Wait for the READY signal to go high.
4. Read results from FAP4's memory.

Algorithms.

- Multiplication with modulo reduction is the fundamental operation of the chip.

- Multiplication without modulo reduction can be done by setting *R* to zero.

- Exponentiation is performed by scanning the exponent from left to right and issuing command *10* (square and multiply) if a bit is 1 or *00* (square) if it is 0. These commands can be precomputed and a DMA

channel used to pass them to the array: there is then about a microsecond's latency between commands.

Faster operation could be achieved in an implementation with more registers: for instance, the cube of the base could be stored in a separate register, and the sequence "square, multiply, square, multiply" replaced by "square, square, multiply by cube": but no such extension can do more than double the speed of exponentiation.

- Division can be performed by multiplication and modulo reduction. A simple example in base 10 is:

  314159 x 10000 modulo 1469999 = 202137

  so 314159 / 147 = 2137 remainder 20.

- Highest common factor computations can be speeded up by modifying Euclid's algorithm to work with left-justified operands.

- Modular division (finding y such that $x * y = z \bmod r$) normally uses Euclid's algorithm with a parallel calculation building up the inverse as the algorithm progresses. A similar technique to that used for division can combine the two processes into one.

- Primality testing is best done by using a procedure such as Knuth's Algorithm P [2], which involves repeated squaring and hence no reloading of operands.

Performance.

At a clock rate of 5MHz, the single multiplication commands *00000000* and *00000001* are executed in $c/5 + 20$ microseconds, where c is the operand length stored in the *C* register. For full 512-bit operations, this gives a time of 124μs per operation. The double-multiplication commands *00000010* and *00000011* take twice as long.

The time taken for a full exponentiation depends on the number of *1* bits in the binary representation of the exponent. The worst-case time for a 511-bit exponentiation is 133ms, and the average time is 100ms.

Applications.

FAP4 can be used for:

- Implementation of the RSA or similar number-theoretic public-key cryptosystems.
- Key setup and exchange for other cryptosystems.
- Authentication of messages and electronic mail by means of "digital signatures".
- Generation of prime numbers and sets of keys for public-key cryptography.
- Add-on acceleration of high-precision fixed-point operations for a host computer.
- Number theoretic research: e.g. evaluation of factorisation algorithms.

Availability.

FAP4 is available now, as chip sets; or on a 512-bit array board with a generic microprocessor interface. Also available are an interface card for the IBM PC; and a special *XBASIC* interpreter, which provides a version of the Basic language which includes support for high-precision arithmetic and for arithmetic operations with built-in modulo reduction.

REFERENCES:

[1] E.F. Brickell, "A Fast Modular Multiplication Algorithm with Applications to Cryptography", *Advances in Cryptology: Proceedings of CRYPTO 82*, Plenum Press, New York: 1983.

[2] Donald M. Knuth, *The Art of Computer Programming, 2: Seminumerical Algorithms*, Addison-Wesley, New York: 1981 (2nd edition) p.379.

[3] R.L. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher", *LAMBDA Magazine* 1, 3 (Fourth Quarter 1980), 14-18.