An Energy-Efficient Reconfigurable Public-Key Cryptography Processor

James Goodman, *Member, IEEE,* and Anantha P. Chandrakasan, *Member, IEEE*

***Abstract*—The ever-increasing demand for security in portable energy-constrained environments that lack a coherent security architecture has resulted in the need to provide energy-efficient algorithm-agile cryptographic hardware. Domain-specific reconfigurability is utilized to provide the required flexibility, without incurring the high overhead costs associated with generic reprogrammable logic. The resulting implementation is capable of performing an entire suite of cryptographic primitives over the integers modulo N, binary Galois Fields and nonsupersingular elliptic curves over GF(2^n), with fully programmable moduli, field polynomials and curve parameters ranging in size from 8 to 1024 bits. The resulting processor consumes a maximum of 75 mW when operating at a clock rate of 50 MHz and a 2-V supply voltage. In ultralow-power mode (3 MHz at 0.7 V) the processor consumes at most 525 μW. Measured performance and energy efficiency indicate a comparable level of performance to previously reported dedicated hardware implementations, while providing all of the flexibility of a software-based implementation. In addition, the processor is two to three orders of magnitude more energy efficient than optimized software and reprogrammable logic-based implementations.**

I. Introduction

THE FIELD of cryptographic algorithms can be divided into two basic types, symmetric and asymmetric, which have distinctly different properties. Symmetric, or secret-key, algorithms require two parties to share some secret piece of information (i.e., the key) that is then used to encrypt/decrypt messages between them. The existence of a shared piece of secret information enables secret-key algorithms to be very computationally efficient. Hence, symmetric algorithms are used to encrypt the bulk of the messages being passed. Asymmetric, or public-key, algorithms on the other hand rely on the presumed existence of hard number-theoretic problems that enable two sets of keys to be created: public (encryption) and private (decryption). Public keys are stored in the open so that anyone can encrypt a message. However, because of the number-theoretic properties of the algorithms used, only the intended recipient who generated the public–private key pair can decode the message correctly. Hence, no secret needs to be shared by the communicating parties. Unfortunately, the underlying mathematics which enables this asymmetry requires a great deal more computation than symmetric-key algorithms. For example, a single public-key operation can consume as much time and energy as encrypting tens of megabits using a secret-key cipher. Thus, public-key algorithms are used primarily for establishing secret keys throughout the network in a secure manner, as well as for user authentication and identification. The work described within this paper addresses the implementation of public-key cryptographic algorithms only.

In the past, several standards for implementing various asymmetric techniques have been proposed, leading to a multitude of incompatible systems that are based upon different underlying mathematical problems and algorithms. For example, the IEEE 1363 Standard Specification for Public Key Cryptography [1] recognizes three distinct families of problems upon which to implement asymmetric techniques: integer factorization (IF), discrete logarithms (DL), and elliptic curves (EC).

As a result, system developers have had to utilize software-based techniques in order to achieve the algorithm agility required to maintain compatibility. Unfortunately, software-based approaches lead to slow implementations that are very energy inefficient. Hence, these approaches are not well suited to the migration to portable battery-operated nomadic computing terminals. Hardware-based implementations, on the other hand, while being very energy and computationally efficient, are very inflexible and capable of supporting only a limited subset of asymmetric cryptography. A compromise between these two extremes is achieved by taking advantage of the fact that the range of operations is small enough that domain-specific reconfigurable hardware can be developed that is capable of implementing the various asymmetric algorithms without incurring the overhead associated with generic reconfigurable logic devices. Furthermore, this is done in an energy-efficient manner that enables operation in the portable energy-constrained environments where this algorithm agility is required most of all. The resulting implementation is known as the domain-specific reconfigurable cryptographic processor (DSRCP).

In conventional reconfigurable applications such as field-programmable gate arrays (FPGAs), the architectural goals of the device are to provide a large number of small yet powerful programmable logic cells, embedded within a flexible programmable interconnect. Unfortunately, the overhead associated with making such a general purpose computing device ultimately limits its energy efficiency and hence its utility in energy-constrained environments. Kusse [2] quantifies this overhead by breaking down the energy consumption of a conventional FPGA (Xilinx XC4003A [3]) into its architectural

Manuscript received March 22, 2001; revised June 4, 2001. This work was supported in part by the Defense Advanced Research Project Agency (DARPA) Power Aware Computing/Communication Program and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement F30602-00-2-0551, and in part by National Semiconductor Corporation.

J. Goodman is with Chrysalis-ITS, Ottawa, ON K2G 6P9, Canada.

A. P. Chandrakasan is with the Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Publisher Item Identifier S 0018-9200(01)08228-2.

TABLE I
DSRCP INSTRUCTION SET

Mnemonic	Description
SET_LENGTH length	sets width of processor to be (length + 1)
REG_CLEAR rd,rs0	clears registers specified in mask formed by (rd,rs0) = R<7:0>
REG_MOVE rd,rs0	rd = rs0
REG_LOAD rd	rd is loaded from I/O interface
REG_UNLOAD rs1	rs1 is unloaded to I/O interface
COMP rs0,rs1	sets the gt and eq flags, where gt = (rs0 > rs1) and eq = (rs0 == rs1)
ADD/SUB rd,rs0,rs1,rs2	rd = rs0 + rs1 + rs2<0> rs2<2:1> = 00 rd = (rs0 + rs1 + rs2<0>) >> 1 rs2<2:1> = 01 rd = rs0 - rs1 rs2<2:1> = 10 rd = (rs0 - rs1) >> 1 rs2<2:1> = 11
MOD_ADD rd,rs0,rs1,rs2	rd = (rs0 + rs1 + rs2<0>) mod N
MOD_SUB rd,rs0,rs1	rd = (rs0 - rs1) mod N
MONTRED_A	(Pc,Ps) = A · 2^{-n} mod N
MONTMULT	(Pc, Ps) = A·B· 2^{-n} mod N
MONTRED	(Pc,Ps) = (Pc,Ps)· 2^{-n} mod N
MOD rd,rs0,rs1,rs2	rd = (rs1·2^n + rs0) mod N, correction factor of 2^{2n} mod N stored in rs2
MOD_MULT rd,rs0,rs1,rs2	rd = (rs0·rs1) mod N, correction factor of 2^{2n} mod N stored in rs2
MOD_INV rd,rs0	rd = (1 / rs0) mod N
MOD_EXP rd,rs0,rs2,length	rd = $rs0^{Exp}$ mod N, Exp has (length + 1) bits, correction factor of 2^{2n} mod N stored in rs2
GF_ADD rd,rs0,rs1	rd = rs0 + rs1 over GF(2^n) (equivalent to rs0 ⊕ rs1)
GF_MULT	Pc = A·B
GF_INV	A = 1 / Pc
GF_INVMULT	A = B / Pc
GF_EXP rd,rs0,length	rd = $rs0^{Exp}$ mod N, Exp has (length + 1) bits
EC_ADD rd,rs0,rs1,rs2,wb	(rd,rd+1) = (rs0,rs0+1) + (rs1,rs1+1), over curve defined by parameters in (rs2, N) NOTE: if (wb = 0) computation is performed but result is not written back to (rd,rd+1)
EC_DOUBLE rd,rs0,rs2	(rd,rd+1) = 2·(rs0,rs0+1), over curve defined by parameters in (rs2, N)
EC_MULT length	(R4,R5) = Exp·(R2,R3), Exp has (length + 1) bits, over curve defined by parameters in (R6,N)

components. The analysis reveals that only 5% of the total energy is used to perform useful computation, while approximately 65% is dissipated in the programmable interconnect.

The DSRCP differs from conventional reconfigurable implementations in that its reconfigurability is limited to the subset of functions, called a domain, required for asymmetric cryptography as defined in IEEE 1363. This domain requires only a small set of configurations for performing all of the required operations over all possible problem families defined within the standard. As a result, the reconfiguration overhead, particularly that of the reprogrammable interconnect, is much smaller in terms of performance, energy efficiency, and reconfiguration time, making the DSRCP feasible for algorithm-agile asymmetric cryptography in energy-constrained environments.

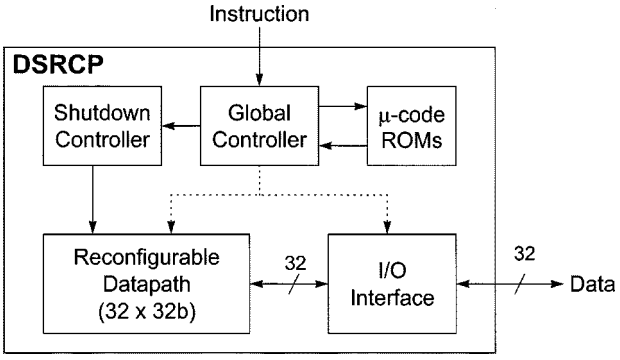

Fig. 1. Top-level system architecture.

II. ARCHITECTURE

A. Instruction Set Architecture

The instruction set definition of the DSRCP is dictated by the IEEE 1363 Public Key Cryptography Standard document [1]. A list of the arithmetic functions required to implement the various primitives defined in the standard was tabulated in a functional matrix, which was then used to define the instruction set architecture (ISA) of the processor (Table I). The ISA contains 24 instructions broken up into six types of operations: conventional arithmetic, modular integer arithmetic, GF(2^n) arithmetic, elliptic curve field arithmetic over GF(2^n), register manipulation, and processor configuration.

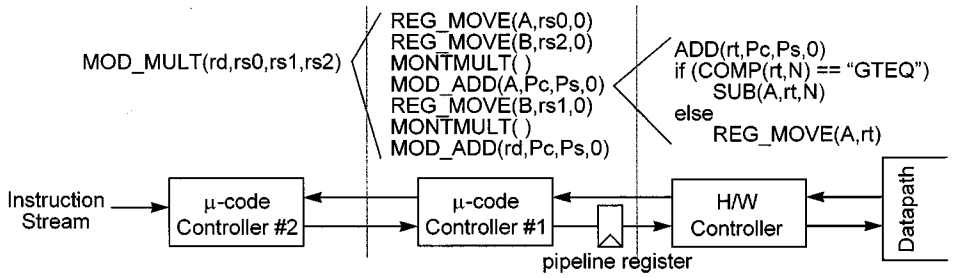

Fig. 2. Hierarchical instruction structure of the DSRCP.

B. Hardware Architecture

Fig. 1 shows a top-level block diagram of the DSRCP. The processor consists of four main architectural blocks: the global controller and microcode ROMs, the I/O interface, the shutdown controller, and the reconfigurable datapath.

The global microcontroller is responsible for all high-level control within the DSRCP. The controller utilizes a three-tiered control approach that uses both hardwired and microsequenced control functions. This multitiered approach is required as various instructions within the DSRCP's ISA are implemented using other instructions, as illustrated by the MOD_MULT instruction example shown in Fig. 2.

The microcode approach is used due to its simplicity and extensibility, as modifications and enhancements of the ISA can be accomplished with minimal design effort by modifying the microcode ROM. The drawback of using this approach is the additional latency that is incurred by accessing the ROMs sequentially, which can end up consuming a significant portion of the processor's cycle time. This performance issue is addressed by pipelining the instruction decoding/sequencing at the output of the first-level microcode ROM, as shown in Fig. 2.

The DSRCP features a shutdown controller that is responsible for disabling unused portions of the datapath in order to minimize any unnecessary switched capacitance. The shutdown strategy is dictated by the current width of the datapath, as set by the last invocation of the SET_LENGTH instruction and enables the datapath to be shut down in 32 32-b increments.

Operands used within the processor can vary in size from 8 to 1024 bits (1025 bits in the case of field polynomials for GF(2^{1024})), requiring the use of a flexible I/O interface that allows the user to transfer data to/from the processor in a very efficient manner. Ultimately, the I/O interface width is dictated by the physical implementation of the processor, which makes a 32-b interface the most economical width. The choice of a 32-b interface maps well to existing systems, as well as allowing for relatively fast operand transfer onto and off of the processor, requiring at most 32 cycles to transfer the largest possible operand. The additional bit required for GF(2^{1024}) field polynomials is input as part of the REG_LOAD instruction word.

The primary component of the DSRCP is the reconfigurable datapath, whose architecture is shown in Fig. 3. The datapath consists of four major functional blocks: an eight-word register file, a fast adder unit, a comparator unit, and the main reconfigurable logic unit.

The register file size is chosen to be eight words as that is the minimum number required to implement all of the functions of the datapath. The limiting case for this architecture is that of elliptic curve point multiplication in which registers R2 and R3 are used to store the point that is going to be multiplied by the value stored in Exp register, R4 and R5 are used to store the result, R0 and R1 are used to store an intermediate point used during the computation, R6 is used to store the curve parameter a, and R7 is used as a dummy register in order to provide resilience to timing attacks. The number of read and write ports within the register file is dictated by the requirement to be able to perform single-cycle two-operand instructions that generate a writeback value. In certain cases, two write ports could have proven useful (e.g., elliptic curve point transfers), but the infrequency of the operation did not merit the additional overhead that it would have introduced.

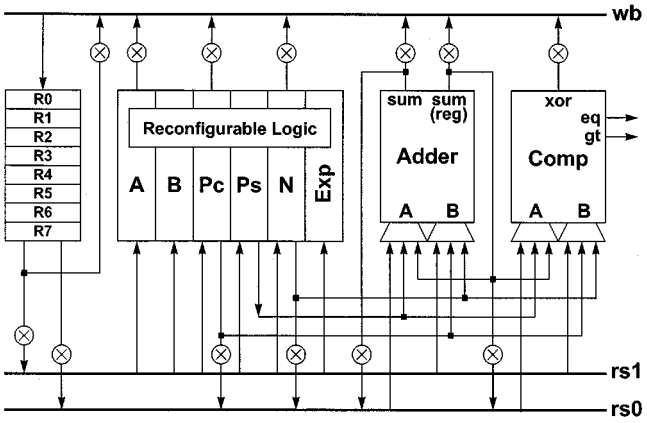

Fig. 3. Reconfigurable datapath architecture block diagram.

The fast adder unit is capable of adding/subtracting two n-bit ($8 \leq n \leq 1024$) operands in three cycles using the hybrid carry–bypass and carry–select technique described in [4] and optimized for a bitsliced implementation (Fig. 4).

The comparator unit performs single-cycle magnitude comparisons between two n-bit operands, as well as computing the XOR of the two operands (i.e., GF(2^n) addition). The comparator generates two flags, `gt` and `eq`, that can be decoded into all possible magnitude relations.

The reconfigurable logic unit consists of six local registers (Pc, Ps, A, B, Exp, and N) and a reconfigurable logic block that is capable of implementing all of the required datapath operations. The Pc and Ps registers are used primarily in modular operations to store the carry–save format partial product and in Galois Field operations as two separate temporary values. A and B store the input operands used in all modular and Galois Field

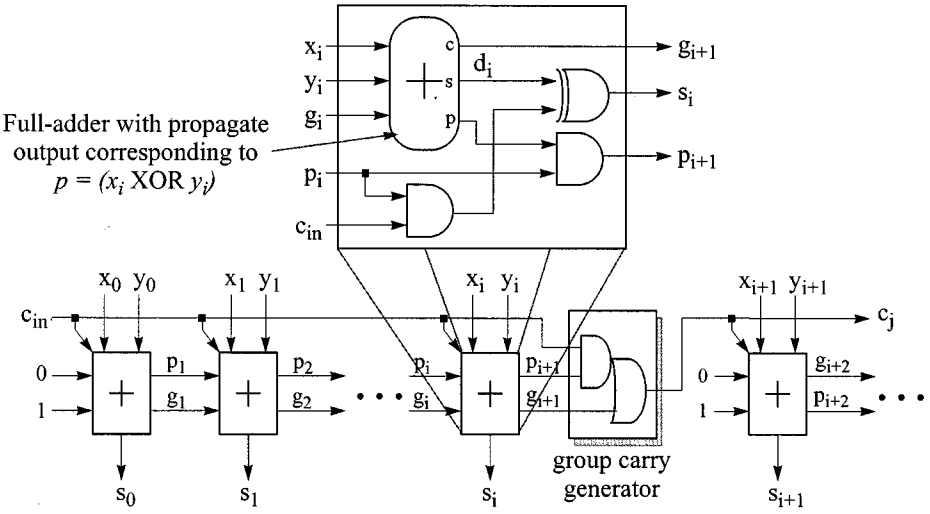

Fig. 4. Modified bitsliced carry–bypass adder [4].

operations. The Exp register is used for storing either the exponent value in the case of exponentiation operations or the multiplier value in the case of elliptic curve point multiplication. The N register also serves a dual purpose; for modular operations it is used as the modulus value, and in Galois Field operations, it stores the field polynomial in a binary vector form (e.g., x^7+x^2+1 is stored as [10000101]). In all relevant operations, it is assumed that both the Exp and N registers are preloaded with their required values.

Using local memory within the datapath eliminates the need to continually access the register file every cycle, eliminating the associated overhead of repeated register file accesses and minimizing the amount of reprogrammable interconnect by effectively isolating the reconfigurable logic from the rest of the processor. In addition, several operations requires four register reads and two writes in any given cycle, requiring additional read and write ports to be added to the register file. This would in turn increase the size of the register file, as well as its decoding complexity, thereby offsetting any advantage that might be gained by going to a unified memory model that eliminates the local memory.

The datapath utilizes three separate busses for distributing data between the various functional units: the two operand busses (`rs0` and `rs1`) and the writeback bus (`wb`). Not all registers and busses are interconnected, as analysis dictated that not all connections were required. The unnecessary connections are removed in order to minimize the capacitive load on the busses. `rs0` is also used as a secondary writeback bus to enable values within the datapath to be transferred between the local registers.

III. Algorithm Implementation

The DSRCP performs a variety of algorithms ranging from modular integer arithmetic to elliptic curve arithmetic over GF(2^n). All operations are universal in that they can be performed using any valid n-bit modulus ($8 \leq n \leq 1024$), GF(2^n) field polynomial and nonsupersingular elliptic curve over GF(2^n). Given the wide range of functionality, some explanation regarding how the various algorithms are implemented is warranted.

A. Modular Arithmetic

The various complex modular arithmetic operations (multiplication, reduction, inversion, and exponentiation) are implemented using microcode, while simple operations (addition and subtraction) are implemented directly in hardware using the wide adder and comparator units. Multiplication is performed using Montgomery multiplication [5], which computes the value MONTMULT $(A, B, N) = A \cdot B \cdot 2^{-n} \bmod N$. An additional Montgomery multiplication with a correction factor of $2^{2n} \bmod N$ is then performed to undo the division by 2^n inherent in Montgomery's method. The correction factor is assumed to be preloaded into the register file and is then specified via a third source operand (`rs2`) in the instruction word. Modular reduction is performed using a similar technique with Montgomery reduction at its core.

Modular inverses are computed using the extended binary euclidean algorithm [6]. This technique requires special architectural considerations, such as the ability to right shift the output of the adder unit, and explicit access to the LSB of R1, R2, and R3 in order to check the looping conditions of the algorithm.

Modular exponentiation is performed using a standard square-and-multiply algorithm [7] with an exponent scanning window of size two. The algorithm (Fig. 5) precomputes and stores the values $\{2^n, \texttt{rs0} \cdot 2^n, \texttt{rs0}^2 \cdot 2^n, \texttt{rs0}^3 \cdot 2^n\}$ in {R0, R1, R2, R3}, respectively. During each iteration, the current value is squared twice and then the exponent is scanned two bits at a time. Scanning is done nondestructively so exponent values need not be reloaded prior to each operation. The value read corresponds to the register that is used during the subsequent multiplication (e.g., if "01" is read, then R1 is used).

Note that multiplication by R0 is essentially a null operation (NOP) due to Montgomery multiplication's implicit division by 2^n. The use of NOPs provides protection from timing attacks[8], and simple power analysis [9] as a multiplication is always performed, thereby eliminating any variation in execution based

Input: rs0: n-bit register containing the value to be exponentiated
rs2: n-bit register containing the Montgomery correction value 2^{2n} mod N
length: 10-bit value representing the length of the exponent stored in Exp

Output: rd = $rs0^{Exp}$ mod N

Algorithm:

```
REG_MOVE(Ps,rs2)                        // Ps = 2^2n mod N
MONTRED( )                              // (Pc,Ps) = 2^n mod N
MOD_ADD(R0,Pc,Ps,0)                     // R0 = 2^n mod N
REG_MOVE(A,rs0)                         // A = rs0
REG_MOVE(B,rs2)                         // B = 2^2n mod N
MONTMULT( )                             // (Pc,Ps) = rs0·2^n mod N
MOD_ADD(R1,Pc,Ps,0)                     // R1 = rs0·2^n mod N
REG_MOVE(A/B,R1)                        // A,B = rs0·2^n mod N
MONTMULT( )                             // (Pc,Ps) = rs0^2·2^n mod N
MOD_ADD(R2,Pc,Ps,0)                     // R2 = rs0^2·2^n mod N
REG_MOVE(B,R2)                          // B = rs0^2·2^n mod N
MONTMULT( )                             // (Pc,Ps) = rs0^3·2^n mod N
MOD_ADD(R3,Pc,Ps,0)                     // R3 = rs0^3·2^n mod N
REG_MOVE(A/B,R0)                        // A,B = 2^n mod N
for (i=length-1;i>=0;i=i-2)
    MONTMULT( )                         // (Pc,Ps) = P^2·2^n mod N
    MOD_ADD(A/B,Pc,Ps,0)                // A,B = P^2·2^n mod N
    MONTMULT( )                         // (Pc,Ps) = P^4·2^n mod N
    MOD_ADD(A,Pc,Ps,0)                  // A = P^4·2^n mod N
    REG_MOVE(B,R<Exp[2i:2i-1]>)         // B = R<Exp[2i:2i-1]>
    MONTMULT( )                         // (Pc,Ps) = P^4+j·2^n mod N
    MOD_ADD(A/B,Pc,Ps,0)                // A,B = P^4+j·2^n mod N
endfor
MONTRED_A( )                            // (Pc,Ps) = P^Exp mod N
MOD_ADD(rd,Pc,Ps,0)                     // rd = P^Exp mod N
```

Fig. 5. Modular exponentiation algorithm implementation pseudocode.

on the exponent's value. The expense of this immunity is that conventional performance optimizations, such as skipping over strings of zeros in the exponent, cannot be exploited to speed up the operation. The loss in efficiency, in terms of the number of modular multiplications that must be performed due to this fixed performance, assuming that the exponent is uniformly distributed, is only 9%.

The use of the length operand in the MOD_EXP instruction enables the length of the exponent and the operands to be decoupled, leading to much more efficient exponentiation when the exponent value is significantly shorter than the operands, such as in public-key operations.

B. GF(2^n) Arithmetic

GF(2^n) addition is performed using the XOR function of the comparator unit, and both GF(2^n) multiplication and inversion are implemented directly in hardware using the reconfigurable datapath. GF(2^n) exponentiation is implemented in the same manner as modular exponentiation, with $\{1, \mathrm{rs0}, \mathrm{rs0}^2, \mathrm{rs0}^3\}$ being pre-computed and stored in {R0, R1, R2, R3}. NOPs are once again exploited to provide immunity to timing attacks and simple power analysis.

C. Elliptic Curve Arithmetic

The DSRCP performs affine-coordinate elliptic-curve operations on nonsupersingular elliptic curves over GF(2^n) of the form

$$E : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where $a, b \in \mathrm{GF}(2^n)$. The corresponding point addition and doubling formulae, assuming that P_1 and P_2 are distinct points on E, are given by

$$\begin{aligned} P_1 + P_2 = (x_3, y_3), x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= (x_2 + x_3)\lambda + x_3 + y_2 \\ \lambda &= \frac{(y_1 + y_2)}{(x_1 + x_2)} \end{aligned} \tag{2}$$

$$\begin{aligned} 2P_1 = (x_3, y_3), x_3 &= \lambda^2 + \lambda + a \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1 \\ \lambda &= x_1 + \frac{y_1}{x_1}. \end{aligned} \tag{3}$$

Note that the ISA of the DSRCP enables it to also perform elliptic-curve operations over fields of prime characteristic using an external sequencer and the appropriate formulae (e.g., [10]).

Point addition and doubling are implemented in microcode using the above formulae, with curve points stored as register pairs $(R_i, R_{i+1}) = (x, y)$. Point addition features an additional input in the form of a writeback enable bit which must be set for the result to be written back to the destination register pair. If the enable bit is not set, then the computation is performed and the result is discarded, leaving the destination register pair unaffected. This feature is used to provide immunity to timing attacks and simple power analysis during elliptic-curve point multiplication.

Point multiplication is performed using a repeated double-and-add algorithm, with a window size of one. Larger window sizes are not possible on the current DSRCP architecture due to

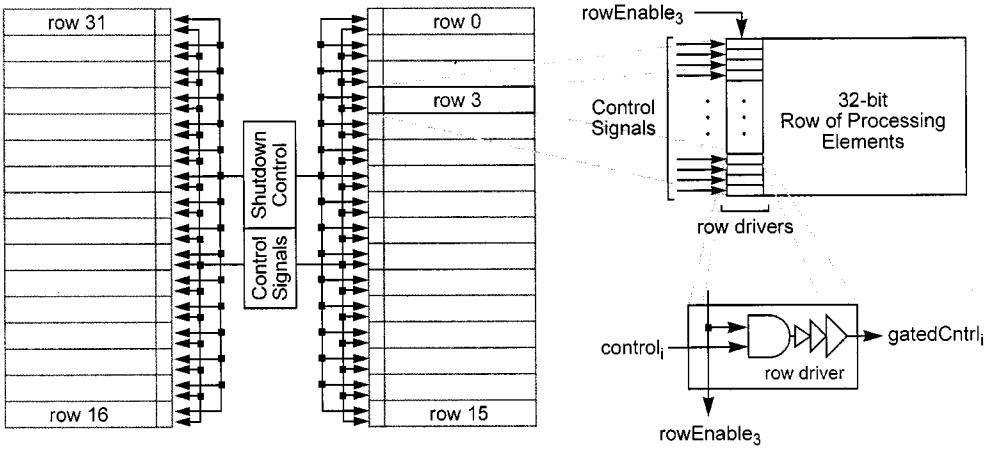

Fig. 6. Shutdown circuitry used in the DSRCP.

TABLE II
DSRCP INSTRUCTION MAPPING WITHIN THE CONTROL HIERARCHY

Tier	Instructions
I	ADD, SUB, COMP, GF_INV, GF_INVMULT, GF_MULT, GF_ADD, MONTRED, MONTMULT, MONTRED_A, SET_LENGTH
II	EC_DOUBLE, EC_ADD, MOD_ADD, MOD_SUB, GF_EXP
III	EC_MULT, MOD _MULT, MOD, MOD_INV, MOD_EXP

memory limitations of the register file (e.g., four precomputed values would require eight additional registers). The issue of timing attacks is once again addressed by using NOPs via the writeback enable bit of the point addition operation. The overhead associated with using NOPs is 33% relative to a conventional implementation where NOPs are skipped, and 50% if a signed radix-2 representation is used for the multiplier [7].

IV. IMPLEMENTATION

A. Controller and Microcode ROMs

The instruction set partitioning of the three-level control hierarchy is shown in Table II. The first tier of control corresponds to those instructions that are implemented directly in hardware. The second tier of control represents the first level of microcoded instructions that are composed of sequences of first-tier instructions. Similarly, the third tier of control represents instructions that consist of sequences of both first- and second-tier instructions.

Each microcode controller consists of a small ROM core, an input selector which gates the appropriate values onto the corresponding operand signals, and a control FSM that also serves as the ROM address generator. The resulting controllers emulate small microcontrollers. The microcode ROMs are implemented using static ROMs which eliminate the need for any precharged circuit techniques, making for a more robust implementation at the cost of requiring complementary word-select lines and larger bit cells due to the use of larger pMOS devices. However, given the small size of the ROMs and their relatively low duty cycle, the resulting energy and area overhead penalties are much less than 0.1% of the total DSRCP area and energy consumption.

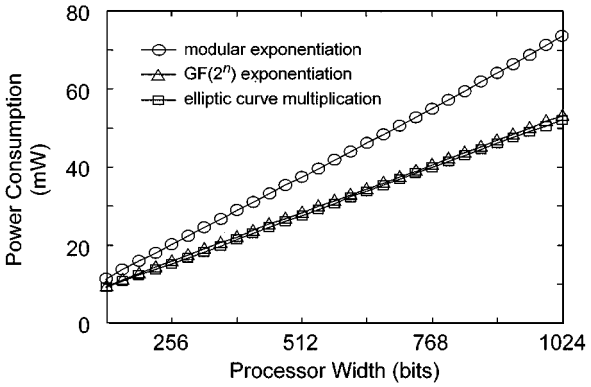

Fig. 7. Power consumption as a function of processor width.

B. Shutdown Controller

The shutdown controller is capable of shutting down the datapath row by row, in 32-b increments using both clock and control signal gating, which is performed using simple AND structures in the row drivers that are found along the inside edge of the two halves of the datapath, as shown in Fig. 6. All clock gating signals are generated off the falling edge of the main clock to ensure that edge-triggered signals generated from the main clock (e.g., register file clocks) are gated during the low phase of the clock to eliminate any spurious glitches that may occur by ANDing the clock with a late-arriving enable signal while the clock is high.

The result of this shutdown strategy is a linear reduction in power consumption as a function of the datapath width, as illustrated in Fig. 7.

There is a subtle feature of the shutdown control scheme, due to the way Galois Field multiplication is performed within the DSRCP, that warrants additional explanation. When performing operations over the field GF(2^n), the field polynomial is an nth degree polynomial that is stored as an $(n+1)$ bit value. Hence, enabling only the least significant n bits of the datapath may result in errors, as the effects of the MSB may not be accounted for if the MSB lies within a disabled portion of the datapath. This condition occurs when n is a multiple of 32, so the shutdown controller detects this condition and enables an additional 32-b block. Given the operand sizes that are typically used when

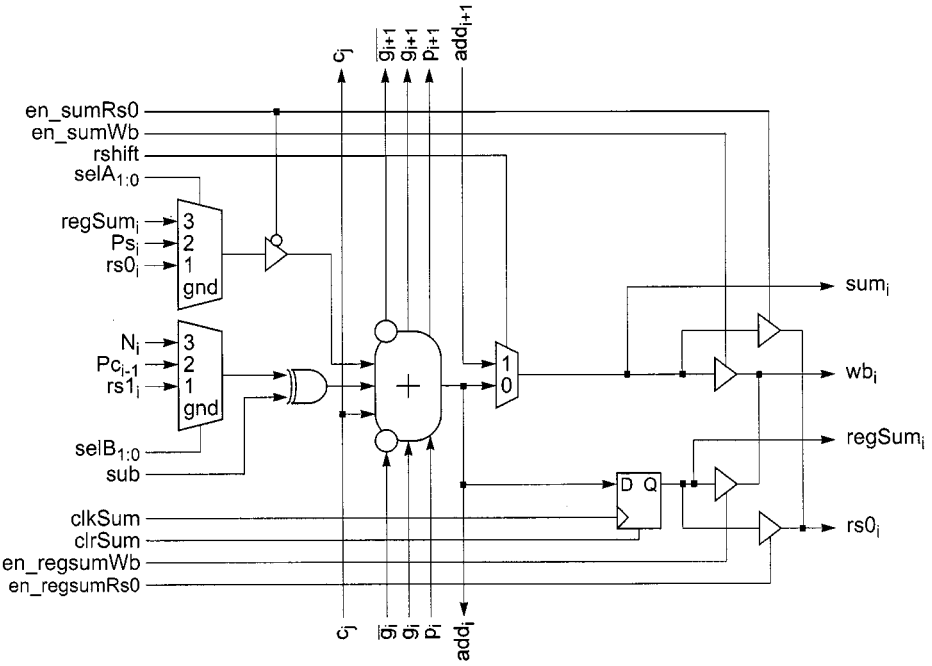

Fig. 8. Adder unit bitslice.

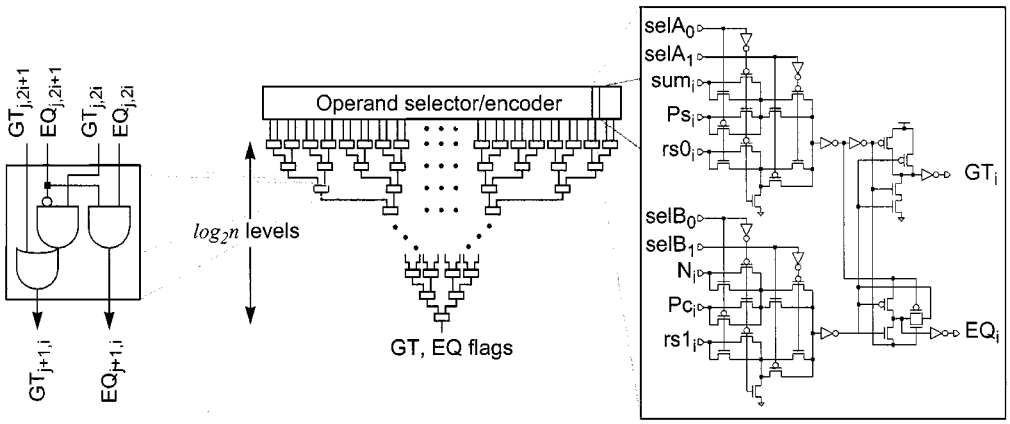

Fig. 9. Tree-based magnitude comparator topology used in the DSRCP.

this condition will occur (512–1024-b), the overhead associated with enabling an additional datapath block is on the order of 3%–6% extra energy consumption.

C. I/O Interface

The processor's floorplan is based on two banks of processing elements (PEs), each with 16 rows of 32 processing elements, as shown in Fig. 6. Each bank contains a set of 32-bit-wide vertically routed input and output busses. Separate input and output busses are used to enable static bus repeaters/latches to be inserted into the busses at the vertical midpoint of the two banks, allowing the bus to be segmented in order to minimize the capacitive load seen by any given driver on the bus. This allows minimum sized drivers to be used and eliminates unnecessary charging/discharging of large portions of the bus capacitance by near-end drivers. The serpentine distribution of PEs within the datapath causes each row to be flipped in relation to those above and below it. A single level of output muxes at the chip interface is used to reverse the order of both the input and output busses as required to provide a consistent 32-b interface at the pads.

D. Reconfigurable Datapath—Register File

The register file is implemented using TSPC-style registers [11]. A more typical SRAM-based register file design was not used due to the small number of registers required and the increased robustness of using an edge-triggered memory element. The drawback of this approach is an increase in both area and energy consumption. The energy consumption penalty is negligible as the register file is accessed very infrequently due to the local data storage in the reconfigurable logic unit. The area penalty is more significant as the TSPC register is twice as large as a simple 6T SRAM cell. Given that the register file represents 20% of the bitslice area, the area overhead is 10%, which is deemed acceptable for this application.

The register outputs are driven onto the `rs0` and `rs1` source operand busses via two 8-to-1 passgate multiplexors and their

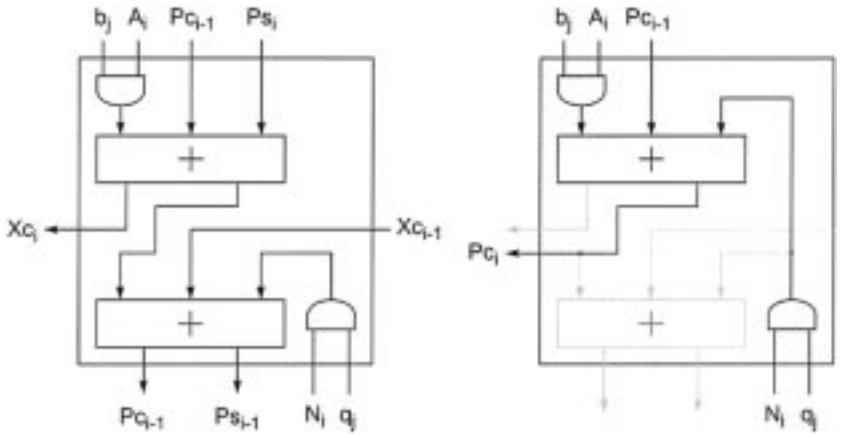

Fig. 10. Multiplier architectures for (a) modular multiplication and (b) GF(2^n) multiplication.

Input: W: a, the element of GF(2^n) that is to be inverted
X: N, the binary representation of the field polynomial $f(x)$
Y: b, the element of GF(2^n) that is to be multiplied by the computed inverse
Z: 0, just plain old zero!

Output: Z = b/a

Algorithm:

```
while (W != 0)
    while (W0 == 0) || (X0 == 0)
        if (W0 == 0)                    if (X0 == 0)
            W = W/2;                        X = X/2;
            Y = (Y + Y0·N)/2;               Z = (Z + Z0·N)/2;
        endif                           endif
    endwhile
    if (W >= X)
        W = W + X; Y = Y + Z;
    else
        X = W + X; Z = Y + Z;
    endif
endwhile
```

Fig. 11. Extended binary euclidean algorithm used for GF(2^n) inversion operation.

inputs are all connected to the wb writeback bus. The eight registers feature individual clock and reset lines, with the clock lines also serving as the writeback register select lines. As mentioned before, the register file features architectural features to improve the efficiency of the modular inversion operation by having R0 having a reset value of 1 and providing the LSBs of R0, R1, R2, and R3 to the global control logic.

E. Reconfigurable Datapath—Wide Adder Unit

The design requirements for the DSRCP call for a wide adder capable of performing 1024-b binary addition/subtraction in at most three processor cycles, using an area-efficient bitsliced implementation with a minimal amount of long interconnect. The area and interconnect requirements precluded the use of conventional structures such as carry–lookahead, hierarchical carry–select, and carry–bypass/skip implementations. However, the modified carry–bypass/skip adder proposed in [4] yields a critical path of approximately 45 full adder delays for a 1024-b operation, while mapping to a very efficient bitsliced implementation. The main difference between this adder and that of a conventional implementation is the serialization of the group propagation signal generation within the bitslices of the group. Distributing the propagation signal generation in this manner eliminates the need to have a wide fan-in AND gate and allows each bit within the group to determine whether the group carry-in will affect its output. Hence, each block can generate its valid sum outputs one XOR delay after the carry-in is valid. By matching delays through proper group sizing, the carry-in becomes valid just as the group propagate and generate signals are valid, leading to the minimal overall adder delay.

The adder unit bitslice is shown in Fig. 8. The adder consists of the aforementioned modified carry–bypass/skip adder cell, a local register for storing intermediate results and multiplexors for both input operand selection and right shifting of the result. Both the output of the adder (sum_i) and its registered version (regSum_i) can be driven onto either the rs0 or writeback busses. The B input selection muxes utilize a left-shifted version of the Pc operand to simplify the conversion of the redundant carry–save value stored in (Pc,Ps) into a nonredundant binary form. Note that the A operand's signal path includes a tristate buffer which is required to eliminate the race condition that results when the A operand is read from the rs0 bus and the

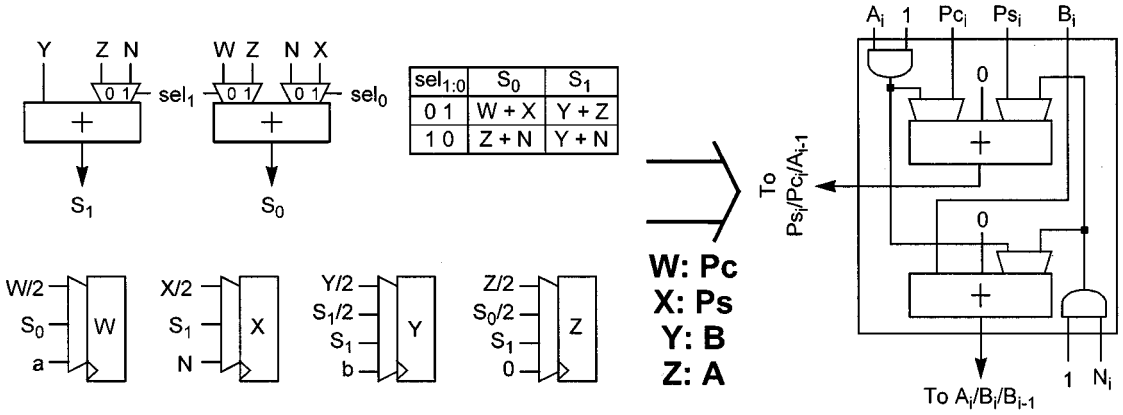

Fig. 12. Basic GF(2^n) inversion architecture and resulting datapath cell.

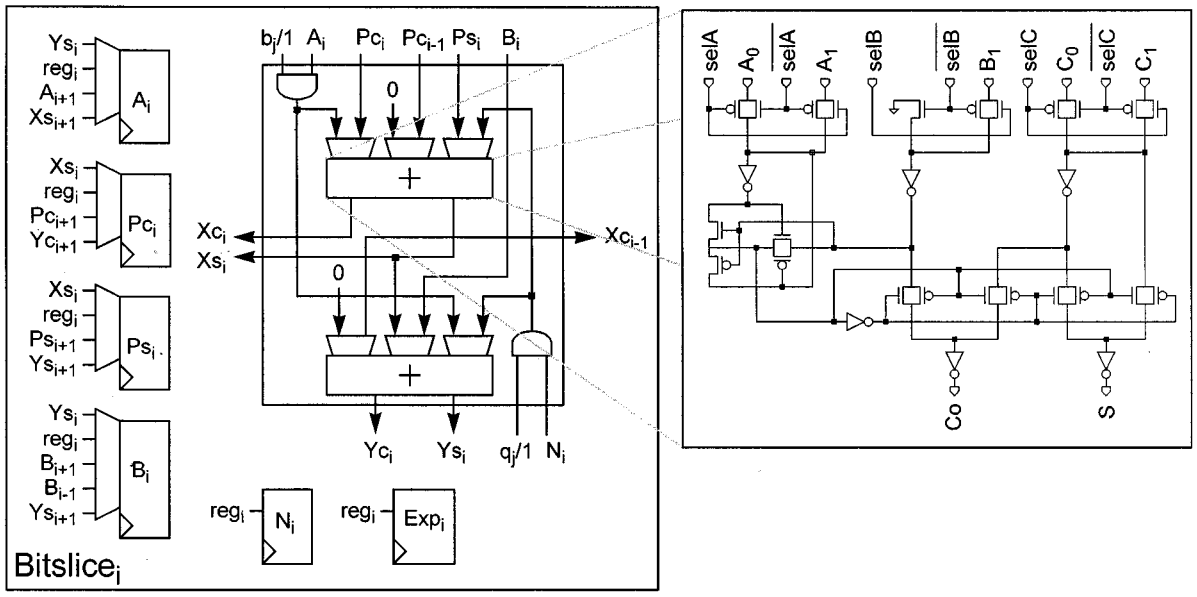

Fig. 13. Reconfigurable logic bitslice.

adder's nonregistered output is then driven onto the same bus. The tristate buffer breaks the feedback path.

F. Reconfigurable Datapath—Wide Comparator Unit

The DSRCP controller utilizes the wide comparator unit outputs for determining branch conditions within a microcoded instruction's execution. Hence, to eliminate branch delays the processor requires that two 1024-b operands be compared within a single processor cycle. This is accomplished using the fast tree-based comparator circuit shown in Fig. 9 which is capable of comparing two n-bit operands in $O(\log_2 n)$ gate delays. The comparator first encodes the inputs based on a bit-by-bit comparison of the two operands to form the signals $EQ_i = \text{op1}_i \oplus \text{op2}_i$ and $GT_i = \text{op1}_i \cdot \overline{\text{op2}_i}$. Once in this form, two adjacent encoded bit positions can be compared using the relations $EQ_{j+1,i} = EQ_{j,2i} \cdot EQ_{j,2i+1}$ and $GT_{j+1,i} = GT_{j,2i+1} + GT_{j,2i} \cdot \overline{EQ_{j,2i+1}}$, the outputs of which are passed to the next level of the comparator tree. At each stage, the number of comparisons are halved, hence the tree has depth $\log_2 n$.

The comparator is partitioned into 32 32-b sections, or one per row. The final stage of each of these 32 comparator blocks utilizes an enable signal that either performs the aforementioned comparison if the row is enabled, or outputs an equal signal in the event that the row has been disabled to prevent any data remaining in the upper unused portions of the register from corrupting the comparison.

G. Reconfigurable Datapath—Reconfigurable Logic Cell

The DSRCP is capable of performing a variety of algorithms using both conventional and modular integer fields, as well as binary Galois Fields. These operations are implemented using a single computation unit that can be reconfigured on the fly to perform the required operation. The possible configurations are Montgomery multiplication/reduction, GF(2^n) multiplication, and GF(2^n) inversion. All other operations are either handled by other units (e.g., the fast adder and comparator), or implemented in microcode.

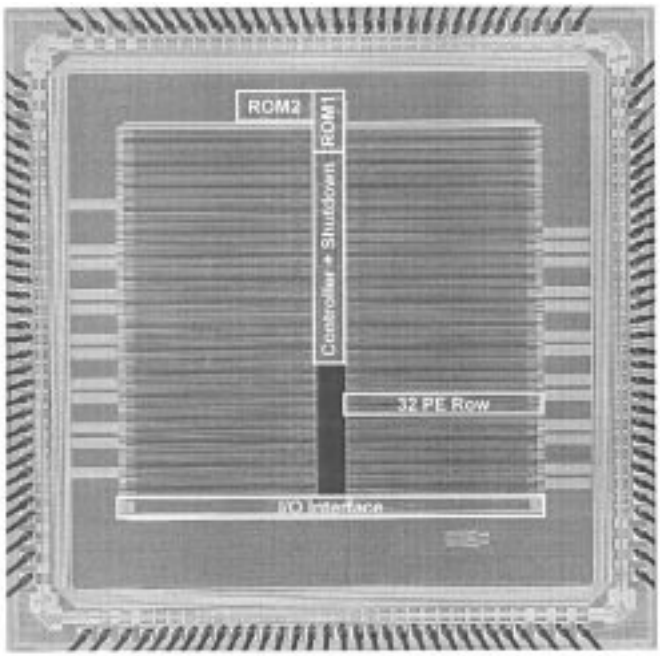

Fig. 14. DSRCP die photograph.

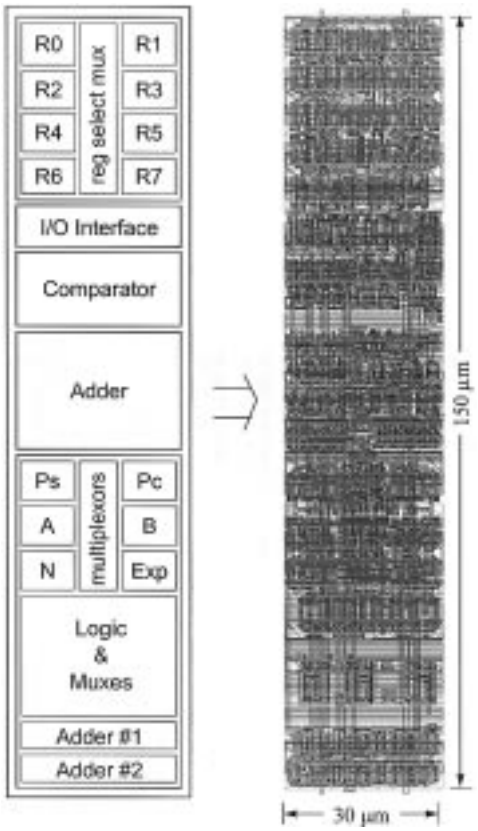

Fig. 15. DSRCP bitslice layout.

Montgomery multiplication utilizes the simple iterated radix-2 implementation

$$(P_c, P_s)_{j+1} = \frac{(P_c, P_s)_j + b_j A + q_j N}{2}, \quad j = 0, \ldots, (n-1) \tag{4}$$

where $q_j = P_{s_0} \oplus b_j A_0$ and b_j is the j th bit of operand B. A redundant carry–save representation of the partial product accumulator (Pc,Ps) is exploited in order to minimize the cycle time. This operation can be implemented using the basic computational resources of Fig. 10(a): two full adders and two AND gates. Montgomery reduction of A can be performed by setting $B = 1$ (i.e., $b_0 = 1$, $b_i = 0$, $i = 1, \ldots, n-1$). Similarly, reduction of (Pc, Ps) can be performed by setting $B = 0$.

Mastrovito's thesis [12] serves as an extensive reference of hardware architectures for performing GF(2^n) multiplication. Given our choice of a polynomial basis, the most efficient multiplier architecture is an MSB-first approach as it minimizes the number of registers that are clocked in any given cycle. In addition, the MSB-first approach can be mapped to the existing hardware of the Montgomery multiplier [Fig. 10(b)] by exploiting the fact that a full adder's sum output computes a three-input addition. Hence, GF(2^n) multiplication can be performed using the iteration

$$Pc_j = 2 \cdot Pc_{j-1} \oplus A \cdot b_{n-j-1} \oplus q_j N \tag{5}$$

where q_j is bit $n-1$ of Pc_{j-1}, which is used to modularly reduce the partial product Pc_j. The field polynomial $f(x)$ is stored as a binary vector in N and the resulting approach is universal in the sense that it can operate with any valid field polynomial over GF(2^n) for $8 \leq n \leq 1024$.

The limiting operation in affine-coordinate elliptic-curve point operations is typically the inversion operation. In hardware using a polynomial basis, the extended binary euclidean algorithm [6] can be used to compute inverses in a very efficient manner (Fig. 11). The basic algorithm is modified to perform a multiplication concurrently with the inversion by initializing the Y variable to be the multiplier value (if no multiplication is required, the Y register can simply be initialized with the value 1). This optimization provides significant savings during elliptic-curve point operations as it eliminates one multiplication, reducing the total cycle count by approximately 18%. The resulting algorithm combines two embedded `while` loops into a single parallel operation, which effectively halves the number of cycles required as the dominant portion of time is spent in this part of the algorithm. The net result of these optimizations is a universal GF(2^n) invert-and-multiply operation that takes at most four multiplication times ($T_{\text{mult}} = n$ cycles) and on average $3.3 \cdot T_{\text{mult}}$ in order to invert (and multiply) an element of GF(2^n).

Inversion is implemented using the same datapath cell used in both Montgomery and GF(2^n) multiplication by providing a small degree of reconfigurability such that computational resources can be reused to perform different parts of the algorithm. The basic requirements are two two-input adders over GF(2) to perform each of the parallel `if` operations and the two summations in each branch of the final `if` clause. Each iteration of the inner `while` loop requires one cycle as all operations are performed in parallel. An additional cycle is incurred when the exit condition of the inner `while` loop is satisfied (i.e., $W_0 = X_0 = 1$), as it must be detected via an additional iteration of the loop. The second part of the algorithm requires a single

TABLE III
REPORTED IMPLEMENTATIONS OF MODULAR EXPONENTIATION FUNCTIONS

Design	Power Consumption (W)	Operand Size (bits)	Time per Operation (ms)	Cycles per Operation (cycles/bit^2)	Clock Rate (MHz)	Energy per Operation (mJ/op)
Ishii [17]	2	1024/512	100/25	3.81/3.81	40	200/50
Ivey [18]	-	512	< 8	< 4.58	150	-
Orup [19]	-	512	5	0.48	25	-
Chen [20]	-	512	21	4.01	50	-
Yang [21]	-	512	4.3	2.05	125	-
Guo [22]	-	512	1.8	0.98	143	-
Leu [23]	-	512	4.6	2.02	115	-
Royo [24]	-	768	10.6	0.90	50	-
Satoh [4]	0.33	1024	23	0.99	45	7.59
Vandemeulebroecke [25]	0.5	1024	125	2.98	25	62.5
Shand [26]	-	1024/512	6/0.85	0.23/0.13	40	-
Yuliang [27]	-	1024	650	12.40	20	-
Motorola MPC-180	0.6	1024	32	2.01	66	19.2
Pijnenberg PCC-ISES	1	2048/1024	6.02/1.45	0.07/0.07	50	6.02/1.45
Broadcom BC5820	"Low"	1024	1	0.1	100	-
DSRCP	**< 0.075**	**1024/512**	**32.1/8.2**	**1.53/1.56**	**50**	**2.41/0.37**
DSRCP w/CRT[28]	**< 0.045**	**1024/512**	**17/4.5**	**0.81/0.86**	**50**	**0.77/0.11**

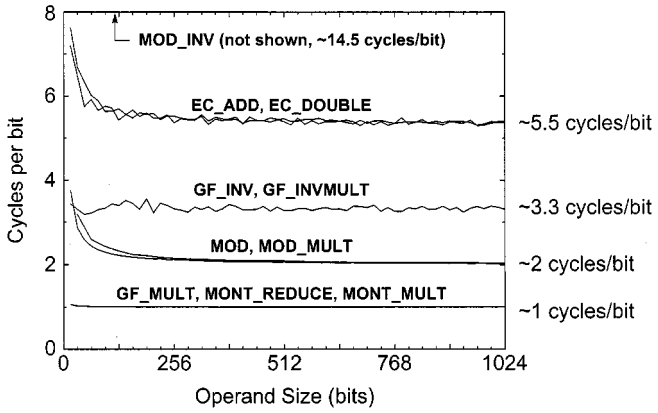

Fig. 16. Performance of several DSRCP arithmetic instructions.

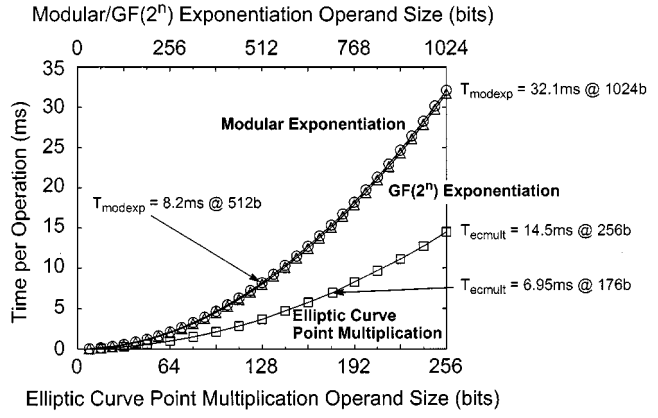

Fig. 17. Performance of various cryptographic primitives using the DSRCP at 50 MHz.

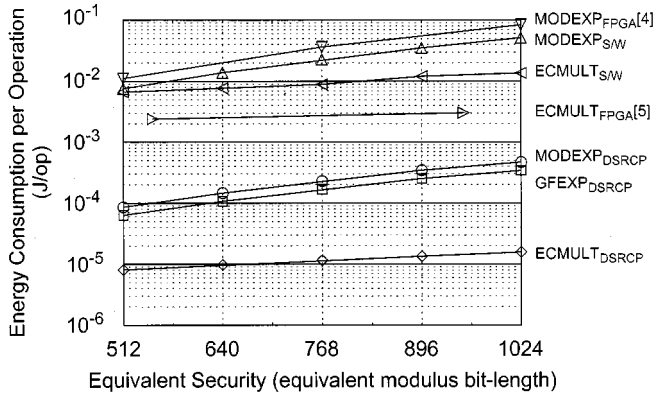

Fig. 18. Comparison of energy consumption per operation for software, FPGA, and DSRCP.

cycle as well. The two datapath adders can be used as two-input $\mathrm{GF}(2)$ adders by zeroing one of their inputs and then utilizing multiplexors to allow the adder inputs to be changed on the fly. The corresponding architecture and its resulting mapping to the datapath cell is shown in Fig. 12.

The final reconfigurable datapath cell is shown in Fig. 13 and contains two reconfigurable full adders, two AND gates, and six local register cells with multiplexed inputs. The reconfigurable adders are implemented using high-performance small-area pass-transistor-based full-adder cells with multiplexed inputs. The adder and register reconfiguration muxes are configured through the use of eight control lines, three for the adder muxes and five for the register muxes, that are exposed to the control hardware, allowing for single-cycle reconfigurability.

V. Experimental Results and Evaluation

The processor is fabricated in a 0.25-μm CMOS technology with five levels of metallization. Fig. 14 depicts a microphotograph of the processor whose core contains 880 000 devices and measures 2.9×2.9 mm^2. The datapath consists of 1024 processing bitslices, each of which measures 30×150 μm^2 (Fig. 15). At 50 MHz, the processor operates at a supply voltage of 2 V and consumes at most 75 mW of power. In ultralow-power mode (3 MHz at $V_{DD} = 0.7$ V), the processor consumes at most 525 μW.

Fig. 16 shows the performance of those DSRCP instructions whose execution time is proportional to the size of the operands. The results are normalized relative to the operand size in order to better illustrate this proportionality. The performance of the cryptographic primitives required for IF, DL, and EC-based cryptography are shown in Fig. 17. Important performance points are denoted and compared with other reported implementations in Table III. The DSRCP's performance compares favorably; although several solutions quote higher rates, they represent dedicated solutions with no algorithm agility. For those dedicated solutions that report their power consumption, the energy consumption per operation of the DSRCP is found to be at least a factor of two better.

Fig. 18 demonstrates the energy efficiency of the DSRCP relative to software-based implementations on the StrongARM SA-1100 [13] and previously reported programmable-logic-based implementations ([14], [15]) on Xilinx XC4000 parts. The software-based energy consumption is measured using a StrongARM SA-1100 evaluation platform that is executing hand-optimized assembly language implementations of the various cryptographic primitives. The FPGA-based energy consumption is estimated using the implementation details provided in [14] and [15] and the power consumption guidelines described in [16]. The DSRCP is approximately two to three orders of magnitude more energy efficient than both software and programmable-logic-based solutions, while providing the same degree of flexibility and algorithm agility.

VI. Conclusion

Given a specific domain of functionality such as public-key cryptography, it is possible to provide a limited degree of domain-specific reconfigurability to provide flexibility while minimizing the overhead that is typically associated with reprogrammable logic. Domain specific integrated circuits (DSICs) utilize interconnect-centric architectures to exploit locality in order to minimize the interconnection overhead, which is the dominant source of energy consumption in generic reconfigurable logic.

The resulting public-key cryptography DSIC provides a comparable level of performance and twice the energy efficiency as previously reported dedicated hardware solutions, while providing all of the flexibility of a software-based implementation. In addition, the processor is two to three orders of magnitude more energy efficient than both optimized software and reprogrammable-logic-based implementations.

Acknowledgment

The authors would like to thank the National Semiconductor Corporation for providing fabrication facilities.

References

[1] *Standard Specifications for Public Key Cryptography—Draft 13*, IEEE P1363, Nov. 1999.
[2] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," in *Proc. 1998 Int. Symp. Low Power Electronic Design (ISLPED)*, pp. 155–160.
[3] "XC4000 field programmable gate arrays: programmable logic databook," Xilinx Corp., San Jose, CA, 1996.
[4] A. Satoh *et al.*, "A high-speed small RSA encryption LSI with low power dissipation," in *Proc. 1st Int. Information Security Workshop (ISW'97)*, pp. 174–187.
[5] P. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 48, pp. 243–264, 1987.
[6] D. E. Knuth, *The Art of Computer Programming—Seminumerical Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1981, vol. 2.
[7] D. M. Gordon, "A survey of fast exponentiation methods," *J. Algorithms*, vol. 27, no. 1, pp. 129–146, Apr. 1998.
[8] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems," in *Proc. Advances in Cryptology (CRYPTO'96)*, pp. 104–113.
[9] P. Kocher, J. Jaffe, and B. Jun. (1998) Introduction to differential power analysis and related attacks. [Online]. Available: http://www.cryptography.com/dpa/technical
[10] G. Seroussi, N. P. Smart, and I. F. Blake, *Elliptic Curve Cryptography*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[11] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *IEEE J. Solid-State Circuits*, pp. 62–70, Feb. 1989.
[12] E. D. Mastrovito, "VLSI architectures for computation in Galois fields," Ph.D. dissertation, Linköping Univ., Linköping, Sweden, 1991.
[13] "StrongARM SA-1100 microprocessor for portable applications brief datasheet," Intel Corp., Chandler, AZ, 1999.
[14] T. Blum, "Modular exponentiation on reconfigurable hardware," Master's thesis, Worcester Polytechnic Inst., Worcester, MA, 1999.
[15] M. C. Rosner, "Elliptic curve cryptosystems on reconfigurable hardware," Master's thesis, Worcester Polytechnic Inst., Worcester, MA, 1998.
[16] "A simple method of estimating power in XC4000Xl/EX/E FPGAs," Xilinx Corp., San Jose, CA, Application Brief XBRF 014 v1.0, 1997.
[17] S. Ishii, K. Ohyama, and K. Yamanaka, "A single-chip RSA processor implemented in a 0.5-mm rule gate array," in *Proc. 7th Annu. IEEE Int. ASIC Conf. Exhibit*, 1994, pp. 433–436.
[18] P. A. Ivey, S. N. Walker, J. M. Stern, and S. Davidson, "An ultrahigh-speed public-key encryption processor," in *Proc. IEEE 1992 Custom Integrated Circuits Conference (CICC'92)*, pp. 19.6.1–19.6.4.
[19] H. Orup, E. Svendsen, and E. Andreasen, "VICTOR: an efficient RSA hardware implementation," in *Proc. Advances in Cryptography (EUROCRYPT'90)*, pp. 245–252.
[20] P. -S Chen, S. -A Hwang, and C. -W Jen, "A systolic RSA public-key cryptosystem," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS'96)*, pp. 408–411.
[21] C.-C. Yang, T.-S. Chang, and C.-W. Jen, "A new RSA cryptosystem hardware design based on montgomery's algorithm," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 908–913, July 1998.
[22] J.-H. Guo, C.-L. Wang, and H.-C. Hu, "Design and implementation of an RSA public-key cryptosystem," in *Proc. 1999 IEEE Int. Symp. Circuits and Systems (ISCAS'99)*, vol. 1, pp. 504–507.
[23] J.-Y. Leu and A.-Y. Wu, "A scalable low-complexity digit-serial VLSI architecture for RSA cryptosystem," in *1999 IEEE Workshop Signal Processing Systems (SIPS'99)*, pp. 586–595.
[24] A. Roya, J. Moran, and J. C. Lopez, "Design and implementation of a co-processor for cryptography applications," in *Proc. 1997 European Design and Test Conf. (ED&TC'97)*, pp. 213–217.
[25] A. Vandemeulebroecke, "A single-chip 1024-bits RSA processor," in *Proc. Advances in Cryptology (EUROCRYPT'89)*, pp. 219–236.
[26] M. Shand and J. Vuillemin, "'Fast implementations of RSA cryptography," in *Proc. 11th IEEE Symp. Computer Arithmetic*, 1993, pp. 252–259.
[27] D. Yuliang, M. Zhigang, Y. Yizheng, and W. Tao, "Implementation of RSA cryptoprocessor based on Montgomery algorithm," in *Proc. 5th Int. Conf. Solid-State and Integrated Circuit Technology*, 1998, pp. 524–526.
[28] H. Garner, "The residue number system," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 140–147, 1959.

James Goodman (S'97–M'00) received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1994. He received the S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1996 and 2000, respectively. His primary research areas were reconfigurable energy-efficient architectures for implementing cryptographic algorithms and protocols.

He is currently with Chrysalis-ITS, Ottawa, ON, Canada, as Senior IC Security Architect, developing next-generation integrated-security solutions for a variety of applications. He is a member of the Signal Processing Subcommittee for the ISSCC 2002, the Technical Program Committee for VLSI 2001, and the Program Committee for the Workshop on Cryptographic Hardware and Embedded Systems (CHES).

Dr. Goodman received the 1999 Design Automation Conference Design Contest Award.

Anantha P. Chandrakasan (S'92–M'95) received the B.S, M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1989, 1990, and 1994, respectively.

Since September 1994, he has been with the Massachusetts Institute of Technology, Cambridge, where he is currently an Associate Professor of electrical engineering and computer science. He held the Analog Devices Career Development Chair from 1994 to 1997. His research interests include the ultralow-power implementation of custom and programmable digital signal processors, distributed wireless sensors, multimedia devices, emerging technologies, and CAD tools for VLSI. He is a co-author of the book *Low Power Digital CMOS Design* (Norwell, MA: Kluwer) and a co-editor of *Low Power CMOS Design* and *Design of High-Performance Microprocessor Circuits* (New York: IEEE Press). He has served on the technical program committee of various conferences including the ISSCC, VLSI Circuits Symposium, DAC, and the International Symposium on Low-power Electronics and Design (ISLPED). He has served as a technical program co-chair for the 1997 ISLPED, VLSI Design '98, and the 1998 IEEE Workshop on Signal Processing Systems, and as a general co-chair of the 1998 ISLPED. He served as an elected member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the Signal Processing Society. He was the Signal Processing Sub-committee chair for ISSCC 1999 through 2001. He is Program Vice-Chair for the ISSCC 2002.

Dr. Chandrakasan received the NSF Career Development award in 1995, the IBM Faculty Development award in 1995, and the National Semiconductor Faculty Development award in 1996 and 1997. He has received several best paper awards, including the 1993 IEEE Communications Society's Best Tutorial Paper Award, the IEEE Electron Devices Society's 1997 Paul Rappaport Award for the Best Paper in an EDS publication during 1997, and the 1999 Design Automation Conference Design Contest Award. He was an Associate Editor for the IEEE JOURNAL OF SOLID-STATE CIRCUITS from 1998 to 2001.