A Hyperelliptic Curve Crypto Coprocessor for an 8051 Microcontroller

Alireza Hodjat, David Hwang
Electrical Engineering Department
University of California, Los Angeles
Los Angeles, USA
ahodjat@ee.ucla.edu and dhwang@ee.ucla.edu

Lejla Batina, Ingrid Verbauwhede
COSIC Group
Katholieke Universiteit Leuven
Leuven, Belgium
lbatina@esat.kuleuven.ac.be and ingrid@ee.ucla.edu

Abstract—**This paper presents a microcode instruction set coprocessor which is designed to work with an 8-bit 8051 microcontroller and implements a Hyperelliptic Curve Cryptosystem (HECC). The microcode coprocessor is capable of performing a range of Galois Field operations using a dual-multiplier/dual-adder datapath and storing the intermediate results in the local storage unit of the coprocessor (RAM). This coprocessor is programmed using the software routines from the 8051 microcontroller which implements the HECC divisor's doubling and addition operations. The Jacobian scalar multiplication was computed in a 656 msec (7.87 M cycles) at 12 MHz clock frequency.**

I. INTRODUCTION

High speed implementation of Public Key Cryptography (PKC) is required for providing security in various communication systems. The best-known and most commonly used public-key cryptosystem is RSA [1]. However, it is not a feasible solution for low-power and low foot-print devices. Emerging areas such as RFID tags and sensor networks put new requirements on implementations of PKC algorithms with firm constraints in terms of number of gates, power, bandwidth, etc. A promising candidate appears to be a Hyper/Elliptic Curve Cryptosystem (H/ECC), but the previously mentioned requirements can probably be achieved only with the synergy of hardware and software. ECC has already proven its potential as it offers shorter certificates, lower power consumption and better performance on some platforms. In addition, ECC offers more "security per bit" than RSA, as no sub-exponential algorithm is known that solves the discrete logarithm problem in this group. However, HECC maintains all those advantages with even shorter bit-lengths. More precisely, the operand size for HECC is at least a factor of two smaller than the one of ECC, with the same level of security. This fact makes HECC a very good choice for platforms with limited resources.

Algorithms for HECC and their implementations have been studied intensively in the past years. A significant amount of work has been performed on optimizing the formulae for the group operation [2, 4, 5, 7]. Explicit formulae for genus 2 curves are given by Lange [2] for arbitrary fields and for various types of coordinates. For embedded processors, a large amount of work is performed for the ARM platform [3, 9, 10]. Pelzl et al. [9] implemented the group operation of genus 2 and 3 for HECC on an ARM7 processor. They compared the results with ECC implementation (with corresponding security) and showed that HECC performance is comparable to the one of ECC. The performance for divisor scalar multiplication on the ARM microprocessor for genus 2 was further optimized in [10] and compared to genuses 3 and 4. Gura et al. [11] compared ECC and RSA on 8-bit CPUs and proved that Public-Key Cryptography is viable on small devices, with the results favoring ECC substantially.

The first complete hardware implementation of HECC was given by Boston [6]. They used Cantor's algorithm [8] to implement HECC on the VirtexII FPGA. Wollinger et al. investigated HECC implementation on a VLSI coprocessor [12, 13]. In [14] three different architectures on a FPGA have been examined for a vast area of applications. Most of the published work dealt with binary fields. The only exception is work of Baktır et al. [3] which investigated implementation over an extension field of odd characteristic i.e. over Optimal Tower Fields (OTF) on an ARM7.

This paper presents a microcode instruction set coprocessor which is designed to work with an 8-bit 8051 microcontroller to implement a Hyperelliptic Curve Cryptosystem. More precisely, we have implemented the HECC divisor multiplication operation on the 8051 microprocessor, which uses a hardware coprocessor to optimize the performance. This extra hardware is a coprocessor with dual-multiplier/dual-adder datapath, which allows for a speed-up of factor 228 when compared with the software-only solution. We have re-written the formulae of Byramjee and Duquesne [7] to facilitate the divisor operations in this special case. In this way we achieved optimized divisor doubling and addition. Namely, we take advantage of a special dual-multiplier/dual-adder datapath, which allowed us to explore the parallelism in field multiplications.

The remainder of this paper is organized as follows. In section 2 some background information on HECC is given. Details of our implementation are specified in section 3. Section 4 gives details of a microcode instruction set coprocessor. Results are listed in section 5 and conclusions are given in section 6.

II. HYPERELLIPRIC CURVE CRYPTOGRAPHY

Hyperelliptic Curve Cryptography was proposed in 1988 by Koblitz [15] as a generalization of Elliptic Curve Cryptography. In particular, elliptic curves can be viewed as a special case of hyperelliptic curves i.e. an EC is an HEC with genus $g=1$.

A. Hyperelliptic curves

Here we consider a hyperelliptic curve C of genus $g=2$ over GF(2^m), which is defined by an equation of the form:

$$C: y^2 + h(x)\cdot y = f(x) \text{ in } GF(2^m)[x,y],$$

where $h(x) \in GF(2^m)$ is a polynomial of degree at most g $(deg(h) \leq g)$ and $f(x)$ is a monic polynomial of degree $2g + 1$ $(deg(f) = 2g + 1)$. There are some more conditions that have to be fulfilled. More details can be found in [16]. For genus 2 curves, in the general case the following equation is used:

$$y^2 + (h_2x^2+h_1x+h_0)\cdot y = x^5+f_4x^4+f_3x^3+f_2x^2+f_1x+f_0.$$

For our implementation we used so-called type II curves [7], which are defined by $h_2 = 0, h_1 \neq 1$. In particular, the authors of [7] recommend curves of the form:

$$y^2 + x.y = x^5+f_3x^3+x^2+f_0,$$

since they combine simpler arithmetic with a good security level. More precisely, those curves allow for much faster divisor doubling while addition stays the same as for a general curve.

Now we introduce a group structure for specific objects created on a hyperelliptic curve. A divisor D is a formal sum of points on the hyperelliptic curve C. Let Div denote the group of all divisors on C and Div_0 the subgroup of Div of all divisors with degree zero. The Jacobian J of the curve C is defined as the quotient group $J = Div_0/P$. Here P is the set of all principal divisors, where a divisor D is called principal if $D = div(f)$, for some element f of the function field of C. In practice, the Mumford representation is typically used; in this representation each divisor is represented as a pair of polynomials $[u,v]$. Here, u is monic of degree 2, $deg(v) < deg(u)$ and $u \mid f\text{-}hv\text{-}v^2$ (so-called reduced divisors). For implementations of HECC, we need to implement the multiplication of elements of the Jacobian i.e. divisors with some scalar.

B. HECC algorithms

Following a top-down approach, the highest-level operation is the divisor scalar multiplication. It is implemented by the use of the so-called "non-adjacent form" i.e. as the NAF algorithm [17], which has the lowest weight among all other signed digit representations. The fact that the subtraction of divisors is as expensive as the divisor addition makes this representation beneficial. In this way the scalar multiplication is implemented as a sequence of divisor additions/subtractions and doublings. We use projective coordinates which allow us to complete all divisor operations without inversion. Only one inversion and four multiplications are required at the end to convert back from projective to affine coordinates. We have re-written the formulae from [7] for the doubling to achieve almost full parallelism for field multiplications. We also used the same approach to get the formulae for the addition in the case of mixed coordinates. The formulae for both, the parallelized doubling and addition are given in Tables I and II, respectively.

III. COPROCESSOR ARCHITECTURE

This section presents the architecture of the proposed crypto coprocessor. First the system architecture and the interface of the coprocessor with the 8-bit microcontroller are described. Then, different units of the crypto coprocessor which are the coprocessor's datapath, the storage unit, and the controller are presented.

A. System Architecture

Figure 1 shows the block diagram of the hardware architecture. There are four 8-bit ports that are used for communication between the 8051 microcontroller and the coprocessor. Two of them are for the input and output data and the other two are for coprocessor's instruction and the address to access the local storage. Every data transfer to the local storage (RAM) is through the *input_word* and the *output_word* registers that are 84 bits wide which is the word length of the operation in the coprocessor's datapath.

The 8051 is an 8-bit microcontroller originally designed by Intel that consists of several components: a controller and instruction decoder, an ALU, 128 bytes of internal memory, up to 64 KB of external memory addressed by a 16-bit DPTR register, and up to 64 KB of external program memory or 4 KB of internal program memory (ROM). The 8051 also has 28 bytes of special function registers (SFRs), which are used to store system values such as timers, serial port controls, input/output registers, etc. In our architecture using the Dalton 8051 core from UC Riverside [18], all four ports are available as "memory-mapped" interface to the microcode coprocessor.

TABLE I. DIVISOR'S DOUBLING OPERATION USING DUAL MULTIPLICATION

Step	Calculations		# mult
1	Pre-computation and resultant r:		2M
	$t_0 = Z^2$ $r = U_0.Z$	$t_1 = U_1^2$ $a = Z + V_1$	
2	Compute almost inverse (useless):		
	$inv_0 = U_1.Z$	$inv_1 = Z$	
3	Compute k:		2M
	$k_1 = f_3.t_0 + t_1$ $k_0 = U_1.k_1 + Z.b$	$b = V_1.a + t_0$	
4	Compute s:		2M
	$t_2 = k_0.U_1$ $s_0 = k_1.r + t_2$	$s_1 = k_0.Z$	
5	Compute l:		4M
	$t_0 = t_0.r$ $r = t_0.s_1$ $l_2 = s_1.t_2$ $l_1 = (t_2 + t_3).(s_0 + s_1)$ $l_1 = l_1 + l_2 + l_0$	$t_1 = s_1.k_0$ $t_3 = U_0.k_0$ $l_0 = s_0.t_3$	
6	Compute U':		1M
	$U_0' = s_0^2 + r$	$U_1' = t_0^2$	
7	Precomputation:		4M
	$a = s_0.s_1 + U_1'$ $l_2 = l_2 + a$ $Z' = s_1.r$ $t_0 = U_0'.l_2 + l_0.\ s_1$ $t_1 = U_1'.l_2 + s_1.b$	$s_1 = s_1^2$ $b = U_0' + l_1$ $t_2 = r.t_1$	
8	Adjust:		1M
	$U_1' = U_1'.r$	$U_0' = U_0'.r$	
9	Compute V':		1M
	$V_1' = t_0 + t_2.V_0$	$V_1' = t_1 + t_2.V_1 + Z'$	
Total			**17M**

TABLE II. DIVISOR'S ADDITION OPERATION USING DUAL MULTIPLICATION

Step	Calculations		# mult
1	Pre-computation and resultant r:		3M
	$t_1 = U_{11}.Z_2 + U_{21}$ $t_0 = U_{11}.t_1 + t_2$ $r = t_0.t_2 + a.U_{10}$	$t_2 = U_{10}.Z_2 + U_{20}$ $a = t_1^2$	
2	Compute almost inverse (useless):		
	$t_1 = inv_1$	$t_3 = inv_0$	
3	Compute almost s:		4M
	$t_4 = V_{10}.Z_2 + V_{20}$ $w_2 = t_0.t_4$ $b = t_0 + t_1$ $b = b.(t_4 + t_5)$ $b = w_2 + b$ $s_1 = a + b$	$t_5 = V_{11}.Z_2 + V_{21}$ $w_3 = t_1.t_5$ $a = w_3.(1 + U_{11})$ $s_0 = w_2 + U_{10}.w_3$	
4	Pre-computations:		5M

	$R = Z_2.r$
	R_tilda = R.s_3
	$S = s_0.s_1$
	S_tilda = $s_3.s_1$
	R_2tilda = R_tilda. S_tilda
		$s_3 = s_1.Z_2$
		$s_0 = s_0.Z_2$
		$S_3 = s_3^2$
		S_2tilda = $s_0.s_3$	

5	Compute l:		2M
	l_2 = S_tilda.U_{21} l_1 =(S_tilda + S).($U_{21} + U_{20}$) $l_2 = l_2$ + S_2tilda	$l_0 = S.U_{20}$ $l_1 = l_0 + l_1 + l_2$	
6	Compute U':		9M
	$a = t_1.r$ $c = s_1.Z_2$ $a = R.(a + c)$ $d = t_2$.S_tilda + s_0^2 $U_0' = b + d + a$ U_1' = S_tilda.$t_1 + R^2$ $b = S_3.(U_0' + l_1)$ $l_2 = l_2 + U_1'$ $t_4 = U_0'.l_2 + S_3.l_0$ Z' = R_tilda.S_3 U_1' = R_tilda.U_1'	$b = s_1^2$ $b = b.t_1$ $b = b.(t_1 + U_{21})$ $t_5 = U_1'.l_2 + b$ U_0' = R_tilda.U_0'	
7	Compute V':		1M
	V_0'= R_2tilda.V_{20} $V_0' = t_4 + V_0'$	V_1'= R_2tilda.V_{21}+ Z' $V_1' = t_5 + V_1'$	
Total			**24M**

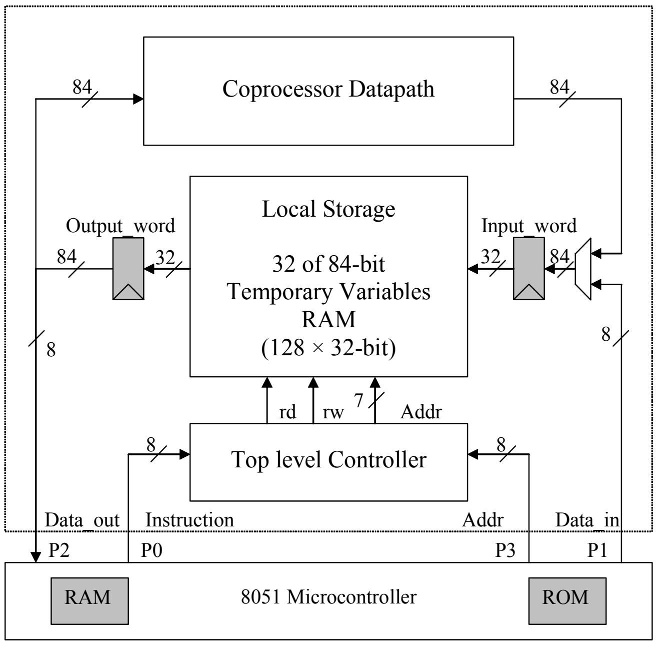

Figure 1. Microcode coprocessor connected to 8051 microcontroller

B. *Coprocessor's datapath*

Figure 2 shows the coprocessor's datapath that is designed based on the dual-multiplier/dual-adder in GF(2^{83}). The main reason for this implementation is that the divisor's operations in Tables I and II are scheduled so that two multiplications or additions can be performed concurrently in order to increase the overall performance. This means that the datapath has to be capable of performing every line of the schedules in Tables I and II. This can be done as the following. Before starting the GF(2^{83}) operations, the input operands are loaded into A, B, and D registers. After the completion of the multiplication or addition, the output results can be either sent out to the local storage or be moved from C registers to the input registers (A, B, D) for further processing. Therefore, a combination of the Galois Field operations which include multiple multiplication/addition over multiple input operands can be performed. This way every line of the divisor's doubling and addition schedules can be implemented over the proposed datapath. Moreover, the bit-serial GF(2^{83}) multipliers that perform multiplication in 84 cycles and the bit-parallel GF(2^{83}) adders that perform addition in a single clock cycle are used.

C. *Local storage unit*

The local storage unit consists of 128 memory locations of 32-bit width. In order to have easy addressing, every four locations are used to store each temporary variable of the GF(2^{83}) field. Therefore, there are total of 32 memory locations that can store the elements of GF(2^{83}). The input data is first loaded into addresses 0x00 to 0x10 and the doubling and addition result is overwritten to the same locations for every step of the scalar multiplication algorithm.

In the end the same memory locations contain the final result which is sent back to the 8051 microcontroller after the projective-to-affine conversion is performed. The memory address bus is 7 bits wide to cover the 128 locations (variables) and the coprocessor controller asserts the required values for the memory read (rd) and write (wr) signal. Also notice that the input into and out of the local RAM has to go through the input_word and output_word registers.

D. *Coprocessor's controller*

The controller takes care of reading the instructions and addressing different locations of the local storage. It also controls the datapath elements in order to implement the microcode instructions.

IV. INSTRUCTION SET

There are two basic type of instructions for the proposed coprocessor: single and microcode instructions, described as follows.

A. *Single instructions*

Table III shows the single instructions and their definitions. These instructions are used to transfer data between the coprocessor and 8051, load and store data to the RAM through the *input_word* and *output_word* registers, perform single operations using each of the adders and multipliers, moving the content of the registers in the coprocessor's datapath.

B. *Microcode instructions*

The microcode instructions are the main instructions that are used to implement the divisor's addition and doubling algorithms. These instructions implement a combination of Galois Field additions and multiplications with multiple input operands. Table IV lists these instructions, their definitions, and their microcode implementations. Any line of the divisor's doubling or addition schedules in Tables I and II can be implemented by one of these microcode instructions. It should be noted that before calling these instructions, the input operands, (values of A1, B1, D1, A2, B2, and D2) are loaded from the RAM using the single instructions (*Read_from_RAM, Outword_to_A1, Outword_to_B1, Outword_to_D1, Outword_to_A2, Outword_to_B2*, and *Outword_to_D2*). On the other hand after running any of the above microcode instructions, the results (registers C1 &C2) are stored back into the RAM.

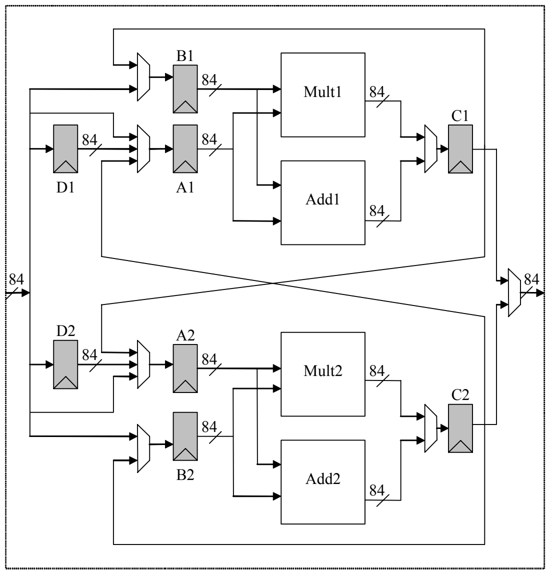

Figure 2. The datapath of the coprocessor

TABLE III. COPROCESSOR'S SINGLE INSTRUCTIONS

Instruction	Definition
Load_data_in	Loads 8 bits of data from Data_in port to the Input_word register
Get_data_out	Returns 8 bits of data from Output_word register to the Data_out port
Load_to_RAM	Loads the Input_word register into RAM from addr to addr+2
Read_from_RAM	Returns content of RAM from addr to addr+2 to the Output_word register
Do_mult1	Runs the first multiplier (Mult1) C1 = A1 * B1
Do_add1	Runs the first adder (Add1) C1 = A1 + B1
Do_mult2	Runs the second multiplier (Mult2) C2 = A2 * B2
Do_add2	Runs the second adder (Add2) C2 = A2 + B2
Outword_to_A1	Moves the content of output_word to A1
Outword_to_B1	Moves the content of output_word to B1
Outword_to_D1	Moves the content of output_word to D1
C1_to_inword	Moves the content of C1 to input_word
Outword_to_A2	Moves the content of output_word to A2
Outword_to_B2	Moves the content of output_word to B2
Outword_to_D2	Moves the content of output_word to D2
C2_to_inword	Moves the content of C2 to input_word
C1_to_B1	Moves the content of C1 to B1
D1_to_A1	Moves the content of D1 to A1
C2_to_B2	Moves the content of C2 to B2
D2_to_A2	Moves the content of D2 to A2
C1_to_A2	Moves the content of C1 to A2
C2_to_A1	Moves the content of C2 to A1

TABLE IV. COPROCESSOR'S MICROCODE INSTRUCTIONS

Instruction	Definition	Implementation

Mult_Mult	C1 = (A1 * B1)
	C2 = (A2 * B2)
		Domult1 & Domult2

Add_Mult	C1 = (A1 + B1)
	C2 = (A2 * B2)
		Doadd1 & Domult2

Mult_Add	C1 = (A1 * B1)
	C2 = (A2 + B2)
		Domult1 & Doadd2

Add_Add	C1 = (A1 + B1)
	C2 = (A2 + B2)
		Doadd1 & Doadd2

Twoadd_Mult	C1 = (A1+B1) * (A2+B2)	Doadd1 & Doadd2 C1TOB1 & C2TOA1 Domult1
Twomult_Add	C1 = (A1*B1) + (A2*B2)	Domult1 & Domult2 C1TOB1 & C2TOA1 Doadd1
Mult_and_Add	C1 = (A1*B1)+ D1	Domult1 C1TOB1 & D1TOA1 Doadd1

Mult&add_Mult	C1 = (A1*B1)+ D1
	C2 = (A2*B2)
		Domult1 & Domult2
		C1TOB1 & D1TOA1
		Doadd1

Mult&add_add	C1 = (A1*B1)+ D1
	C2 = (A2+B2)
		Domult1
		C1TOB1 & D1TOA1
		Doadd1 & Doadd2

Two_Mult&add	C1 = (A1*B1)+ D1
	C2 = (A2*B2)+ D2
		Domult1 & Domult2
		C1TOB1 & D1TOA1
		C2TOB2 & D2TOA2
		Doadd1 & Doadd2

Add_and_Mult	C1 = (A1+B1)* D1	Doadd1 C1TOB1 & D1TOA1 Domult1

Add&mult_Add	C1 = (A1+B1)* D1
	C2 = (A2+B2)
		Doadd1 & Doadd2
		C1TOB1 & D1TOA1
		Domult1

Add&mult_mult	C1 = (A1+B1)* D1
	C2 = (A2*B2)
		Doadd1
		C1TOB1 & D1TOA1
		Domult1 & Domult2

Two_Add&mult	C1 = (A1+B1)* D1
	C2 = (A2+B2)* D2
		Doadd1 & Doadd2
		C1TOB1 & D1TOA1
		C2TOB2 & D2TOA2
		Domult1 & Domult2

C. *Programming from 8051 Micro-controller*

In order to program the microcode coprocessor using the 8051, the proper instructions (single or microcode) are assigned to the ports of the 8051 microcontroller. Figure 3 shows an example that implements step 3 of the doubling algorithm (compute k) shown in Table I. Each line of the program in figure 3 puts the required binary opcode on the ports of 8051 (P0-P3). This is done by writing assembly software codes for the 8051 microcontroller.

V. PERFORMANCE RESULTS

The proposed HW/SW co-design of the HECC system was implemented and co-simulated using GEZEL [19]. GEZEL is a design environment for the exploration of domain-specific coprocessor and multiprocessor micro architectures, which can provide cycle-true HW/SW co-simulation with various embedded core instruction set simulators. In our application, we used the Dalton 8051 ISS to perform cycle-accurate simulation. The microcode coprocessor is designed in GEZEL hardware description language which is a FSMD (finite state machine plus datapath) system model. The coprocessor is attached to the input/output ports of the 8051 ISS using the GEZEL design environment and timing and functional verification is performed. In the end, the GEZEL code was automatically converted to RTL VHDL and synthesized for FPGA.

The detailed timings of different parts of the HECC co-design implementation are presented in Table V. The delay is given in terms of number of cycles and msec at the 12 MHz clock frequency for 8051. Sizes of RAM and ROM are given in bytes. Table VI compares the performance of the scalar multiplication of the presented HECC system with related work. Our 83-bit HECC system takes 7.8 M cycles of the 8051 micro-controller which results in the total delay of 656 msec at a 12 MHz clock frequency. This implementation is more than 228 times faster than the pure software implementation of HECC on 8051 and is 7 times faster than 160-bit ECC implementation on 8051 as is reported by [11]. Moreover, compared to 80-bit HECC implementation of ARM7, the number of clock cycles is a better metric because ARM7 is clocked at 80 MHz. In terms of number of clock cycles, our design is at the same order with [10] and around 4 time faster that [3].

```
Instruction                 Address
P0 = Read_from_RAM          P3 = 0xF0
P0 = Outword_to_A1
P0 = Read_from_RAM          P3 = 0xB0
P0 = Outword_to_B1
P0 = Outword_to_D2
P0 = Read_from_RAM          P3 = 0xB4
P0 = Outword_to_D1
P0 = Read_from_RAM          P3 = 0x88
P0 = Outword_to_A2
P0 = Read_from_RAM          P3 = 0xE0
P0 = Outword_to_B2
P0 = Two_Mult&add
P0 = C1_to_inword
P0 = Load_to_RAM            P3 = 0xBC
P0 = C2_to_B2
P0 = Read_from_RAM          P3 = 0x90
P0 = Outword_to_A2
P0 = Read_from_RAM          P3 = 0x80
P0 = Outword_to_A1
P0 = Read_from_RAM          P3 = 0xBC
P0 = Outword_to_B1
P0 = Twomult_Add
P0 = C1_to_inword
P0 = Load_to_RAM            P3 = 0xC0
```

Figure 3. Programming example for step 3 of Table I

TABLE V. PERFOMANCE OF DIFFERENT OPERATIONS FOR HECC

	RAM For the 8051 [Bytes]	**ROM For the 8051 [Bytes]**	**# of clock cycles [M cycles]**	**Delay At 12 MHz [m sec]**
Input and Output Data Transfer	149	1551	0.089	7.4
Divisor's Doubling	149	2414	0.11	9.9
Divisor's Addition	149	2844	0.13	11.1
Projective to Affine transformation	158	2111	0.20	17.2
Scalar Multiplication	196	6744	7.87	656

Design	**PKC**	**CPU**	**Freq. [MHz]**	**Delay [m sec]**	**# of clock cycles M Cycles**
[3]	80-bit HECC	ARM7	80	374	29.9
[10]	83-bit HECC	ARM7	80	71.56	5.72
[11]	160-bit ECC	8051	12	4580	54.9
Software only	**83-bit HECC**	**8051**	**12**	**149.8 $\times 10^3$**	**1798**
HW/SW Codesign	**83-bit HECC**	**8051**	**12**	**656**	**7.87**

TABLE VI. COMPARISON TO THE RELATED WORK

VI. CONCLUSION

This paper presented a microcode crypto coprocessor that is designed to accelerate the Hyperelliptic Curve scalar multiplication using the 8051 microcontroller. The microcode coprocessor is capable of performing the combination of GF(2^{83}) operations. The divisor's addition and doubling operations are implemented using SW routines based on the coprocessor's microcode instructions. The scalar multiplication is developed in C and compiled into 8051 assembly instructions. The total delay of 656 msec (7.8 Mcycles) was achieved for the 83-bit HECC scalar multiplication at 12 MHz.

ACKNOWLEDGMENT

This research has been supported by the Space and Naval Warfare Systems Center - San Diego under contract No. N66001-02-1-8938 and the FWO projects (G.0141.03) and (G.0450.04). Moreover, the authors thank Patrick Schaumont for the support on GEZEL co-simulation environment.

REFERENCES

[1] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, 1997.

[2] T. Lange. *Formulae for arithmetic on genus 2 hyperelliptic curves*. Applicable Algebra in Engineering, Communication and Computing, Vol. 15, Nr. 5, pages 295-328, Feb 2005.

[3] S. Baktır, J. Pelzl, T. Wollinger, B. Sunar, and C. Paar. *Optimal tower fields for hyperelliptic curve cryptosystems*. In Proceedings of 38th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA, Nov. 2004.

[4] J. Pelzl, T. Wollinger, and C. Paar. High performance arithmetic for hyperelliptic curve cryptosystems of genus two. In Proceedings of ITCC, April 5-7, 2004, Las Vegas, Nevada, USA, 2004.

[5] T. Wollinger, Software and Hardware Implementation of Hyperelliptic Curve Cryprosystem. PhD thesis, Ruhr-University Bochum, Germany, 2004.

[6] N. Boston, T. Clancy, Y. Liow, and J. Webster. *Genus two hyperelliptic curve coprocessor.* In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), number 2523 in Lecture Notes in Computer Science, pages 400--414. Springer-Verlag, 2002.

[7] B. Byramjee and S. Duquesne. *Classification of genus 2 curves over F_{2^n} and optimization of their arithmetic*. Cryptology ePrint Archive: Report 2004/107.

[8] D. Cantor. Computing the Jacobian of a Hyperelliptic Curve. Math. of Computation, 48:95--101, 1987.

[9] J. Pelzl, T. Wollinger, J. Guajardo, and C. Paar. Hyperelliptic curve cryptosystems: Closing the performance gap to elliptic curves. In C. Walter, Ç. K. Koç, and C. Paar, editors, Proceedings of 5th International Workshop on Cryptograpic Hardware and Embedded Systems (CHES), number 2779 in LNCS, pages 351--365. Springer-Verlag, 2003.

[10] J. Pelzl, T. Wollinger, and C. Paar, *Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation,* Chapter in Embedded Cryptographic Hardware: Design and Security. Nova Science Publishers, 2004.

[11] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs*. In M. Joye and J. J. Quisquater, editors, Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS 3156, pages 119--132, 2004.

[12] G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. *Finding optimum parallel coprocessor design for genus 2 hyperelliptic curve cryptosystems*. In Proceedings of ITCC, April 5-7, 2004, Las Vegas, Nevada, USA, 2004.

[13] G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. *Hyperelliptic Curve Cryptosystem: What is the Best Parallel Hardware Architecture?*, chapter in Embedded Cryptographic Hardware: Design and Security. Nova Science, 2004.

[14] H. Kim, T. Wollinger, Y. Choi, K. Chung, and C. Paar. *Hyperelliptic curve coprocessors on a FPGA*. In Workshop on Information Security Applications - WISA, Jeju Island, Korea, 2004.

[15] N. Koblitz. *A family of Jacobians suitable for Discrete Log Cryptosystems*. In S. Goldwasser, editor, Advances in Cryptology: Proceedings of CRYPTO'88, N. 403 in LNCS, pages 94--99. Springer-Verlag, 1988.

[16] A. Menezes, Y.-H. Wu, and R. Zuccherato. *An elementary introduction to hyperelliptic curves*, chapter Appendix, pp 155-178. Springer-Verlag,1998. Koblitz: Algebraic Aspects of Cryptography.

[17] IEEE P1363/D13, Standard Specification for Public-key Cryptography, November 1999.

[18] Dalton 8051, http://www.cs.ucr.edu/~dalton/8051/

[19] P. Schaumont, I. Verbauwhede, "Interactive cosimulation with partial evaluation," 2004 DATE 2004, pp. 642-647, February 2004.