A VLSI implementation of RSA and IDEA encryption engine.

Ahto Buldas
e-mail: ahtbu@ioc.ee
phone: +372-5102-160
fax: +372-6202-253

Jüri Põldre
e-mail: jp@pld.ttu.ee
phone: +372-5103-168
fax: +372-6202-253

Abstract- *Data communication uses RSA for key exchange and IDEA for block encryption. The presented design employs both modular arithmetic and IDEA using the same 96-bit ALU for calculations. The one chip 1.0 µm 104 mm^2 CMOS design can also generate and hold keys for asymmetric key exchange systems and has internal self-test.*

I. INTRODUCTION

Data encryption based on asymmetric key exchange algorithms and symmetric block encryption has been used in data communications for many years. These systems usually consist of general purpose processor and some additional logic to speed up modular calculations. As the key generation for RSA [RSA78] is complicated and time-consuming most hardware implementations use externally generated keys. This presents serious security problem connected with the possibility to access sensitive data. If the key generation, inversion and the key exchange field handling can be done in a single tamper-proof device, there is no need to enable user access to these procedures. As these procedures can be realised using a reasonable amount of memory, it becomes feasible to realise such a tamper-proof device in a single integrated circuit. Such a realisation would not only improve security but is also cost effective.

Secure key exchange with RSA in a reasonable time requires a lot of hardware resources. Most of it is used for modular arithmetic. Also this device should include a hardware for a secure block cipher. The calculations used by most block ciphers are bit operations and table lookups which are hard to share with integer arithmetic and have to be realised by separate hardware. One solution would be to use IDEA cipher for block encryption. IDEA has 128-bit key length. The encryption process consists of 8 rounds. The operations in one round contain 16-bit modular additions and multiplications which can be easily shared with integer calculations used in RSA. Also the key inversion algorithms for both ciphers are similar. When using the same ALU for both asymmetric key exchange algorithms and block encryption, it is possible to save silicon area.

The combination of RSA and IDEA has also been used with success in freeware e-mail encryption system PGP.

In the following pages a VLSI implementation of the device discussed above is presented starting with data path description. The control part, self-test and future directions are discussed.

II. DATA PATH

ALU is divided into 4 different units, each 24-bit wide. For IDEA calculations these units can be configured as two 16 bit multipliers. The calculations are carried out using low-high algorithm [LAI91],[LAI92]. Each 24 bit block can calculate 8*16-bit multiplication. Using 2 such multipliers we can multiply 16*16 bits in one cycle.

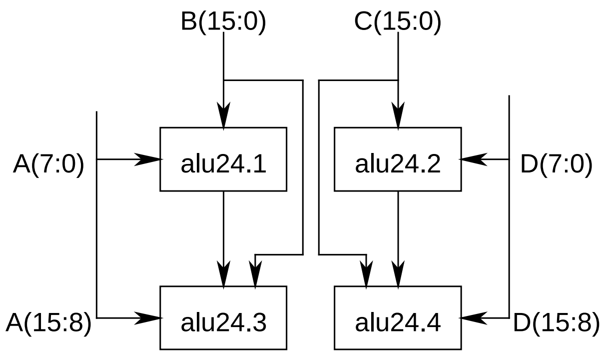

Fig. 1 ALU in IDEA multiply mode

The second cycle will be used for modular reduction step of low-high algorithm. During these steps lower parts of calculations from alu24.1 and alu24.2 are fed into CSA tree of alu blocks alu24.3 and alu24.4, avoiding carry propagation delay [HEN90]. The rest of IDEA calculations do not take much silicon area and consist of some XORs and two 16-bit adders. For more detailed description see IDEA control below.

In long modular calculations mode the ALU is configured as 8*96 bit multiplier or 96-bit adder/negator with additional carry logic.

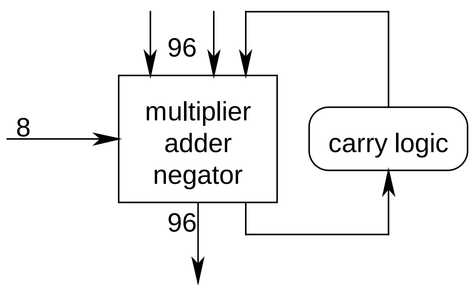

Fig. 2 ALU in modular calculation mode

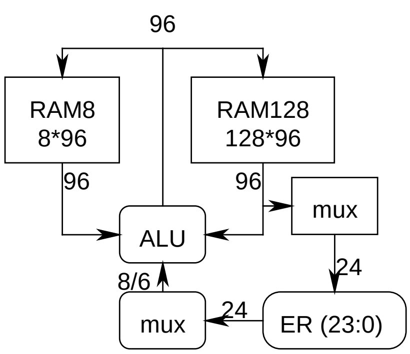

FIG. 3 The structure of ALU datapath

This mode enables us to handle long numbers 8 bits at a time. The datapath consists of ALU, two RAM modules and one 24-bit register ER (Fig.3). This small register is used for multiply and modular reduction as a source of multiplicand. RAM modules can be used as two data register groups of 1 and 16 registers. All data registers are 768 bits wide.

III. ALU CONTROL STRUCTURES

ALU control structures are shown on fig. 5. The processor has two levels of code. Microcode is fixed and contains two types of commands.

Index calculating commands. For that purposes the INDEX_CALC state machine has 4 internal 8-bit registers enabling to address 6-bit entities inside the 768-bit data registers as shown on Fig. 4.

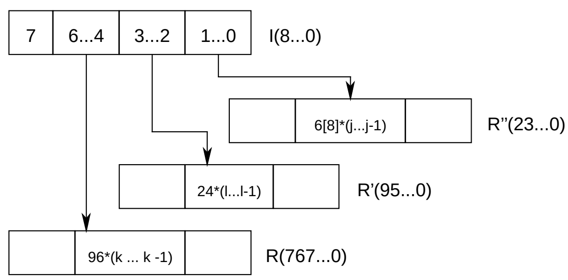

Fig. 4. Index register layout

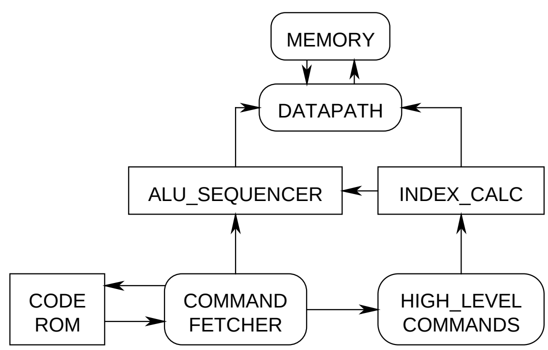

Fig. 5. ALU control structure

The index registers are used for loading parts of 768-bit registers into 24-bit register ER. Also it is possible to do logic operations between immediate values, index registers and ER.
Arithmetic commands operate on long data registers. Typical commands of that type are adding, subtracting, multiplying and transfer of data registers. These are executed in ALU_SEQUENCER in parallel with index calculations.
To ease the programming job a level of hierarchy is built upon microcode.
HIGH_LEVEL COMMANDS state machine decodes these commands and executes the necessary microcode program. The microcode program memory contains jump table at the start of code area. Each time a high level command gets executed the state machine checks the entry in this table to find the appropriate microcode.
At the highest level it is possible to choose between 32 external commands. These are selected with setting logic levels on circuits pads. The high level code has similar table as microcode to decode all external commands of the circuit.

IV. MODULAR MULTIPLY ALGORITHM

Let us now look more closely at the modular multiply step. Let A, B and C be data registers and S be an index register. First the operands are reduced so that for *A*B* mod *C* the first step would be

$$A := A \bmod C \quad \text{and} \quad B := B \bmod C.$$

Then the larger of the arguments *A* and *B* is loaded into RAM8 (Fig.3), after what the higher 24 bits of second argument are loaded into 24-bit register ER. Then we multiply the higher 6 bits of the first argument with the second argument, and do the modular reduction step. For that we use binary search algorithm to find out the higher bits of constant *m* what we must use in reduction step. Assuming $A<B<C$, modular multiply consists of the following steps:

1. *RAM8 := B*
2. *C := -C*
3. S := length of A in 24 bit fields
4. *ER := A*[S]
5. for I=1 to 4
6. CALL MODMUL
7. S:= S -1
8. if S ≠ 0 GOTO 4

MODMUL:

1. *RAM8 := ER*(higher 6 bits) * *RAM8*
2. Find the largest 7-bit *m* using binary search so that $C * m + RAM8 \geq 0$
3. Reduce $RAM8 := RAM8 + C * m$
4. Shift ER 6 bits
5. RETURN

The binary search algorithm starts with the highest bit and then moves through all bits. To speed up the calculations we use only the higher 96-bit part of RAM8 for compare. Only when *C*m* and RAM8 are "almost equal" the full compare is used. Fortunately, this happens very rarely (with the probability 2^{-89}) and can be safely ignored in speed calculations. The number *N* of cycles for multiplication can be calculated as follows:

$$N = (R + R + 1 + m) * R * L / m{,}000$$

where:
N -- number of cycles for multiply,
m -- ALU multiplier length
R -- length of register in RAM fields
L -- length of one RAM field in bits.
In our case:

$$N := (8 + 8 + 7) * 8 * 96 / 6 := 23 * 8 * 16 := 2944$$

using 25 MHz clock we get 8491 modular multiplications per second.
The small register can also be used for right shifting arguments in divide operations and for bit operations.

V. I/O AND IDEA

The I/O control structure takes care of all I/O and starts IDEA cipher process when the circuit is in block encryption mode. The circuit has two I/O modes. One is active when command fetcher has been stopped by the WAIT micro-command. During this it is possible to read out all internal registers. The other is active when circuit is in I/O mode. I/O mode lets user read out only last 2 data registers. This is necessary to avoid the leak of secret information from the circuit.
In IDEA mode the I/O starts with reading and filling temporary input registers. After the last word of 64 bits has been read the IDEA transform starts and the temporary registers are ready to accept new input data. After IDEA has finished, it gives out the IDIO_RDY signal and fills the output registers, what can then be read out from the circuit. All this is necessary, because IDEA transform is rather fast. During 50 cycles of operating time it must be possible to read in next 4 words and read out previous ones. As one I/O operation takes 4 cycles, the total time for I/O is 4*4*2 = 32 cycles. The encryption speed will be 32 Mbit/s using 25 MHz clock.
The I/O control has at its final stage the possibility to select between byte-parallel, word-parallel and serial transfer mode.
The IDEA control system takes us through the IDEA transform in 50 cycles. To see how it is achieved let us first look at IDEA cycle on Fig. 6. Here and after F4 stands for Fermat's 4^{th} digit, namely $2^{16}+1$. The IDEA transform consists of 8 similar rounds and output transform. One IDEA round contains 6 multiply and 4 add operations. It transforms 64 bit of input data to 64 bit of output data using 96 bits of key information. Output transform contains only the upper 2 additions and multiplications.

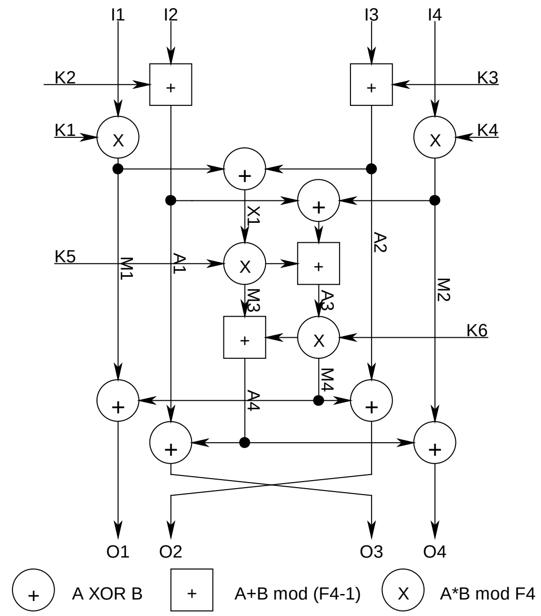

Fig. 6 One IDEA round.

The time-consuming operation here is modular multiply. Two16 bit additions at beginning are cheap and can be done in parallel with multiplications. XOR is also no problem.
As it was pointed out before we use low-high algorithm for modular multiplication. This algorithm for $C := A * B$ mod F4 consists of the following steps:

1. D := A * B,
   D.lo := D(15..0), D.hi :=D(31..16)
2. if(D.lo >= D.hi) C := D.lo - D.hi
   else C := D.lo - D.hi + F4

When describing IDEA datapath we noted that ALU can be used for two 16-bit multiplications in one cycle. So the first step of the algorithm is quite easy. For second step we use ALU units alu24.1 and alu24.3 for the M1 and alu24.2 and alu24.4 for the M2 multiplication. They both work alike so I will take a look only at the first multiplication. Alu24.1 calculates D.lo-D.hi, alu24.3 calculates D.lo-D.hi+F4. It is possible to use them like this, because ALU units are multipliers and consist of 8 argument CSA adder with CLA at the end. For M3 and M4 multiplication we use all units and calculate the multiplication in the alu24.1 and alu24.3. Alu24.2 and alu24.4 will have at reduction stage the addition argument added. Then we check the result of alu24.1 and alu24.3 and select the result from alu24.2 and alu24.4. That enables us to run all multiplications in 2 clock cycles. So for each IDEA cycle we have 2*3:=6 clock cycles. As the output transform contains only the beginning of the cycle it adds 2 clock cycles to the total time leaving us with:

$$N = 2*3*8 + 2 = 50$$

clock cycles per IDEA transform.

Finally, the following table presents the ALU datapath schedule in IDEA mode. Here .lo and .hi represent the lower and higher part of calculation. The M1 of next cycle is selected from M1 and M1’ depending on the value of M1 according to low-high algorithm. M2 is selected similarly. M3 is selected from M3’’ and M3’’’ depending on the value of M3. M4 is selected as M3.

The keys are selected from RAM128 with the possibility to change between the starting location externally before each transform, thereby selecting between encryption and decryption.

VI. SELF TEST

The circuit contains self-test system for evaluating error condition and verifying hardware. To accomplish it we use state hashing of all FSMs. The bits are fed into 8 bit analysers running in parallel. After 255 states the registers are shifted into 32 bit analyser and the readout of that analyser is possible in test mode. Datapath can be checked by running arithmetic test program and IDEA transform. The state hashing help us to verify that the desired result was achieved in the correct way. By making the source code available it is possible by checking the state hashing to verify that the hardware has not been tampered with to include backdoor.

Table 1. IDEA datapath schedule.

nr	alu24.1	alu24.3	alu24.2	alu24.4
1	T1 := I1*K1	T2:=I1*K1	T3:=I4*K4	T4:=I4*K4
2	M1:=T1.lo-T1.hi	M1’:=T2.lo-T2.hi+F4	M2:=T3.lo-T3.hi	M2’:=T4.lo-T4.hi+F4
3	T1:=X1 * K5	T2:=X1 * K5	T3:=X1 * K5	T4:=X1 * K5
4	M3:=T1.lo-T1.hi	M3’:=T2.lo-T2.hi+F4	M3’’:=T3.lo-T3.hi+X2	M3’’’:=T4.lo-T4.hi+X2+F4
5	T1:=A3 * K5	T2:=A3 * K5	T3:=A3 * K5	T4:=A3 * K5
6	M4:=T1.lo-T1.hi	M4’:=T2.lo-T2.hi+F4	M4’’:=T3.lo-T3.hi+M3	M4’’’:=T4.lo-T4.hi+M3+F4

VII. FUTURE DIRECTIONS

This circuit has several deficiencies:

1. No internal code rom. This makes it easy for the attacker to rewrite control programs.
2. No internal cryptographically secure random number generator. It has a built-in PRNG for testing purposes only. External random generator must be used to run it in real applications. But external generators can be manipulated by the attacker.
3. The moduli length is too short- only 768 bits. For RSA to be secure it should be at least 1024 bits. In the next version we will double the moduli length by adding second ALU unit. This will also increase IDEA speed two times. Also by running two ALUs in parallel and using Chinese Remainder

theorem we can increase RSA decryption speed by a factor of 4.

4. Datapath design is not the best. Registers should be added to increase the speed, also the layout information should be extracted from VHDL files to produce compact layout. This was one important by-product of the project. By using VHDL generate statements and some control pragmas the program extracts layout control information from source file. This information is then converted into Cadence tile generator form to produce layout for datapath.
5. The possibility to control clock frequency with built-in PLL synthesiser should be included to enable us run the circuit at lower speeds what is crucial for mobile equipment where high encryption speeds must give a way to low power consumption.
6. Some Message Authentication Code calculation should be included to internally test the integrity of bypassing data.
7. The support for signed public key database should be included.
8. The switch to faster technology should increase clock speed and decrease area.

VIII.PROJECT DEVELOPMENT

The project started on year 1993 with the financial and educational aid from Europe. TEMPUS JEP4772 project with Prof. Manfred Glesner from Germany, Darmstadt, Prof. Bernard Courtois from France, Grenoble and prof. Raimund Ubar from Tallinn Technical University gave us the tools for synthesis and mapping and the know-how.

In the beginning we chose the architecture and then wrote the circuit simulator on PC. At that time Ahto Buldas held theoretical seminars about cryptology in Institute of Cybernetics. Using the PC simulator we refined the code for calculations. At the same time we started to write the behavioural description of the circuit using SYNOPSYS development software. Whole development was carried out using VHDL language. Finally we added the self-test feature. The behavioural description is in 22 files with total size 509 KB. At the end of 1996 the circuit was ready for prototype run. The placement and routing was done with CADENCE development system. The prototype silicon run was financed by Institute of Cybernetics and manufactured via Europractice in ES2 1.0 µm technology. We received the prototypes at the beginning of March, 1997. Since then we have developed the interface to PC using 2 XILINX FPGAs and rewrote the simulator to include the possibility to download and test the code on real device. As the result we have found these devices comply with the expectations, with 3 out of 20 prototypes not functional due to production faults. The calculated worst-case speed was 20 MHz. As experiments showed the real maximal operating speed was 25 MHz.

Now we are developing the add-in card for PC to carry out disk and network encryption and are rewriting the simulator to allow other people to experiment with the circuit.

ACKNOWLEDGEMENTS

The authors are grateful to prof. Raimund Ubar for the advises and patience, European Union TEMPUS office and EUROPRACTICE organisation for giving us the opportunity and dr. Ülo Jaaksoo and Institute of Cybernetics for financial support.

REFERENCES

[HEN90] Hennessy,J.L., Patterson,D.A., “Computer architecture: a quantitative approach,” Morgan Kaufman Publishers, Inc., 1990.

[LAI91] Lai,X., Massey,J.L., “A proposal for a new block encryption standard,” *Advances in Cryptology—EUROCRYPT’90*, 389—404, 1991.

[LAI92] Lai, X., “On the design and security of block ciphers,” ETH Series in Information Processing, J.L.Massey (editor), vol. 1, Hartung-Gorre Verlag Konstanz, Technische Hochshule (Zurich), 1992.

[RSA78] Rivest,R.L., Shamir,A., Adleman,L.M., “A method obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, v. 21, n.2, 120—126, Feb 1978.