HW - SW codesign project

Boran Car Victor Statescu

July 1, 2011

The goal of this project was to design a cryptographic system that would support 1024-bit encryption RSA and ElGamal. The processor given was an 8051-based. The choice is similar to what many smart-card manufactures offer [2, 3]. Basically, the 8051 is an industry proven, well tested and widespread micro-controller. This it owes to having been around for more than 20 years.

Since the processor was not fast or powerful enough, HW-SW co-design was employed to offload some operations onto a separate coprocessor.

1 System design

The 8051 has 4 8-bit ports available for IO. Due to port sharing, not all of these ports can be used in specific configurations (e.g. using an external memory takes away the ports P0 and P2; using interrupts, UART or external ROM takes away port P3). This was the reason why only port P1 was chosen as a means of port IO signalization with the coprocessor. Only 2 pins were used for the signalization so the rest can still be used for other purposes.

Shared memory was used as a means of communicating the data to the coprocessor. Addresses 0x600–0x801 are mapped to the coprocessor's addresses 0x000–0x201. Addresses 0x000–0x600 are left for the main processor.

The operations chosen for offloading to the coprocessor were the Montgomery multiplication and Montgomery inversion for speed and flexibility reasons.

The fastest the Montgomery multiplication could be made in SW was 1.3 s@12 MHz which is not fast enough. HW realization on the other hand took 2300 cycles, which goes roughly to 0.2 ms@12 MHz.

For the inversion case it took 16 s@12 MHz, while in the HW realization 1.3 mil.cycles, roughly 100 ms@12 MHz.

For the modular exponentiation case it was decided to keep it in SW using the Montgomery multiplication of the coprocessor. Since the main processor is 12 times slower some techniques have been applied to eliminate the communication overhead.

The frequency of the system was lowered to 4 MHz to match most of the manufacturers.

2 Coprocessor design

The coprocessor has:

- 6 1024-bit registers (u, v, p, R, S, Q)
- 1 1024-bit datapath (adder and a shifter)
- 10-bit datapaths and registers for loop and address counters

The shared memory is used in the following way:

- 0x600–0x680 1024-bit number
- 0x680–0x700 1024-bit number
- 0x700–0x780 1024-bit number
- 0x780–0x800 command queue (up to 128 commands)
- 0x800 state signaling from the coprocessor

The instructions are the following:

- Halt - stops the execution of the coprocessor and signals the done to the main processor
- Init - initializes the coprocessor (just sets the modulus for now)
- Montgomery multiplication - executes the Montgomery multiplication on the registers u and v and stores the result back to u
- Montgomery squaring - executes the Montgomery squaring of the register u and stores the result back to u
- Montgomery inversion - executes the Montgomery inversion of the register u and stores the result back to u (destroys the previous values of u and v)
- Load u from the shared memory - loads the register u from the shared memory location 0x00 or 0x80 or 0x100
- Load v from the shared memory - loads the register v from the shared memory location 0x00 or 0x80 or 0x100

- Store result to shared memory - stores the result of the operation to memory location 0x00 or 0x80 or 0x100
- Store quotient to memory (Montgomery only) - stores the quotient of the Montgomery operation to memory location 0x00 or 0x80 or 0x100

Special instructions were provided for loading from shared memory because the coprocessor is 24 times faster with loading from memory (12 clock cycles for 1 instruction cycle, 2 instruction cycles for loading on the processor). This way, a third parameter can be stored in the shared memory for faster fetching.

2.1 Command queueing

The idea is to allocate special storage in the shared memory for the commands the coprocessor should execute. The main processor then simply fills this memory with the commands the coprocessor needs to execute. The main processor then starts the coprocessor which will execute these instructions.

This is also useful if the main processor should be able to perform other operations while the coprocessor executes the commands provided. The method has also been tested with the main processor being slower 144 (additional 12x slowdown) times than the coprocessor and it gave satisfying results (only 5x increase in cycle numbers for RSA).

2.2 Datapath

The datapath design revolves around a 1024-bit adder and a shifter in series. The shifter can shift one position to the left, one position to the right or just pass the input.

The inputs to the adder are multiplexed between 0, 1, u, v, p, R, S, power for the x operand. For the v operand, the multiplication step is multiplexed instead of the power.

The result always gets assigned back to u. This was to allow chaining of Montgomery multiplications when exponentiation is performed. This way, no redundant copying is necessary from/to the shared memory. What is also possible this way is to update the shared memory while the coprocessor is executing (this helps the elimination of the communication overhead). The technique was not tested within this project.

2.3 Modifications to the Montgomery product

The Montgomery product computation algorithm has been modified to include computing the quotient (as is called in [4]). This allows for doubling the bitlength ([4, 1]) of the crypto-coprocessor in software. This technique was not tested within this project.

3 Implementation

3.1 Software in C

A library is provided exposing primitives which are executed on the coprocessor:

- montpro - Montgomery product
- montinv - Montgomery inversion
- modexp - Modular exponentiation

Also provided are the following software methods:

- add1024 - adding 1024-bit numbers
- subtract1024 - subtracting 1024-bit numbers
- multiply1024 - multiplies 1024-bit numbers to produce a 2048-bit
- larger_or_equal - checks if the number is larger or equal than a number

These operations were left in software as they were already fast enough there. Multiplication was left in case someone would want to use the bit doubling method. These functions are documented in the lib.h header file.

3.2 Hardware in VHDL

During the co-design phase, separate datapaths were made for the adder and the bit shifter. This was to allow a different implementation in VHDL as the GEZEL-to-VHDL tool (fdlvhd) generates one .vhd file per datapath. The target devices usually have sophisticated CLA logic implemented so that efficient adders in terms of space and speed can be generated.

The same reasoning goes for ASIC design. Usually the libraries provided from the foundries will include efficient designs of adders which can be combined to produce the required adder. Even if they don't, having a separate block allows the designer to concentrate more on where optimization really counts (critical data path, major effect on area).

In our specific case, the GEZEL-to-VHDL tool generated 2 1024-bit adders, one with carry-in and one without it. Manual intervention was required to change it to a single 1024-bit adder with carry-in. Later on 2 513-bit adders were used to reduce the slice count even further. Putting 2 smaller adders helped the synthesis tool spread out and perform better interconnect. Custom add_sub.customxx.vhd are provided in the vhdl subfolder.

4 Results

The RSA encryption and decryption with provided exponents (9-bit for encryption and 1024-bit for decryption) take around 5 mil.cycles to complete in total. On the 4 MHz clock frequency this amounts to around 1.25 s (average for encryption/decryption 600 ms).

The ElGamal encryption and decryption with provided exponents (128-bit for encryption and decryption) take around 4.5 mil.cycles to complete in total. On the 4 MHz clock frequency this amounts to around 1.125 s (average for encryption/decryption 560 ms).

These times put the design along with most of the systems presented in [2, 3].

Eventually the implemented design did not meet the area constraint. It used 101% of the device (Table 1). The maximal frequency is 20.449 MHz which is well above the required 4 MHz.

	Used	Available	%
Slices	13951	13696	101%
Flip-flops	6267	27392	22%
4-input LUTs	26033	27392	95%

Table 1: Implementation results

Following is an excerpt from the synthesis tools report (available as the file sys.syr):

```
# Adders/Subtractors                                        : 5
 10-bit adder                                               : 2
 11-bit subtractor                                          : 1
 513-bit adder carry in                                     : 2
# Registers                                                 : 6251
 Flip-Flops                                                 : 6251
# Comparators                                               : 1
 1024-bit comparator equal                                  : 1
# Multiplexers                                              : 1
 1026-bit 4-to-1 multiplexer                                : 1
```

References

[1] W. Fischer and J.-P. Seifert. Increasing the bitlength of a crypto-coprocessor. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 71–81, London, UK, UK, 2003. Springer-Verlag.

[2] H. Handschuh and P. Paillier. Smart card crypto-coprocessors for public-key cryptography. In *CARDIS*, pages 372–379, 1998.

[3] H. Handschuh and P. Paillier. Smart card crypto-coprocessors for public-key cryptography. In J.-J. Quisquater and B. Schneier, editors, *Smart Card Research and Applications*, pages 386–394. Springer, 2000.

[4] M. Yoshino, K. Okeya, and V. Camille. Unbridle the bit-length of a crypto-coprocessor with montgomery multiplication. In *Proceedings of the 13th international conference on Selected areas in cryptography*, SAC'06, pages 188–202, Berlin, Heidelberg, 2007. Springer-Verlag.