TFE4141 – Design of Digital Systems 1

Introductory Presentation of the Term Project

13. September 2018

Advisors and helpers:

Magnus Hirth **(**magnuhi@ntnu.no**)**
Eirik Andre Moltu **(**eiram@stud.ntnu.no**)**
Meeyad Mohd Shabab **(**meeyadms@stud.ntnu.no)
Øystein Gjermundnes **(**oystein.gjermundnes@ntnu.no**)**

Outline

- Term project - goals
- RSA
- Plan – Specification

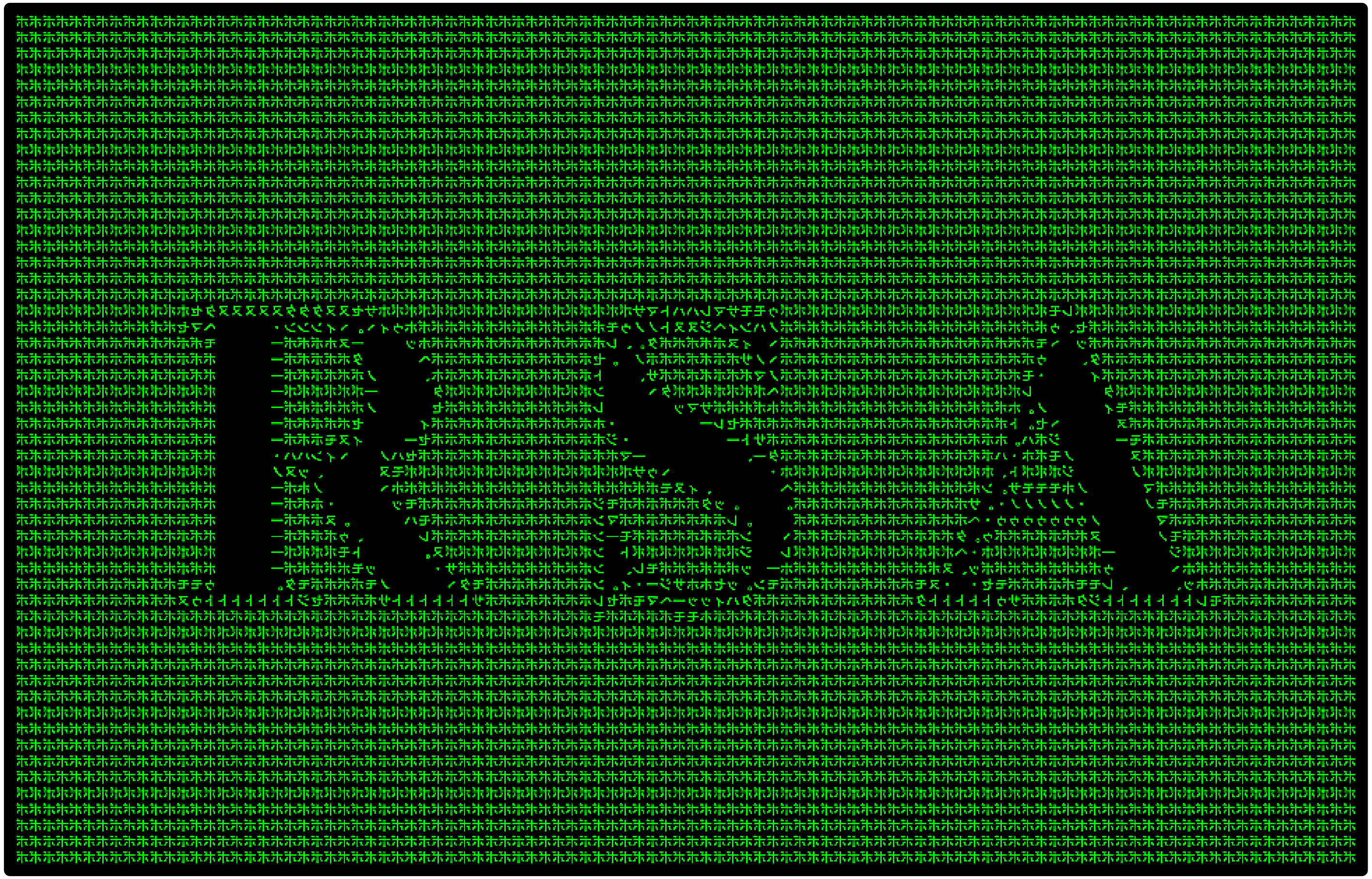
Term Project

Learning goals

- **Important part of learning in TFE4141**
- **Learning – design and practical skills in using EDA tools**

- The work on this term project will give you experience with :
  - Implementation of algorithms in hardware
  - Verification by simulation
  - Use of tools for synthesis
  - To understand and evaluate other people's work by a peer-review system. This will also help you to receive critique (and praise..)

- **30 %** of the final grade is based upon the project report



RSA

Ron **R**ivest, Adi **S**hamir, Len **A**dleman

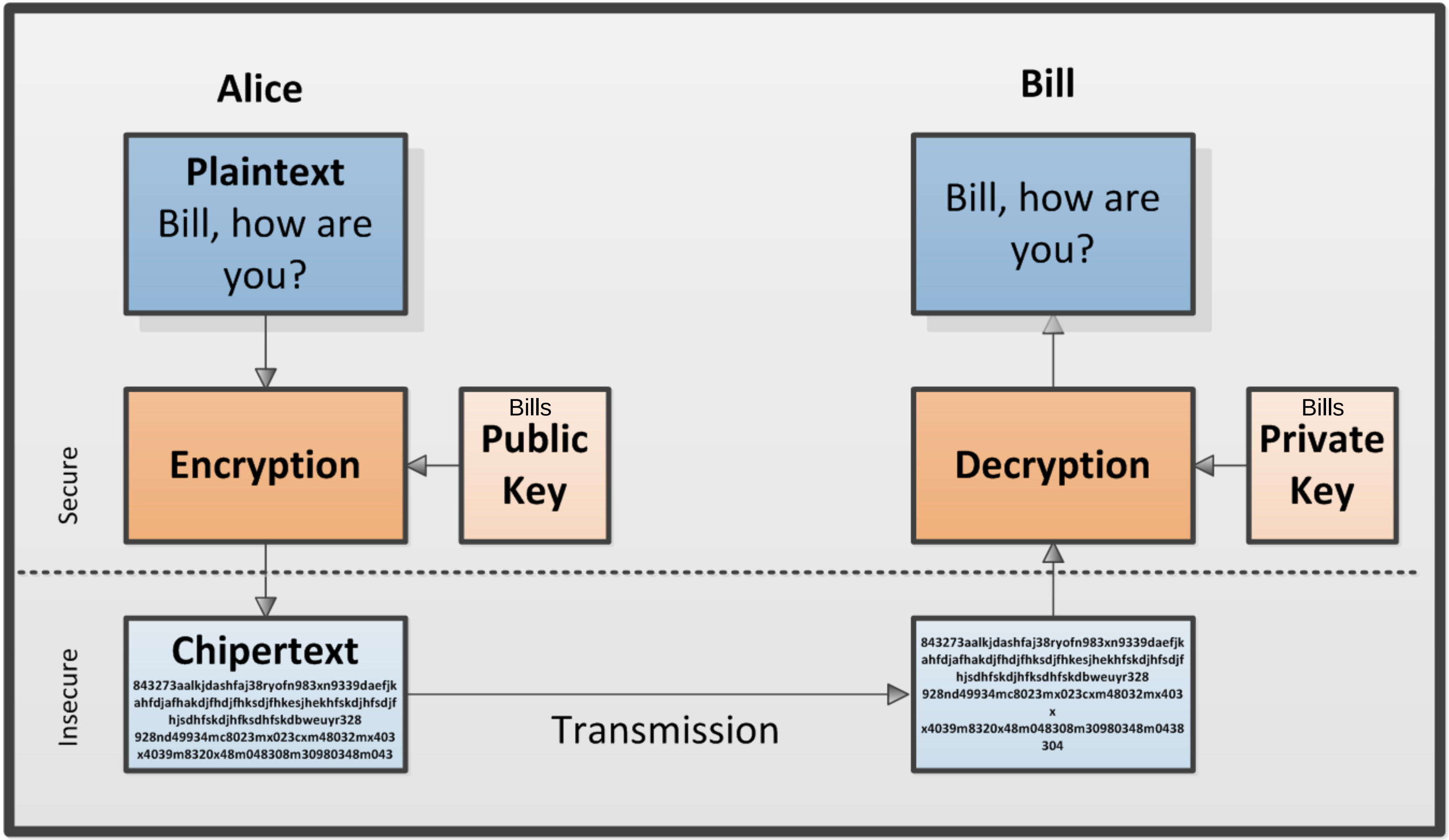
- Block cipher: both plaintext and cipher text are integers between 0 and n-1.

- RSA uses two different keys. One is for encryption (public key); the other is for decryption (private key). This is an **asymmetric** algorithm.

- The method relies on a one-way function f such that it is easy to compute $Y = f(X)$, but impossible to compute $X = f^{-1}(Y)$, if you don't have the correct key k.

- RSA is widely used in electronic commerce protocols, and is believed to be secure, given sufficiently long keys and the use of up-to-date implementations.

RSA Encryption/Decryption Mechanism



RSA

- The **same computation** is performed for encryption and decryption, but with different parameters. The encryption and decryption keys (***d, e***) are related through the following equation:

- **Encryption**: $\mathbf{C = M^e \bmod n, M < n}$
  public key: {e, n}

- **Decryption**: $\mathbf{M = C^d \bmod n = (M^e)^d \bmod n}$
  private key: {d, n}

- $\mathbf{M = M^{ed} \bmod n}$ restricts the values for **e, d** and **n.**
- to be found by **Euler's theorem**.

Wikipedia http://en.wikipedia.org/wiki/RSA

Generating the keys

1) Select two prime numbers **p** and **q**
2) Compute **$n = pq$**
3) Compute **$\Phi(n) = \Phi(p) * \Phi(q) = (p-1)(q-1)$**
4) Select **e** such that **$gcd(\Phi(n), e) = 1, 1 < e < \Phi(n)$**
5) Compute **$d = e^{-1} \bmod \Phi(n)$**

gcd: greatest common divisor

Φ(k): Euler's totient function.

A simple example

Key generation:

- Find two primes p = 7, q=17
- n = pq = 7*17=119
- $\Phi(n)$ = 6*16=96
- Select *e* such that *e* and *$\Phi(n)$* are relative primes. e=5 can be used.
- Determine *d* such that de=1 mod 96 and d < 96.
  77*5=385=4*96+1 => d=77.
- Public key : {e,n} = {5,119}
- Private key: {d,n} = {77,119}

Encrypt/Decrypt:

- Message: M=**19**
- Encryption: 19^5 mod 119 = 2476099 mod 119 = 20807*119 + 66 mod 119 = 66
- Decryption: 66^{77} mod 119 = …"Large numbers"… = **19**

M^e mod n

"Brute force"

```
# 1) Very naive method for computing M^E:

#    Say we were to compute M^55. One possible solution would be:
#    C = M*M*M*M*M*M*M*M*M*M*
#        M*M*M*M*M*M*M*M*M*M*
#        M*M*M*M*M*M*M*M*M*M*
#        M*M*M*M*M*M*M*M*M*M*
#        M*M*M*M*M*M*M*M*M*M*
#        M*M*M*M*M
#    which requires 54 multiplication operations. It cannot be a very good
#    method since it would take us quite some time with very large numbers.
#
#    This is an algorithm with complexity O(E)
```

- Modular exponentiation is achieved by
- replacing all multiplications in the
- algorithm by modular multiplications.

M^e mod n

"Repeated Squaring"

```
•   # 2) A much better solution for computing M^E
•   #
•   #    The exponent written with different base numbers:
•   #    E = "55"dec = "37"hex = "00110111"bin
•   #
•   #    The exponent written in order to show the bit indexes
•   #    E : 0  0  1  1  0  1  1  1
•   #    i : 7  6  5  4  3  2  1  0
•   #
•   #    The exponent written as a sum of powers of two:
•   #    E = 0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 1*2^2 + 1*2^1 + 1*2^0= 55
•   #
•   #    We can now easily observe that M**55 can be rewritten like this:
•   #    M^55 = M^[0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 1*2^2 + 1*2^1 + 1*2^0]
•   #         = M^[2^5] * M^[2^4] * M^[2^2] * M^[2^1] * M^[2^0]
•   #
```

M^e mod n

"Repeated Squaring"

```
• #    From this we can easily device a very efficient algorithm:
• #
• #    E0 = 1:  C = M    = M^[2^0]
• #             P = M*M  = M^[2^1]
• #
• #    E1 = 1:  C = C*P  = M^[2^0 + 2^1]
• #             P = P*P  = M^[2^2]
• #
• #    E2 = 1:  C = C*P  = M^[2^0 + 2^1 + 2^2]
• #            P = P*P  = M^[2^3]
• #
• #    E3 = 0:  C = C*P  = M^[2^0 + 2^1 + 2^2]
• #            P = P*P  = M^[2^4]
• #
• #    E4 = 1:  C = C*P  = M^[2^0 + 2^1 + 2^2 + 2^4]
• #            P = P*P  = M^[2^5]
• #
• #    E5 = 1:  C = C*P  = M^[2^0 + 2^1 + 2^2 + 2^4 + 2^5]
• #            P = P*P  = M^[2^6]
• #
• #    E6 = 0:  C = C*P  = M^[2^0 + 2^1 + 2^2 + 2^4 + 2^5]
• #            P = P*P  = M^[2^7]
• #
• #    E7 = 0:  C = C*P  = M^[2^0 + 2^1 + 2^2 + 2^4 + 2^5]  <-- CORRECT
  ANSWER
• #            P = P*P  = M^[2^8]
• #
• #    Only 15 multiplications were needed here....
```

```
#    Pseduocode for the algorithm:
#
#    C := 1
#    P := M
#    for i=0 to k-1
#      if Ei = 1
#        C := C*P
#      P := P*P
#    return C
#
#    This is an algorithm with
#    complexity O(log E) which is a lot
#    better than O(E).
```

A*B

```
•    # 1) Paper and pencil method for multiplying two nonnegative integers:
•    #
•    #    A = 1001 = 9
•    #    B = 1011 = 11
•    #    P = 110011 = 2**6 + 2**5 + 2**1 + 2**0 = 64 + 32 + 2 + 1 = 99
•    #    P = A*B = 9*11 = 99
•    #
•    #
•    #           1001 * 1011
•    #       ------------------
•    #          1
•    #         1-1001
•    #         -1001
•    #         0000
•    #        1001
•    #       ------------------
•    #     =  1100011
•    #       ------------------
•    #       ------------------
•    #
•    #    An algorithm for multiplying two nonnegative integers:
•    #
•    #    P := 0                    | Initial value for partial sum
•    #    for i=0 to k-1            | k is the number of bits in B
•    #      P = 2P + A*B_{k-1-i}       | Bi is the i'th bit in B
•    #    return P
```

A*B mod n

```
•    # 2) Paper and pencil method for modular multiplication of two nonnegative
•    #    integers:
•    #
•    #    This can be done very similar to the algorithm explained in 1)
•    #
•    #
•    #    P := 0                    | Initial value for partial sum
•    #    for i=0 to k-1            | k is the number of bits in B
•    #      P = 2P + A*B_{k-1-i}   | Bi is the i'th bit in B
•    #      P = P mod N             | Interleaved reduction of P
•    #    return P
•    #
•    #    The largest possible number produced by P mod N is: N-1
•    #    The largest possible number produced by A*Bi is: N-1
•    #    Due to the interleaved reduction we will always have P in the range
•    #    0 <= P <= 2*(N-1) + (N-1) = 3N - 3
•    #    In order to reduce P to the range 0 <= P <= N-1 it is necessary with
•    #    at most two subtractions. The new pseudocode will then be:
•    #
•    #    P := 0                    | Initial value for partial sum
•    #    for i=0 to k-1            | k is the number of bits in B
•    #      P = 2P + A*B_{k-1-i}    | Bi is the i'th bit in B
•    #      if P >= N               |
•    #        P = P - N             |
•    #      if P >= N               | Reduction ...
•    #        P = P - N             |
•    #    return P                  |
```

Design specification

Design an RSA encryption/decryption circuit that meets the following requirements:

REQ1: [MUST]
The design must implement the RSA encryption algorithm.

REQ2: [MUST]
Encrypt/decrypt a message of length 256 bits as fast as possible.

REQ3: [MUST] The design must fit inside the Zynq-7000 FPGA on the Pynq-Z1 board.

REQ4: [MUST]
The RSA design must be integrated as an hardware accelerator inside the Zynq SoC. It must be managed by the CPU and made accessible through the Juniper notebook interface.

REQ5: [SHOULD]
The design should implement memory mapped status registers, performance counters and other mechanisms for debugging of features and performance at system level.

Design specification

Design an RSA encryption/decryption circuit that meets the following requirements:

REQ6: [MUST]

The design must have one AXI-Lite Slave interface to enable access of memory-mapped registers.

The design must have one AXI stream slave interface for messages that shall be encrypted(decrypted) and one AXI stream master interface for messages that have been encrypted(decrypted).

REQ7: [COULD]

The design could support a configurable message/key size {128, 256}.

Design specification

Design an RSA encryption/decryption circuit that meets the following requirements:

REQ8: [MUST]
A verification plan must be created and the design must be verified and meet the quality targets set in the verification plan. The verification plan must include the development of unit level testbench(es) as well as testing on the FPGA. The design must work.

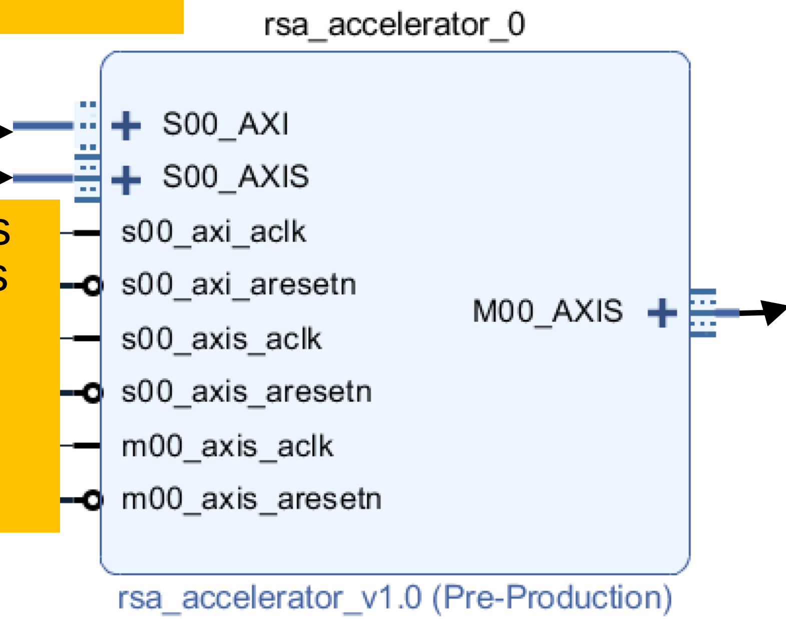
REQ9: [COULD]
The verification plan could include collection of code coverage, functional coverage as well as simple assertions.

REQ10: [MUST]
A functional high-level model of the algorithm must be created.

REQ11: [MUST]
A microarchitecture diagram must be created before any VHDL code is written.

Interface description

AXI SLAVE IF (S00_AXI)
used for reading and writing
-Key registers
-Status registers
-Performance counters



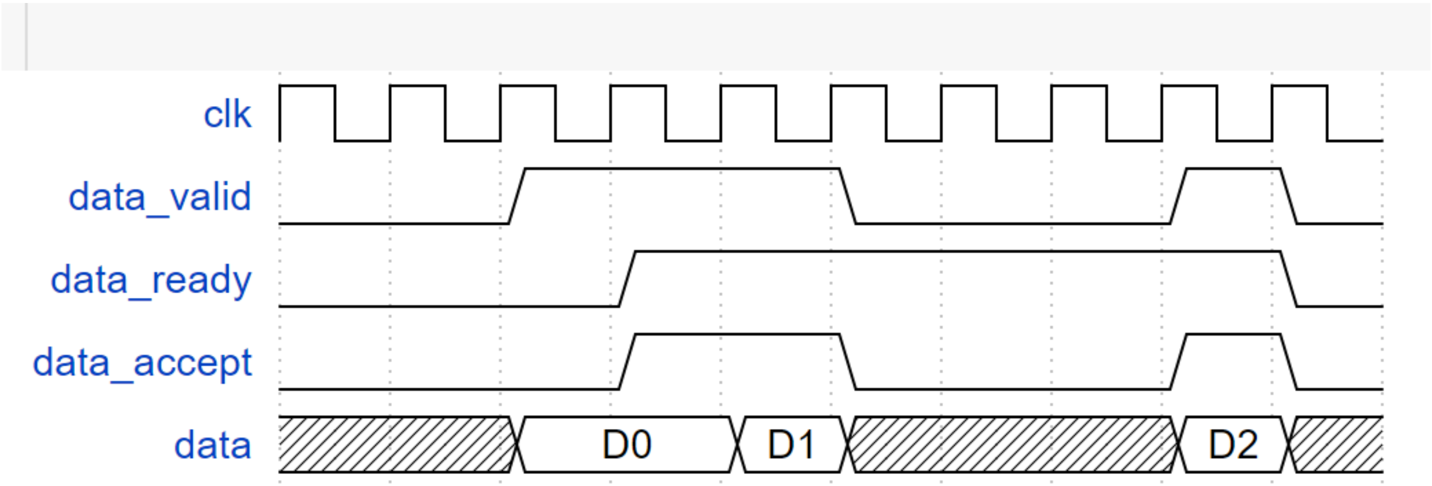
clocks
resets

AXI STREAM MASTER IF (M00_AXIS)
used for sending out messages that has been encrypted/decrypted

AXI STREAM SLAVE IF (S00_AXIS)
used for sending in messages to encrypt/decrypt

Interface description

Ready valid handshaking used for transferring information on all interfaces.



Ready valid handshaking (Chapter 22)

Data is transferred when both ready and valid are high at the same time.

This is illustrated with the derived helper signal named data_accept in the figure above

The handshaking is exactly the same independent of what information that is transferred. Common information includes *data*, *addresses*, *stream data*, *write responses etc.*

Memory mapped registers

Address	Register
0-7	Key N (8*32bit =256 bit)
8-15	Key E (8*32bit =256 bit)
16-23	User defined
24-31	User defined
32	Status register

Proposed use of the memory mapped registers, but this is entirely up to the group to decide.

Example integration

Will be available week 38

Summary of your task

- Major Goal - Design and Implement the RSA Circuit:

The circuit should be able to encrypt and decrypt a message of 256 bits as quick as possible. Design must fit inside the FPGA.

- Understand Algorithm and create model

Literature study of RSA and HW realization of RSA.

Create a high level model of the RSA algorithm.

- Create microarchitecture for HW design

Suggest a design that implements RSA according to the specification. Create block diagram for your solution. It must be possible to identify mux-es, registers, adders etc.

- Write RTL code and test

Write RTL code for your design based on the microarchitecture.

Summary of your task

- Subsequent works of this project:
- Simulate and Verify your circuit function
- Construct the system in hardware and perform physical implementation on a FPGA
- (Circuit shall be tested and verified in the lab)
- Expected result
- After the circuit has been verified and implemented on an FPGA, then it should be able to encrypt and decrypt messages.

Important points to consider…

If you choose to implement Montgomery modular multiplication as a part of your design, then you may want to use the extra user defined parameters for sending in:

1. r mod n
2. r^2 mod n

Practical Questions and Ans…

Actually the encryption and decryption basic formulas are,

Encryption: $C = M^e \bmod n$, $M < n$,
public key: $\{e, n\}$

Decryption: $M = C^d \bmod n$
private key: $\{d, n\}$

Note: Encryption and Decryption are essentially the same operations

Question:
When we implement them using either Normal or Montgomery algorithm in VHDL, what should be the first, second and so on?

Answer:
Normal: e (exponent) sent first, n (modulator) sent second
Montgomery: e (exponent) sent first, n (modulator) sent second, third R mod n and fourth R^2 mod n.

These are configuration packages during the setup of the chip.

Practical Question and Ans…

Question:
Do we just repeat the same modules/operations for decryption?

Answer: Yes
RSA is just reversible. You can just repeat the same procedure for decryption.

The configuration packages for decryption is:

Normal: d first, then n
Montgomery: d first, n second, third R mod n and fourth R^2 mod n

Note:
1. The same configuration package as with encryption, just the symbol e exchanged with the symbol d

2. When the chip is set up (has received a configuration package), it is ready to receive messages. It will then receive the message (M or C, depending on whether you are decrypting or encrypting) for each message-block you want encrypted/decrypted. (usually we send 0-31 first).

Recommended literature

1. C. K. Koc. *RSA Hardware Implementation*. RSA Laboratories, August 1995. (http://islab.oregonstate.edu/koc/papers/r02rsahw.pdf) (Se spesielt avsnitt 4 og avsnitt 7.1. **Les denne først**.)

2. R. L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communication of the ACM, Feb. 1978

Time schedule term project - tentative

Date		Milestone
Thursday	13. Sept	Term project introduction
Thursday	11. Oct	Microarchitecture review day. Whiteboard presentations for all groups . Each group prepare 2-3 powerpoint slides that describe the microarchitecture of their design. Each group will present for two one other groups as well as one representative from the staff.
Tuesday	22. Nov	Each group presents their solution. Focus on presenting the architecture, performance and area.
Sunday	25. Nov	Hand in the term project report