

NTUEE DCLAB

LAB 2: RSA256 解密機

Graduate Institute of Electronics Engineering
National Taiwan University

Outline

- Introduction
 - Lab requirements
- RSA256 cryptosystem
- System-on-Chip (SoC) and Qsys
- Implementation
 - RSA256 core
 - RSA256 wrapper
- Code template
- System setup and run testing program
- Report regulations

Introduction

- RSA256解密機
 - PC 端透過 RS232 傳輸金鑰與密文給 FPGA
 - FPGA 接收資料並進行解密運算
 - 解密完成後 FPGA 透過 RS232 將答案傳回給 PC 端
- 實驗目的
 - 實作巨大整數運算，了解不同運算方式對硬體效率的影響，體會硬體加速的不可取代性
 - 實作大型的輸入輸出界面，理解模組溝通的基礎模式與系統間通訊的匯流排(bus)觀念

Lab Requirements

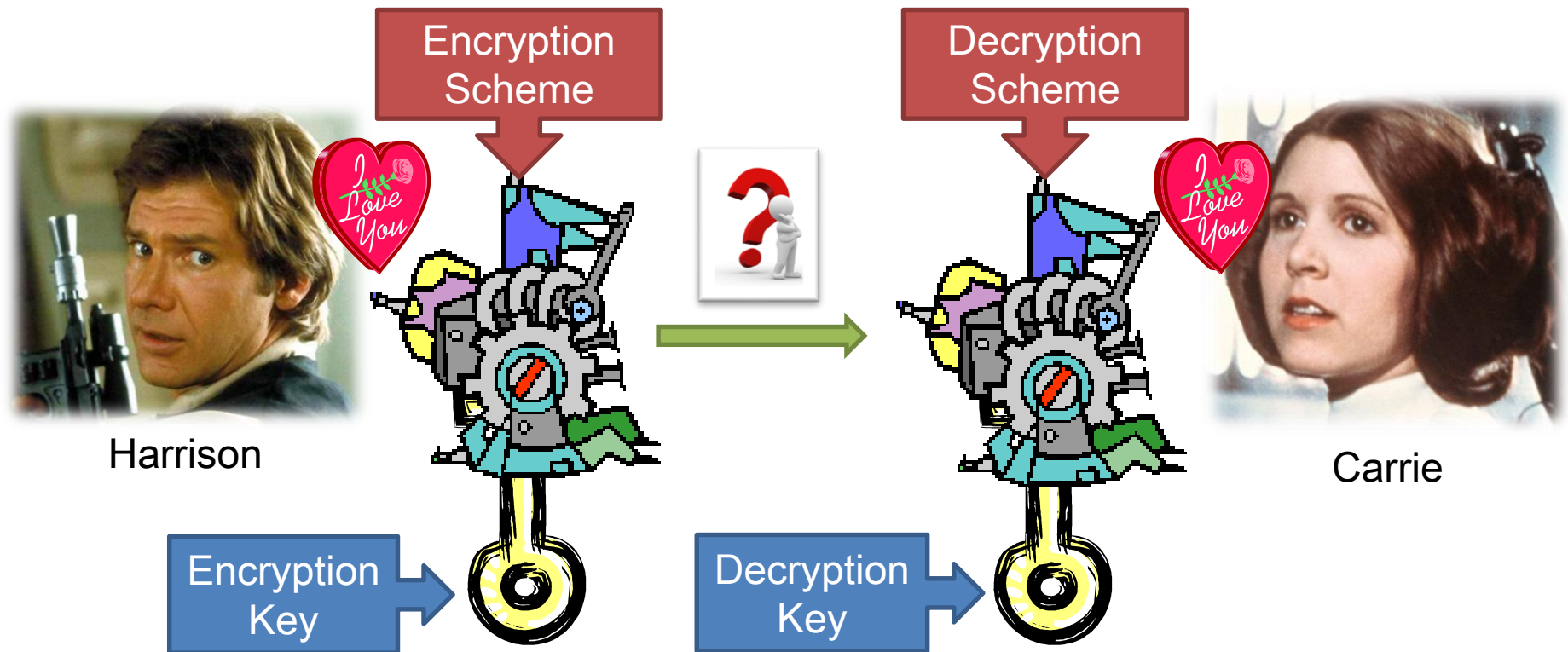
- Key0 可以reset
- 通過測資，正確解密
 - Code template 助教有提供三筆
 - 另外一筆隱藏測資會在 demo 當天公佈
 - 設計可為單次使用(每次解密前要先按reset)
- Bonus (demo 時與 report 中皆應清楚詳細說明)
 - 不需 reset 即可連續解密多份密文(不同金鑰)
 - 進行更多bit數字(> 256b)的解密運算
 - 其他能想到的變化

Outline

- Introduction
 - Lab requirements
- **RSA256 cryptosystem**
- System-on-Chip (SoC) and Qsys
- Implementation
 - RSA256 core
 - RSA256 wrapper
- Code template
- System setup and run testing program
- Report regulations

Introduction to Cryptography

- Communication is insecure
- Use cryptosystems to protect communications



RSA Cryptosystem

- Carrie select a key pair and release the **encryption key/public key (公鑰)**
 - Everyone knows the encryption key
 - RSA is a public cryptosystem
- If Harrison wants to communicate with Carrie, he uses the encryption key released by Carrie to encrypt
- Carrie uses the **decryption key/private key (私鑰)** to decipher the encrypted message
 - Only Carrie knows the decryption key
 - Message sent from Harrison to Carrie is secured

Key Pair Selection Scheme

- Randomly select 2 distinct prime numbers p and q
 - For security reason, $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also prime numbers.
- Evaluate $N = pq$ and $\Phi(N) = (p - 1)(q - 1)$
- Choose e such that $\gcd(e, \Phi(N)) = 1$
- Find the integer d ($0 < d < \Phi(N)$) such that $ed - k\Phi(N) = 1$
- Finally, release the number pair (N, e) and keeps $(d, p, q, \Phi(N))$ in secret
 - (N, e) is the public key
 - (N, d) is the private key

RSA Encryption and Decryption

- x is the message to be sent
- y is the encrypted message actually being sent

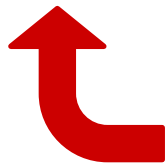
- Encryption

- $y = x^e \bmod N$

- Decryption

- $x = y^d \bmod N$

(N, e) is the public key
 (N, d) is the private key



For RSA256: p, q are 128b, the rest are 256b

Hard to calculate!

Need an efficient way to compute!

But how?

Exponentiation by Squaring

- **Reduce the amount of multiplication**
- Use the binary representation of the exponent
- Example: assume $d = 12$
 - $y^d = y^{12_{10}} = y^{1100_2} = (1 \cdot y^8) \cdot (1 \cdot y^4) \cdot (0 \cdot y^2) \cdot (0 \cdot y^1)$
 - $y^d \pmod{N} = [(y^8) \cdot (y^4)] \pmod{N}$

Algorithm 1 RSA256 with exponentiation by squaring

```
1: function EXPONENTIATIONSQUARING( $N, y, d$ )
2:    $t \leftarrow y$ 
3:    $m \leftarrow 1$ 
4:   for  $i \leftarrow 0$  to 255 do
5:     if  $i$ -th bit of  $d$  is 1 then
6:        $m \leftarrow m \cdot t \pmod{N}$ 
7:     end if
8:      $t \leftarrow t^2 \pmod{N}$ 
9:   end for
10:  return  $m$ 
11: end function
```

$\triangleright t \rightarrow t^2 \rightarrow t^4 \rightarrow \dots$

Modulo of Products

- Now, $y^d \bmod(N)$ becomes several $ab \bmod(N)$ operations
 - **Further replace multiplication with additions**
- Example: assume $a = 12$
 - $ab \bmod(N) = 12_{10} \cdot b \bmod(N) = 1100_2 \cdot b \bmod(N) = (8b + 4b) \bmod(N)$
 - $8b = 2 \cdot 4b = 2 \cdot 2 \cdot 2b$ (can be compute with similar way)
- Perform modulo operation every iteration to prevent overflow

Algorithm 2 Modulo of products

```
1: function MODULOPRODUCT( $N, a, b, k$ ) ▷  $k$  is number of bits of  $a$ 
2:    $t \leftarrow b$ 
3:    $m \leftarrow 0$ 
4:   for  $i \leftarrow 0$  to  $k$  do
5:     if  $i$ -th bit of  $a$  is 1 then
6:       if  $m + t \geq N$  then
7:          $m \leftarrow m + t - N$  ▷ perform modulo operation in each iteration
8:       else
9:          $m \leftarrow m + t$ 
10:      end if
11:    end if
12:    if  $t + t > N$  then
13:       $t \leftarrow t + t - N$  ▷ perform modulo operation in each iteration
14:    else
15:       $t \leftarrow t + t$ 
16:    end if
17:  end for
18:  return  $m$ 
19: end function
```

Montgomery Algorithm

- However, Algorithm 2 still needs comparison (for modulo operation) in each iteration
- Alternative method to prevent overflow
 - Calculating $ab \cdot 2^{-i} \bmod(N)$
- Example: assume $a = 12, i = 4$
 - $ab \cdot 2^{-4} = 4'b1100 \cdot b \cdot 2^{-4} = \left(\left((0 \cdot 2^{-1} + 0) \cdot 2^{-1} + b \right) \cdot 2^{-1} + b \right) \cdot 2^{-1}$
 - 2^{-1} is multiplied in every iteration so it won't overflow

More on Montgomery Algorithm

Algorithm 3 Montgomery algorithm for calculating $ab2^{-256} \bmod N$

```
1: function MONTGOMERYALGORITHM( $N, a, b$ )
2:    $m \leftarrow 0$ 
3:   for  $i \leftarrow 0$  to 255 do
4:     if  $i$ -th bit of  $a$  is 1 then
5:        $m \leftarrow m + b$ 
6:     end if
7:     if  $m$  is odd then
8:        $m \leftarrow m + N$ 
9:     end if
10:     $m \leftarrow \frac{m}{2}$ 
11:  end for
12:  if  $m \geq N$  then
13:     $m \leftarrow m - N$ 
14:  end if
15:  return  $m$ 
16: end function
```

▷ 4~6: replace multiplication with successive addition

▷ 7~10: calculate the modulo of $a \cdot 2^{-1}$
→ Montgomery reduction

RSA256 with Montgomery Algorithm

- The final algorithm
 - Remember to multiply y (b) by 2^{256} in advance

Algorithm 4 RSA256 with exponentiation by squaring and Montgomery algorithm

```
1: function RSA256MONT( $N, y, d$ )
2:    $t \leftarrow \text{ModuloProduct}(N, 2^{256}, y, 256)$   $t = y * 2^{256}$ 
3:    $m \leftarrow 1$ 
4:   for  $i \leftarrow 0$  to 255 do
5:     if  $i$ -th bit of  $d$  is 1 then
6:        $m \leftarrow \text{MontgomeryAlgorithm}(N, m, t)$   $m * t * 2^{-256} \pmod{N}$ 
7:     end if
8:      $t \leftarrow \text{MontgomeryAlgorithm}(N, t, t)$   $t * t * 2^{-256} \pmod{N}$ 
9:   end for
10:  return  $m$ 
11: end function
```

- For RSA256, we use $i = 256$
 - $ab \pmod{N} = ab \cdot 2^{256} \cdot 2^{-256} \pmod{N} = ab' \cdot 2^{-256} \pmod{N}$
 - $b' = b * 2^{256}$

Outline

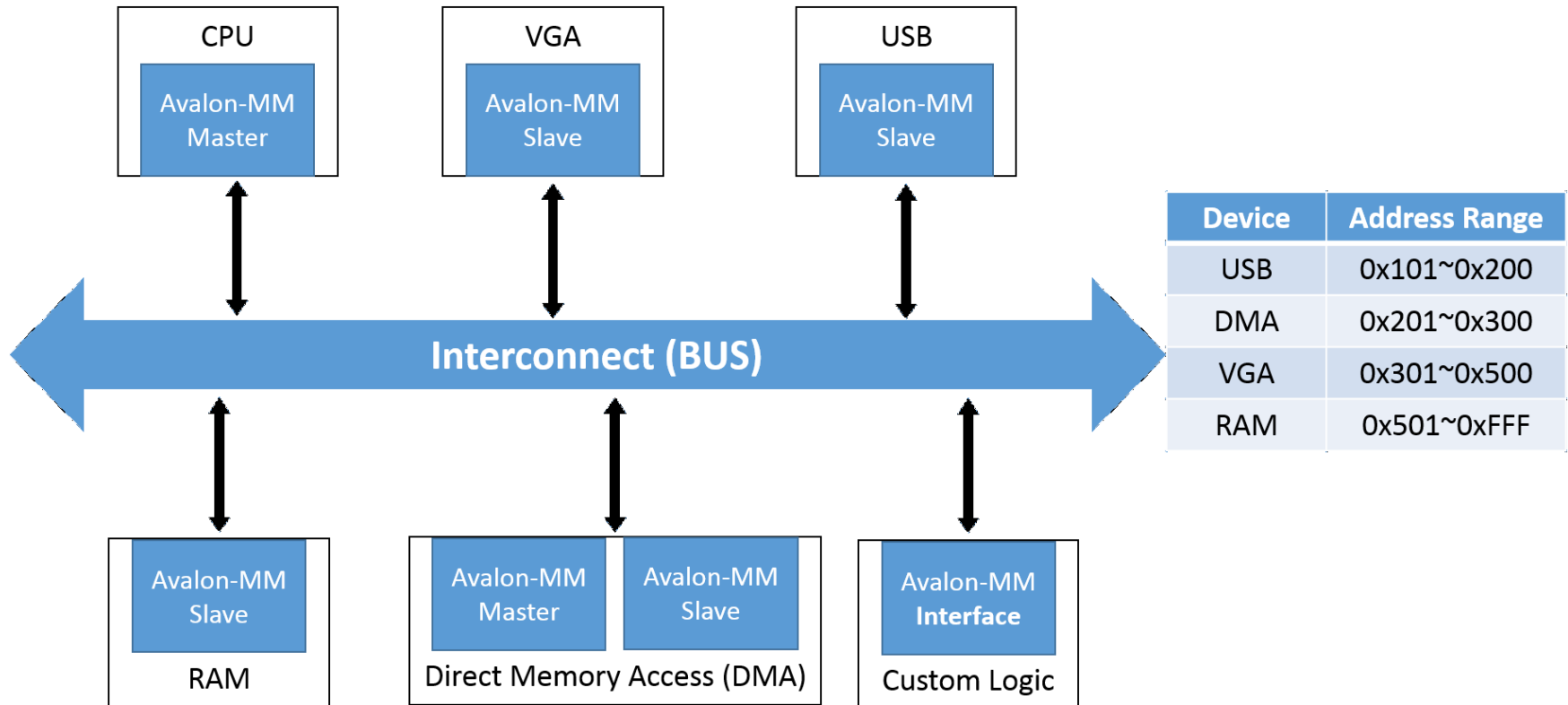
- Introduction
 - Lab requirements
- RSA256 cryptosystem
- **System-on-Chip (SoC) and Qsys**
- Implementation
 - RSA256 core
 - RSA256 wrapper
- Code template
- System setup and run testing program
- Report regulations

System-on-Chip (SoC)

- Integrate the entire system onto a single chip
 - May contain digital, analog, mixed-signal, RF functions
 - Allows large systems to be built with existing modules
- Master
 - initialize requests
- Slave
 - respond to requests
- Bus
 - interconnection between master and slave IPs
 - The protocols are similar to memory read/write
 - Ex: AMBA/AHB/AXI (ARM), Avalon (Altera)

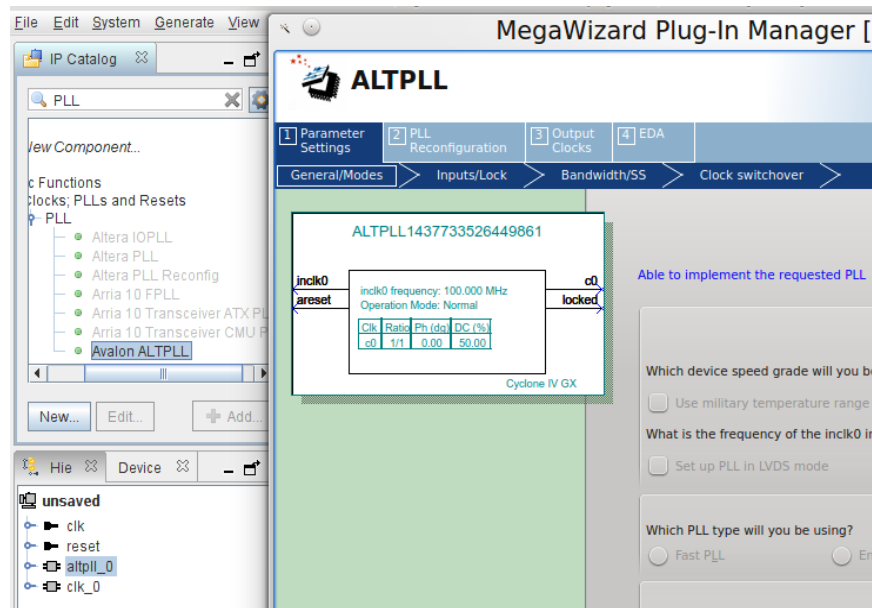
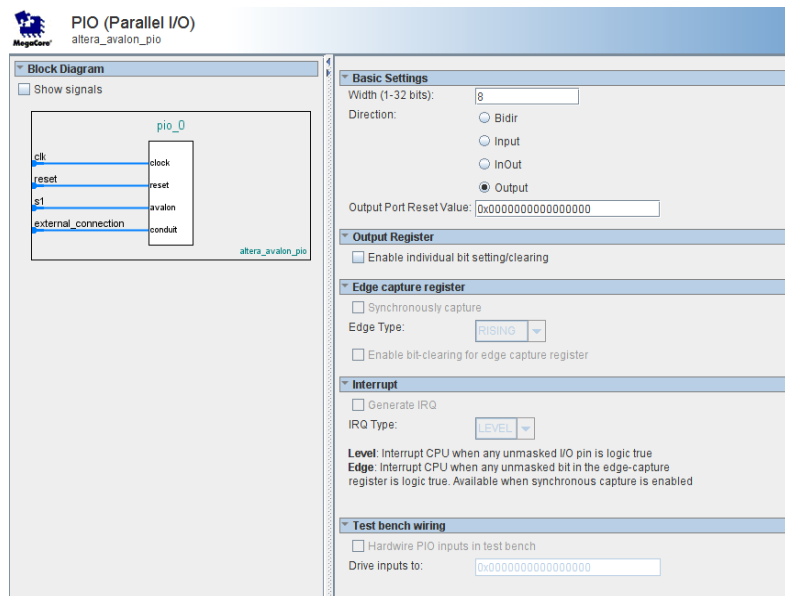
Memory Mapped I/O (MMIO)

- CPU uses address to access the RAM
- Some addresses in SoC are mapped to I/O of IP
 - Access them just like accessing the RAM



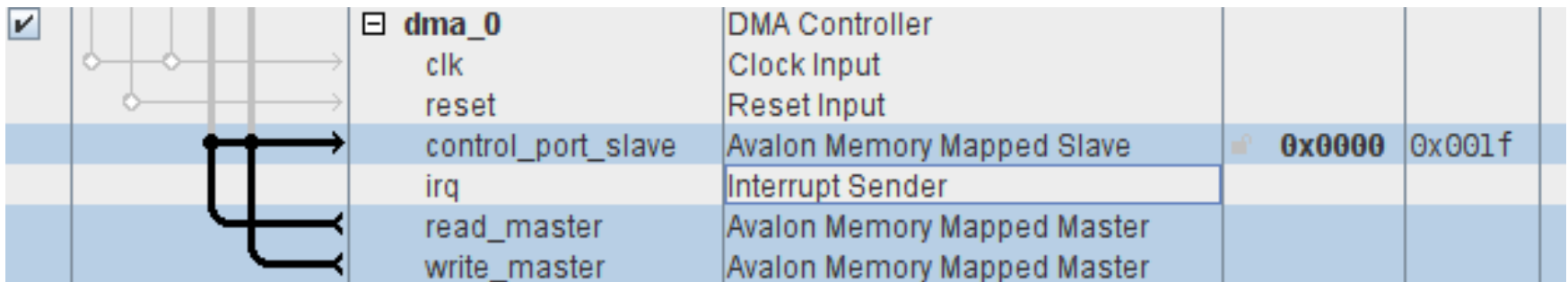
Qsys: Altera SoC Tool

- Upon opening Qsys, there will be a **clock source module**
 - Converts raw signals (conduits) to clock and reset (negedge)
- **Parallel I/O modules** can create read/write slave interface
 - For key, switch, LED, ...
- **PLL** converts 50MHz (default clock) to almost any frequency



More on Qsys

- You can add more modules to the design
 - Like the core and wrapper you implemented (as master)
 - Possible connection will be displayed, click to enable
- Connected signals are colored black
 - Slaves are associated with address ranges
 - Masters uses this address to access the slaves

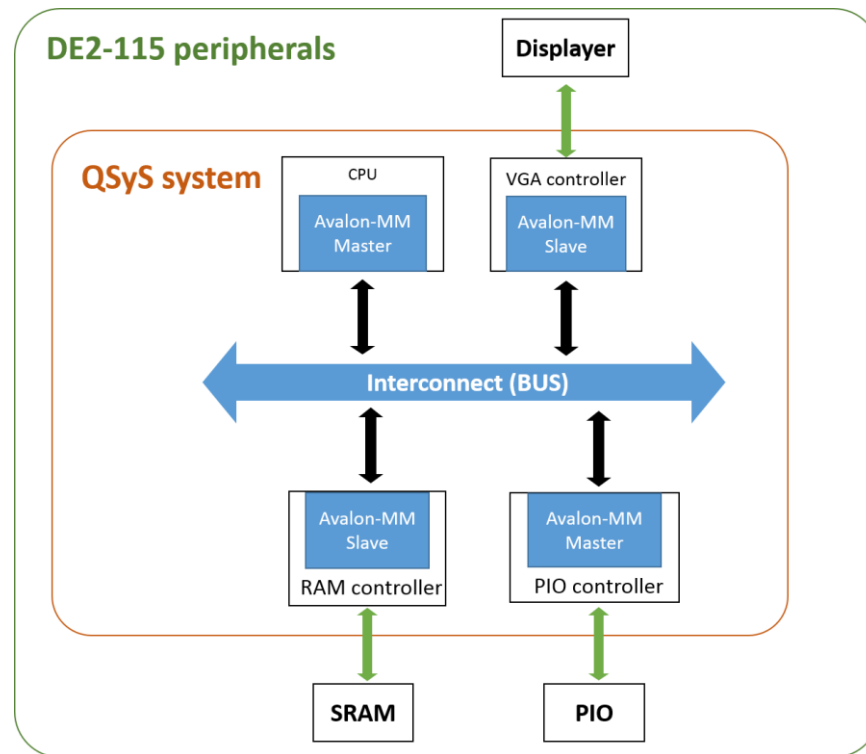


The screenshot shows the Qsys component browser with a search filter 'dma_0'. The component 'dma_0' is expanded, showing its ports and their connections. The connections are as follows:

| Port | Connection | Address Range |
|--------------------|-----------------------------|-----------------|
| clk | Clock Input | |
| reset | Reset Input | |
| control_port_slave | Avalon Memory Mapped Slave | 0x0000 - 0x001f |
| irq | Interrupt Sender | |
| read_master | Avalon Memory Mapped Master | |
| write_master | Avalon Memory Mapped Master | |

Add Qsys Module to Your Design

- Generate Qsys qip module and Verilog file
- Add the qip to your project and connect the corresponding wires under the top module (DE2_115.sv)



Follow the step-by-step tutorial to generate your Qsys module for Lab2!

Outline

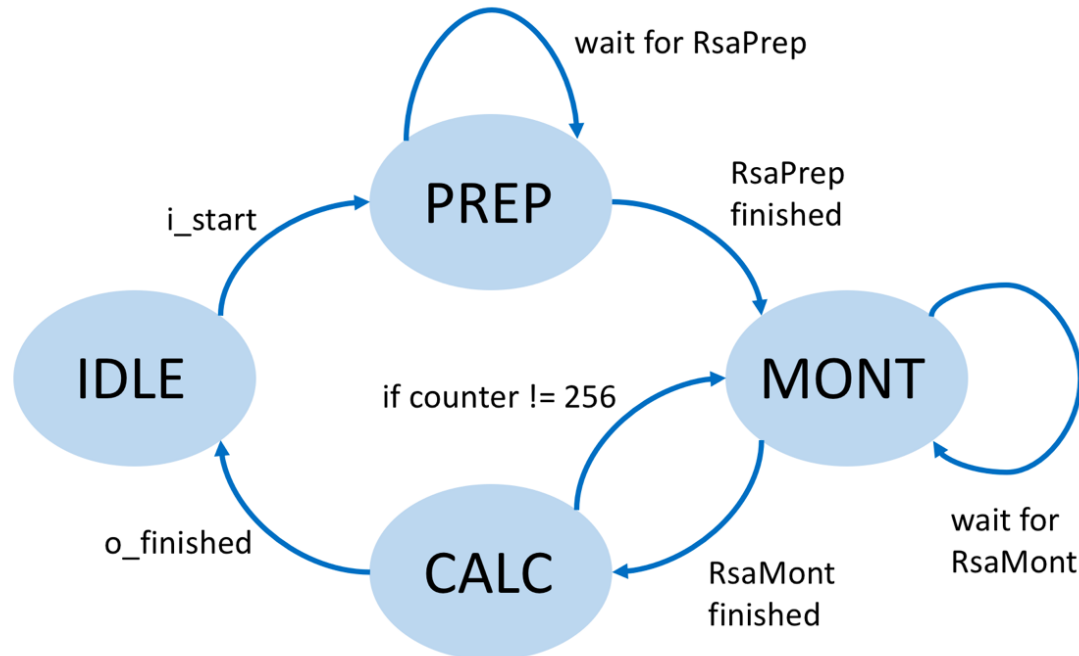
- Introduction
 - Lab requirements
- RSA256 cryptosystem
- System-on-Chip (SoC) and Qsys
- **Implementation**
 - RSA256 core
 - RSA256 wrapper
- Code template
- System setup and run testing program
- Report regulations

General Roadmap

- Create a project
- Implement the RSA256 core
- Implement a wrapper to control RS232 and your core
- Build Qsys system
- Compile and program

RSA256 Core

- Divide the functions based on Algorithm 4 into submodules
 - During PREP, execute submodule **RsaPrep** (ModuloProduct)
 - Calculate $y \cdot 2^{256} \bmod(N)$
 - During MONT, execute submodule **RsaMont**
 - Calculate $t \leftarrow t^2 \cdot 2^{-256} \bmod(N)$ and $m \leftarrow mt \cdot 2^{-256} \bmod(N)$
 - Dedicate one instant to each calculation for parallel processing



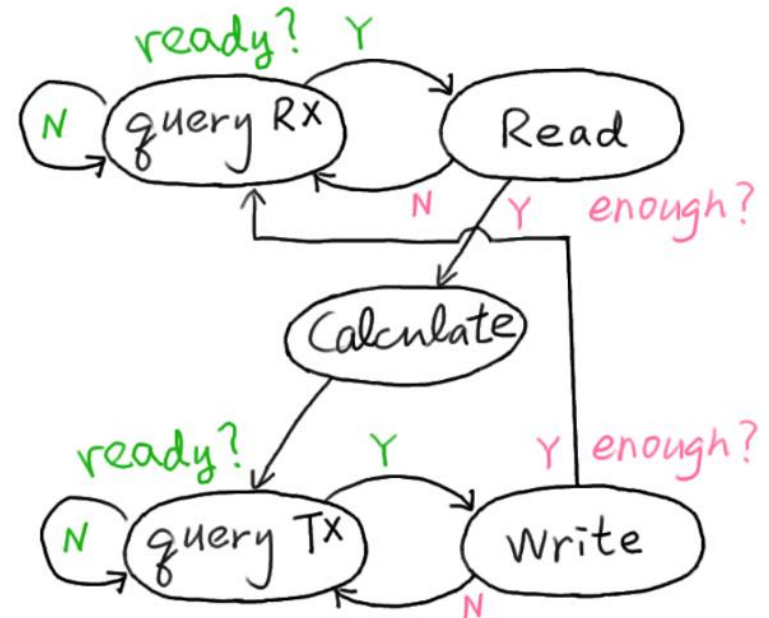
RS232

- Very old (1969) and very simple protocol
 - Only has two signal lines receiver/transmitter (RX/TX)
- But very slow (~10KB/s)
- Here, we use Qsys IP
 - Access different data by address BASE+0, 4, 8, ...

| Offset | Register Name | R/W | Description/Register Bits | | | | | | | | | | | | | |
|--------|--------------------|-----|---------------------------|------|-----|-------|------|----|-----------|---------------------|------|------|------|------|-----|-----|
| | | | 15:13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | rxdata | RO | Reserved | | | | | 1 | 1 | Receive Data | | | | | | |
| 1 | txdata | WO | Reserved | | | | | 1 | 1 | Transmit Data | | | | | | |
| 2 | status 2 | RW | Reserved | eop | cts | dcts | 1 | e | rrd y | trd y | tmt | toe | roe | brk | fe | pe |
| 3 | control | RW | Reserved | ieop | rts | idcts | trbk | ie | irrd y | itrdr y | itmt | itoe | iroe | ibrk | ife | ipe |
| 4 | divisor 3 | RW | Baud Rate Divisor | | | | | | | | | | | | | |
| 5 | endof- packet 3 | RW | Reserved | | | | | 1 | 1 | End-of-Packet Value | | | | | | |

RSA256 Wrapper

- 操作Qsys生成的RS232 IP
 - 先讀入資料(key & cipher text)
 - 讀取完後交給core進行解密
 - 將解密完資料(plain text)寫出



- 在讀寫前要先確定IP準備好了
 - 讀取BASE+8的[7]和[6]
(前頁螢光筆標示處)
 - Ex: 當addr給BASE+8，readdata[7]代表RX準備情況
- 讀寫時每次只有8 bits
 - 所以每一筆256b資料要分32次讀
 - Ex: 當addr給BASE+0，readdata[7:0]是RX送來的8b資料

Outline

- Introduction
 - Lab requirements
- RSA256 cryptosystem
- System-on-Chip (SoC) and Qsys
- Implementation
 - RSA256 core
 - RSA256 wrapper
- **Code template**
- System setup and run testing program
- Report regulations

Code Template

- DE2_115/
 - Design setup files
- pc_python/
 - Python executable test program for PC
- tb_verilog/
 - Verilog testbench for core and wrapper
- [Rsa256Core.sv](#)
 - Implement RSA256 decryption algorithm here.
- [Rsa256Wrapper.sv](#)
 - Implement controller for RS232 protocol
 - Including reading check bits and read/write data.

Core and Wrapper Modules

```
module Rsa256Core (  
    input        i_clk,  
    input        i_rst,  
    input        i_start,  
    input [255:0] i_a, // cipher text y  
    input [255:0] i_d, // private key  
    input [255:0] i_n,  
    output [255:0] o_a_pow_d, // plain text x  
    output        o_finished  
);
```

```
module Rsa256Wrapper (  
    input        avm_rst,  
    input        avm_clk,  
    output [4:0] avm_address,  
    output        avm_read,  
    input [31:0] avm_readdata,  
    output        avm_write,  
    output [31:0] avm_writedata,  
    input        avm_waitrequest  
);
```

Debug Core and Wrapper

- Testbench for core and wrapper are provided in tb_verilog/
- To run simulation for core
 - `ncverilog +access+r tb.sv Rsa256Core.sv`
- To run simulation for wrapper
 - `ncverilog +access+r test_wrapper.sv PipelineCtrl.v \ PipelineTb.v Rsa256Wrapper.sv Rsa256Core.sv`
- Use `nWave` to check the waveform and happy debugging!
 - It is advised to test individual modules first

Outline

- Introduction
 - Lab requirements
- RSA256 cryptosystem
- System-on-Chip (SoC) and Qsys
- Implementation
 - RSA256 core
 - RSA256 wrapper
- Code template
- **System setup and run testing program**
- Report regulations

System Setup



Testing Program

- Environment setup
 - Install Python2
 - ez_setup.py (<https://pypi.org/project/setuptools/>) or sudo apt-get install python-pip
 - (sudo) pip install pySerial
- Usage
 - Copy key and encrypted data (enc.bin and key.bin) next to the executable
 - ./rs232.py [COM? | /dev/ttyS0 | /dev/ttyUSB0]
- Several test data are already provided

Decryption Flow

- The executable will
 - Send 32-byte divisor N
 - Send 32-byte exponent d
 - Loop
 - Send 32B cipher text y
 - (Your module calculates the result)
 - Receive 31-Byte plain text x
- Note: a zero byte is padded to the front of each 32-byte plain text to prevent overflow
 - The size of plain text is 31 bytes
 - The size of cipher text in enc.bin is 32 bytes

Outline

- Introduction
 - Lab requirements
- RSA256 cryptosystem
- System-on-Chip (SoC) and Qsys
- Implementation
 - RSA256 core
 - RSA256 wrapper
- Code template
- System setup and run testing program
- Report regulations

Report Regulations

- 內容應包含
 - 層級架構
 - Block Diagram (必須包含Data Path，control signal可有可無)
 - FSM or Scheduling
 - Fitter Summary 截圖
 - Timing Analyzer 截圖
 - 遇到的問題與解決辦法
- 一組交一份，以pdf檔繳交
- 命名方式：teamXX_lab2_report.pdf
 - Ex: team01_lab2_report.pdf
- 繳交期限：demo當天午夜
 - 遲交每三天*0.7

Questions?