

2010

MAHOHard Software Inc.

Hardware Security Module

DETAILED DESIGN REPORT

e1502228 Ali CANDER
e1502152 Ozan Erdem BAYCAN
e1502764 Haldun TOPÇUOĞLU
e1503028 Mustafa ŞİŞMAN

MAHOHard Software Inc.
01/18/2010



Contents

1-Introduction.....	5
1-1 Project Definition and Goals	5
1-2 Purpose of Document	6
2-Design Constraints.....	6
2-1 Resource Constraints.....	6
2-2 Power Constraints	6
2-3 Time Constraints.....	7
2-4 Ergonomic Constraints	7
2-5 Performance Constraints	7
2-6 Experience of Members	7
3-System Architecture	7
3-1 Overview of HSM.....	7
3-2 Architectural Design	8
3-2-1 Hardware Design	8
3-2-2 Software Design.....	12
4- Modeling	13
4-1 Data Flow Diagrams	13
4-1-1 Level 0.....	13
4-1-2 Level 1 : HSM	14
4-1-3 Level 2 : FPGA MODULE	15
4-2 Class Diagrams	17
4-2-1 Ethernet Module	17
4-2-2 FPGA Module	19
4-3 Activity Diagrams.....	26
4-3-1 Ethernet Activity Diagram	26
4-3-1 FPGA Activity Diagram.....	27
4-4 Sequence Diagrams.....	28



4-4-1- Packet Transportation	28
4-4-2 Microcontroller – Crypto Module Interaction	29
5- Microprocessor/Microcontroller	30
5-1- TSK3000A	31
5-1-1 Pin Description	31
6- INTERFACE.....	36
6-1 External Interface.....	36
6-1-1 Ethernet Port.....	36
6-1-2 Ethernet Interface.....	37
6-2- Internal Interface	39
6-2-1- Wishbone Interface.....	39
7- Language Specifications	40
7-1- Embedded C /C++	40
7-2- VHDL	40
8- Testing and Debugging	41
8-1- Testing	41
8-1-1- Unit Testing.....	41
8-1-2- Integration Testing.....	41
8-1-3- System Testing	42
8-2- Debugging.....	42
9- Gantt Chart.....	43
9-1- Term 1 Gantt Chart.....	43
9-2- Term 2 Gantt Chart.....	44
10- References	45



Table of Figures

Figure 1: Design Overview	8
Figure 2: Functional Overview of nanoboard.....	10
Figure 3: Overall Component of Hardware Architecture.....	11
Figure 4: Features of Altium Designer.....	12
Figure 5: Context Level DFD.....	13
Figure 6: Level1 DFD : HSM.....	14
Figure 7: Level 2 DFD: FPGA Modules	16
Figure 8: Ethernet Module	17
Figure 9: FPGA Module.....	19
Figure 10: General Structure of Encryption/Decryption.....	22
Figure 11: Each round of AES	22
Figure 12: Working principal of DES.....	23
Figure 13: Each round of DES	24
Figure 14: Ethernet Activity Diagram	26
Figure 15: FPGA Activity Diagram.....	27
Figure 16: Packet Transportation.....	28
Figure 17: Interaction between Microcontroller and Crypto Module	29
Figure 18: TSK3000A.....	31
Figure 19: Ethernet network packet holding an IP packet.....	38
Figure 20: High-level illustration of C-to-Hardware compilation in Altium Designer	40



1-Introduction

1-1 Project Definition and Goals

Securely managing keys is one of the most important and resource consuming tasks required to guarantee the security on a public key crypto system. This is due to a close relationship between security and the proper management of private keys. A public key crypto system can be considered secure as long as the private keys are secured. Taking this as a premise, it should be guaranteed that a (private) key is strictly secure during all events in its life cycle. This goal can be achieved by designing systems to securely create, manage and destroy (private) keys, maintaining an audit trail of every operation which was done during their existence. Such systems are known as Hardware Security Modules (HSMs).

HSMs are specialised tamper-proof devices in which cryptographic functions and embedded software have been built to properly manage keys and control their life cycles. They are designed in such a way that if an unauthorised attempt to access them is made, this is considered an attempt to tamper and all critical internal parameters and keys are destroyed.

Although very common in the banking industry, HSMs are also desirable in PKI, but not always implemented. As shown in Table 1, their common usage in the banking industry leads to specialisation of the HSMs to perform tasks such as PIN calculations or payment protocols, that are suitable in such industry.

Bank HSMs	PKI HSMs
PIN Calculation	Strong Authentication
Role Based Authentication	Identity Based Authentication
Dual Key Entry	Strict Key-life Cycle Control
Payment Protocols	Fully Auditable Operation
Cryptographic Speed	Triggered Group Mechanisms

Table 1: Comparison Between Bank HSMs and PKI HSM



In this project, it will be tried to develop a PKI HSM. The goals of this HSM are :

- onboard secure generation
- onboard secure storage
- use of cryptographic and sensitive data material
- offloading application servers for complete asymmetric and symmetric cryptography

HSMs provide both logical and physical protection of these materials from non-authorized use and potential adversaries. In short, they protect high-value cryptographic keys.

1-2 Purpose of Document

The purpose of this document is to show the detailed design concepts about HSM project. In this document it will be given details of this project according to requirements explained in the requirement analysis report.

2-Design Constraints

2-1 Resource Constraints

There will be need of datasheets of the devices that will be used for this project and manuals of the software development environment that will be used for coding. These documents will be supplied by teaching assistant and whenever extra information is needed, internet resources will be used. Since this project is an hardware project and similar projects are commercial and are not open source, it will be hard to find related resources. That is why there will be limitations in the development progress.

2-2 Power Constraints

Since Hardware Security Module (HSM) has very critical task, which has not to be interrupted, the power must have some features:



- Power must be supplied continuously without any drop and rising.
- Power supply must supply a voltage in a range. For example, 90-132 and 175-264 VAC.

2-3 Time Constraints

The deadline of the project is June and a prototype should be provided at the end of this semester. Since this project is an embedded project, time is very important constraint. Time have to be used very effective in order to achieve some results.

2-4 Ergonomic Constraints

Since new platforms such as “Altium Designer” will be used which is new for all team members, there may be some problem.

2-5 Performance Constraints

First, the HSM must provide a significant speed for data transferring and all other functionality. Besides, when number of transferred data increase, the HSM must also provide parallelism. For example if there are more than one data will be encrypted, the HSM must share these data between suitable modules. By that way, in one time more than one data can be encrypted. Supplying these features will be big deal.

2-6 Experience of Members

Lack of experience of the team members on coding for embedded device is one of the restrictions. Sometimes, some difficulties may be faced with managing unexpected problems and unforeseen details of the project.

3-System Architecture

3-1 Overview of HSM

As it is explained throughout the report, the HSM system needs a complex architecture because lots of modules will work cooperatively. Therefore, the architecture should be easily modifiable according to changes and it should allow developers for developing new modules. Moreover, it should make this complex system's development phase less difficult with good separation of layers. General overview of the HSM can be seen at Figure 1.



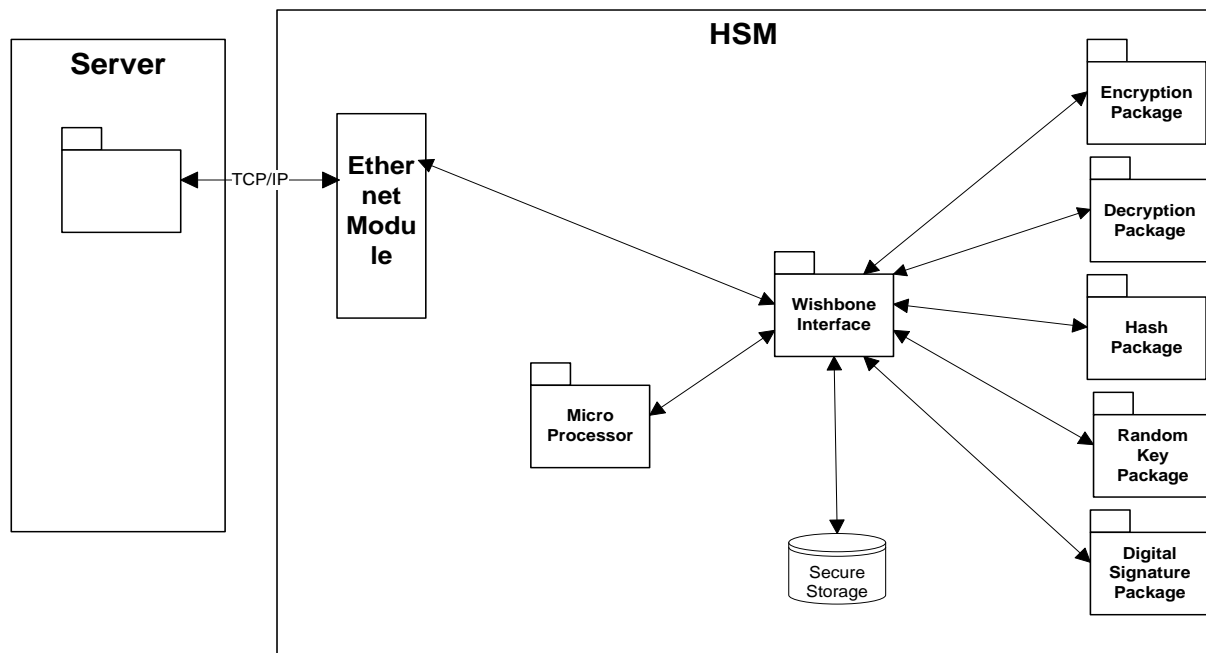


Figure 1: Design Overview

3-2 Architectural Design

3-2-1 Hardware Design

The HSM will be designed on a board which is called Altium Nanoboard 3000. This board has been chosen because of some of its features and advantages. Some advantages of this board are:

- Perfect entry-point to discover and explore the world of FPGA-based embedded systems design. Programmable hardware realm allows designer to update the design quickly and many times over without incurring cost or time penalties
- Reprogrammable hardware development platform that harnesses the power of a dedicated high-capacity, low-cost programmable device to allow rapid and interactive implementation and debugging of our designs
- High-capacity FPGA located on the motherboard, and provision for a single plug-in peripheral board (Altium or user's own) for additional system flexibility.
- Automatic peripheral board detection and configuration.



This board has some basic and important features which make us choose it. Some of them are:

- NanoBoard 3000XN – with fixed Xilinx® Spartan™-3AN device (XC3S1400AN-4FGG676C)
- Variety of standard communications interfaces: RS-232, RS-485, PS/2, 10/100 Fast Ethernet, USB 2.0, S/PDIF, MIDI.
- On-board memories accessible by user FPGA 256KB x 32-bit common-bus SRAM (1MB), 16M x 32-bit common-bus SDRAM (64MB), 8M x 16-bit common-bus 3.0V Page Mode Flash memory (16MB), dual 256KB x 16-bit independent
- SRAM (512KB each).
- Four 8Mbit SPI flash memory devices – one containing Primary boot image for Host Controller, one containing golden boot image for Host Controller, two for use by user FPGA (for boot/embedded purposes).
- Host (NanoTalk) Controller hosts the NanoBoard firmware. Responsibilities include managing JTAG communications (with Altium Designer/User FPGA/connected peripheral board), as well as access to common-bus SPI resources.
- High-speed PC interconnection through USB 2.0 allows for fast downloading and debugging. ⁱ

Since the HSM implementation will be on Altium nanoboard, all basic functionality and peripherals of board have to be known. Below in Figure 2, there is functional overview of this board.



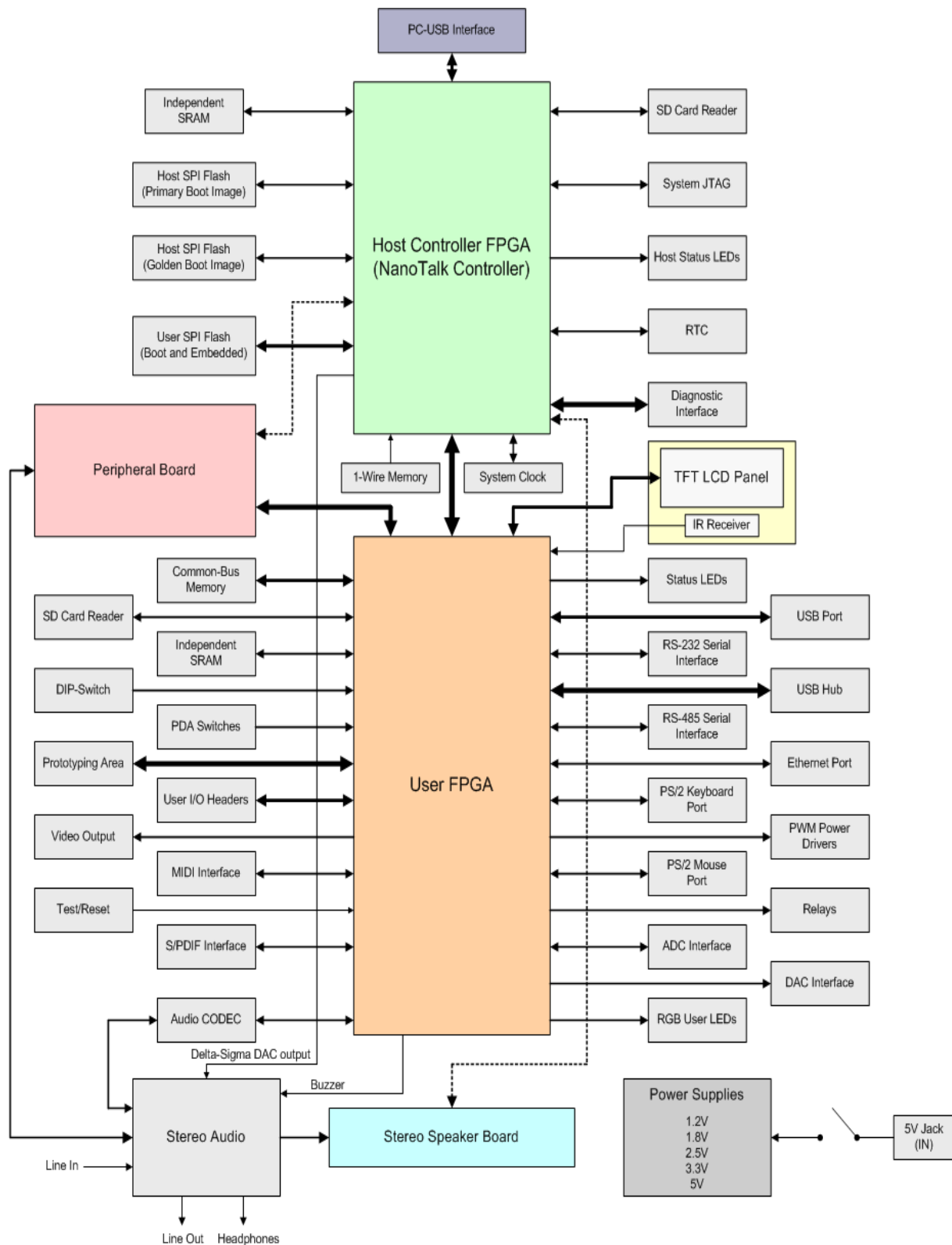


Figure 2: Functional Overview of nanoboard.ⁱⁱ

3-2-1-1 Hardware Components

The hardware components are on the User FPGA part of nanoboard. Other peripherals of nanoboard will be used in order to communication, storage, debugging etc. For example, Ethernet port will be used for communication with server. Besides, all components inside User FPGA will talk with each other by wishbone interface. Main picture of components are shown in Figure 3:

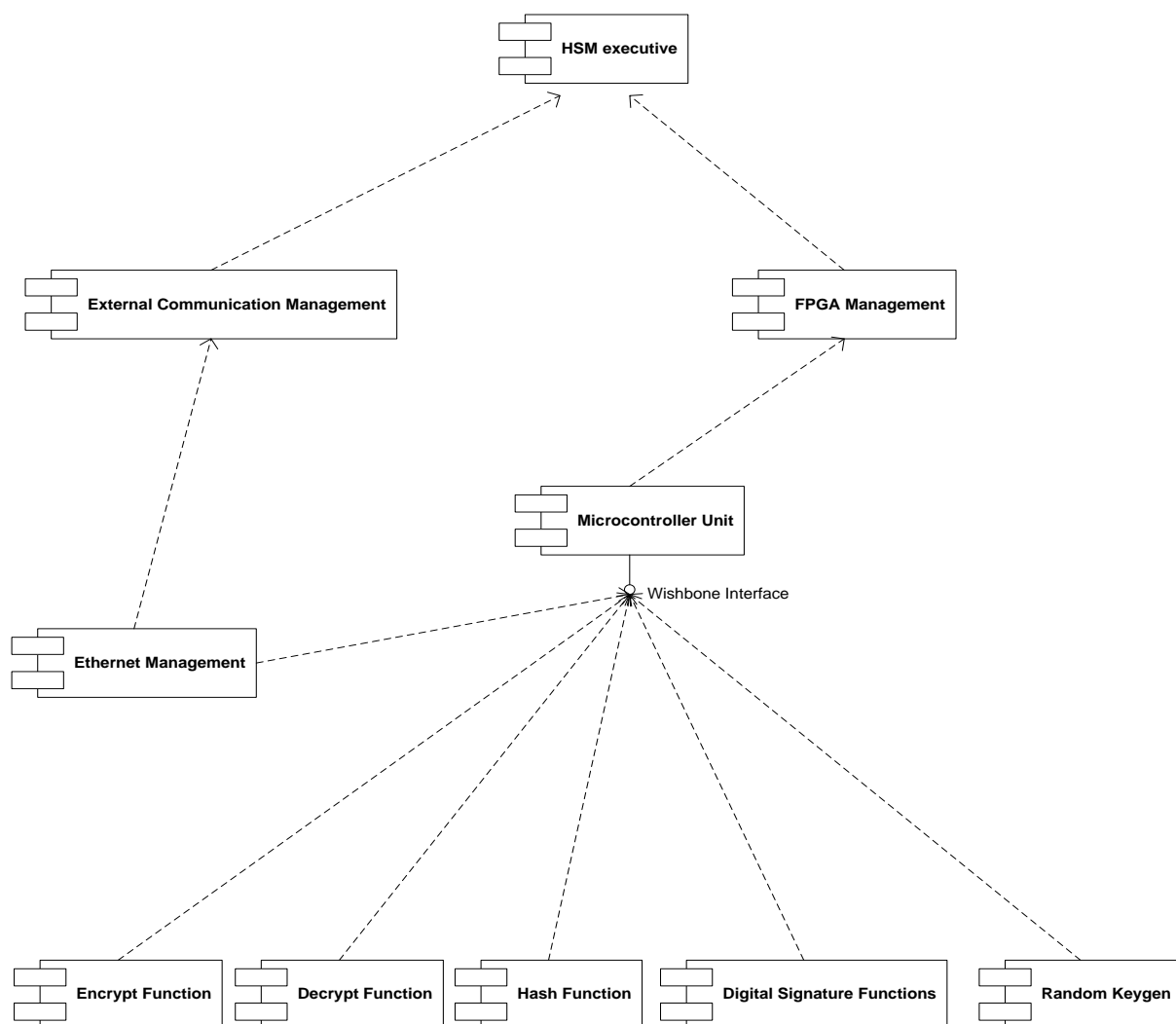


Figure 3: Overall Component of Hardware Architecture



HSM Executive is main picture of our HSM project. It has two main parts, which are external communication and FPGA management parts. External communication part is used for communication with server. FPGA management part is used for cryptographic functions, time scheduling and data management. FPGA will be composed of 6 parts. Five of them are for cryptographic functions and one is for microcontroller/microprocessor. All data management and time scheduling will be controlled by using microcontroller unit. In this report working hierarchy of all parts will be explained in detail with data flow and class diagrams.

3-2-2 Software Design

Since Altium Nanoboard will be used, “Altium Designer” is chosen as a product development system. Altium Designer involves all needed libraries for the board and other devices that is need in the project development. This system offers a single solution to develop hardware, programmable hardware and software. Besides, it is very easy to debug the work by using this system. Figure 4 briefly shows features of Altium Designer.

DXP Platform	Software integration platform, consistent GUI provided for all supporting editors and viewers, Design Insight for design document preview, design release management, design compiler, file management, version control interface and scripting engine
Schematic – Viewer	Open, view and print schematic documents and libraries
PCB – Viewer	Open, view and print PCB documents, additionally view and navigate 3D PCBs
CAM File – Viewer	Open CAM and mechanical files
Schematic – Soft Design Editing	All schematic and schematic library editing capabilities (except in PCB Projects and Free Documents), netlist generation
Simulation – VHDL	VHDL simulation engine, integrated debugger and waveform viewer, with third-party support for ModelSim and Active-HDL
NanoBoard Support	Range of auto-configured, swappable target FPGA daughter boards (from all chip vendors) are supported plus plug-in peripheral boards for complete flexibility in system architecture, Power Monitor for FPGA designs
FPGA Design	Custom FPGA Logic Development in C, OpenBus, Schematic, VHDL and Verilog design synthesis, Custom Wishbone Interface Component
FPGA Processor Cores	Support for a range of 32-bit soft processors for use in FPGA design: TSK3000A, Xilinx MicroBlaze®, Altera Nios II®, Actel CoreMP7®. Also support for the PowerPC (PPC405A) discrete processor, immersed in the Xilinx Virtex II Pro®, as well as a number of legacy, 8-bit Microcontrollers (TSK51, TSK52, TSK80 and TSK165)
Processor Core Embedded Tools	Full software development tool chain – C compiler/assembler/source-level debugger/profiler for each supported 32-bit processor; Plug-n-Play Software Platform Builder for easier hardware access
Programmable FPGA-Based Instruments	Presynthesized FPGA-ready instruments including Custom Instrument, Terminal Emulator, Digital I/O, Crosspoint Switch, Logic Analyzer, Frequency Generator, Frequency Counter, Field Dashboard for remote access
Soft Device JTAG Support	Live connection to soft devices such as virtual instruments and processors running inside an FPGA
Hard Device JTAG Support	Interactive monitoring of pin status for any JTAG device
IP Core Design Re-Use	Support for importing third-party FPGA IP cores, developing and reusing IP libraries
Import/Export	Supports import and/or export of designs and library data created in OrCAD, Allegro, PADS, DxDesigner, Cadstar, P-CAD, CircuitMaker, Protel and more
Schematic – Editing	All schematic document and library editing capabilities, netlist generation

Figure 4: Features of Altium Designerⁱⁱⁱ

4- Modeling

4-1 Data Flow Diagrams

4-1-1 Level 0

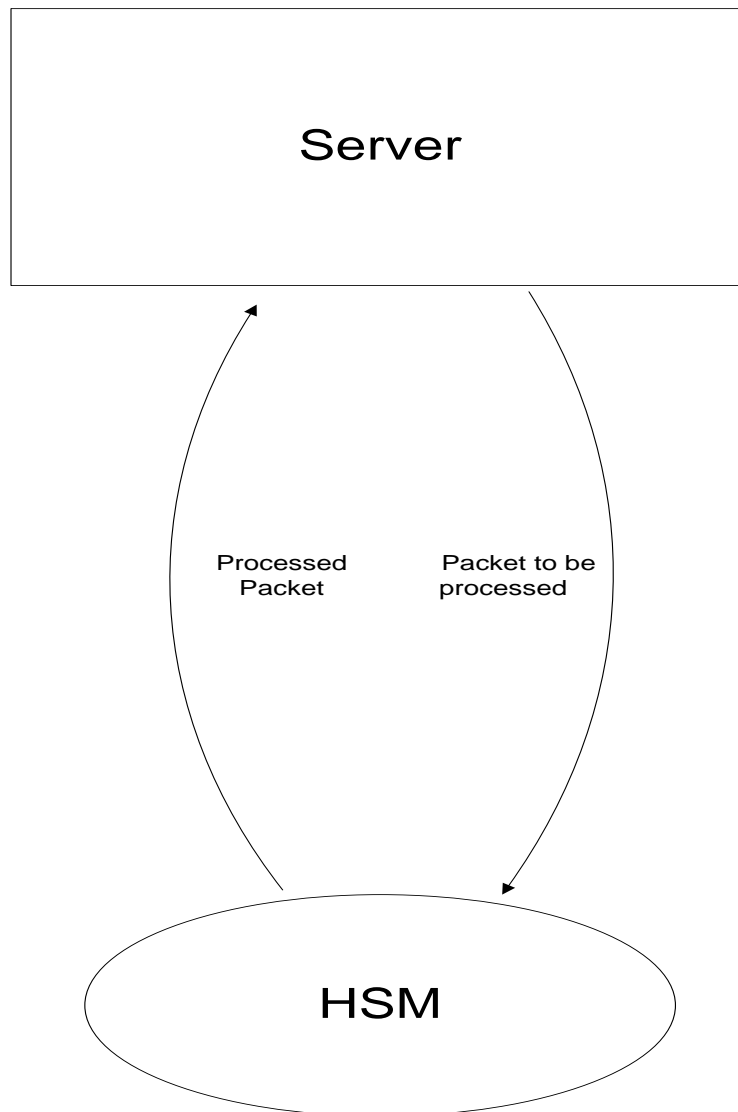


Figure 5: Context Level DFD

As it can be seen from the Figure 5, basically HSM will be in communication with server. Server will send a packet which cover request and data. HSM will process the packet and will return an answer. The answer will change according to request.



4-1-2 Level 1 : HSM

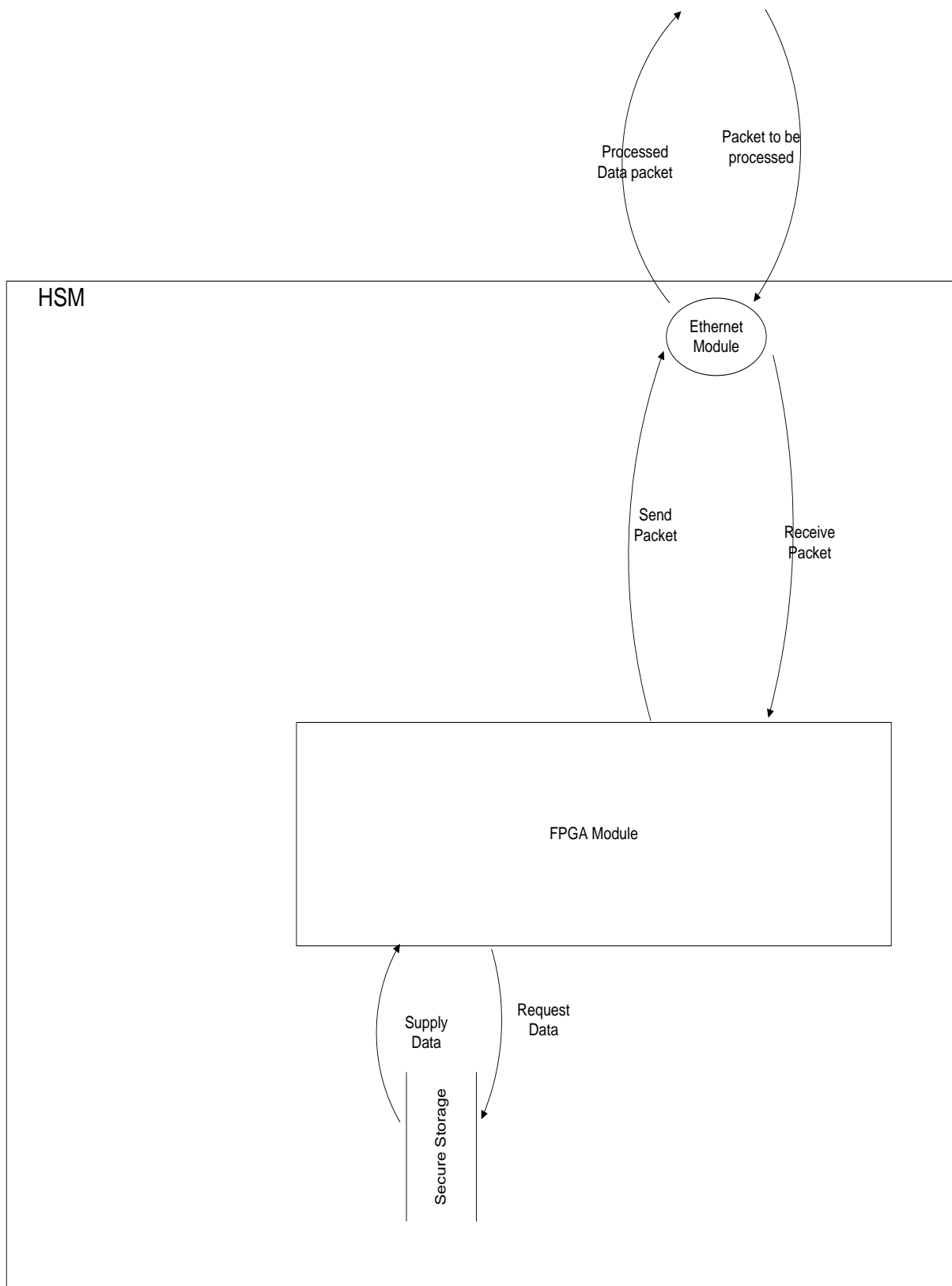


Figure 6: Level1 DFD : HSM



Packet coming from server comes to ethernet module by TCP/IP protocol at first. Then packet is transferred to the FPGA by wishbone interface. FPGA modules process packet and if there is a request about secure storage, FPGA module reaches to secure storage again with wishbone interface. After process, FPGA sends packet back to the ethernet module by wishbone interconnection. Finally, ethernet module sends answers back to server.

4-1-3 Level 2 : FPGA MODULE

FPGA module includes six processes :

- **Encryption** : It will have encryption function with strong cryptographic algorithm such AES,DES.
- **Decryption** : It will have decryption function.
- **Hashing** : It will have hash function with strong algorithm.
- **Key Generation** : It will have a function that generate random keys when key is needed.
- **Digital Signature** : It will have a function that produce digital signature or verify the signature that is produced by itself.
- **Microcontroller** : Microcontroller will manage all data flows in the system.

As it can be seen below, wishbone interface is used for all connection between all FPGA components and peripherals. All Cryptographic functions will be explained in class hierarchy in this report. Besides, wishbone interface will be explained in interface chapter in detail. The most important component of FPGA module, microcontroller, will briefly be explained in microcontroller/microprocessor chapter.



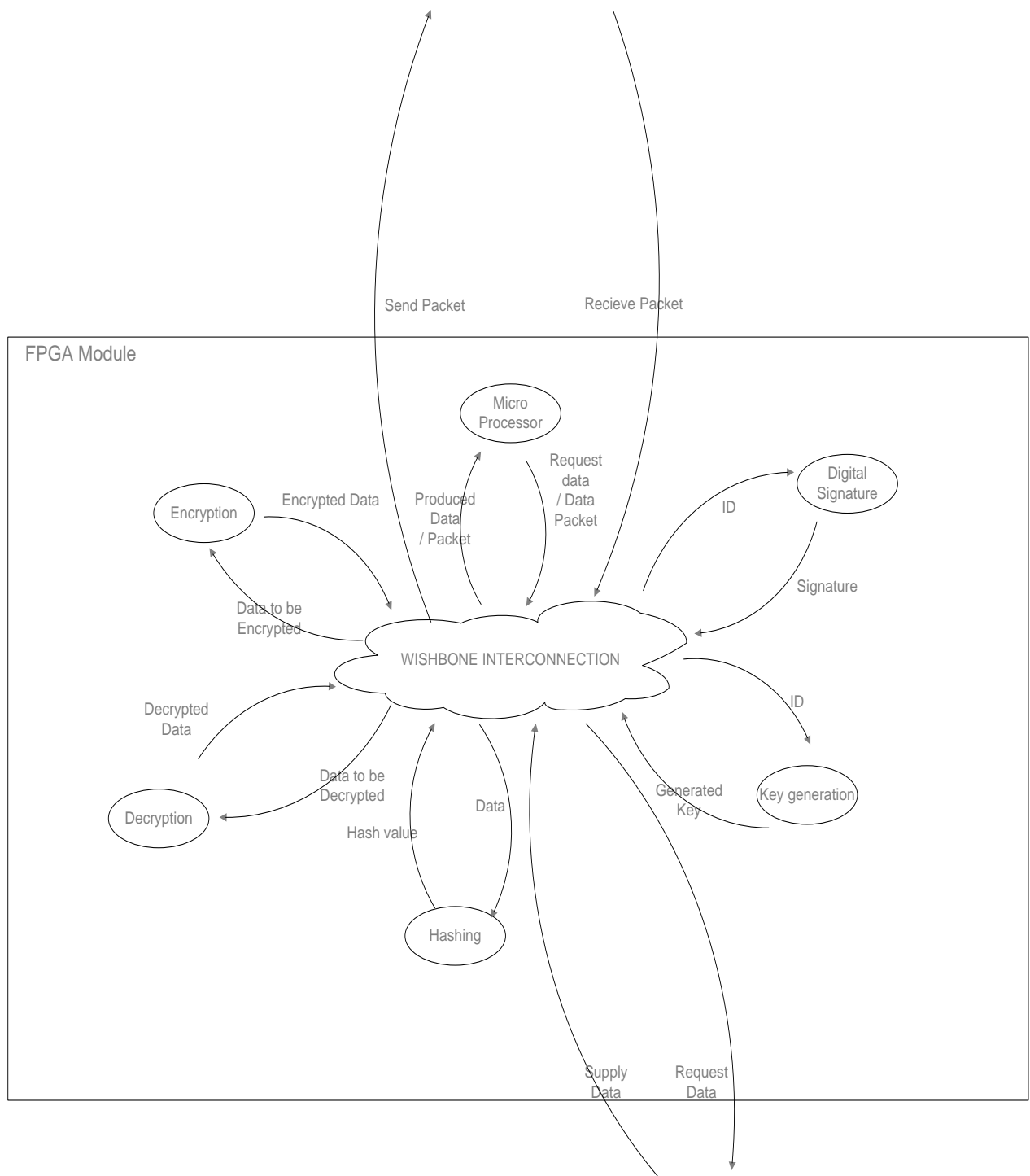


Figure 7: Level 2 DFD: FPGA Modules

4-2 Class Diagrams

4-2-1 Ethernet Module

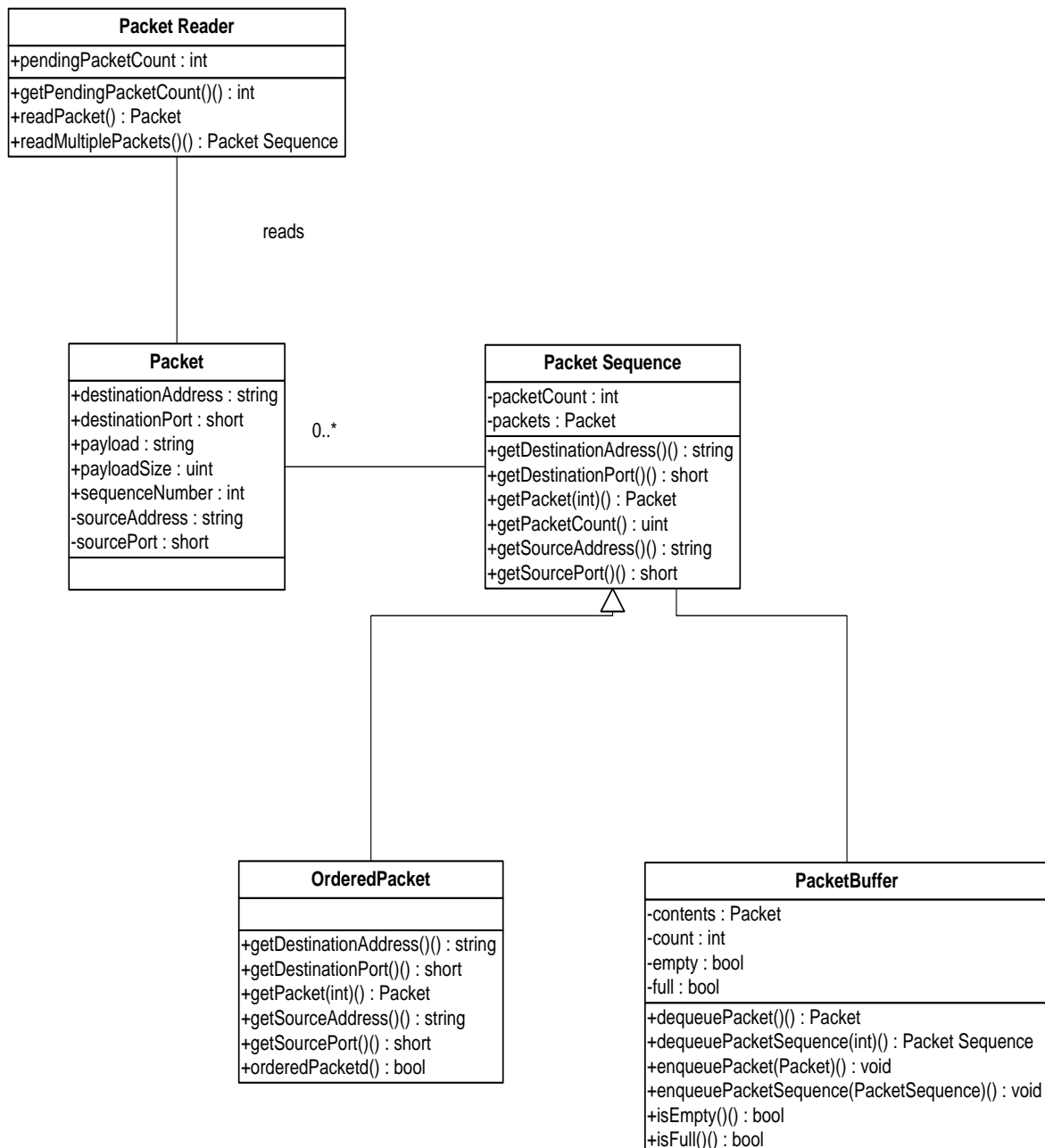


Figure 8: Ethernet Module

There are explanations of Ethernet module's classes below:

PacketReader is an abstract class that is responsible for reading packets from an input source. It includes methods for reading either a single packet or multiple packets at once. The `pendingPacketCount` member returns the unread packet count waiting at the input source.

Packet class is the data structure that is used to define a single packet. The member variables of this class are filled by the object that reads the packet from the input source.

PacketSequence is the collection class for packets that have the same source and destination addresses and same ports. It includes methods that provide random access to packets stored in the collection.

OrderedPackets extends the `PacketSequence` class to add functionality that orders the packets in the sequence based on their sequence number.

PacketBuffer implements a simple FIFO queue mechanism that is able to store a predefined number of packets in a queue data structure.



4-2-2 FPGA Module

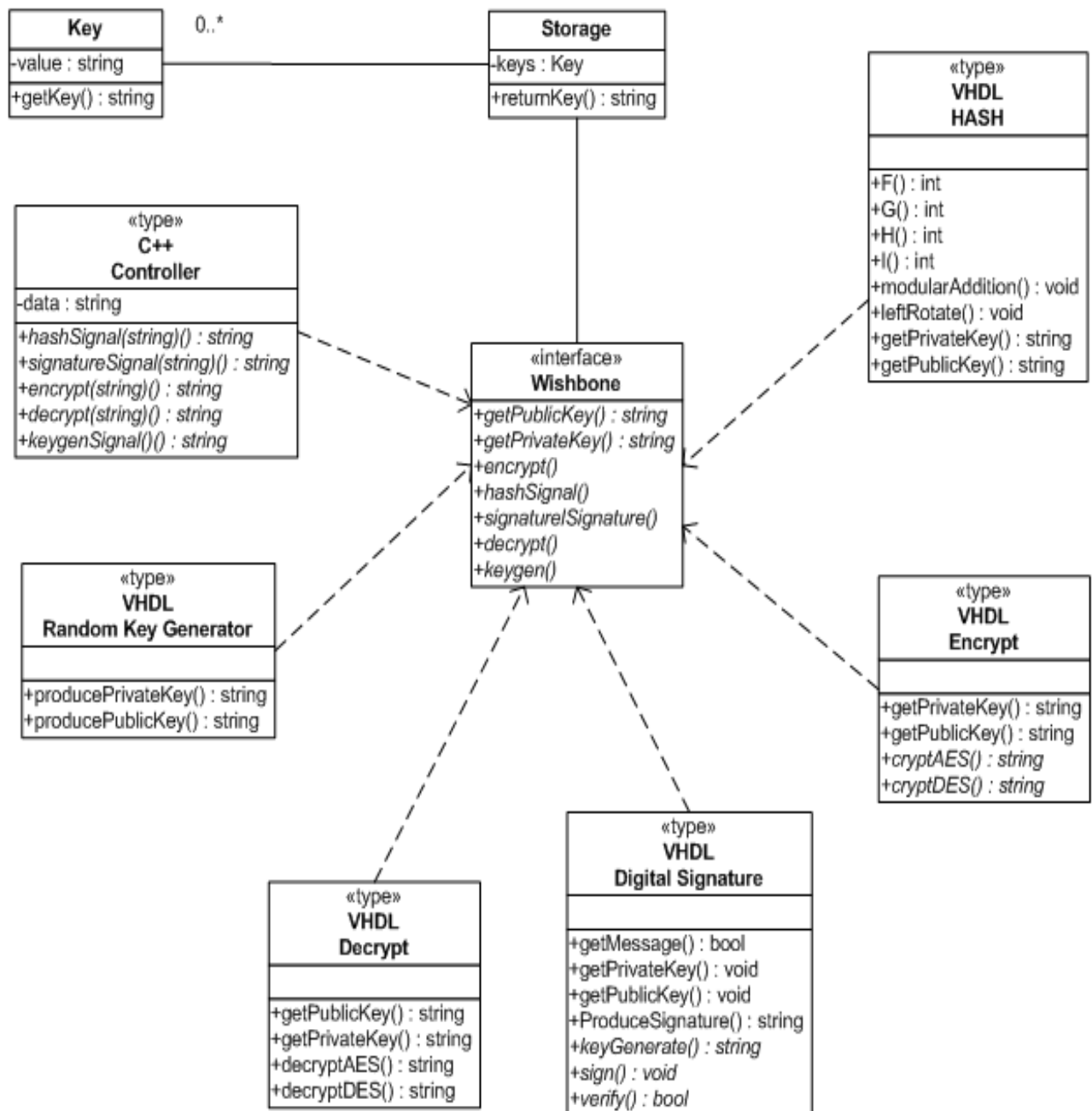


Figure 9: FPGA Module

Classes of FPGA modules are explained below in details:

Wishbone is the interface which is responsible for data transfers between Controller and other modules. Therefore every module is dependent to this Wishbone interface. `getPrivateKey` and `getPublicKey` functions are generic and polymorphic functions that can be redefined in its subclasses. Other methods are redefined in Controller class.

Key is a simple class that holds a single private key. This key is used in other classes. `getKey()` method returns the value of key that is stored in "key" field.

Storage is the class where exists an array of keys which are stored in the secure storage. Requested key values which are needed by other classes (i.e. AES, DES) are found on this class' "keys" field. When a request comes, requested key is going to be found and sent back from here.

Controller is the class where all request are going to be sent. It holds a data which is going to be processed by AES, DES, Digital Signature, Hash classes' methods.

- **hashSignal** method sends the data with a signal which is used to tell Wishbone interface that the request is a hash request.
- **signatureSignal** method sends the data with a signal which is used to tell Wishbone interface that the request is a digital signature request.
- **encrypt** method sends the data with a signal which is used to tell Wishbone interface that the request is an encryption request.
- **decrypt** method sends the data with a signal which is used to tell Wishbone interface that the request is a decryption request.
- **Keygen** method sends only a signal that requests random key generation.

HASH is the class where the hash requests are handled. The hash algorithm will be MD5 algorithm. The main MD5 algorithm^{iv} operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C and D. These are initialized to certain fixed constants. The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a non-linear function F, modular addition, and left rotation. There are four possible functions F; a different one is used in each round:



- $F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
- $G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
- $H(X, Y, Z) = X \oplus Y \oplus Z$
- $I(X, Y, Z) = Y \oplus (X \vee \neg Z)$

Encrypt class is the class where encryption requests are handled. There will be several encryption functions to encrypt data.

- **encryptAES** method uses Advanced Encryption Standard algorithm in order to encrypt data.

Advanced Encryption Standard Algorithm :

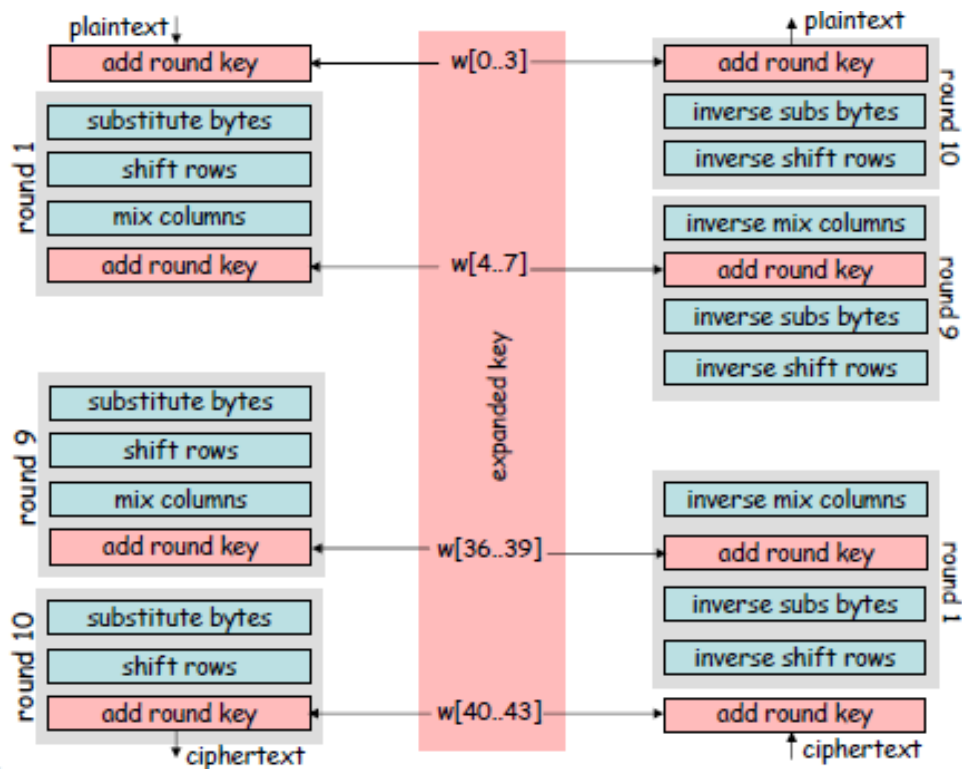
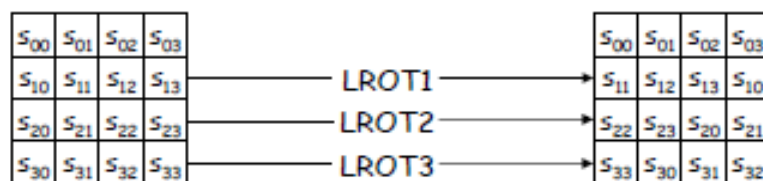
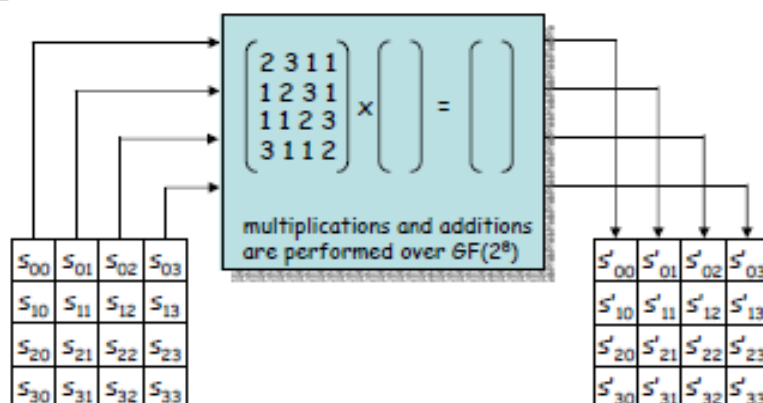
AES is based on a design principle known as a Substitution permutation network. It is fast in both software and hardware. Unlike its predecessor, DES, AES does not use a Feistel network.

AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits.

AES operates on a 4×4 array of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field.

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of cipher text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher text back into the original plaintext using the same encryption key.



Figure 10: General Structure of Encryption/Decryption^vshift rowmix columnFigure 11: Each round of AES^{vi}

- **encryptDES** method Data Encryption Standard algorithm in order to encrypt data.

Data Encryption Standard Algorithm:

There are 16 identical stages of processing, termed rounds. There is also an initial and final permutation, termed IP and FP, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have almost no cryptographic significance, but were apparently included in order to facilitate loading blocks in and out of mid-1970s hardware, as well as to make DES run slower in software.^{vii}

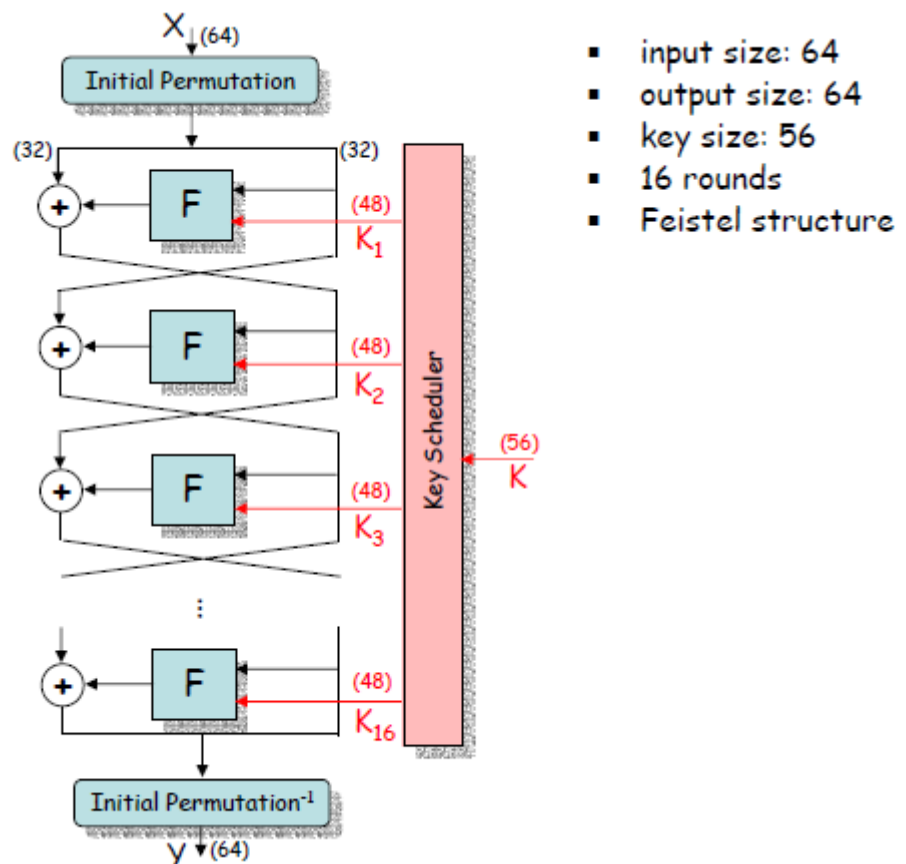


Figure 12: Working principal of DES^{viii}

Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistel scheme. The Feistel structure ensures that decryption and encryption are very similar processes — the only difference is that the sub keys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly

simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

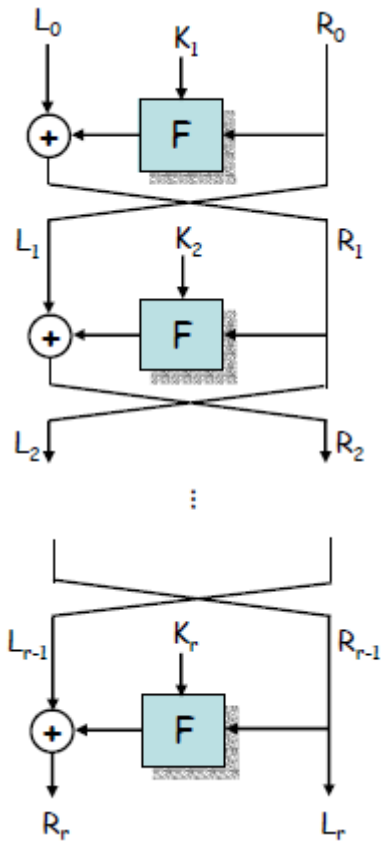


Figure 13: Each round of DES

Decrypt class is the class where decryption requests are handled. There will be several decryption functions to decrypt data.

- **decryptAES** uses the same AES algorithm with sub keys in reverse order to decrypt data.
- **decryptDES** uses the same DES algorithm with sub keys in reverse order to decrypt data.

Random Key Generator is the class that produces public and private keys.

producePrivateKey: Produces the private key and returns it.

producePublicKey: Produces the public key and returns it.



Digital Signature class will produce unique signatures according to the public key, private key and the message.

A digital signature scheme typically consists of three algorithms:

- A key generation algorithm that selects a private key uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding public key.
- A signing algorithm which, given a message and a private key, produces a signature.
- A signature verifying algorithm which given a message, public key and a signature, either accepts or rejects the message's claim to authenticity.

Two main properties are required. First, a signature generated from a fixed message and fixed private key should verify the authenticity of that message by using the corresponding public key. Secondly, it should be computationally infeasible to generate a valid signature for a party who does not possess the private key.



4-3 Activity Diagrams

Basic activity diagrams of HSM system are below. Activity diagrams enclose FPGA and Ethernet modules.

4-3-1 Ethernet Activity Diagram

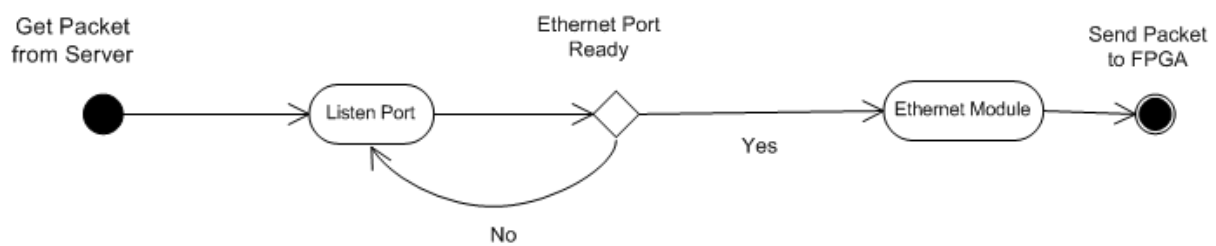


Figure 14: Ethernet Activity Diagram

4-3-1 FPGA Activity Diagram

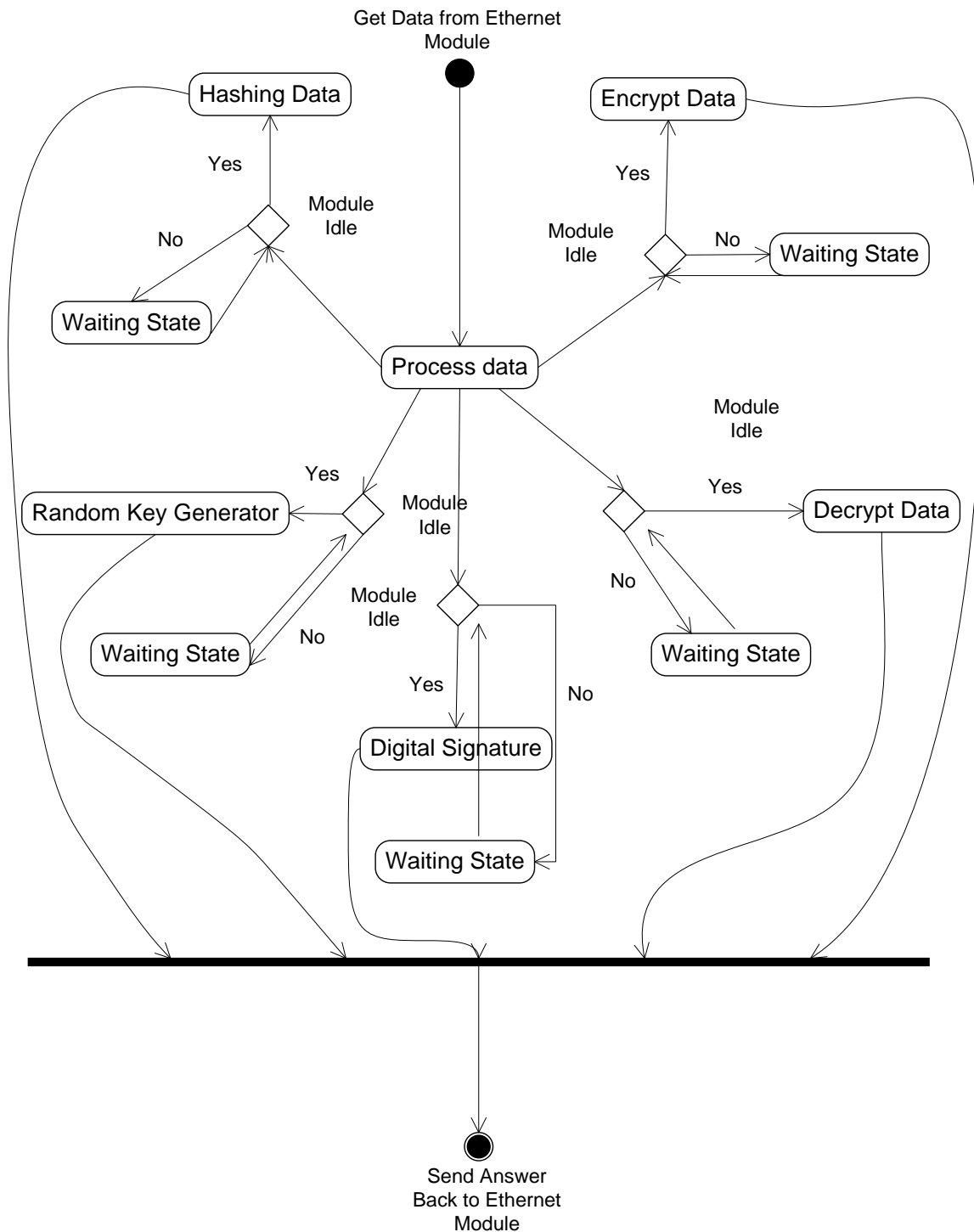


Figure 15: FPGA Activity Diagram

4-4 Sequence Diagrams

4-4-1- Packet Transportation

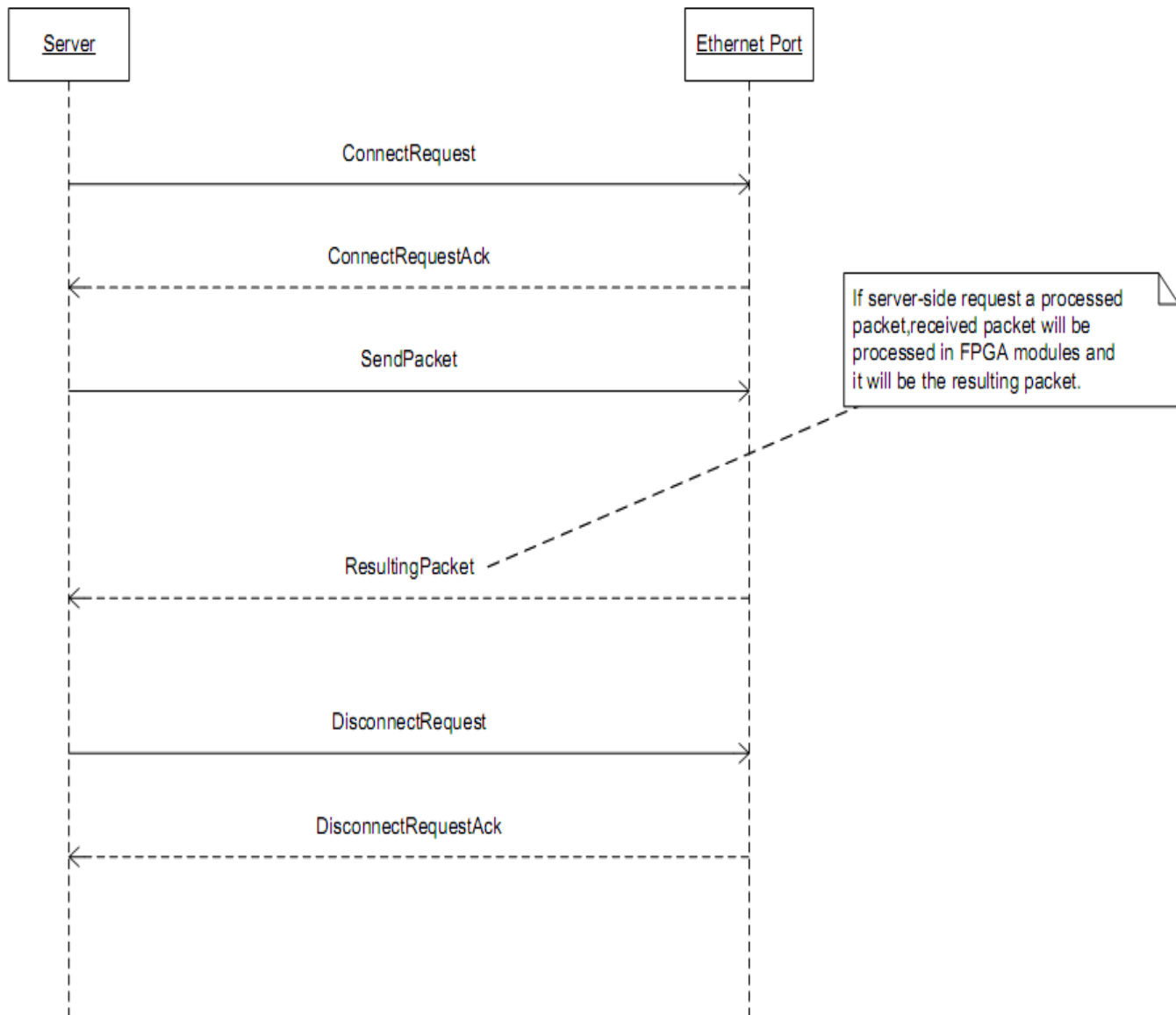


Figure 16: Packet Transportation

Figure 16 shows the packet transportation between server side and HSM side. The connection between server side and HSM side is accomplished via ethernet connection. The procedure for data transported is explained below:

- 1- Server side sends a request in order to establish a connection with the HSM.
- 2- HSM sends an acknowledgement signal as a response to a connection request.
- 3- Server side sends a packet to HSM which is to be processed.
- 4- If server side request a processed packet, this data in the packet will be processed and the result is returned as “resulting packet”.
- 5- After server side finished its job with the HSM, it will send a disconnection request.
- 6- In response to disconnection request, HSM will send a disconnection acknowledgement signal.

4-4-2 Microcontroller – Crypto Module Interaction

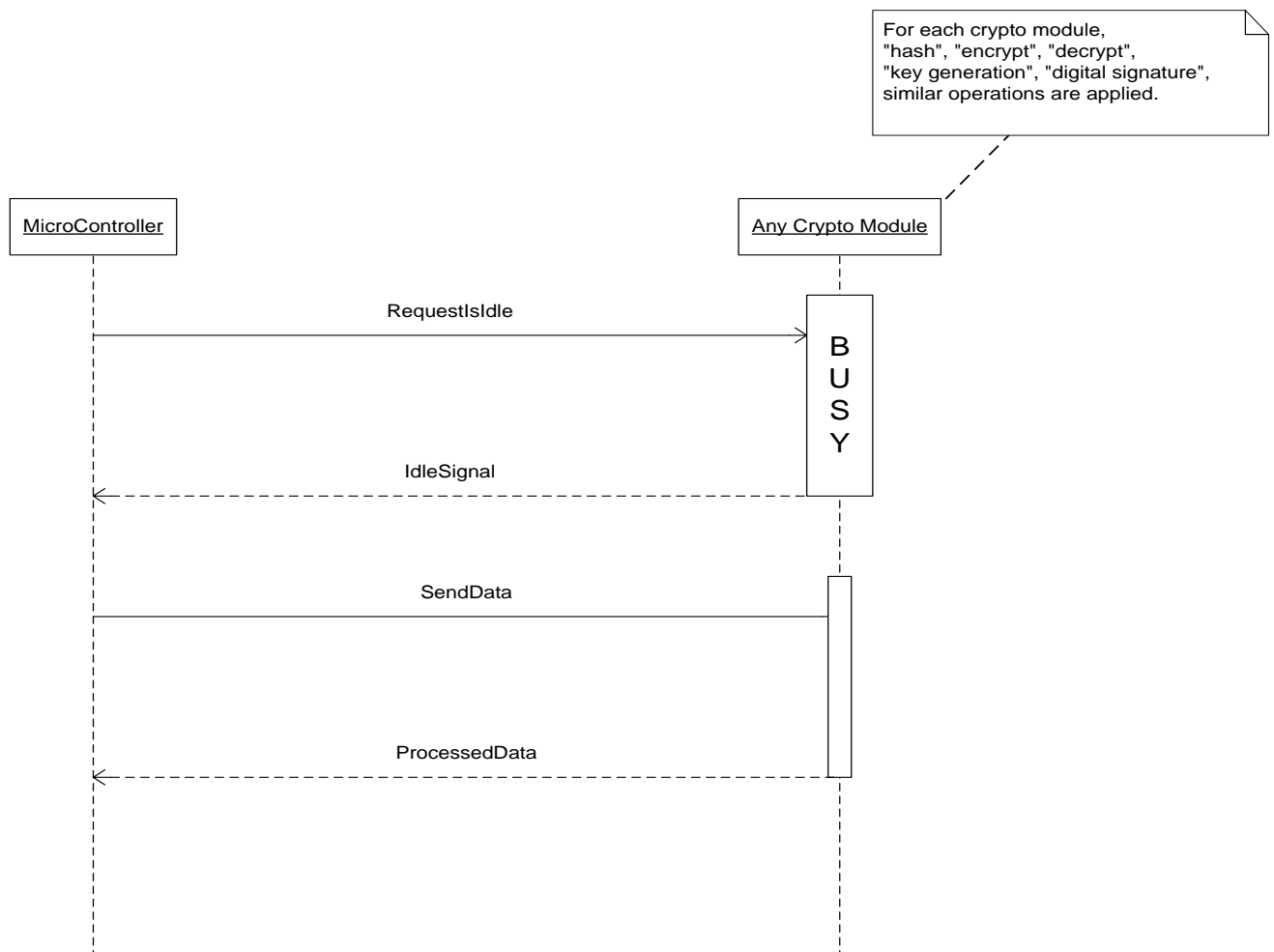


Figure 17: Interaction between Microcontroller and Crypto Module

The sequence diagram in figure 17 shows the interaction between microcontroller unit and a cryptographic module basically. Since the logic behind all of the modules are similar, it is enough to explain all of them in this sequence diagram.

Firstly, microcontroller requests information from the crypto module to learn that whether the module is idle or not. If the module is busy, after the module finishes its job it will send idle signal to the microcontroller. Then, microcontroller will send the data to be processed to related crypto module. After the data is processed it will be send back to microcontroller from the crypto module.

5- Microprocessor/Microcontroller

Microprocessor/Microcontroller is used in order to accomplish data management, time scheduling and easily communicate with peripheral devices. There are number of benefits to be gained from using soft processors on reconfigurable hardware.

The following sections are some of the more significant of these benefits.

- coprocessors Field reconfigurable hardware
- Faster time to market
- Improving and extending product life-cycles
- Creating application-specific
- Implementing multiple processors within a single device
- Lowering system cost
- Avoiding processor obsolescence

In this project, Altium TSK3000A is going to be used as the processor. TSK 3000A is a 32-bit, Wishbone-compatible, RISC processor. Instructions are 32-bits and execute in single clock cycle.



5-1- TSK3000A

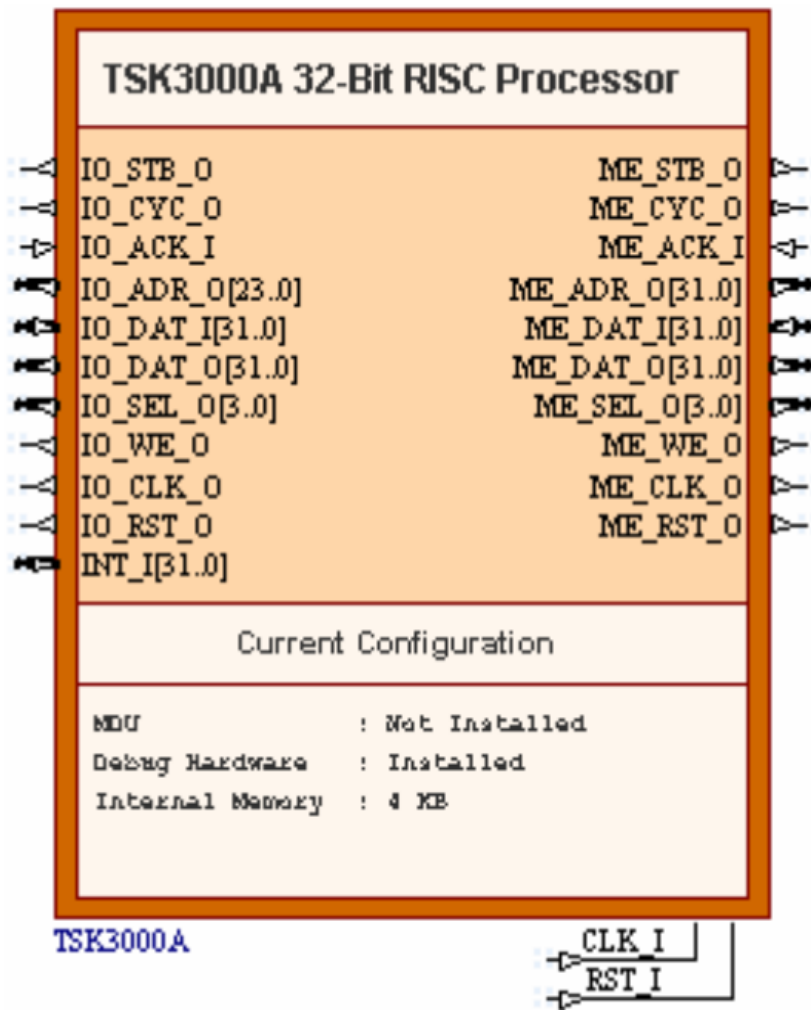


Figure 18: TSK3000A^{ix}

5-1-1 Pin Description

The pin out of the TSK3000A has not been fixed to any specific device I/O - allowing flexibility with user application. The TSK3000A contains only unidirectional pins (inputs or outputs).



Name	Type	Polarity/Bus size	Description
Control Signals			
CLK_I	I	Rise	External (system) clock
RST_I	I	High	External (system) reset
Interrupt Signals			
INT_I	I	32	Interrupt inputs. Each input can be configured to operate as level-sensitive or edge-triggered by clearing or setting the corresponding bit in the IMode register respectively. Interrupts can be configured in one of two modes – Standard or Vectored – determined by the VIE bit of the Status register (Status.9)
Wishbone External Memory Interface Signals			
ME_STB_O	O	High	Strobe signal. When asserted, indicates the start of a valid Wishbone data transfer cycle
ME_CYC_O	O	High	Cycle signal. When asserted, indicates the start of a valid Wishbone bus cycle. This signal remains asserted until the end of the bus cycle, where such a cycle can include multiple data transfers
ME_ACK_I	I	High	Standard Wishbone device acknowledgement signal. When this signal goes High, an external Wishbone slave memory device has finished execution of the requested action and the current



			bus cycle is terminated
ME_ADR_O	O	32	Standard Wishbone address bus, used to select an address in a connected Wishbone slave memory device for writing to/reading from
ME_DAT_I	I	32	Data received from an external Wishbone slave memory device
ME_DAT_O	O	32	Data to be sent to an external Wishbone slave memory device
ME_SEL_O	O	4	Select output, used to determine where data is placed on the ME_DAT_O line during a Write cycle and from where on the ME_DAT_I line data is accessed during a Read cycle. Each of the data ports is 32-bits wide with 8-bit granularity, meaning data transfers can be 8-, 16- or 32-bit. The four select bits allow targeting of each of the four active bytes of a port, with bit 0 corresponding to the low byte (7..0) and bit 3 corresponding to the high byte (31..24)
ME_WE_O	O	Level	Write enable signal. Used to indicate whether the current local bus cycle is a Read or Write cycle. 0 = Read 1=Write
ME_CLK_O	O	Rise	External (system) clock signal (identical to



			CLK_I), made available for connecting to the CLK_I input of a slave memory device. Though not part of the standard Wishbone interface, this signal is provided for convenience when wiring your design
ME_RST_O	O	High	Reset signal made available for connection to the input of a slave memory device. This signal goes High when an external reset is issued to the processor on its RST_I pin. When this signal goes Low, the reset cycle has completed and the processor is active again. Though not part of the standard Wishbone interface, this signal is provided for convenience when wiring your design
Wishbone Peripheral I/O Interface Signals			
IO_STB_O	O	High	Strobe signal. When asserted, indicates the start of a valid Wishbone data transfer cycle
IO_CYC_O	O	High	Cycle signal. When asserted, indicates the start of a valid Wishbone bus cycle. This signal remains asserted until the end of the bus cycle, where such a cycle can include multiple data transfers
IO_ACK_I	I	High	Standard Wishbone device acknowledgement signal. When this signal goes High, an external Wishbone slave peripheral device has finished execution of the requested action and the current bus cycle is terminated
IO_ADR_O	O	24	Standard Wishbone address bus, used to select an internal register of a connected Wishbone slave peripheral device for writing to/reading from
IO_DAT_I	I	32	Data received from an external Wishbone



			slave peripheral device
IO_DAT_O	O	32	Data to be sent to an external Wishbone slave peripheral device
IO_SEL_O	O	4	Select output, used to determine where data is placed on the IO_DAT_O line during a Write cycle and from where on the IO_DAT_I line data is accessed during a Read cycle. Each of the data ports is 32-bits wide with 8-bit granularity, meaning data transfers can be 8-, 16- or 32-bit. The four select bits allow targeting of each of the four active bytes of a port, with bit 0 corresponding to the low byte (7..0) and bit 3 corresponding to the high byte (31..24)
IO_WE_O	O	Level	Write enable signal. Used to indicate whether the current local bus cycle is a Read or Write cycle. 0 = Read 1 = Write
IO_CLK_O	O	Rise	External (system) clock signal (identical to CLK_I), made available for connecting to the CLK_I input of a slave peripheral device. Though not part of the standard Wishbone interface, this signal is provided for convenience when wiring your design
IO_RST_O	O	High	Reset signal made available for connection to the input of a slave peripheral device. This signal goes High when an external reset is issued to the processor on its RST_I pin. When this signal goes Low, the reset cycle has completed and the processor is active again. Though not part of the standard Wishbone interface, this signal is provided for convenience when wiring your design

Table 2: TSK3000A Pins Description^x

6- INTERFACE

According to HSM's working principal, two interface will be used. Contrary to expectations, these interfaces will not be graphical user interfaces. These interfaces are called physical interfaces and they will be used for connecting hardware component to each other and making connection available between them. One of the interfaces is internal interface which connects HSM's internal components with each other and the other one is external interface which is used for connecting HSM with outside world.

6-1 External Interface

External interface is between HSM and "Server". This physical connection will be provided by ethernet interface. Altium Nanoboard provides a fast Ethernet connection, supporting 10Base-T and 100Base-TX, for operational speeds of up to 10Mbps and 100Mbps respectively. Before explaining ethernet interface, some information about ethernet port will be given in order to understand easily how ethernet connection works.

6-1-1 Ethernet Port

An 8P8C ('RJ45') modular connector is used to provide the Ethernet port (a FC0901238, from Konvee). The connector has integrated 10/100Base-T Ethernet Isolation Transformers and two indication LEDs. The latter – one yellow and one green – have been wired to reflect the Link status and 100Mbps activity, respectively. Connection to the external network is made using standard Category 5 unshielded twisted pair (UTP) network cable.

"Providing the interface between an Ethernet Media Access Controller in an FPGA design and the external network, is an RTL8201CL 10/100M Fast Ethernet PHYceiver device."

xi



6-1-2 Ethernet Interface

Table 2 summarizes the available design interface component that can be placed from the FPGA Nanoboard 3000 Port-Plugin.IntLib to access the Ethernet interface. Port-Plugin.IntLib is coming ready with “Altium Designer” and it will make job easier for the project. Ethernet interface is going to be used for transferring network packets. The network packet is explained in the next chapter.

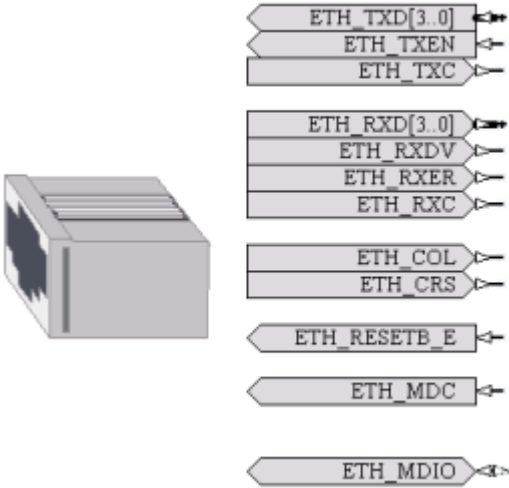
Component Symbol	Component Name	Description
 <p>The component symbol for the Ethernet PHY interface. It features a network port icon on the left. To its right, there are several signal pins: ETH_TXD[3..0] (bidirectional), ETH_TXEN (output), ETH_TXC (output), ETH_RXD[3..0] (bidirectional), ETH_RXDV (input), ETH_RXER (input), ETH_RXC (input), ETH_COL (input), ETH_CRS (input), ETH_RESETB_E (input), ETH_MDC (input), and ETH_MDIO (bidirectional).</p>	ETH_PHY	Place this component to access the RTL8201CL PHYceiver device and subsequent Ethernet port.

Table 2: Ethernet Interface port-plugin component.^{xii}

6-1-2-1 Network Packet

Network packets are like Russian dolls. An IP-packet resides within an Ethernet-packet. A TCP-packet resides within an IP-packet. A HTTP-packet resides within a TCP-packet.

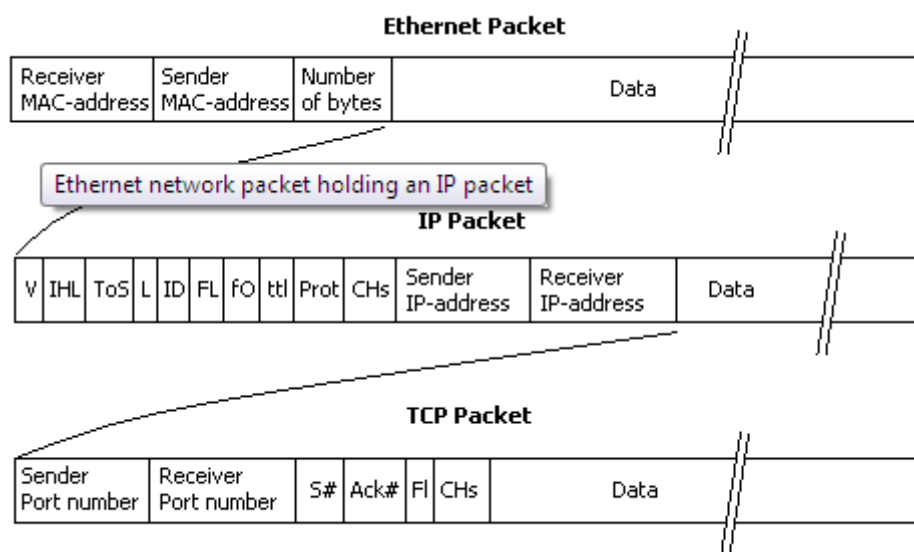


Figure 19: Ethernet network packet holding an IP packet

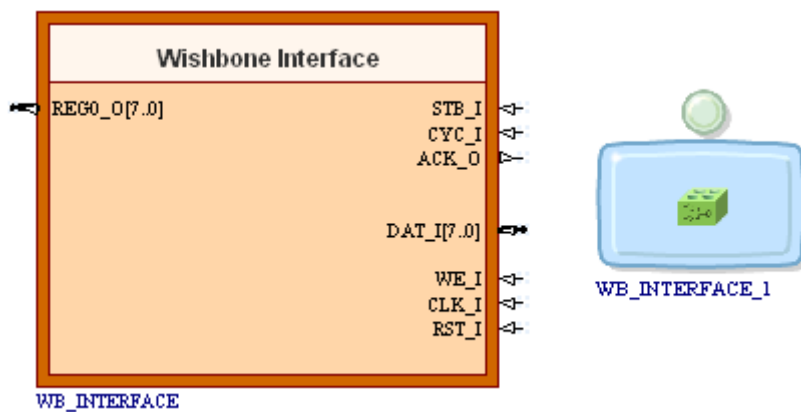
The data part of an Ethernet packet can hold up to 1500 bytes. All requests that are used in HSM system are introduced in first four bits of data part. MAC-addresses (48bits) are 6 bytes wide each and the Number of Bytes field is 2 byte wide. That gives the maximum size of an Ethernet frame to be 1514 bytes.

When introducing the IP-protocol on Ethernet the Number of Bytes field is used to mark that the Ethernet frame holds an IP-packet by the number of 0x0800. By using a number that is greater than the maximum length an Ethernet frame can hold indicates that the Ethernet frame holds another protocol frame in the data part. The Ethernet standard says that if the Number of Bytes field is greater than 0x0600, the Ethernet frame holds another protocol.

6-2- Internal Interface

Wishbone interface will be used for making connection between HSM's internal components with each other. "Altium Designer" will also provides wishbone interconnection for designer. There are various wishbone interconnections for different aims. Basically custom wishbone interface will be explained. Detailed information about wishbone interface will be given in Detailed Design Report.

6-2-1- Wishbone Interface



The Wishbone Interface component (WB_INTERFACE) enables designer to build a custom Wishbone peripheral in a design, extending your 32-bit FPGA systems through the creation of custom FPGA logic.

The Wishbone Interface component has a fully configurable interface for transferring data to/from connected logic, and a Wishbone bus to interface with a host processor. The individual units of this configurable interface are referred to as 'items'. The interface can include a combination of one or more of the following items:

- Internal Registers – which allow values to be read from, and/or written to, connected logic.
- Command Sets – which allow operations to be enabled on connected logic.
- External Address Ranges – which allow access to blocks of addresses on connected

In addition to making the task of building Wishbone peripherals far easier, the Wishbone Interface component also provides the ability to generate C code based on the items specified in the interface simplifying interaction with the component from the embedded code running on the host processor.^{xiii}

7- Language Specifications

7-1- Embedded C /C++

C and C++ are general programming languages and can be used for implementing software system and portable application software. Programmers around the world embrace C and C++ because it gives maximum control and efficiency to the programmer.

Microprocessor (TSK 3000A) and wishbone connections will be implemented by using these languages.

Altium Designer which will be used for developing HSM, provides C-to-Hardware Compilation (CHC) technology. The Compiler takes C source code as input and produces FPGA logic as output.



Figure 20: High-level illustration of C-to-Hardware compilation in Altium Designer

7-2- VHDL

VHDL is a programming language that has been designed and optimized for describing the behavior of digital systems. VHDL has many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and

custom chips. Features of VHDL allow electrical aspects of circuit behavior (such as rise and fall times of signals, delays through gates, and functional operation) to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits (using schematics, block diagrams or system-level VHDL descriptions) for the purpose of simulation. VHDL will be used in order to design FPGA. Cryptographic modules such as Encryption and Decryption module will be designed by using VHDL programming Language.^{xiv}

8- Testing and Debugging

8-1- Testing

The aim of this part is to detect errors and bugs of the hardware security module. A good testing strategy hopefully will make the project work fully. There are the testing strategies which are going to be used in testing part.

8-1-1- Unit Testing

The aim of this kind of testing is to verify whether the smallest testable pieces of the application are working properly or not. In this phase of testing each unit will be tested separately before integrating it to the whole system. ,Since finding the possible error in the integrated project is crucial, this stage is relatively important. Also this stage ensures that integration test may only have integration errors and hopefully has no unit dependent errors appear.

8-1-2- Integration Testing

This testing stage is a little extended form of unit testing. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using Black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their



input interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

8-1-3- System Testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

8-2- Debugging

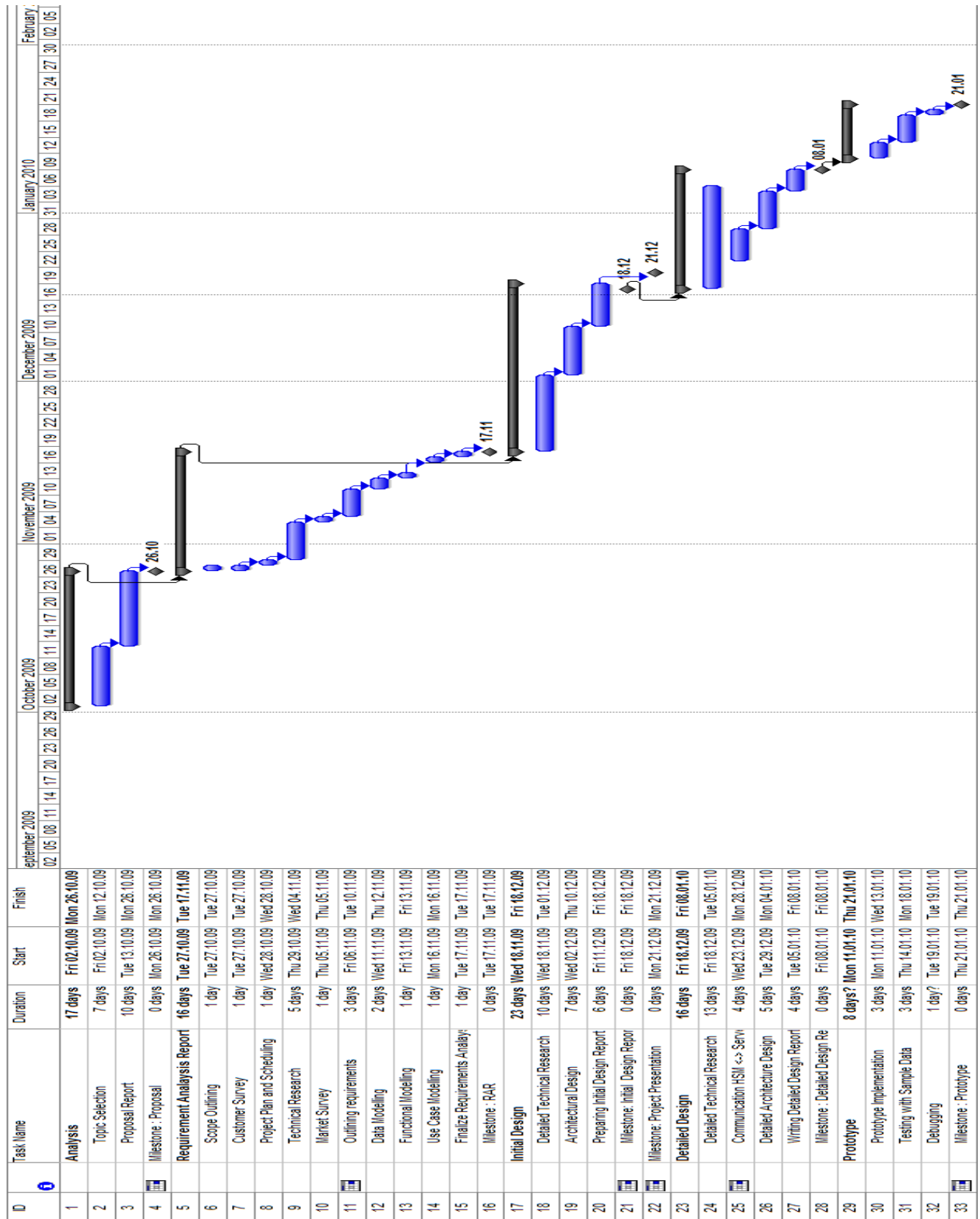
Debugging is the act of testing designer hardware design and any embedded software (running on 'soft' processors therein), to obtain the desired (correct) performance and functionality. Debugging is an important element of the overall design strategy, and effective debugging can save a lot of time and money when it comes time to deploy your end design in the field.

Since "Altium Designer" will be used for this project, Altium Designer's debugging environment will be used. In Altium Designer, debugging of hardware is provided courtesy of 'virtual' instruments – components which are 'wired' into the actual FPGA design but which, on programming the physical device, offer software-based controls for interrogation and control of nodes within the design. Imagine being able to walk around inside the physical FPGA device, armed with your favorite test instruments, and programmer will have some idea of what these instruments can offer as part of a 'live' debugging environment.

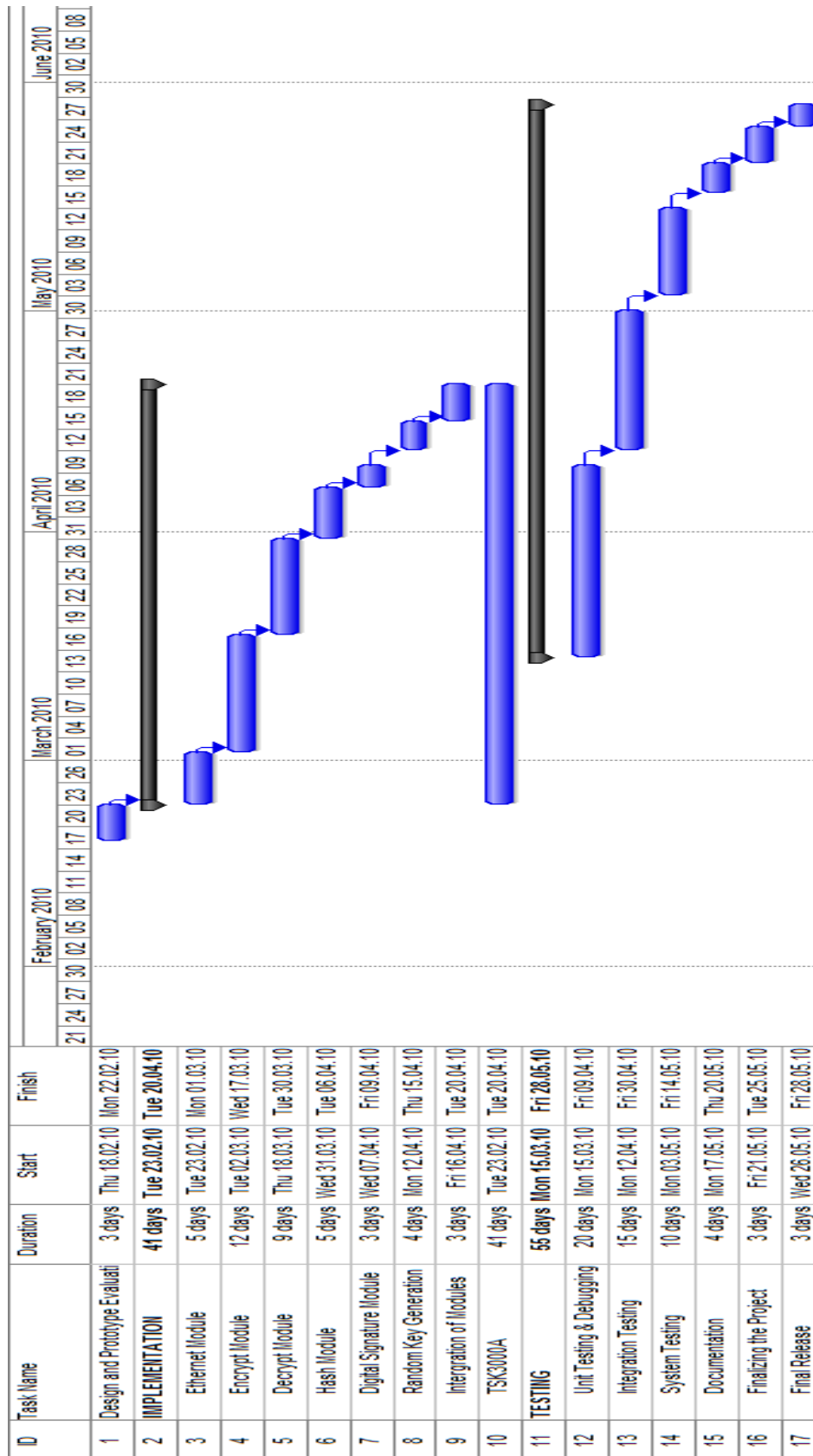


9- Gantt Chart

9-1- Term 1 Gantt Chart



9-2- Term 2 Gantt Chart



10- References

<http://wiki.altium.com/display/ADOH/Key+Features+of+the+NanoBoard+3000> ⁱ

<http://wiki.altium.com/display/ADOH/Functional+Overview+of+the+NanoBoard+3000> ⁱⁱ

<http://www.altium.com/files/pdfs/Altium-Designer-Feature-Set-Summary.pdf> ⁱⁱⁱ

<http://www.ietf.org/rfc/rfc1321.txt> ^{iv}

<http://www.crysys.hu/courses/adatbiztonsag/DES-AES.pdf> ^v

<http://www.crysys.hu/courses/adatbiztonsag/DES-AES.pdf> ^{vi}

<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf> ^{vii}

http://www.absoluteastronomy.com/topics/Feistel_cipher ^{viii}

<http://www.altium.com/files/learningguides/.%5CCR0121%20TSK3000A%2032%20bit%20RISC%20Processor.pdf> ^{ix}

<https://altium.onconfluence.com/display/ADOH/TSK3000A+Data+Organization> ^x

<http://wiki.altium.com/display/ADOH/Ethernet+Protocol> ^{xi}

<http://wiki.altium.com/display/ADOH/Ethernet+Protocol> ^{xii}

http://wiki.altium.com/display/ADOH/WB_INTERCON+-+Configurable+Wishbone+Interconnect ^{xiii}

<http://www.altium.com/files/AltiumDesigner6/LearningGuides/TR0114%20VHDL%20Language%20Reference.pdf> ^{xiv}

