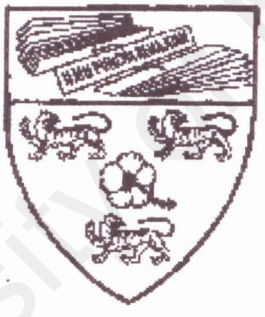
Keyboard – Smart Card Reader Controller using VHDL

By

MOHD ZUWAIRI B. LANI
WEK 98169



A final year project as a partial fulfillment of the requirement for **Degree of *Bachelor of Computer Science***

Faculty of Computer Science and Information Technology

University Malaya
Kuala Lumpur

Session 2000/2001

Abstract

As the time fly by the technology of computers has improves and much security is needed in every transaction of data that being done. In this project, designing a microcontroller for keyboard and smart card reader terminal integration is being done. At first stage, the architecture of the controller will analyze and studied. This will need some practice of knowledge in computer organization and architecture, as well as microprocessors architecture designs. Then the architecture is design using hardware prototyping. This is by using the VHSIC Hardware Description Language (**VHDL**). The codes done using the language then being simulate to get the result of a proper working design. The reports will consists of analysis, designs, simulation results and the final outcome of the project which is the microcontroller itself. It will also cover some survey on books, websites and other sources that related to the project. In a way this project is one of many steps in making a better secure protection in the network for the coming future.

Acknowledgement

First and foremost 'alhamdulillah' to Allah, my supervisor Encik Noor Zaily for the guidance and advices given in making this project a success. I also would like to thank you my moderator, Dr Rosli Salleh for his evaluation on the project. In the way, morale support from family is deeply appreciated, Aizan Fareena (for the patience), Mohd Azfeezal (for the sources given), housemates (for laughs and jokes) and colleague (for the time spent). In the end, all of this cannot be real if not all the support that I get in making this project a success.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.0 Introduction

The needs for security in the era of information technology nowadays have increased for the past year. Although the problem already started since the beginning of Internet, many ways of protecting the data from being exposed or manipulated has been implemented but still it just not enough.

A few years ago, a technology called integrated-circuit cards is introduced. This technology offers more security protection than other technology such as passwords.

Smart cards greatly improve the convenience and security of any transaction. Since the data stored are tamper-proof, this can be the vital component of data exchange through networks.

In the ways of making the convenience of security during transaction or getting access, the idea of making a smart card reader that integrated with keyboards came to mind. This is because in any transaction a keyboard is needed and to insure the data exchange is safe a smart card can be a solution.

Using the normal keyboard port or PS/2 or Universal Serial Bus (USB), the reader will be integrated with the keyboard by a micro controller with handle the tasks of a keyboard and the smart card reader as well. The task of designing the micro

controller is done on a Very High Speed Integrated Circuit Hardware Description Language (VHDL) simulator.

1.1 Objectives

The main goals are to improve the design of the keyboard-smart card reader integration that already existed in the market. These will includes the task of to find out how the keyboard can be integrated with the reader, and then design the controller that can do the integration. Using a **VHSIC Hardware Description Language (VHDL)** simulator the design of the controller will be done.

The controller will work as processor for the keyboard as well as the reader also. It can interface with PC as an ordinary keyboard and a smart card reader in a single communication line such as through the PS/2 port or the usual keyboard port.

As the goals of improving the hardware that already exist, the improvement will target to these areas:

- Makes it low cost
- Simplicity of design
- Ease of use
- Compatibility
- Provide better security

1.2 Scopes and Limitation

The researches will mainly concentrate on the design of a micro controller that will integrate these two devices. The micro controller that integrates the reader and keyboard should handle both device functions. Beside that it has to support several standard and protocol such as T=0 and T=1 and ISO7816-1. Standard such as ISO7816-1 is an international standard for integrated-circuit cards (commonly known as smart cards) that use electrical contacts.

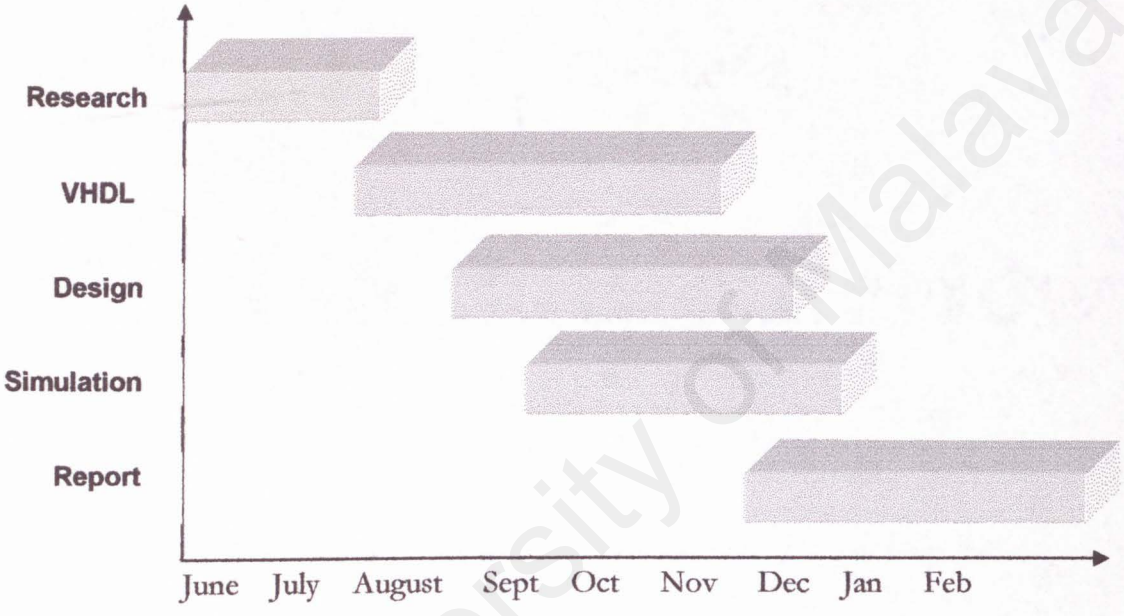
The keyboard also needs to interface with PC using single PS/2 interface. With that the design should also lowered the cost of the existence keyboard.

Although some limitation can be expected in designing the controller, because some standard need to be familiarize so it can be implemented. Standard such as ISO7816 only can be obtained from American National Standards Institute (ANSI). Obtaining these document can be a problems and can limit the probability that the controller can support these standard once it implemented.

Beside that the needs for development tools are limited. Tools such as VHDL simulator and test or demo board to download the design are hard to obtain. Hence, due to these limitations the some of the feature maybe can't be implemented fully.

1.3 Work plans and Scheduling

The works done were based on the planning and timetable that I've created. This important because each work must be done in time and careful planning will make it possible.



Chapter 2

Literature Review

2.0 Literature Review

2.1 Smart Card

2.1.1 Introduction

A smart card is a type of card that embedded with a computer chip that stores and transacts data between users. This data may associated with either value, information or both and stored and processed within the card's chip, either memory or microprocessor. The data is transacted via a reader that is a part of a computing system. Nowadays, the smart card-enhanced systems are in use in several key applications such as healthcare, banking, entertainment and transportation. These applications benefit the added features and security that smart card provide.

The first smart card was actually produced by CII-Honeywell-Bull with Motorola in 1979 but didn't become popular until the European countries discover the benefit this technology in 1985. French implement the technology for public phones pre payment card. Germany implemented it for the country health care plan.

As time passed by, smart card gained popularity more than the rather conventional magnetic stripe card. The reason is that smart cards can store about 80 times storage capacity than conventional magnetic stripe card. They provide advanced hardware / software security features and multiple function cards.

2.1.2 What Is Smart Card?

Smart card has a variety of type, depending primarily on the type of integrated circuit chip embedded in the plastic card and the physical form of the connection mechanism between the card and the reader. Three such variants are:-

- Cards with surface contacts leading to a memory-only integrated circuit chip
- Cards with an electromagnetic connection to a microprocessor-integrated circuit chip
- Cards with surface contacts leading to a microprocessor-integrated circuit chip.

But the very earliest smart cards were memory cards containing an integrated circuit chip comprised of only nonvolatile memory and the necessary circuitry to read and write that memory. Nowadays, such still exists, beside inexpensive but provide a modest security protection for a variety applications.

A *memory card*, as its name implies, is a card that contains an embedded integrated circuit chip providing nonvolatile memory for storing information in a permanent or semi-permanent fashion. The circuitry of the smart card exposes, through a standard electrical connector, the control lines for addressing selected memory locations as well as for reading and writing those memory locations through the electrical connectors on the face of the card. There is no on-board processor to support a high-level communications protocol between the reader and the card. Rather, memory cards use a

synchronous communication mechanism between the reader and the card. Essentially, the communication channel is always under the direct control of the reader side. The card circuitry responds in a direct (synchronous) way to the very low-level commands issued by the reader for addressing memory locations and for reading from or writing to the selected locations. In some recent memory cards, security enhancements have been incorporated through the provision of memory addressing circuitry within the chip that requires a shared secret between the terminal (which is writing to the card chip) and the chip itself. These are often called *logic cards.*

A *contact-less* card has an integrated circuit chip embedded within the card; however, it makes use of an electromagnetic signal to facilitate communication between the card and the reader. With these cards, the power necessary to run the chip on the card is transmitted at microwave frequencies from the reader into the card. The separation allowed between the reader and the card is quite small—on the order of a few millimeters. However, these cards offer a greater ease of use than cards that must be inserted into a reader. This ease of use can be mitigated by other factors.

With the current technology, the transfer data rate between the reader and the contact less card is restricted by the power level that it achieve in the card. Because of the card does not have internal power supply; the reader must derived power to the card so it can run on-board processor from the signal the being transmitted.

There not much different between the contact-based and contact- less architecture beside the physical mechanism the used to transfer between reader and card. The microprocessor used integrates both memory and central processing unit into single integrated circuit chip.

2.1.3 How it works?

First of all a smart card has to be embedded with the circuit that consists of central processing unit, memory, and I/O electronics. With the integrated circuit, it can conceal all the interconnection and will give a better security of what going on in or what's stored in the computer.

Communicating With Smart Card

Intelligent smart card are depending on how their communicating with the reader or terminal. Contact card used gold plated six or eight contact on the face of the card. Functions of it vary depending on the card configuration and application but all must has connection for power, reset, clock and data I/O. The specific definitions for the contact are shown in *Table 2.1*.

Contact	Designation	Use
C1	V_{cc}	Power connection through which operating power is supplied to the microprocessor chip in the card.
C2	RST	Reset line through which the IFD can signal to the smart card's microprocessor chip to initiate its reset sequence of instructions.
C3	CLK	Clock signal line through which a clock signal can be provided to the smart card's microprocessor chip to control the speed at which it operates and to provide a common framework for data communication between the reader and the smart card.
C4	RFU	Reserved for future use.
C5	GND	Ground line providing a common electrical ground between the reader and the smart card.
C6	V_{pp}	Programming power connection providing a separate source of electrical power (from the operating power) that can be used to program the nonvolatile memory on the microprocessor chip.
C7	I/O	Input/output line that provides a half-duplex communication channel between the reader and the smart card.
C8	RFU	Reserved for future use.

Table 2.1 Contact definition for smart cards

To show how smart card communicates, we take a look on using smart card through a terminal or reader. To illustrate how this works in a real application the following shows how a typical security transaction occurs for access to a locked door:

Door	**Card Reader**		**Smart Card**
	Reset	-->	
		<--	Answer reset
	Request card authenticate	-->	
		<--	Send random number
	Request authentication data	-->	
		<--	Authenticate reader
	Authenticate card	-->	
		<--	Transmit reader authentication data
	Open access file to read	-->	
		<--	Open access file

		Request read access file	-->	
			<--	Transmit access code
		Write date and time of transaction to access file	-->	
			<--	Acknowledge
		Request unload	-->	
			<--	Acknowledge
Open door	<--	Send authorized command		

The transmission characteristics of most smart card ICs are based on an asynchronous half duplex mode. The T=0 protocol involves the transmission of bytes while the T=1 protocol defines a block mode of operation. Note that two I/O signal connectors are required for full duplex operation so that transmission can take place in both directions at the same time. ISO defines only a single line for interchange of data between the IC and the interface device. In this mode of operation the line must change direction depending on whether the IC is transmitting or receiving.

Once the card is inserted into the reader, no power is applied to any of the contacts. When the reader detects that the card is properly inserted, power is applied to the card. First, the contacts are brought to a coherent idle state, as shown in *Table 2.2.* A reset signal is then sent to the card via the RST contact line. The idle state is characterized as being when the power (VCC) contact is brought up to a normal, stable operating voltage of 5v. An initial power setting of 5v is always applied first, even though some microprocessor chips being introduced operate at 3v when in an I/O

state. The I/O contact is set to a reception mode on the reader side and a stable clock (CLK) is applied. The reset line is in a low state. It must remain in a low state for at least 40,000 CLK cycles before a valid reset sequence can be started by the reader, raising the reset line to a high state.

Contact	State
V_{CC}	Powered and stable
V_{PP}	Stable at idle state
RST	State—low
CLK	Suitable and stable clock signal applied
I/O	Reception mode in interface device

Table 2.2 Contact states prior to card reset

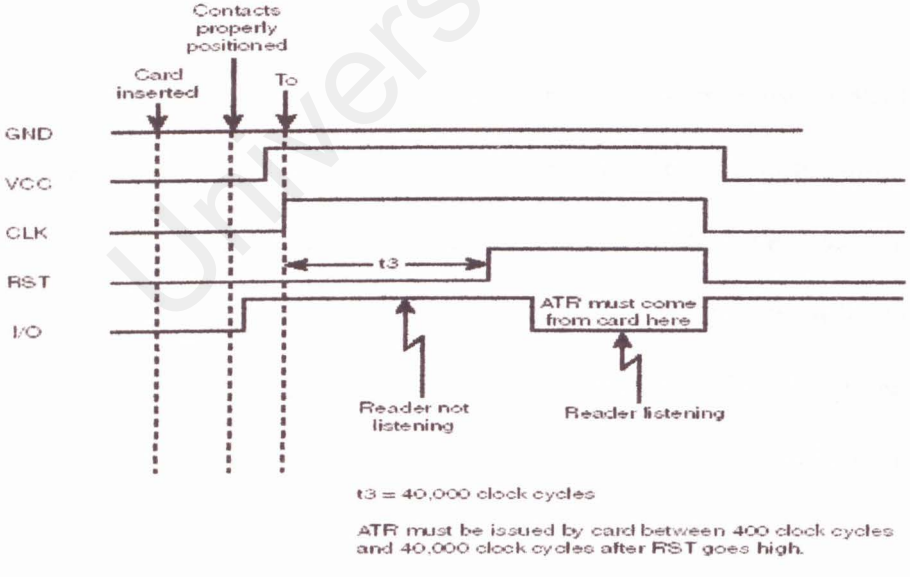


Figure 2.1 The reader reset sequence

Data transfer between the reader and the card occurs through the concerted action of two of the contact lines: CLK and I/O. The I/O line conveys a single bit of information per unit of time as defined by the CLK depending on its voltage relative to GND. A 1 bit can be conveyed either through a +5 v value or through a 0 v value. The actual convention used is determined by the card and is conveyed to the reader through the "initial character" of the ATR, which is referenced as *TS*. To transfer 1 byte of information, 10 bits are actually moved across the I/O line; the first is always a "start bit" and the last is always a parity bit used to convey even parity. Considering that the I/O line can be (in one bit period) either in a high (H) state or a low (L) state, the TS character of the form HLHHLLLLLLH signals that the card wants to use the "inverse convention," meaning that H corresponds to a 0 and L corresponds to a 1. A TS character of the form HLHHLHHHLLH signals that the card wants to use the "direct convention," meaning that H corresponds to a 1 and L corresponds to a 0.

The direct convention and the inverse convention also control the bit ordering with each byte transferred between the card and the reader. In the direct convention, the first bit following the start bit is the low-order bit of the byte. Successively higher-order bits follow in sequence. In the inverse convention, the first bit following the start bit is the high-order bit of the byte. Successively lower-order bits follow in sequence. Parity for each byte transferred should be even; this means that the total number of 1 bit in the byte, including the parity bit, must be an even number.

Protocols Used in Smart Card Communications

T=0 protocols

T=0 is an asynchronous protocol, meaning there is no strict timing connection between one command sent from the reader to the card and the next command sent from the reader to the card. When the card receives a command from the reader, it performs the requested operations and sends back to the reader a response relative to that command. The reader is then free to send the next command to the card whenever it needs to.

T=1 protocols

The T=1 protocol is a block-oriented protocol. This means that a well-defined collection of information—or *block*—is moved as a single unit between the reader and the card. Embedded within this block structure may be an APDU defined for a specific application. This facility is a good illustration that the T=1 protocol provides excellent layering between the link protocol layer and the application protocol layer. Moving information in a block, however, requires that the block be transferred (between the reader and the card) error free, or else the protocol can easily get lost. The error detection and correction, then, is a significantly more complex operation than was the case with the T=0 protocol.

Error detection in the T=1 protocol is done by using either a *longitudinal redundancy character*, which is essentially a slightly more complex form of parity checking than was done in the T=0 protocol, or by using a *cyclic redundancy check* character, which is guaranteed to detect any single-bit errors in a transmitted block. The specific CRC

algorithm used is defined in detail in the ISO 3309 standard. When an error is detected within a block by the received end of the channel, it signals the transmitting end to repeat sending the block received in error.

The T=1 protocol makes use of three different types of blocks, as illustrated in *Figure 2.5.*

Each has the same structure, but serves a different purpose:

- *Information block*—this block is used to convey information between application software in the card and application software on the reader-side of the channel.

- *Receive ready block*—this block is used to convey either positive or negative acknowledgments from one end of the channel to the other. A positive acknowledgment indicates that a block was correctly received while a negative acknowledgment indicates that an error was detected (via checking the LRC or CRC) in the received block.

- *Supervisory block*—this block is used to convey control information between the card and the reader.

Each T=1 block comprises three fields:

- Prologue field—a mandatory field in the block which is 3 bytes in length. It includes the following three elements:

    1. NAD—Node address

2. PCB—Protocol control byte
3. LEN—Length

- Information field—an optional field in the block that may be up to 254 bytes in length.
- Epilogue field—a mandatory field in the block that is either 1 or 2 bytes in length.

2.2 Keyboard Interfacing

In this segment, we will find out how the keyboard interfaces with the computer and how it can be integrated with the smart card.

2.2.1 How the keyboard communicates with the system?

The keyboard communicates on a character-by-character basis. Each key, including SHIFT, CTRL and ALT, sends a specific code when depressed and another code when it is released (the break code is the make code proceeded by F0h). ASCII is impractical for communication with a keyboard (especially with special features such as typematic rate) so special scan codes are used and converted to ASCII by the BIOS.

Certain keys produce series of scan codes called scan sets. There are 3 scan sets in use with set 2 being the most common. (Sets 2 & 3 used on AT keyboards.)

The 101/102 keyboard is laid in a 16 row by 8 column matrix. With set 1 or 2, for cursor control keys: ALT, CTRL, DEL, PgUp, PgDn, Ins, Home, End, the keyboard issues a series of codes dependant on shift keys (alt, ctrl, shift) and on the status of the indicator of the Num lock key. Since these keys are duplicated, the basic scan codes are identical - to identify the alternate key, an extra code E0h is added to the basic code. With set 3, each key generates just one code unaffected by the status of other keys.

With set 1, the up code is obtained by adding 80h to the down code. For set 2 and 3, the 1st byte is F0h and the 2nd byte is the down code for that key.

If a key is pressed, the keyboard acknowledges the key and issues the down key code. If a 2nd key is pressed while the 1st is still depressed, the 2nd key is acknowledged and it's down code sent. If the 2nd key is released before the first, the 1st key is deactivated. To reactivate the 1st key, it must first be released.

If two or more keys are pressed simultaneously, all are validated and all codes are sent (no error is generated).

With the exception of the pause key, all keys when held down for a certain time, auto-repeat. The down code is repeated until the key is released. If two or more are pressed together, only the last key pressed is repeated. The repeat stops when the last

key pressed is released, even if the other keys are still depressed. The delay is usually 500ms and the repeat 10/s and can be modified.

The interface is serial, using the KBCLK (generated by the keyboard) and KBDTA lines. KBDTA is bidirectional! Data format is 11 bits for scan sets 2 & 3 - 1 start bit, 8 data bits (LSB 1st), 1 odd parity bit and 1 stop bit.

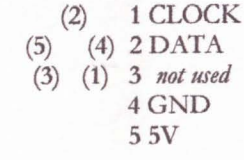
Data from controller to keyboard always has priority. As long as the keyboard has not yet transmitted the 10th bit, the controller can take over the interface, which it does by pulling KBCLK low for 100ms. Within 5ms the keyboard will start sending low clock pulses on KBCLK. KBDTA must be set to the relevant logic level each time KBCLK is low and must remain there until KBCLK goes high then low again. The keyboard reads this data bit when the clock pulse is high. If there is no clock within 20ms or transmission takes more than 2ms assume a transmit time-out and send a resend command to the keyboard (FEh).

For the keyboard to send the controller data, both KBCLK and KBDTA must be high. The keyboard begins by pulling KBDTA low and then starts sending clock pulses on KBCLK. The data bit is output while the clock is high and remains while it goes low then high again. The controller should read the data while the clock is low. If the controller holds the clock low the transmission will be aborted and retried once the line is free.

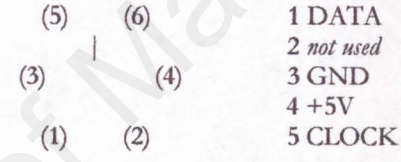
After the POST (power on self test) the keyboard issues an AAh to the system (FCh if failure). The signals source is open-collector TTL (low <0.8V high>2.4V). Signals are on a 6-pin DIN or SDL (PS/2) connector (5 wires - 1 supply, 2 gnd, dta and clk):

1. Data In/Out
2. N/C
3. Gnd
4. +5V
5. Clock
6. Test (not used / Gnd)

AT pin out diagram



PS/2 pin out diagrams



Chapter 3

VHDL

3.0 VHSIC Hardware Description Language

In this project, all the design will be done by using a hardware description language or better known as **VHDL. VHDL** stand for ***Very high-speed integrated circuit Hardware Description Language.*** The language now is the most used design and modeling language for digital systems. This leads to systems design and synthesis. It has been used in field of robotics and development of microprocessor. Beside that it is useful for describing hardware for simulation, testing, design, modeling and documentation. In the sense of hierarchical representation for functional and wiring details of digital systems, it provides the most convenient and compact format.

3.1 What Is VHDL?

It is a language that can be used for modeling a digital system at many level of abstraction, from algorithmic level to the gate level. The complexity could vary from a simple gate to a complete digital electronic system, or anything in between. VHDL always regarded as integrated amalgamation of some languages, which is: -

- *Sequential language* +
- *Concurrent language* +
- *Net-list language* +
- *Timing specification*
- *Waveform generation language*

The language has feature or constructs that enable users to express the concurrent or sequential behavior of a digital system with or without timing. It also can model the system as an interconnection of components. By using this constructs, test waveforms can be generated. In the end, the constructs can provide a comprehensive description of the system in a single model.

Not only it defines syntax but also clearly defines simulation semantics for each construct. Thus all the models written can be verified using the VHDL simulator. Since it is a strong typed language, it is often verbose to write. VHDL inherits the sequential language part features from ADA programming language. Because of the language provide an extensive range of modeling capabilities; it is often very difficult to understand. However it is possible to assimilate a core subset of the language that is both easy and simple to understand without learning the more complex features. The subset is sufficient to model most applications. The complete language however has sufficient power to capture the descriptions of the most complex chip to a complete electronic system.

3.2 The Advantages

The advantages that the VHDL offers are: -

- *Standard*: VHDL is like an EKE standard. Just like any standard, it reduces confusion and makes interfaces between tools, companies and products easier.

Development to the standard will give better chance to lasts longer and less chance of being obsolete due to incompatibility with others.

- *Industry support*: It has became tools for the industry and supported by the electronic industry

- *Portability*: Because of the codes used can be simulated and used in many design tools and in different stages, it reduces dependency for the set of tools whose limited in capability. The VHDL standard also transforms design data much easier than a design database of a proprietary design tool.

- *Modeling capability*: It is developed to model all level of designs, from electronic boxes to transistors. It can accommodate behavioral constructs and mathematical routines that describe complex models, such as queuing network and analog circuits. It allows use of multiple architectures and associates with the same design during various stages of the design process.

- *Reusability*: Designs can be described, verified, and modified for future use. This eliminates reading and marking changes to schematics pages that are time consuming beside subject to error.

- *Technology and foundry independence*: The functionality and behavior of the design can be described with VHDL and verified, making it foundry and technology independent. This frees the designer to proceed without having to wait for the foundry and technology to be selected.

- *Documentation*: VHDL is a description language, which allows documentation to be located in single place by embedding it in the code. The combining of comments and the code actually dictates what the design should do reduces the ambiguity between specification and implementation.

- *New design methodology*: Using VHDL and synthesis creates a new methodology that increases the design productivity, shortens the design cycle, and lower costs. It amount to a revolution comparable to that introduced by the automatic semi-custom layout synthesis tools of the last few years.

3.3 New Design Methodology

The introduction of VHDL and synthesis enables the design community to explore a new design methodology. With the traditional approach, starts with schematics drawing and then performs functional and timing simulation based on the same schematics. If occur errors it back to update the schematics again. After the layout, function and back-annotated timing are verified again with the same schematics. With VHDL, the design is functionally described.

VHDL simulation is used to verify the functionality of the design. In general modifying VHDL source code much faster then changing schematics. This allows designers to make faster functionally correct designs, to explore more architecture trade-offs and to have more impact on the designs. After the functions meet the requirement, the code then synthesized to generate schematics (or equivalent net lists). These net lists can be used to layout the circuit and to verify the timing requirement (both before and after the layout). Changes can be made by modifying the code or the constraints (timing, area and so on) in the synthesis. This new design approach and methodology has improved the design process by shortening design time, reducing the number of design iterations, and increasing the design complexity that designers can manage.

3.4 Hardware Abstraction

VHDL is used to describe a model for a digital hardware device, which specifies the external view of the device and one or more internal views. The internal view specifies the functionality or structure, while the external view specifies the interface of the device through which it communicates with the other models in its environment.

The device-to-device model mapping is strictly one-to-many. For example, a device modeled at high level of abstraction may not have a clock as one of it inputs, since the clock may not have been used in the description. Also, the data transfer at the interface may be treated in terms of integer values, instead of logical values. In VHDL,

each device model is treated as a distinct representation of a unique device, called an *entity* in this text.

3.5 Elements

Constructs of the VHDL language are designed for describing hardware component, packing parts and utilities, and use of libraries. It also designed for specifying design libraries and parameters. In simplest form, the description of component in VHDL consists of an interface specification and architecture specification.

- ✓ *Interface specification* begins with the **ENTITY** keyword and contains the input-output ports of the component. The external characteristic of a component, such as time and temperature dependencies, can also be included in the interface description.

- ✓ *Architectural specification* begins with the keyword **ARCHITECTURE,** which describe the functionality of a component. This functionality depends on the input-output signals and other parameters that are specified in the interface description. Several architectural specification with different identifiers can exist with one component with a given interface description.

3.6 Basic Concept

Since VHDL is a hardware description language, it has features which are conceptually different than other languages. These represents special characteristic of hardware components and carries. These are timing and concurrency.

3.6.1 Timing

Timing is associated with the values that are assigned to the hardware carriers. Signal represents real wires, where the delays of the transfer through wire are concern, thus the assignment to signal in VHDL, involve timing. Consider the assignments in software as shown below:

```
a = x;
b = x;
```

These assignment only consider the values transfer from **x** to **a** and **b**, ignore the timing of such transfer.

3.6.2 Concurrency

The terms refer to the simultaneous operation of various components. The VHDL has constructs that allow a virtually concurrent environment to be created. These constructs satisfy concurrency required for the description of the hardware. Through the use of concurrent constructs, timing of the interconnecting signals and order of the simulation

construct or components, a VHDL simulator makes us think that the execution is being done concurrently.

3.7 Objects and Classes

An object is an entity that has a value of a given type. Some of the object in VHDL is port signals, loops index, variables, and signals for interconnecting components, temporary variables and files. In VHDL there a four classes in VHDL and an object may be one of these classes:-

- *Signal class* – object in this class represent the hardware wires and have timing associated with them. Value assigned to the signals is placed on the signal driver and will appear on the signal after a specified delay value.

- *Variable class* – objects are for storage of temporary values and have no hardware significance. The variables can only be declared or have values assigned to them in sequential bodies of VHDL.

- *Constant class* – object represent the values of a given type. Can be declared and used in concurrent and sequential bodies. These values cannot be changed.

- *File class* – objects contain data of the same type. Also can be declared and be used in both concurrent and sequential bodies.

3.8 Signals Assignment

One of the important issues in VHDL, in it simplest form it consist of a target signal on the left-hand side of a left arrow and expression for defining a waveform on the right-hand side. An assignment can include an **AFTER** clause specifying that a physical time delay occurs before the assignment to the left-hand side takes place. Signal assignment can have inertial, inertial with reject and transport mechanism.

3.9 Signal Driver

A signal has driving value and there may be several pending transactions on this signal waiting to become current. When a transaction become current, its value becomes the driving value of the signal. Assignments to a signal in multiple concurrent bodies create multiple drivers to the signal. Such a signal must be resolved and a resolution function must exist to resolve the value from the multiple driving values.

3.10 Packages

Often, in a hardware design environment it becomes necessary to group components or utilities used for description of components. Examples of VHDL constructs for describing utilities and environments are type definitions and subprograms. Components and such utilities can be grouped by used of packages.

Chapter 4

Design Development

4.0 Design Development

4.1 Microcontroller Design Steps

The microcontroller design will incorporate some design that will be able to control the two devices. Initially will start with the design idea, a complete and thorough definition of the intended hardware must then be developed from the initial design idea. This stage will determine overall functionality of the input to output mapping and the functional specification will be decided. Then, the data path of the interconnected components is designed and developed. Later the logic design then is done by using primitive gates. Finally if the design is successful then the physical hardware components are wired and manufactured which are will be done on this thesis. The work will only focus on the designs.

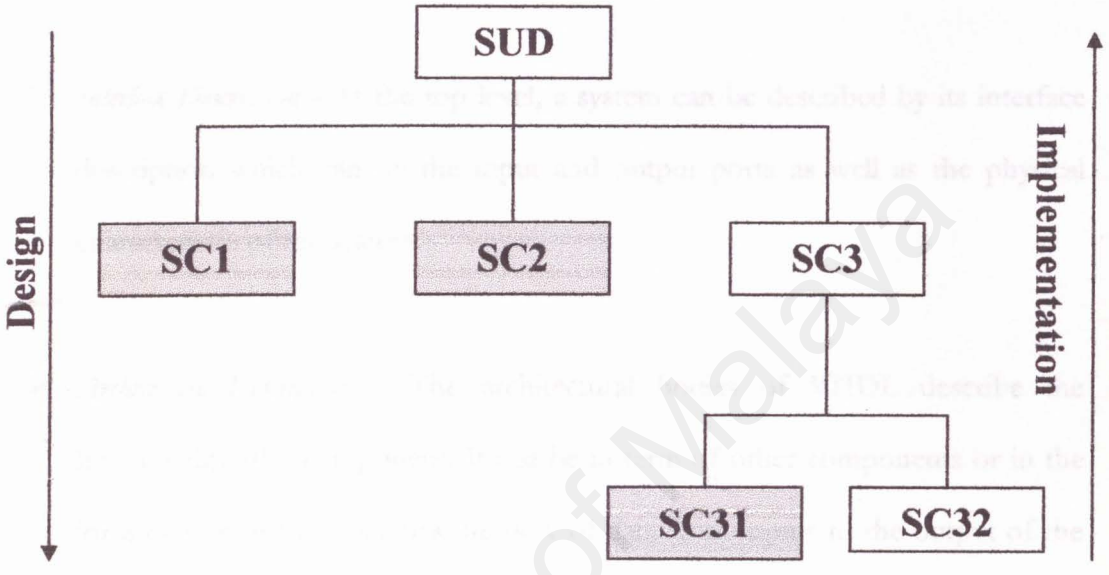
4.2 Analysis of Designs

To make the design is a success two methodologies being implemented which is typical digital system design and top-down design with VHDL.

4.2.1 Top-Down Design

Top-down design technique is recursively partitions a system into its sub-components until all sub-components become manageable design parts. Design became manageable when the component is available as part of the library. It can be implemented by modifying an already available part.

Mapping to hardware depends on target technology, available libraries and tools. Generally, a system can be further partitioned into its simpler components. Figure 4.1 shows the implementation of top-down and bottom-up design.



SUD – system under design
SSC – system sub-component
Shaded areas designate sub-components with hardware implementation

Figure 4.1 Top-down design and bottom-up implementation

4.2.2 Designing with VHDL

When designing with VHDL, synthesis tools, complex library elements and a set of configurable parts are available as the manageable parts.

Synthesis tools – Current tools are capable of translating high-level description into an interconnection of logic cells or **FPGA** (Field Programmable Gate Arrays) layouts. Such examples are combinatorial circuits.

Libraries – Prepackaged libraries are available in most technologies. In fact, libraries consist of pre-designed, tested and commonly used functional units. Such examples are multiplexer, flip-flop and ICs.

- *Interface Description* - At the top level, a system can be described by its interface description which can be the input and output ports as well as the physical characteristic of the system.

- *Architecture Description* – The architectural bodies of VHDL describe the functionality of a component. It can be in term of other components or in the form definition that specifies the flow of data from inputs to the output of the circuit (I/O mapping). *Dataflow* is referred to the description where functionality of a hardware component is mainly described in term of data flow through buses, logic units and registers.

- *Behavioral Description* – In this level, assigning appropriate value to circuit is the concern. The structure and details of the hardware are out of concern; instead I/O mapping and functionality are main concern. At this level also whereas ports of models correspond to the actual inputs and outputs of the actual circuit.

Configurable Parts – Pre-designed parts in the previous design can be used in the current design by reconfiguring them.

4.2.3 Design Scenarios

i. **Analyzing the requirement**

Design begins with analyzing the requirement and developing an implantable model of the system. In this stage, interface and behavioral description are developed. Before any next step is taken, the behavioral description must be simulated.

ii. **Recursive partitioning**

After simulating the behavioral description, the code is studied for generating the general layout. Specify the functionality that can be implemented by a correspondence hardware.

- *First-level of partitioning*

  Every terminal must be checked so it can be synthesize, availability of libraries, or as a pre-designed parts. After that, the VHDL description will be developed to study way of designing them.

- *Second-level of partitioning*

  If there is sub-component that cannot be synthesized, second-level of partitioning will be performed. The design is not complete until all terminals have corresponding hardware.

iii. **Design implementation**

Bottom-up implementation consists of wiring all the elements that have hardware correspondence or that appear as the terminal nodes of the final partition.

4.2.4 Final Step

After that the final layout from the final partitioning is produced.

4.3 Design Idea

The main purpose here is to design a microcontroller that will integrate two devices which is a keyboard and smart card reader. The final out come is a microcontroller that capable to handle these device so both of it can works properly in the same time. When it completed, it can be used with any keyboard and any smart card reader for any computer because it will interfaced with the computer through the normal AT keyboard port or the P/S2 port. The controller should able to:

- ✓ Interface with the computer without any problem
- ✓ Handle the task as a terminal for the smart card
- ✓ Functions as a normal keyboard
- ✓ Interface the reader with the computer

4.4 Keyboard Controller Design

As we all know the keyboard interfaces with the computer using binary code. But the keys are assigned with hexadecimal numbers according to the **ASCII** code. Therefore to enables the computer to understand what the user type in, there must be a decoder for decode the hexadecimal value to binary value. With each transmission is clocked serial (with the keyboard as clock source or 'master') and 11 bits in length. These bits consists of one start bit (logic 0), 8 data bits (LSB first), one odd parity bit and one stop bit (logic 1). The clock rate is approx. 10-20 KHz and can vary from keyboard to keyboard. The keyboard data format resembles 8-odd-1 asynchronous transmission format. However, the bit rate from keyboard to keyboard can vary significantly so it is necessary to use a clocked serial interface (SPI in Motorola parlance) or a *async* (SCI) interface with a 1x receive clock input.

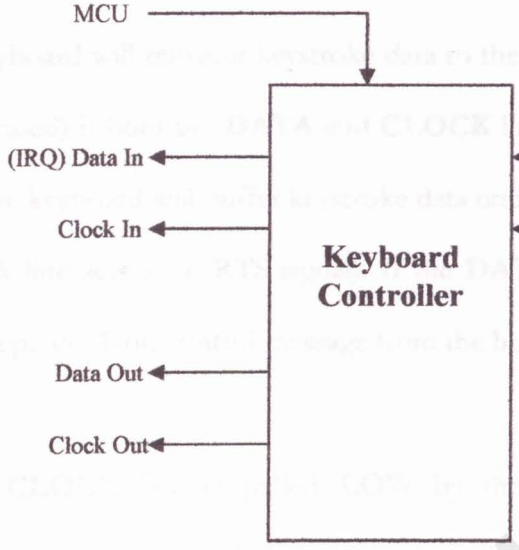


Figure 4.2 Block diagram of a keyboard controller

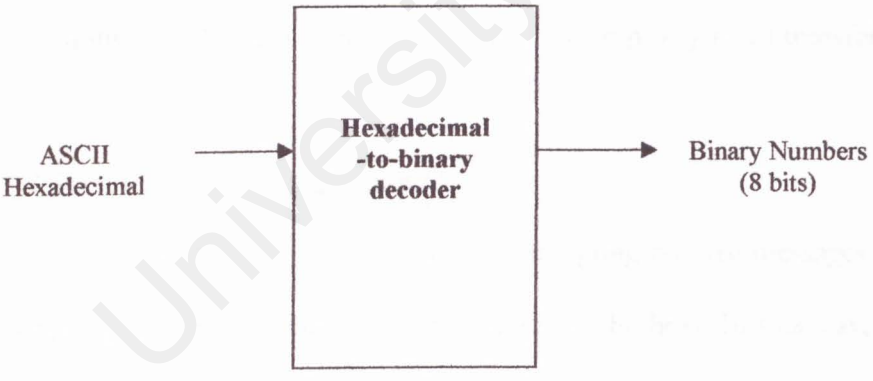


Figure 4.3 ASCII hexadecimal to binary decoder

4.4.1 Keyboard to System Protocol

The keyboard will transmit keystroke data to the host system as soon as a key is pressed (or released) if both the **DATA** and **CLOCK** lines are **HIGH**. If the **CLOCK** line is **LOW**, the keyboard will buffer keystroke data until the **CLOCK** line goes **HIGH** again (the clock line acts as a -RTS signal). If the **DATA** line is **LOW**, the keyboard prepares to accept an 11-bit control message from the host (see below).

If the **CLOCK** line is pulled **LOW** by the host while the keyboard is transmitting data (up to the 10th bit) for at least 60uS, the keyboard will suspend transmission and prepare to receive a system message. The keyboard will eventually re-transmit the data byte that was interrupted, unless the keyboard was successful in transmitting the 10th bit, in which case the keyboard considers the data byte as successfully sent. It is generally a bad idea to interrupt keyboard transfers this way.

4.4.2 System to Keyboard Protocol

AT-style keyboards are capable of accepting control messages from the system in addition to sending scan code information to the host. In this way, the status LEDs may be controlled and the keyboard typematic parameters (repeat delay and rate) can be set. Command transmission to the keyboard is initiated by bringing the keyboard **CLOCK** line **LOW** for at least 60uS. After the 60uS delay, the **DATA** line should be brought **LOW** and the **CLOCK** line released (**HIGH**). Make sure to bring the **DATA** line **LOW** before releasing the **CLOCK** line. Some time later (up to 10 milliseconds) the

keyboard will start to generate clock signals. At each **HIGH** to **LOW** clock transition the keyboard will clock in a new bit. The data stream from the system (on the **DATA** line) should conform to the serial protocol described above.

After the parity bit has been clocked out, the host should release (bring **HIGH**) the **DATA** line and wait for the keyboard to send another **CLOCK** pulse (this will be the 10th **HIGH** to **LOW** transition, not counting the initial H->L transition generated by the host). The KEYBOARD will then bring the **DATA** line **LOW** sometime before the 11th **HIGH** to **LOW** transition of the **CLOCK** line to acknowledge reception of the command byte. If the **DATA** line is not released (allowed to go **HIGH**) after the 10th **CLOCK** bit then the keyboard will continue to issue clock pulses until the **DATA** line is released. In this event, the keyboard will (after some delay) pull the **DATA** line **LOW** and transmit a **RESEND** status byte (FEh).

4.5 Smart Card Reader Controller Design

For interfacing the smart card, the reader terminal firstly must detect is the card is inserted properly. This is because if it not inserted properly when the power is applied, it can and will damage the card.

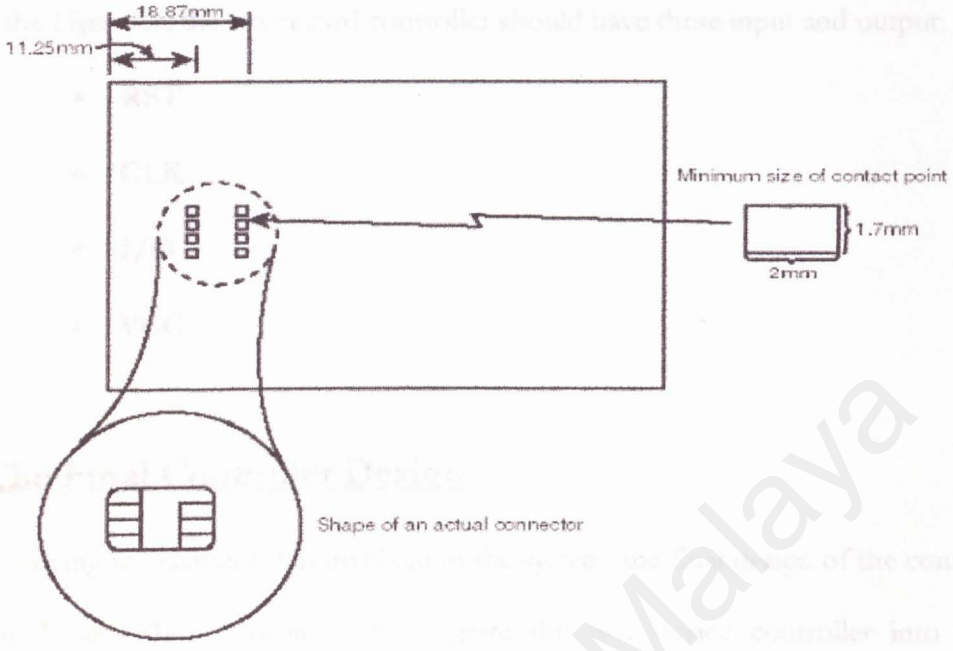


Figure 4.4 Location, size, and shape of contacts

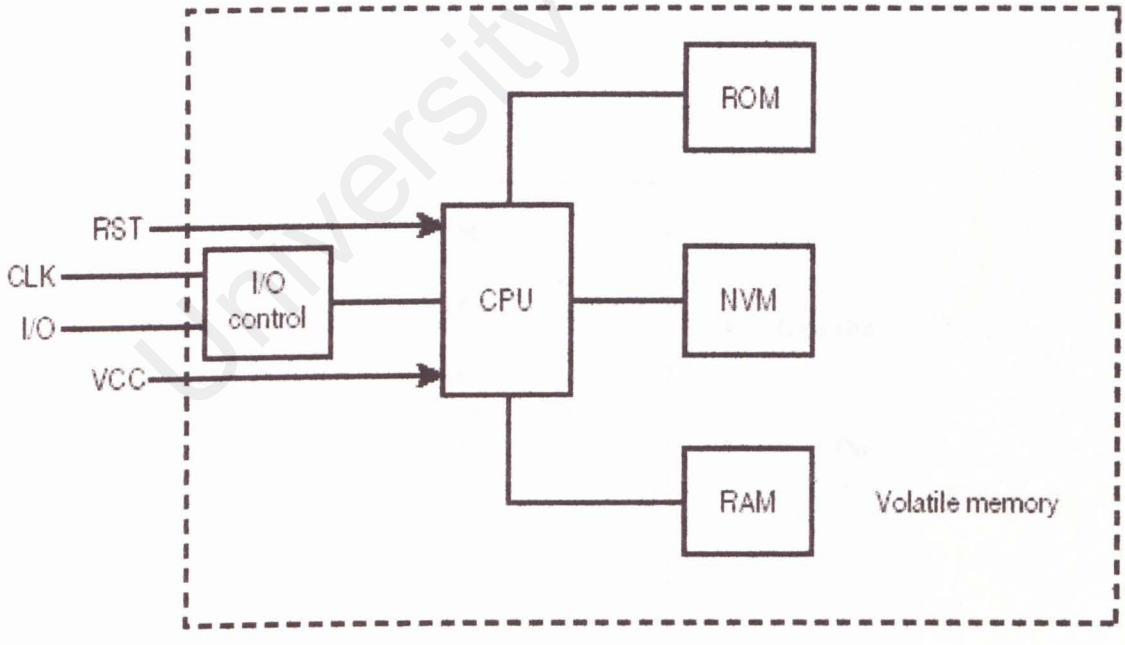


Figure 4.5 The computer element of the smart card

From the *Figure 4.5*, the smart card controller should have these input and output:

- **RST**
- **CLK**
- **I/O**
- **VCC**

4.6 The Final Controller Design

After studying the element that involved in the system, the final design of the controller is figured out. The controller will integrate the two device controller into single microcontroller. It will function as the normal device controller but at the same time also handle the next device function also. *Figure 4.6* shows how the final design looks like.

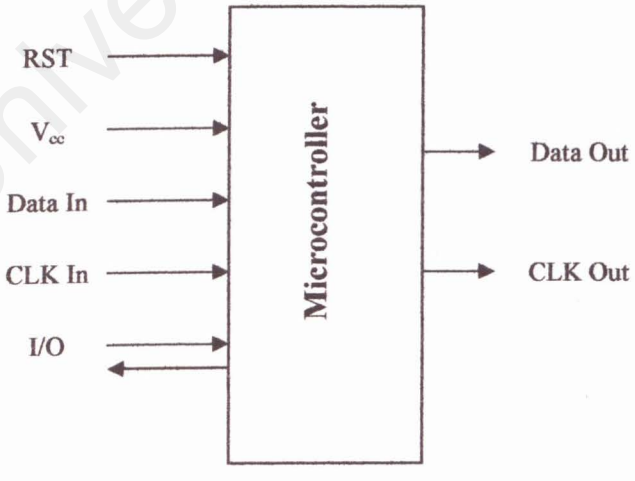


Figure 4.6 Microcontroller proposed pin diagrams

Some modification will be done in the next stage before the implementation, if the design is not working properly. Testing of the design will be done after all the designs correspond correctly. This to make sure there will be no glitch in the design prior the implementation.

Chapter 5

Development and Testing

5.0 Development and Testing

5.1 Introduction

In this phase the design will be used to develop the controller. The design will be develop using the *Xilinx Foundation Series* software. From the design there will be several modules to be developed. By using the software we can develop, compile, simulate and test the design. Each module will be develop and tested individually before being compiled together with the other module as the top level design. The controller will consist of 7 modules that are:

i. *counter8*

ii. *parity*

iii. *shiftregPISO*

iv. *SIPOshiftreg*

v. *ps2dcdr*

vi. *devsync*

vii. *mux_2_to_1*

5.2 Module Development

In this section each function of the module are described accordingly. This is for more understanding of the design of the controller itself. Therefore the development process will be fewer problems to the programmer.

5.2.1 Module Description

In the controller there will be consist of 7 modules, each has it own function and these are:

a) *counter8.vhd*

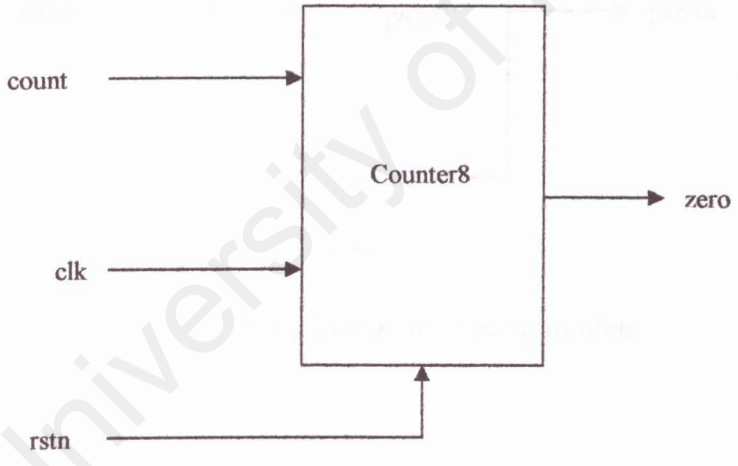


Figure 5.1 Block diagram of *counter8* module

Function:

Mod 8 counter with an output indicating when the value stored is zero.

Input / Output Description:

count	-	assert to count clock
clk	-	clock
rstn	-	asynchronous active low reset
zero	-	indicates counter = 0

b) *parity.vhd*

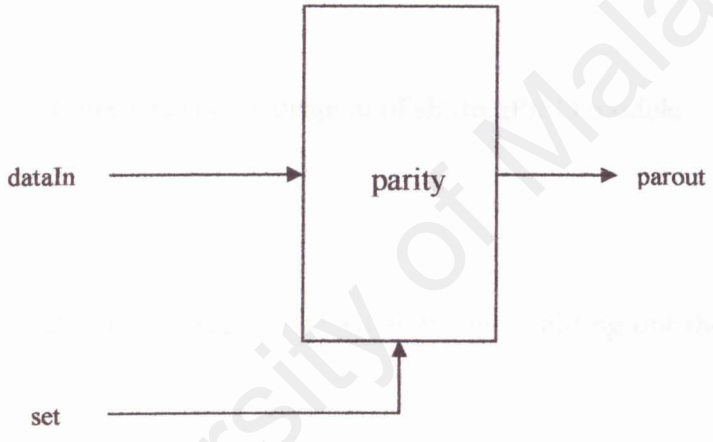


Figure 5.2 Block diagram of *parity* module

Function:

Manages the parity bit of the ps/2 device

Input / Output Description:

dataIn	-	data can be passed serial to calculate parity
set	-	assert to initialize parity (to 1)
parout	-	calculated parity

c) shiftregPISO.vhd

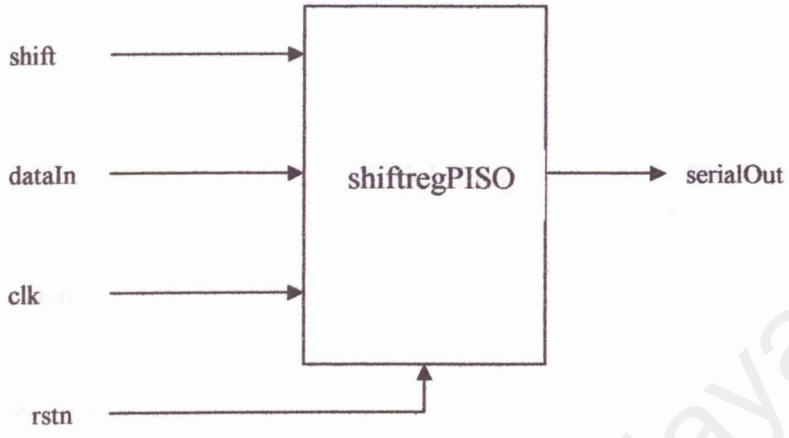


Figure 5.3 Block diagram of shiftregPISO module

Function:

Parallel in / serial out shift register which shift right, shifting out the LSB (**L**east **S**ignificant **B**it)

Input / Description:

shift- when asserted, shifts the register from MSB to LSB

dataIn - parallel data in bus

clk - clock

rstn - asynchronous active low reset

serialout - serial out line

d) SIPOshiftreg.vhd

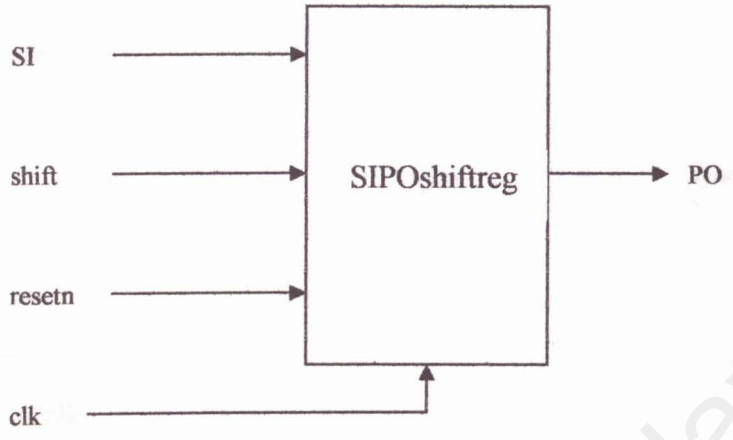


Figure 5.4 Block diagram of *SIPOshiftreg* module

Function:

Implements a SIPO shift register which shift right, shifting into the MSB (**M**ost **S**ignificant **B**it).

Input / Output Description:

SI - serial Input

shift - assert to shift data

resetn - asynchronous active low reset

clk - clock

PO - parallel output

e) devsync.vhd

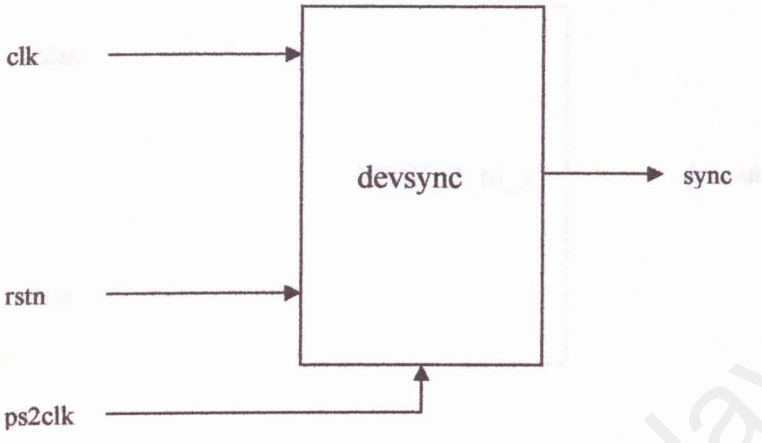


Figure 5.5 Block diagram of devsync module

Function:

Works as device synchronizer, this is when falling edge is seen on the ps/2 clock, sync is held high for one clock cycle.

Input / Output Description:

clk - clock

rstn - asynchronous active low reset

ps2clk - ps/2 clock line

sync - falling edge detected indicator

f) mux_2_to_1.vhd

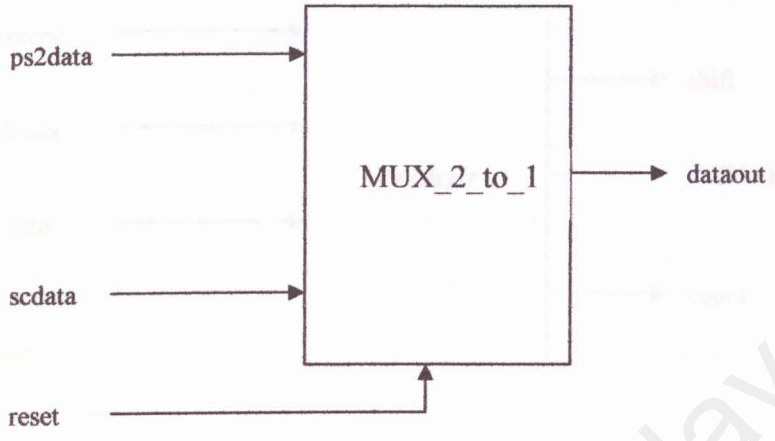


Figure 5.6 Block diagram of *mux_2_to_1* module

Function:

Works as multiplexor, which depends on the reset signal to give the line between the two device.

Input / Output Description:

ps2data - data from the ps/2 device

scdata - data from the smart card reader

reset - asynchronous active low reset

dataout - data that will be sent

g) ps2dcdr.vhd

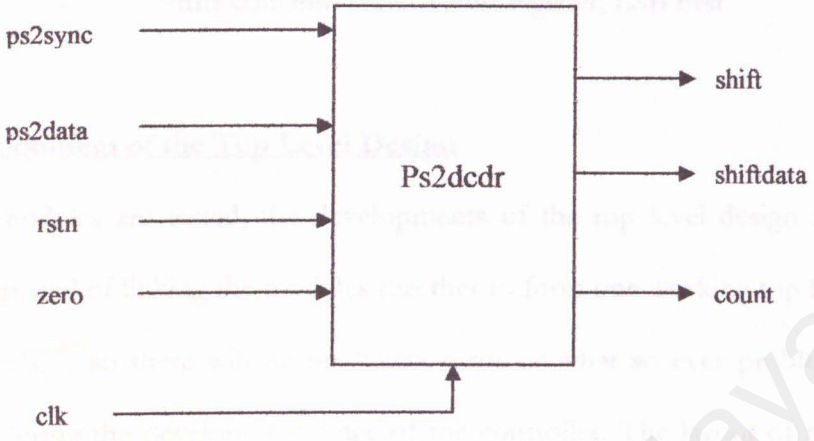


Figure 5.7 Block diagram of *ps2dcdr* module

Function:

State machine which converts serial data from ps/2 device into bytes and adds them into queue. Also control the sending of information to the ps/2 device.

Input / Output Description:

ps2sync - indicates falling edge of ps2 clock line

ps2data - bidirectional ps2 data line

rstn - asynchronous active low reset

zero - output of mod8 counter

clk - clock

count - increment mod8 counter

shiftdata - data to shift to receive shift register

shift - shifts data into receive shift register, LSB first

5.2.2 Development of the Top Level Design

After the modules are tested, the developments of the top level design are initiated. These comprised of linking the modules together to form one working top level module and compiling it so there will be no syntax error or what so ever problem that may encounter during the development stage of the controller. The layout of the top level design can be referred in the appendices section.

5.3 Synthesis

After the modules or entities are described, the next step is to synthesize the entities. It is a process of translation and optimization in digital design. For example, layout synthesis is a process of taking design netlist and translating it into a form of data that facilitates placement and routing, resulting in optimizing timing and / or chip size. Logic synthesis, in other hand, is the process of taking a form of input (VHDL), translating it into a form (a Boolean equations and synthesis tool specific), and then optimizing in propagation delay and / or area.

After the VHDL code is translated, the optimization process can be performed based on constraints such as speed, area, power, and so on. After the synthesis process

completed, the whole modules will be simulated to testify that behavior of the controller is correct. Simulation and tests done will be discussed in the section.

5.4 Tests and Simulation

5.4.1 Introduction

Tests and simulation are carried out in the next stage of development. Test is conducted on each module to determine the workability according to their codes. The test comprised of few steps which is syntax checking, functional simulation and timing verification.

5.4.2 Error Checking / Syntax Checking

The coding is check for any syntax error, starting from the design entry which in form of HDL (***Hardware Description Language***). This checking is done with all the modules by using **Xilinx Foundation Series** *HDL Editor*. After syntax checking return no error, the HDL code then being added to the project. From this we can conclude the steps and move on to the next step of development that is synthesizing the codes.

5.4.3 Functional Simulation

The simulation is done after the synthesis process is completed. The simulation tests at the functional level to testify the behavior of the controller is correct. Thus, simulation is done by using event driven stimulator.

5.4.4 Implementation

In the implementation stage / phase, verification of timing accuracy is done by using static timing analysis tool and this is important at gate – level timing stage.

5.5 VHDL Design Verification

The next step is simulation and it is done after the model description is successfully compiled. A simulation can be done on either one of the following:

- An entity declaration and architecture body pair
- A configuration

Then actual simulation commence on two phase:

1. *Elaboration phase*

   The entity is expanded and linked according to its hierarchy, components are bound to entities in a library and the top – level is built as a network of behavioral models is ready to be simulated.

2. *Initialization phase*

   Driving and effective values for all explicitly declared signals are computed, implicit signals are assigned values, processes are executed until they suspend, and simulation time is set to 0 ns.

In this design step, the VHDL description for each module of the controller had been created and ready to be simulated to insure proper logic functionality. By using script file, that can be generated from Xilinx's script editor, in order to do the simulation. It uses text file as input which contains the header information for the VHDL files such as port maps.

5.6 Testing

In order to determine that the hardware developed really works, a set of test were done to each module or entities that involve. To get the expected result, a set of data were chosen to be assign as the input values. The data will determine whether the entity working as it should be or not.

Chapter 6

Discussion

6.0 Discussion

6.1 Problems

During the development stage of the project, there are several problems faced and each will be described as below:

1. *Learning process*

   The problem faced here is learning the VHDL language itself. For a new user learning a new language will take time to familiarize with the syntax and so on. Thus, learning VHDL took much of the time than expected in developing the controller. Much time spent on referring to manual on using VHDL instead starting writing the codes. The cause of this problem is lack of time to study the language properly, limited references available that time and not having the good foundation in programming language also contributes to this problem.

2. *Coding problems*

   Because of many modules that involve, sometime syntax error always occurs during writing the codes. This is due to the lacks of VHDL programming knowledge and references. One major syntax error that occurs is 'netlist loaded with warning'. This is cause by type mismatch of entity name with the declared name in the component part of other entity. This problem easily solved by changing the name so it is identical.

3. *Limited resources and references*

   As we all know that VHDL is a new language that getting much support from the industry. It's a tool that makes modeling hardware much simpler and easier. Thus, resources are very limited on the VHDL and its standard. There not much reference book that available in store and library. If it available, the user level are for the advance users. This causes problem for the beginner users. Beside that, VHDL are taught in the faculty syllabus then although now the faculty just made it as one of the compulsory subject for the student.

4. *Hardware and software limitation*

   From the experience that I've been through, developing VHDL require much RAM size of the computer. Sadly the workstation that I've the opportunity to work with is low in memory, therefore making the development quite limited. This problem worsen when came to the synthesis process which took longer time if not with a bigger RAM size.

6.2 Solutions and Recommendations

To solve the problem faced, several of these solutions and recommendations should be considered:

a) Making VHDL as an tools for developing hardware in the faculty, encourage students to developed hardware using this tools for broaden their knowledge of computers hardware.

b) Improve the facilities of the laboratory or set-up a proper laboratory for hardware development in near future so there will be no shortage in skilled computer engineering students in future to come.

c) Lecturer should always encourage student to involve in computer hardware projects then more to software side, because it proven that nearly half of student doesn't know the computer hardware and it architecture very well.

d) Improve the information gathering systems so finding resource much easier such having own faculty library instead of a small document room.

e) Try to buy a new VHDL tools so the development not limited to a certain constraint, buying software like **Aldec** *Active HDL* would be an advantage because this software can do simulation and synthesis much faster than the **Xilinx Foundation Series** *Project Manager.*

6.3 Future Enhancement

The design of this controller is not perfect. For future enhancement, there are few feature can be added such as:

i) More compatible for other hardware also, such as thumb reader.

ii) Modify the controller so it can support other hardware or peripherals so it can function as one.

iii) Clean up the coding so there will be no delays created due to the imperfect codes.

iv) Try to implementing it by downloading into a FPGA chip and test it to determine the functionality of the chip.

However, future development of the controller should not only constraint to this design only but depend on the needs.

Chapter 7

Conclusion

7.0 Conclusion

In conclusion, the project had taught me a lot. Starting with the proposal of such project and managing in how to conduct the project so it covers all the objectives within the given time. Second, time management, without proper time management this project would never be completed on time but I've made it with little trouble.

Experiencing the use of VHDL in modeling a controller leaves me with much benefit for the future to come. The language is a tool that made the simulation of model behavior much easier for user through the graphical output, which easier to trace any fault in the design.

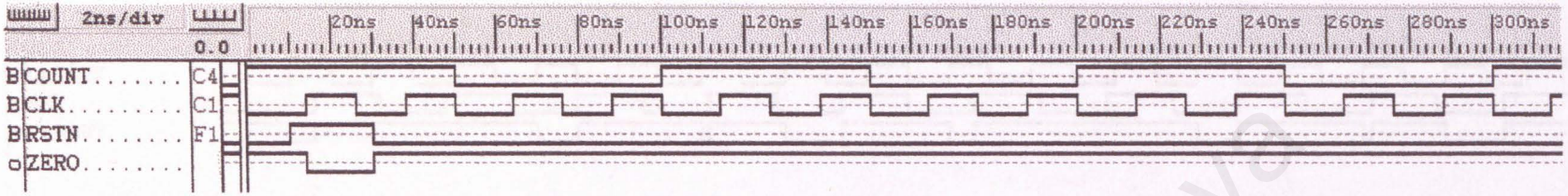
With the help of friends that always around, every problem encountered being resolve through discussion. This problem will be discussed in the next section with solution and recommendations for future use.

Here I've learned that developing source code for certain hardware are more complex then software such as a multimedia package because it involve every situation that involve an electronic circuit. Though the result from are true, but not in real life. This is because it depends on the circuit in the hardware that can effects the output data.

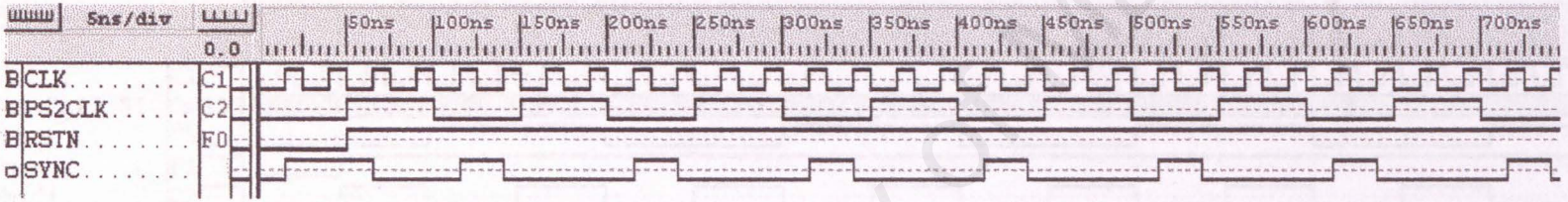
In whole, development of the *Keyboard – Smart Card Reader Controller* gave me one useful experience for me. I've learned the technique of designing and developing hardware that I cannot learn in my majoring in general. With this I can apply the knowledge of electronic that I've learned in class to develop such hardware.

Appendix

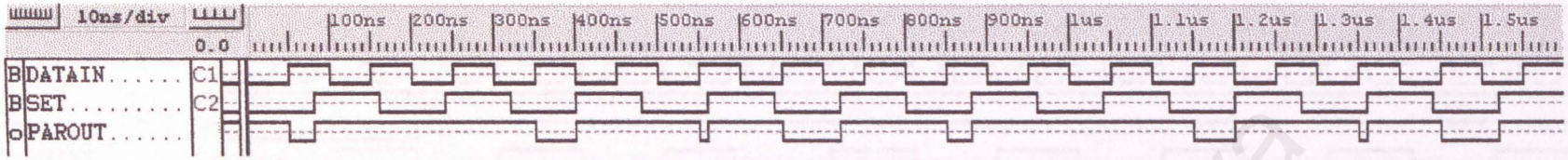
Appendix A



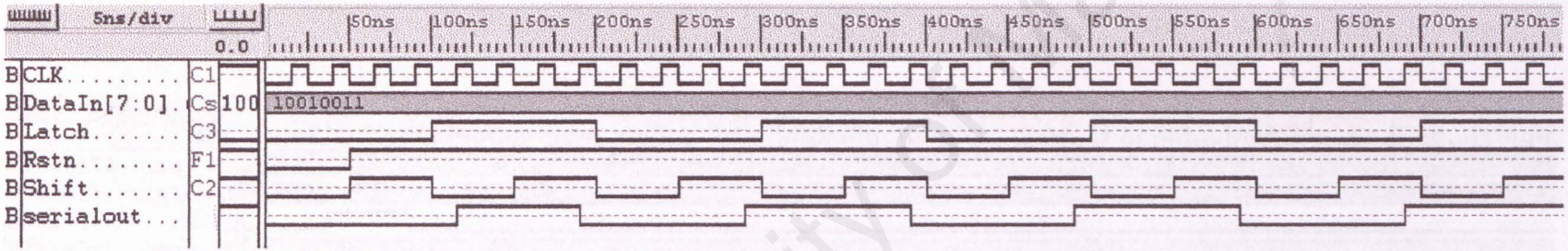
Waveform 1: counter8.vhd



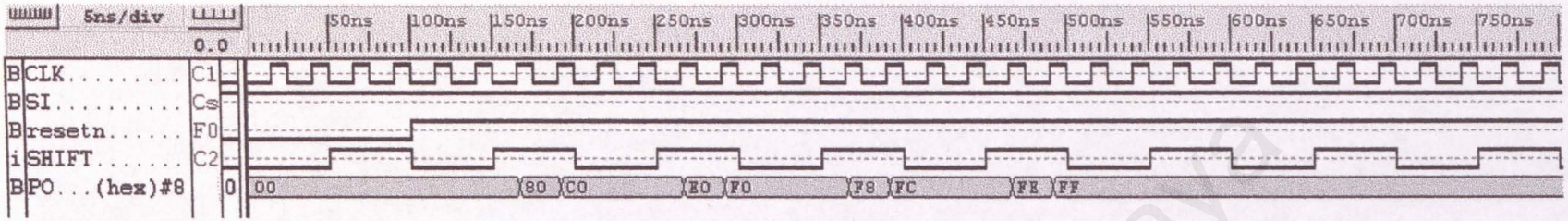
Waveform 2: devsync.vhd



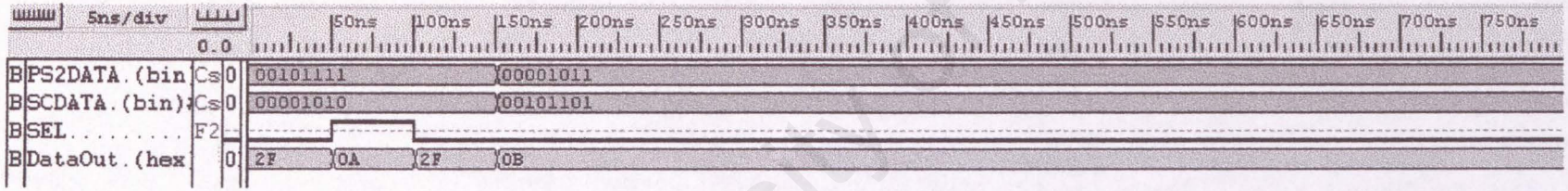
Waveform 3: parity.vhd



Waveform 4: pisoshiftreg.vhd



Waveform 5: siposhiftreg.vhd



Waveform 6: toplevel.vhd

Appendix B

Appendix B: **Optimized Version Reports**

```
====================
Chip ver1-Optimized
====================

Summary Information:
--------------------
Type: Optimized implementation
Source: ver1, up to date
Status: 0 errors, 0 warnings, 1 messages
Export: exported after last optimization

Target Information:
-------------------
Vendor: Xilinx
Family: XC4000XL
Device: 4002XLPC84
Speed: xl-3

Chip Parameters:
----------------
Optimize for: Speed
Optimization effort: Low
Frequency: 50 MHz
Is module: No
Keep io pads: No
Number of flip-flops: 16
Number of latches: 0

Chip Design Hierarchy:
----------------------
ps2keyboard: defined in
c:\fndtn\active\projects\wai\toplevel\ps2keyboard.vhd

Primitive reference count:
--------------------------
BUFG          1
DFF          16
FMAP         23
HMAP          5
IBUF          3
OBUFT         1
STARTUP       1
```

Clocks:

Period (ns)	Rise (ns)	Fall (ns)	Required Freq (MHz)	Estimated Freq (MHz)	Signal
20	0	10	50.00	-1.00	default
-1	-1	-1	-1000.00	100.00	clk_BUFGed

Timing Groups:

Name	Description
(I)	Input ports
(O)	Output ports
(RC,clk_BUFGed)	Clocked by rising edge of clk_BUFGed

Timing Path Groups:

From	To	Required Delay (ns)	Estimated Delay (ns)
(I)	(O)	20.00	0.00
(I)	(RC,clk_BUFGed)	20.00	3.40
(RC,clk_BUFGed)	(O)	20.00	9.00
(RC,clk_BUFGed)	(RC,clk_BUFGed)	20.00	6.80

Input Port Timing:

Port Name	Required Delay (ns)	Estimated Slack (ns)	To-Group
clk	19.90	19.90	(RC,clk_BUFGed)
rstn	n/a	n/a	(RC,clk_BUFGed)
ps2data	16.60	16.60	(RC,clk_BUFGed)
ps2data	16.60	16.60	(RC,clk_BUFGed)
ps2data	20.00	11.00	(RC,clk_BUFGed)
ps2data	20.00	11.00	(RC,clk_BUFGed)
ps2clk	16.60	16.60	(RC,clk_BUFGed)
ps2clk	16.60	16.60	(RC,clk_BUFGed)
ps2clk	n/a	n/a	(RC,clk_BUFGed)
ps2clk	n/a	n/a	(RC,clk_BUFGed)

Output Port Timing:

Port Name	Required Delay (ns)	Estimated Slack (ns)	From-Group
ps2data	16.60	16.60	(RC,clk_BUFGed)
ps2data	16.60	16.60	(RC,clk_BUFGed)
ps2data	20.00	11.00	(RC,clk_BUFGed)
ps2data	20.00	11.00	(RC,clk_BUFGed)
ps2clk	16.60	16.60	(RC,clk_BUFGed)
ps2clk	16.60	16.60	(RC,clk_BUFGed)
ps2clk	n/a	n/a	(RC,clk_BUFGed)
ps2clk	n/a	n/a	(RC,clk_BUFGed)

Critical Path Timing:

Cell Type	Arrival Time (ns)	Required Time (ns)	Fanout Count	Pin-Name
port	9.00	20.00	1	/ver1-Optimized/ps2da
OBUFT	9.00	20.00	1	/ver1-Optimized/C130/
OBUFT	1.90	12.90	1	/ver1-Optimized/C130/
DFF	1.90	12.90	1	/ver1-Optimized/decod
DFF	0.00	11.00	16	/ver1-Optimized/decod

Delays Report

File: toplevel.dly

The 20 Worst Net Delays are:

Max Delay (ns)	Netname
4.129	N82
3.968	decoder_presState<5>
3.618	decoder_presState<1>
3.478	syn560
3.478	syn193
3.427	counter_value<2>
3.390	ps2sync
2.951	syn570
2.888	count
2.837	decoder_presState<3>
2.782	decoder_presState<4>
2.662	decoder_ps2DataInt
2.550	decoder_presState<0>
2.256	counter_value<0>
2.188	decoder_presState<2>
2.082	counter_value<1>
2.079	syn683
2.036	N_ps2data
1.999	clk_BUFGed
1.914	N_ps2clk

Net Delays

N19
   rstn.I1
         1.535 N82.F1

N82
   N82.X
         4.129 C54.GSR

N_ps2clk
   ps2clk.I1
         1.914 edgeDetect_presstate.F2
         1.914 count.G4

N_ps2data
   ps2data.I2
         2.036 decoder_presState<0>.G4
         1.531 decoder_presState<0>.C3

clk

```
         2.860  decoder_presState<2>.F4
         2.770  decoder_ps2DataInt.F1
         1.361  decoder_presState<0>.G3

decoder_presState<2>
   decoder_presState<2>.YQ
         2.163  syn193.F1
         1.011  decoder_presState<2>.F3
         1.011  decoder_presState<2>.G3
         2.188  decoder_ps2DataInt.F3

decoder_presState<3>
   decoder_presState<2>.XQ
         2.837  decoder_presState<0>.F4
         2.809  decoder_ps2DataInt.F2
         2.809  decoder_presState<2>.G4

decoder_presState<4>
   decoder_presState<4>.XQ
         2.264  count.F4
         1.083  decoder_presState<4>.F2
         1.089  decoder_presState<4>.G2
         2.782  decoder_ps2DataInt.G1

decoder_presState<5>
   decoder_presState<4>.YQ
         3.567  syn193.F4

         3.112  decoder_presState<6>.F1
         3.654  syn560.F4
         3.968  syn558.F4
         1.771  decoder_presState<4>.G3

decoder_presState<6>
   decoder_presState<6>.YQ
         1.872  syn683.F4
         1.872  decoder_presState<6>.F4

         1.872  decoder_presState<6>.G4

decoder_presState<7>
   decoder_presState<6>.XQ
         1.834  syn683.F2
         1.573  decoder_presState<0>.F3
         0.609  decoder_presState<6>.G2

decoder_ps2DataInt
   decoder_ps2DataInt.XQ
         2.662  ps2data.T

edgeDetect_presstate
   edgeDetect_presstate.XQ
         1.785  count.G1

ps2sync
   count.YQ
         3.390  syn558.C2
         3.208  syn560.G4
```

```
        3.191  syn560.C4
        0.997  count.C3
        2.443  decoder_presState<0>.F1
        2.443  decoder_presState<0>.G1
        3.328  decoder_presState<2>.F2
        3.328  decoder_presState<2>.G2
        2.443  decoder_presState<4>.F1
        2.443  decoder_presState<4>.G1
        2.389  decoder_presState<6>.F3
        2.389  decoder_presState<6>.G3
        2.215  decoder_ps2DataInt.G2

syn186
   syn186.X
        1.796  count.F3

syn193
   syn193.X
        3.478  count.F1

syn558
   syn558.Y
        1.824  syn683.F3

syn560
   syn560.X

        3.478  syn683.F1

syn570
   syn560.Y
        2.436  decoder_presState<2>.F1
        2.436  decoder_presState<2>.G1
        2.847  decoder_presState<6>.F2
        2.951  decoder_presState<4>.G4

syn683
   syn683.X
        2.079  decoder_ps2DataInt.C4
```

Place and Route Reports

```
PAR: Xilinx Place And Route C.16.
Copyright (c) 1995-1999 Xilinx, Inc.  All rights reserved.

Xilinx PAD Specification File
*****************************

Input file:       map.ncd
Output file:      toplevel.ncd
Part type:        xc4002xl
Speed grade:      -3
Package:          pc84

Pinout by Pin Name:
```

Pin Name	Direction	Pin Number
clk	INPUT	P13
ps2clk	INPUT	P18
ps2data	BIDIR	P84
rstn	INPUT	P47
Dedicated or Special Pin Name		**Pin Number**
/PROGRAM		P55
CCLK		P73
DONE		P53
GND		P21
GND		P64
GND		P76
GND		P52
GND		P43
GND		P12
GND		P31
GND		P1
TDO		P75
VCC		P2
VCC		P74
VCC		P63
VCC		P54
VCC		P11
VCC		P42
VCC		P22
VCC		P33

Pinout by Pin Number:

Pin Number	Pin Name	Direction	Constraint
P1	(GND)		
P2	(VCC)		
P3	---	UNUSED	
P4	---	UNUSED	
P5	---	UNUSED	
P6	---	UNUSED	
P7	---	UNUSED	
P8	---	UNUSED	
P9	---	UNUSED	
P10	---	UNUSED	
P11	(VCC)		
P12	(GND)		
P13	clk	INPUT	

P14	---	UNUSED		
P15	---	UNUSED		
P16	---	UNUSED		
P17	---	UNUSED		
P18	ps2clk	INPUT		
P19	---	UNUSED		
P20	---	UNUSED		
P21	(GND)			
P22	(VCC)			
P23	---	UNUSED		
P24	---	UNUSED		
P25	---	UNUSED		
P26	---	UNUSED		
P27	---	UNUSED		
P28	---	UNUSED		
P29	---	UNUSED		
P31	(GND)			
P33	(VCC)			
P35	---	UNUSED		
P36	---	UNUSED		
P37	---	UNUSED		
P38	---	UNUSED		
P39	---	UNUSED		
P40	---	UNUSED		
P41	---	UNUSED		
P42	(VCC)			
P43	(GND)			
P44	---	UNUSED		
P45	---	UNUSED		
P46	---	UNUSED		
P47	rstn	INPUT		
P48	---	UNUSED		
P49	---	UNUSED		
P50	---	UNUSED		
P51	---	UNUSED		
P52	(GND)			
P53	(DONE)			
P54	(VCC)			
P55	(/PROGRAM)			
P56	---	UNUSED		
P57	---	UNUSED		
P58	---	UNUSED		
P59	---	UNUSED		
P60	---	UNUSED		
P61	---	UNUSED		
P62	---	UNUSED		
P63	(VCC)			
P64	(GND)			
P65	---	UNUSED		
P66	---	UNUSED		
P67	---	UNUSED		
P68	---	UNUSED		
P69	---	UNUSED		
P70	---	UNUSED		
P71	---	UNUSED		
P72	---	UNUSED		
P73	(CCLK)			
P74	(VCC)			
P75	(TDO)			
P76	(GND)			
P77	---	UNUSED		
P78	---	UNUSED		
P79	---	UNUSED		
P80	---	UNUSED		
P81	---	UNUSED		
P82	---	UNUSED		

```
| P83          | ---                                 | UNUSED   |            |
| P84          | ps2data                             | BIDIR    |            |
+--------------+-------------------------------------+----------+------------+

#
# To preserve the pinout above for future design iterations,
# simply run "Lock Pins..." from the Design Manager's Design
# menu, or invoke PIN2UCF from the command line. The location constraints
# above will be written into your specified UCF file. (The constraints
# listed below are in PCF format and cannot be directly used in the UCF file).
#
COMP "clk" LOCATE = SITE "P13" ;
COMP "ps2clk" LOCATE = SITE "P18" ;
COMP "ps2data" LOCATE = SITE "P84" ;
COMP "rstn" LOCATE = SITE "P47" ;
#

--------------------------------------------------------------------------------


PAR: Xilinx Place And Route C.16.
Copyright (c) 1995-1999 Xilinx, Inc.  All rights reserved.


par -w -ol 2 -d 0 map.ncd toplevel.ncd toplevel.pcf


Constraints file: toplevel.pcf

Loading device database for application par from file "map.ncd".
   "toplevel" is an NCD, version 2.28, device xc4002xl, package pc84, speed -3
Loading device for application par from file '4002xl.nph' in environment
C:/Fndtn.
Device speed data version:  C 1.1.2.2 PRELIMINARY.


Device utilization summary:

   Number of External IOBs           3 out of 61      4%
      Flops:                         0
      Latches:                       0
   Number of Global Buffer IOBs      1 out of 8      12%
      Flops:                         0
      Latches:                       0

   Number of CLBs                   14 out of 64     21%
      Total Latches:                 0 out of 128     0%
      Total CLB Flops:              15 out of 128    11%
      4 input LUTs:                 23 out of 128    17%
      3 input LUTs:                  5 out of 64      7%

   Number of BUFGLSs                 1 out of 8      12%
   Number of STARTUPs                1 out of 1     100%



Overall effort level (-ol):   2 (set by user)
Placer effort level (-pl):    2 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    2 (set by user)

Starting initial Placement phase.  REAL time: 3 secs
Finished initial Placement phase.  REAL time: 3 secs

Starting Constructive Placer.  REAL time: 3 secs
```

```
Placer score = 5190
Placer score = 3960
Finished Constructive Placer.  REAL time: 3 secs

Writing design to file "toplevel.ncd".

Starting Optimizing Placer.  REAL time: 3 secs
Optimizing
Swapped 38 comps.
Xilinx Placer [1]   2070   REAL time: 3 secs

Finished Optimizing Placer.  REAL time: 3 secs

Writing design to file "toplevel.ncd".

Total REAL time to Placer completion: 3 secs
Total CPU time to Placer completion: 3 secs

0 connection(s) routed; 90 unrouted.
Starting router resource preassignment
Completed router resource preassignment. REAL time: 3 secs
Starting iterative routing.
Routing active signals.
End of iteration 1
90 successful; 0 unrouted; (0) REAL time: 3 secs
Constraints are met.
Routing PWR/GND nets.
Power and ground nets completely routed.
Writing design to file "toplevel.ncd".
Starting cleanup
Improving routing.
End of cleanup iteration 1
90 successful; 0 unrouted; (0) REAL time: 3 secs

Writing design to file "toplevel.ncd".
Total REAL time: 4 secs
Total CPU  time: 4 secs
End of route.  90 routed (100.00%); 0 unrouted.
No errors found.
Completely routed.

This design was run without timing constraints.  It is likely that much better
circuit performance can be obtained by trying either or both of the following:

  - Enabling the Delay Based Cleanup router pass, if not already enabled
  - Supplying timing constraints in the input design


Total REAL time to Router completion: 4 secs
Total CPU time to Router completion: 4 secs

Generating PAR statistics.

   The Delay Summary Report

   The Score for this design is: 281


The Number of signals not completely routed for this design is: 0

   The Average Connection Delay for this design is:        2.153 ns
   The Maximum Pin Delay is:                               4.129 ns
   The Average Connection Delay on the 10 Worst Nets is:   3.297 ns
```

```
   Listing Pin Delays by value: (ns)

    d < 1.00   < d < 2.00  < d < 3.00  < d < 4.00  < d < 5.00  d >= 5.00
   ---------   ---------   ---------   ---------   ---------   ---------
          10          32          33          14           1           0

Writing design to file "toplevel.ncd".


All signals are completely routed.

Total REAL time to PAR completion: 4 secs
Total CPU time to PAR completion: 4 secs

PAR done.
```

Appendix C

Appendix C: **Source Code**

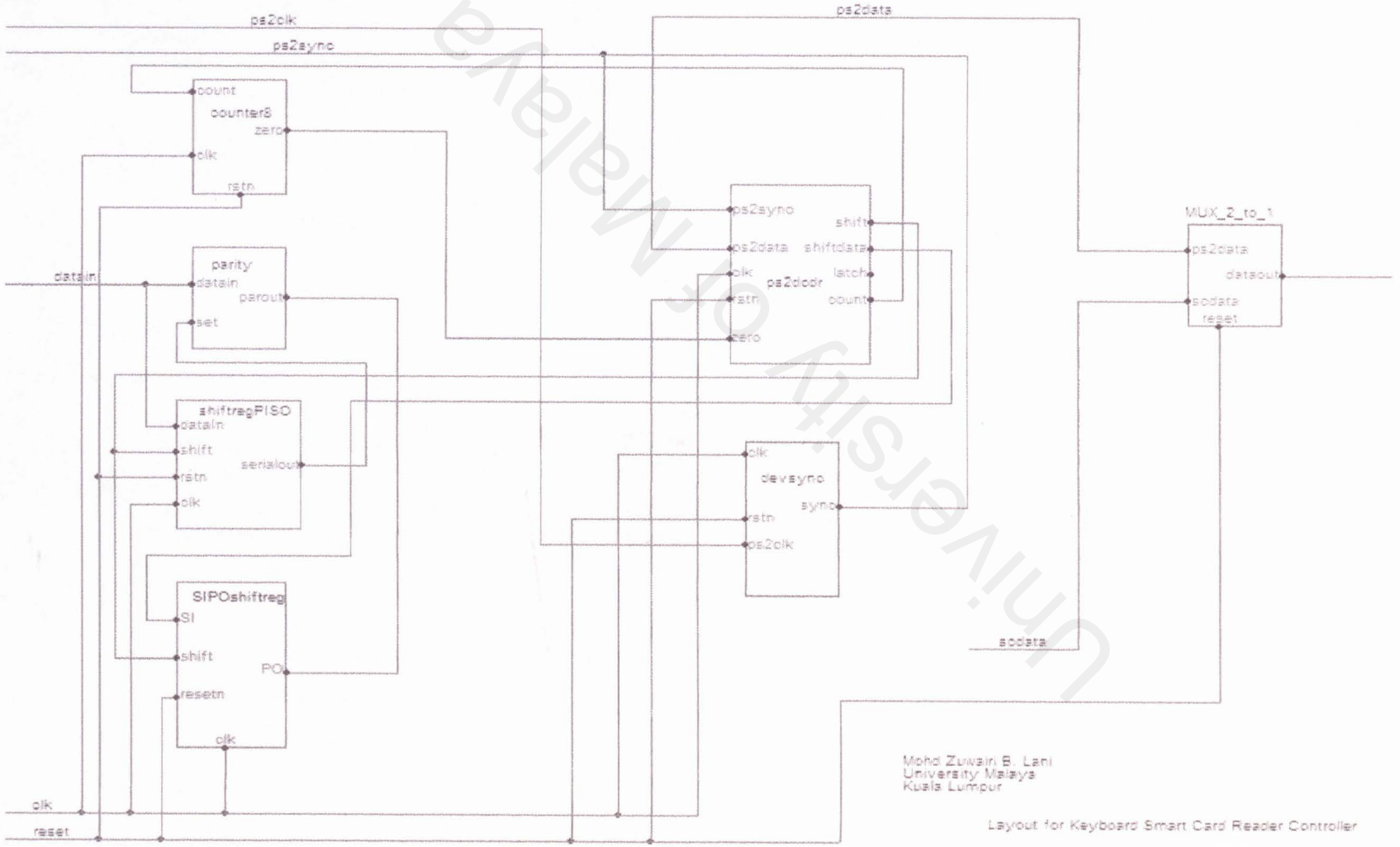
Description: Code for the ps/2 keyboard top level

```
library IEEE;
use IEEE.std_logic_1164.all;

entity ps2keyboard is
    port (
        clk: in STD_LOGIC;
        rstn: in STD_LOGIC;
        ps2data: inout STD_LOGIC;
        ps2clk: inout STD_LOGIC
    );
end ps2keyboard;

architecture ps2keyboard_arch of ps2keyboard is

SIGNAL shiftData: STD_LOGIC;
SIGNAL shift: STD_LOGIC;
SIGNAL count: STD_LOGIC;
SIGNAL zero: STD_LOGIC;
SIGNAL latch: STD_LOGIC;
SIGNAL ps2sync: STD_LOGIC;
SIGNAL valid: STD_LOGIC;
SIGNAL read: STD_LOGIC;
SIGNAL latchedData: STD_LOGIC_VECTOR (7 downto 0);
SIGNAL unlatchedData: STD_LOGIC_VECTOR (7 downto 0);

component SIPOshiftreg
    port (
        resetn: in STD_LOGIC;
        SI: in STD_LOGIC;
        PO: out STD_LOGIC_VECTOR (7 downto 0);
        Shift: in STD_LOGIC;
        clk: in STD_LOGIC
    );
end component;

component counter8
    port (
        count: in STD_LOGIC;
        clk: in STD_LOGIC;
        rstn: in STD_LOGIC;
        zero: out STD_LOGIC
    );
end component;

component devsync
    port (
            clk: in STD_LOGIC;
```

```
            rstn: in STD_LOGIC;
        ps2clk: in STD_LOGIC;
        sync: out STD_LOGIC
    );
end component;

component ps2dcdr
    port (
        ps2sync: in STD_LOGIC;
        ps2data: in STD_LOGIC;
        clk: in STD_LOGIC;
        rstn: in STD_LOGIC;
        zero: in STD_LOGIC;
        shift: out STD_LOGIC;
        shiftData: out STD_LOGIC;
        latch: out STD_LOGIC;
        count: out STD_LOGIC
    );
end component;

begin

      shiftReg: SIPOshiftreg PORT MAP(
        resetn => rstn,
        SI => shiftData,
        PO => unlatchedData,
        Shift => shift,
        clk => clk
    );

      counter: counter8 PORT MAP(
        count => count,
        clk => clk,
        rstn => rstn,
        zero => zero
    );

      edgeDetect: devsync PORT MAP(
      clk => clk,
      rstn => rstn,
        ps2clk => ps2clk,
        sync => ps2sync
    );

      decoder: ps2dcdr PORT MAP(
      ps2sync => ps2sync,
        ps2data => ps2data,
        clk => clk,
        rstn => rstn,
        zero => zero,
        shift => shift,
        shiftData => shiftData,
        latch => latch,
```

```
        count => count
    );

end ps2keyboard_arch;
```

Appendix D



References

References

Books

1. ***Digital Design – Principles and Practices***

   Wakerly, John F., 2000 Prentice Hall, Inc

2. ***A VHDL Primer*** – Revised edition

   Bhasker, J., 1995 Prentice Hall, Inc

3. ***The Designer's Guide to VHDL***

   Ashenden, Peter J., 1996 Morgan Kaufmann Publishers, Inc

4. ***Logic and Computer Design Fundamentals***

   Mano, M. Morris and Kime, Charles R., 2000 Prentice Hall, Inc

5. ***Computer Architecture and Organization: Designing For Performance***

   Stallings, Williams, 5th Edition, Prentice Hall, Inc

Websites

1) http://www.smartcardbasic.com

2) http://www.eduard-rhein-foundation.de/html/t96.html

3) http://www.smartcard.co.uk/finance1.html.

4) http://library.cs.tuiasi.ro/hardware/smart-card-developer-kit/ch03/www.visa.com

5) http://library.cs.tuiasi.ro/hardware/smart-card-developer-kit/ch03/www.opengroup.org

6) http://www.interesting-devices.com/smart.htm