**NAME :** MARINA HARYATI MOHAMAD

**MATRIC NO :** WEK020116

**TITLE :** SMART TRANSDUCER INTERFACE MODULE
(MAIN STATE MACHINE VHDL)

**SUPERVISOR :** EN. NOORZAILY MOHAMMED NOOR

**MODERATOR :** EN. MOHD YAMANI IDNA IDRIS

# ABSTRACT

This project is about the development of Smart Transducer Interface Module in hardware. The IEEE1451.2 smart sensor approach specifies a 'plug and play' capability in a transducer module, which is achieved through transducer electronic data sheet (TEDS). It specifies a digital interface to access this data sheet, read sensor and set actuator. A write and logic function to access TEDS and transducer are defined. This STIM will be implemented using VHSIC Hardware Description Language (VHDL), Peak FPGA software. This report will comprise the STIM phase from the design phase of main state machine until the end of testing phase.

## ACKNOWLEDGEMENT

I would like to take this opportunity to express my gratitude to all those who had helped me throughout this project. First and foremost, I would like to thanksgiving to Allah for his grace and mercy in completing this project. My sincere thank to En. Noorzaily as my Supervisor for all his guidance, advise and was kind enough to help me at times. Also thank to En. Yamani Idris whose always gives suggestion and comment during the duration of this project.

Last but not least, my grateful thank to my friends that also involved in this STIM project, Ana, Ain and Yoges. I would like to thank both my parents En. Mohamad and Pn Masithah for continue support and prayers. All of them always encourage me and made me believe myself even though sometimes I'm low in confidence done this project.

Thank you.

# TABLE OF CONTENT

# CHAPTER 1

# CHAPTER 1 : INTRODUCTION

## 1.0 INTRODUCTION

Transducer serve a wide variety of industry's needs, manufacturing, industrial control, building automotive and biomedicine are but a few. Since the transducer market is very diverse, transducer manufactures are seeking ways to build low-cost and large-scale production. Since then, research has been done by family of IEEE to investigate low cost of data interface for low cost sensor. Family of IEEE 1451 have proposed a standard that VHDL implementation of STIM is suitable for compact solution allowing cost reduction and enhanced product functionality. All these are significant contributes to productivity improvement and will benefit producers, vendor, system integrates and user.

Smart sensor that can extract more information from surrounding environment since it has computational capability. This sensor is equipped with some elaboration unit that treats incoming signals operating, for instances, an analog to digital conversion and a calibration producer. A key characteristic of smart sensor is that it operates on the input signal in a logical fashion to increase the value of the information that it process. The sensor is capable of making logical decisions as the source of the information. Other than that, smart sensors include their capabilities for self-test, adaptive calibration and ease of setup and use.

## 1.1 OBJECTIVE OF PROJECT

There are several objectives to be achieved in completing this project.

- To improve the STIM performances that already exist, by target a low cost of smart sensor with integrated sensor. This can be achieved by using VHDL approach. VHDL is a hardware language that created to describe the behaviors of systems and electronic digital circuit.
- To have a fully understanding of STIM implementation, fundamental and architecture. Also understand of how each module communicate each other.
- To design, coding and simulate the main state machine of STIM using VHDL application software that is Peak FPGA Design Suite by having a good command of VHDL at the end of this project.

## 1.2 SCOPE OF PROJECT

The research will mainly concentrate on the main state machine that manages all the primary operation of the STIM. Since, the main state machine accomplishes more function to implement the STIM, so this project only coordinating data transport and trigger. Hence, due this limitation, some of the features maybe can't fully implement the STIM. This STIM only describe two TEDS, Meta TEDS and Channel TEDS. Also used only two channel, sensor and actuator.

Beside that, it is design the STIM in terms of drawing necessary diagram and developing the necessary sub modules, data transport and trigger using VHDL source code.

## 1.3 PROJECT CONSTRAINT

The STIM cannot be implementing all the features available in the standard of STIM. This is because the complexity of certain features. This STIM only apply two channels than seven channels in the standard and only utilize Meta TEDS and Channel TEDS compare with the standard, there are eight TEDS exist.

There is also the time constraint on developing the source code and simulation of the design and whether the simulation and coding of the modules can be completed within the duration due to the lack of experience

## 1.4 PROBLEM DEFINITION

The main problem that has been detected during the early stages:

**1. Defining the scope**

Standard design of main state machine manages all the features needed. So it is difficult to identify the fundamental of each data transport and trigger because

each of sub modules interconnected with each other.

**2. Design the block diagram**

Block diagram for each module and sub modules are needed to present on Chapter 4.

**3. Developing source code**

It is quite hard to learn VHDL in order to build and implement main state machine of STIM.

## 1.5 PROBLEM SOLVING

1. To define the scope, we have to really understand the fundamental and implementation of STIM. By review the standard and literature related to STIM the function that used in STIM can be identify. All the relevant source code based on the function.
2. To design a complete block diagram, more time is needed. We have to consider all the process and action involved and also consider the input and output for each sub modules. Also frequent meeting with the Supervisor to discuss the design.
3. To develop a good command of source code, more time is needed to learn VHDL language again and again and try to meeting with the Supervisor.

## 1.6 PROJECT LAYOUT

This project consists of seven chapters. The purpose of this layout is to give an overview of phases involved during the project development. The summary of each chapter presented as follows:

**CHAPTER 1:INTRODUCTION**

This chapter gives an overview of major phase of the project that has to be done, the objective, project scope and project schedule.

**CHAPTER 2: LITERATURE REVIEW**

A brief explanation on topic researches that relevant to this project. It covers about STIM, explanation what is STIM, standard of STIM, how itworks and others.

**CHAPTER 3: METHADOLOGY**

Emphasize on the justification for the project methodology, what kind of life cycle and programming language to be used in STIM implementation.

**CHAPTER 4: SYSTEM ANALYSIS**

A brief explanation on the scope of the project, that is data transport and triggering function, description of each sub modules. VHDL plays an important role in the completion of this project. A brief explanation of VHDL programming language will be presented and discussed here also.

**CHAPTER 5: SYSTEM DESIGN**

This chapter focused on the conceptual and technical design of main state STIM and each module. It covers data flow diagram and process for each module.

**CHAPTER 6: SYSTEM IMPLEMENTATION AND TESTING**

This chapter focused on the design implementation and the coding process involved transforming of the design into the programming language. Also discuss about testing and result achieved from the test to assure that the implementation of modules works and to find any error or fault during implementation.

**CHAPTER 7: SYSTEM EVALUATION**

This chapter will touch various areas like conclusion of project, suggestion for future enhancement, problem encountered during development process and others.

## 1.7 PROJECT SCHEDULE

The Gantt chart below shows the activities of each phase that will be carried out through the development of the system. This is important because each phase must be done on time and careful planning will make it possible. It will take approximately time of nine months to finish the whole project.

## GHANTT CHART

| NO | PHASE | JUN 04 | JUL 04 | AUG 04 | SEP 04 | OCT 04 | NOV 04 | DEC 04 | JAN 05 | FEB 05 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | REQUIREMENT ANALYSIS | | | | | | | | | |
| 2 | SYSTEM ANALYSIS | | | | | | | | | |
| 3 | SYSTEM DESIGN | | | | | | | | | |
| 4 | IMPLEMENTATION | | | | | | | | | |
| 5 | SYSTEM TESTING | | | | | | | | | |
| 6 | DOCUMENTATION | | | | | | | | | |

# CHAPTER 2

# CHAPTER 2 : LITERITURE REVIEW

## 2.0 A BRIEF EXPLANATION OF SENSOR

### 2.0.1 Introduction

Just about everything today in the technology area is a candidate for having the prefix smart added to it. The term smart sensor was coined in the mid 1980s, and since then several devices have been called smart sensors. The intelligent required by such devices are available from microcontroller unit (MCU), digital signal processor (DSP) and application specific integrated circuit (ASIC). Today microelectronic technology is applied to sensor. Before availability of microelectronic, the sensor or transducer used to measure physical quantities such as temperature, pressure and flow usually will couple directly to readout device, typically a meter that was read by an observer. The transducer converted the physical quantity being measured to a displacement. The observer initiated system correction to change the reading closer to desired value. A sensor STIM can be used to take measurements of any type with the appropriate analog sensors, such as pressure, temperature, air flow, volume, and digital input sensors like switches.

Commonly, the definition of transducer is a device that converts energy from one domain into another, calibrated to minimize the errors in the conversion

process. A sensor is a device that provides useful output to a specified measurand. The sensor is the basic element of a transducer but it may refer to a detection of voltage in the electrical regime that does not required conversion.

An Institute of Electrical and Electronic Engineers (IEEE) committee has beenactively consolidating terminology that applies to microelectronic sensors. The recently approved IEEE1451.2 specification defines a smart sensor as a sensor that provides function beyond those necessary for generating a correct of a sensed or controlled quantity. This function simplifies the integration of the transducer into application in a network environment.

### 2.0.2 What a sensor

A suite of technologies underlie the rise of sensors, including MEMS, piezo-materials, micromachines, very large scale integration (VLSI) video, and a handful of other technologies MEMS are by far the most important of the technologies enabling the rise of sensors in the near term. In concept, MEMS technology is simplicity itself: it amounts to nothing more than using semiconductor manufacturing techniques to create analog devices. MEMS research has been underway for over a decade, (2) and MEMS-based devices are already finding their way into the marketplace. The automobile industry is a major consumer of MEMS devices, and is likely to be the single largest early market.

Piezo-materials are materials (typically ceramics) that give off an electrical charge when deformed and, conversely, deform when in the presence of an electrical field. (3) Put a charge in, the material deforms; deform the material, it sends out a charge. Piezos are particularly useful as surface-mount sensors for measuring physical movement and stress in materials. But more importantly, piezos are useful not just for sensing, but for effecting -- manipulating the analog world

Micromachines are semiconductor cousins to MEMS technology. Like MEMS, Micromachines are built using semiconductor manufacturing techniques, but unlike MEMS, they are more complex in design, incorporating in some instances micrometer-scale gears and other moving parts.

### 2.0.3 Issues of analog sensor

Research have been done to investigate multi-sensor modules. In addition to the increased number of sensors used in current vehicles, one or more vehicle-wide data-communication networks are being used. These networks link various sensors to actuators and control centers and enable a variety of new automotive functions. penalty goes along with this conversion from a "dumb" analog sensor to a "smart" sensor capable of vehicle-wide communications. This cost can exceed several dollars per sensor.

Examination of sensor function and placement shows that a variety of sensors lend themselves to being placed in localized clusters. These multi-sensor modules can share a single set of communication chips, thereby lowering the overall cost associated with the conversion to a smart sensor network. Packaging is one of the more expensive pieces of sensor manufacturing. By using a single housing, sensor cost is also reduced by eliminating multiple connectors and cables. Reducing the number of connectors, integrated circuits, and cables not only saves costs but also improves reliability and the assembly process and reduces vehicle weight.

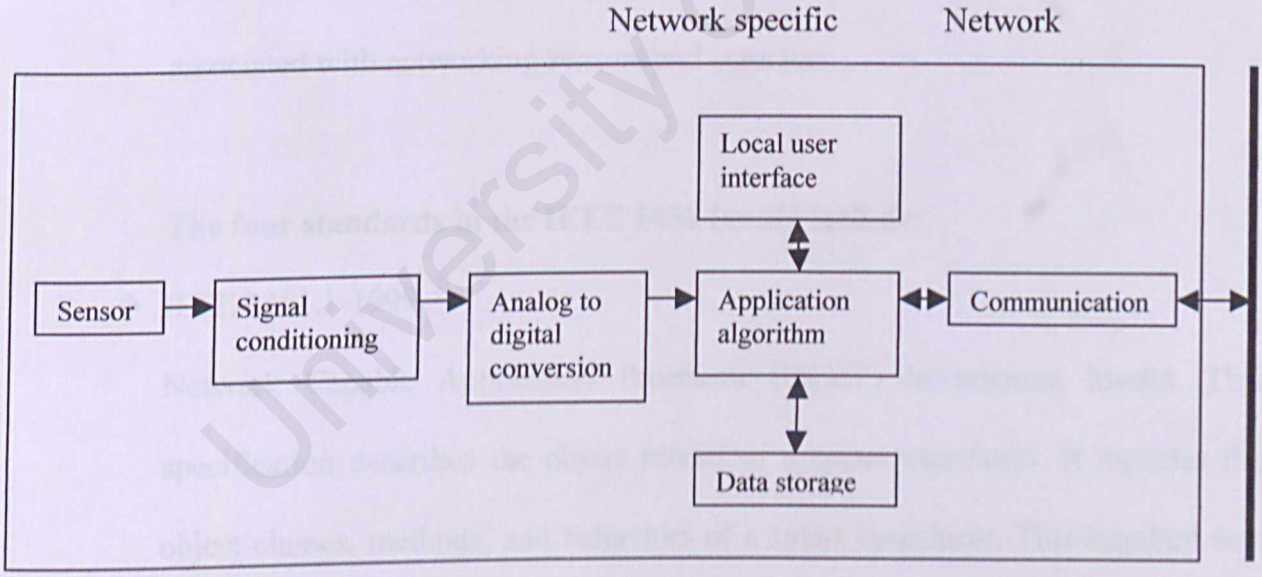



Figure 2.0 A general model of smart sensor

## 2.1 The 1451 Family of Transducer Interfaces

Transducers, defined here as sensors or actuators, serve a wide variety of industry's needs, manufacturing, industrial control, automotive, aerospace, building, and biomedicine are but a few. Since the transducer market is very diverse, transducer manufacturers are seeking ways to build low-cost, networked smart transducers. Accepting the limitations created by the diverse technologies associated with sensors, actuators, networks and processors, the IEEE 1451 family of sensors can be viewed as the cumulative efforts of several experts highly experienced in the area of networking sensors. With this approach, the IEEE 1451 series becomes a reference set containing valuable guidelines, practical rules, and insights into technical problems and possible solutions associated with networking sensors and actuators.

**The four standards in the IEEE 1451 family include:**

- IEEE 1451.1-1999

  Network Capable Application Processor (NCAP) Information Model. This specification describes the object model of a smart transducer. It includes the object classes, methods, and behaviors of a smart transducer. This standard was approved recently.

- IEEE 1451.2-1997

  Standard for a Smart Transducer Interface for Sensors and Actuators: Transducer

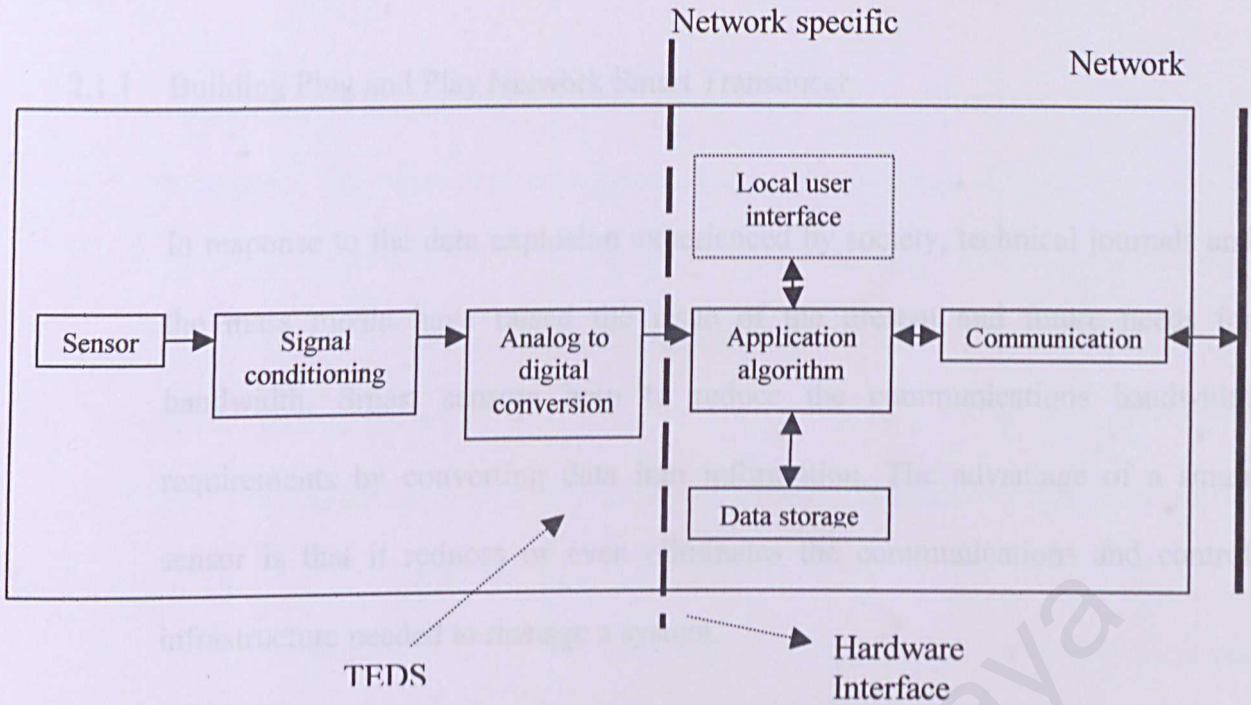
to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. This specification describes the hardware interface and communication protocols between the smart transducer interface module (STIM) and the NCAP. It also includes a definition of the transducer electronic data sheet (TEDS), which allows users to store information about transducer characteristics in the sensor itself.

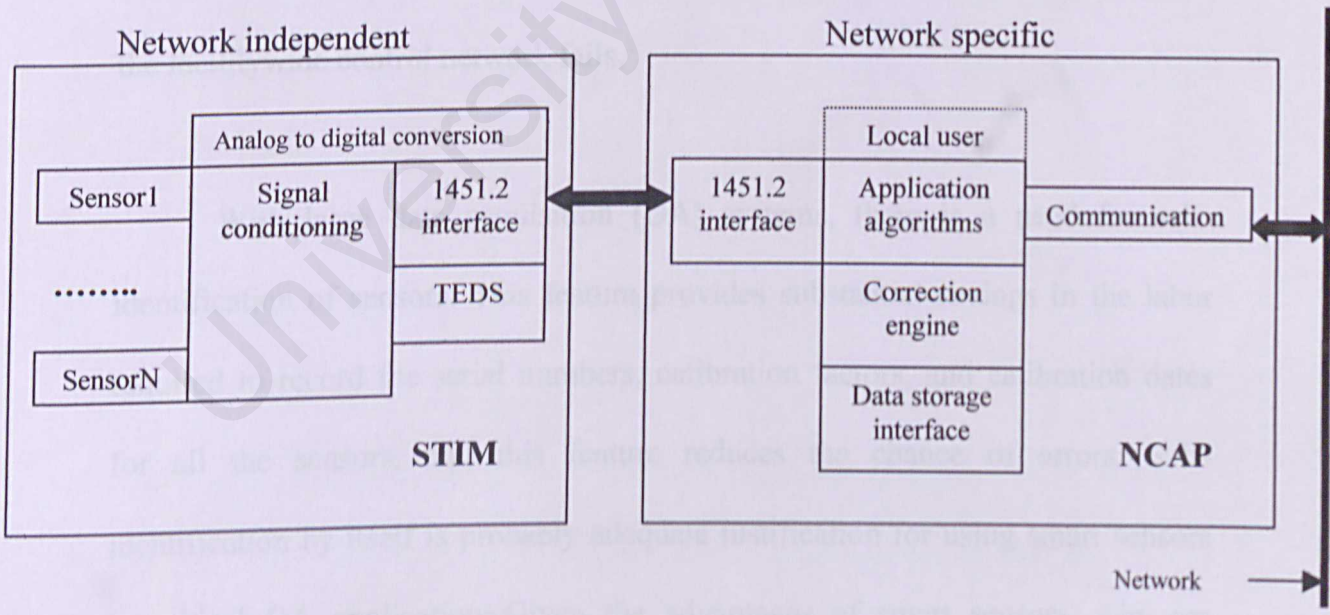
- IEEE P1451.3

  Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems. This standard is still in the approval phase.

- IEEE-P1451.4

  Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. This specification describes how some sensors may use some of the digital features of 1451, such as electronic data sheets, while still allowing analog outputs. This specification is important for applications that require high data-acquisition rates, such as automotive impact testing. This standard is still in the approval phase.



### 2.1.1 Building Plug and Play Network Smart Transducer

In response to the data explosion experienced by society, technical journals and the mass media have raised the issue of the present and future needs for bandwidth. Smart sensors help to reduce the communications bandwidth requirements by converting data into information. The advantage of a smart sensor is that it reduces or even eliminates the communications and control infrastructure needed to manage a system.

In the case of building or factory controls, the systems are large, and the sensor infrastructure is expensive. Smart sensors not only provide significant savings by reducing cabling and monitoring equipment costs, they also provide enhanced reliability and safety because the sensor maintains local control even if the facilitywide control network fails.

With large data acquisition (DA) systems, there is a need for self-identification of sensors. This feature provides substantial savings in the labor required to record the serial numbers, calibration factors, and calibration dates for all the sensors, and this feature reduces the chance of errors. Self-identification by itself is probably adequate justification for using smart sensors in critical DA applications.Given the advantages of smart sensors, why are network-compatible smart sensors not in wider use? One reason is the

fragmented nature of the fieldbus market and the unwillingness or inability of transducer manufacturers to support all the networks now in use. Many sensor network or fieldbus implementations are available, each with its own strengths and weaknesses for a specific class of applications. Interfacing transducers to all these control networks and supporting the wide variety of communications protocols represents a significant and costly effort to transducer manufacturers.

Today, there is no common digital communication interface standard between transducers and network communications nodes; each transducer manufacturer defines and builds its own interface. Consequently, transducer manufacturers cannot afford to support all the control networks for which their products are suited. An open, universally accepted transducer interface standard would facilitate the development of network-capable smart sensors and actuators and should result in faster acceptance and implementation of smart sensors into the market.

### 2.1.2 Issues of Sensor Industry

Issues addressed will include the status of IEEE 1451.3 and 1451.4, wireless sensing using Bluetooth technology, adding intelligence to sensors, smart interfacing, and a variety of smart sensing applications. Since the industry appears to be focusing its future efforts on Ethernet networks, there may no

longer be a need for compatibility with several types of networks and, therefore, no need for several types of NCAPS. The issue now becomes whether make smart sensors that integrate much of the STIM functionality into the sensor product and then connect these smart sensors to someone else's NCAP product.

Sensor Synergy's strategy is to combine the NCAP and STIM into an adaptable smart transducer interface and cost-effectively provide the most useful features of IEEE 1451.2 to end-users and transducer manufacturers. .This approach addresses the reality of an Ethernet-dominated network environment without requiring sensor manufacturers to rework their product offerings by tightly integrating microelectronics intelligence into their sensors.

IEEE 1451.3 defines a digital interface for connecting multiple, physically separated sensors. The multi-drop transducer bus standard is a minibus implementation that is sufficiently small and inexpensive to integrate into a transducer. IEEE 1451.4 addresses analog sensors with respect to their existing wiring and the requirement for wide bandwidth analog measurements. IEEE 1451.4 will allow analog-output, mixed-mode transducers to communicate digital information with a high-level IEEE 1451 object. To fit into the digital network defined by other 1451 standards, the bi-directional digital communication of self-identification, test, and programmable signal conditioning functionality is being defined with an eye toward simplicity and low cost.

In other applications that do not involve multiple protocols, alternative ways of communicating smart-sensor information are preferable, where, Crossbow Technology, Inc. is using the TEDS from 1451.2 in wireless sensor networking solutions based on the Bluetooth™ 2.45 GHz frequency hopping spread spectrum standard. Their CrossNET™ architecture is an integrated hardware/software solution for implementing wireless sensor networks. The solution facilitates the use of sensors in applications that, for example, are difficult or possible to wire or are subjected to harsh conditions.

The CrossNet node incorporates a Bluetooth radio for wireless communication with a computer or handheld device, and can control up to four sensors. Smart I/O cables connect sensors to the CrossNet node and provide self-identifying circuitry using the TEDS. Using a Bluetooth radio, a PC can communicate wirelessly with CrossNet nodes, extending data acquisition and control capabilities to multiple users connected via a network. For users requiring a Bluetooth interface to their personal computer, Crossbow supplies a USB to Bluetooth radio connection. The Bluetooth interface is connected to a personal computer via the USB port, up to a distance of 10 meters.

At Crossbow, we believe that software, combined with the right digital architecture, will be the key to smarter sensing solutions This approach can bring even older mechanical and analog sensors into the digital world by turning them

into smart sensors. Coupling this with wireless technology really bridges the gap between the analog, physical world and the new computer/Interment infrastructure."

## 2.2 SMART TRANSDUCER INTERFACE MODULE (STIM)

### 2.2.1 What is a STIM?

A Smart Transducer Interface Module (STIM) is a module that contains TEDS, logic to implement the transducer interface, the transducer(s) and any signal conversion or signal conditioning. It consists of 1 to 255 sensors or actuators or any combination of them, DAC, ADC and Digital I/O to interface the sensors or actuators. It also consists of conversion circuitry, address logic, and digital electronics or microprocessors to convert the sensor readings into digital form, or to convert digital output to manipulate actuators to and from the network. A STIM is controlled by a Network Capable Application Processor (NCAP) module by means of a dedicated digital interface. This interface is not a network. The NCAP mediates between the STIM and a digital network, and may provide local intelligence. Through the NCAP sensor data is passed to a network. When a STIM contains more than one transducer, it may be referred to as a multichannel STIM or a multi variable STIM.

A transducer channel is denoted "smart" in this context because of the following three features:

- ❖ It is described by a machine-readable, Transducer Electronic Data Sheet (TEDS).
- ❖ The control and data associated with the channel are digital.
- ❖ Triggering, status, and control are provided to support the proper functioning of the channel.

When power is applied to the STIM, the information that it carries in the TEDS is made available to the NCAP for local usage, and for distribution to the rest of the network as necessary. Once the TEDS is read, the NCAP knows how fast it can communicate with the STIM, how many channels the STIM has, and the data format of each channel. It can then send information to the STIM, or ask the sensor to perform a reading or get information about readings from the sensor.

### 2.2.2 Transducer channel types

There six channel types in this STIM. An additional seventh channel type is identified to allow for extensions to STIM behavior. Each channel type has their owns detailed timing and control. The seven channel types are as follows:

- ❖ **Sensor**

A sensor measures some physical parameter on demand and returns digital data representing that parameter. A new data set shall be sampled once for each

triggering event. The data set available to be read shall be the data set acquired as a result of the most recent trigger event.

- **Actuator**

  An actuator causes a physical or virtual action to occur that shall be related to the data set sent to the actuator. The actuator state changes to match the data set most recently sent to it when a triggering event occurs.

- **Buffered sensor**

  A buffered sensor has a single level of data buffering on the output channel. A new data set shall be sampled once for each triggering event. The data set available to be read shall be the data set acquired on the second most recent trigger event.

- **Data sequence sensor**

  A data sequence sensor acquires data continuously, with sampling times under control of the STIM. The data set selected shall be the one acquired immediately following the trigger.

- **Buffered data sequence sensor**

  A buffered data sequence sensor acquires data continuously, with sampling times under control of the STIM. The data set selected shall be the one acquired immediately previous to the trigger.

- **Event sequence sensor**

  An event sequence sensor produces a signal whenever a specific event occurs. The signal shall be the same signal used by sensors and actuators to acknowledge triggering events.

### 2.2.3 Types of TEDS

- **Meta TEDS** provide global information to the client about the STIM unit attached to the NCAP and includes the information necessary to access data in any of the channel TEDS.
- **Channel TEDS** contains information specific to one of the transducers connected to the STIM; each sensor and each actuator connected to the STIM must have its own Channel TEDS
- **Calibration TEDS** an optional TEDS, provides a data calibration capability using a standardized mathematical correction algorithm
- **Meta ID TEDS** an optional TEDS with at most one of these TEDS per STIM unit, provides information about the STIM
- **Channel ID TEDS** another optional TEDS with at most one of these TEDS for each Channel TEDS, provides information about the transducers associated with the specified channel.
- **Calibration ID TEDS** provides a human-readable description of any information deemed relevant to the calibration of each channel
- **End User Application Specific TEDS** is provided as a place to store any additional human-readable data which is not recovered by the specific TEDS
- **Industry Extensions TEDS**

## 2.3 SENSING THE FUTURE

The impact of sensors will be as surprising in the decade ahead as microprocessors were in the 1980s and lasers in the 1990s. And the surprises will be additive because of new interaction among existing generations of technology, with some of the most interesting applications of sensing technology applied to dealing with current problems. It's still early, but in the future, society and business will be saturating the world with communications and information. The future is not going to be people talking to people; it's not going to be people accessing information. It's going to be about using machines to talk to other machines on behalf of people. That's where the growth is going to be. It's also why all of our assumptions about available bandwidth and how much bandwidth we need is wrong by orders of magnitude, once all of these machines start talking to each other. We're talking about washing machines, cars, bank machines, appliances of all kinds, oatmeal in boxes tagged with sensors, not talking machine like computer.

### 2.3.1 Endless Possibilities

RFID (Radio Frequency Identification Devices) is an ID chip, a computer on a chip that you can embed in, say, a box of detergent, and the detergent now can tell what it is, where it's from and where it went once it left the store. Sure, its

obvious application is factory inventory control, but you can use it for all sorts of things. the market for RFID is vast today.

Today, when you're thinking of machines talking to machines, people think about the personal computer or the mobile phone. But that's not even the start of it. Machines talking to machines are all about little devices inside everyday things that we won't even see and won't even know exist. Imagine a home burglary system that's wireless. Each little burglar alarm, each little burglar sensor has its own processor with its own Web page.

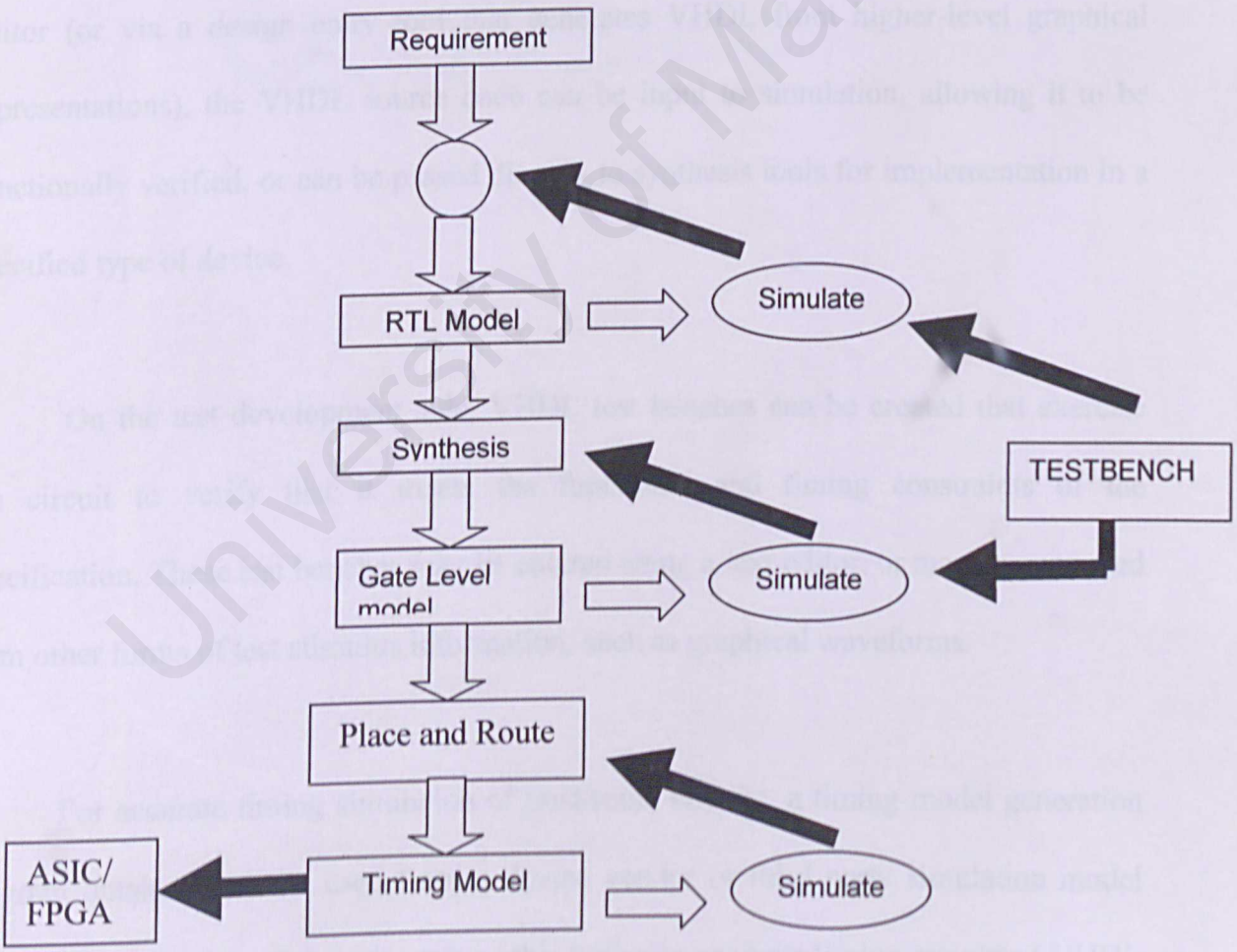
This is another possibilities, that washer machine turns out that its got Internet connectivity but it also has a little wireless transponder in it. And it wakes up and it listens for a radio signal, and it picks up this little signal from the 802.11 box on the side of your house that you already use for wireless Ethernet distribution for your computer. And washing machine starts a conversation with computer and Ethernet box. Electrolux is already doing it. They wanted to do this because there's this huge unserved market of young people who would love to have a washing machine in their house They going to give them a washing machine, and we will charge them by the load. The machine will be in their house, but the title will remain with us. It was basically having the convenience of a machine at home, and yet you as the consumer just paid a monthly bill.

# CHAPTER 3

# CHAPTER 3: METHADOLOGY

A system development methodology is a collection of procedure techniques, tools and documentation which help the developer to implement the product by translate the detail design into code. Therefore choosing the correct methodology for a project is very important because it will ensure the consistency and reproducibility to the product. This chapter illustrates the methodology that has been used in this project.

## 3.0 The FPGA/ASIC Design Process



The following diagram shows a simplified design process including both synthesis and simulation, assuming that the target of the process is one or more programmable logic or ASIC chips. The key to understanding this process, and to understanding how best to use VHDL, is to remember the importance of test development. Test development should begin as soon as the general requirements of the system are known.

VHDL (along with other forms of entry, such as schematics and block diagrams), will be used for design entry: after being captured into a design entry system using a text editor (or via a design entry tool that generates VHDL from higher-level graphical representations), the VHDL source code can be input to simulation, allowing it to be functionally verified, or can be passed directly to synthesis tools for implementation in a specified type of device.

On the test development side, VHDL test benches can be created that exercise the circuit to verify that it meets the functional and timing constraints of the specification. These test benches may be entered using a text editor, or may be generated from other forms of test stimulus information, such as graphical waveforms.

For accurate timing simulation of post-route circuits, a timing model generation program obtained will be used from a device vendor or third party simulation model supplier. Model generation tools such as this typically generate timing-annotated VHDL source files that support very accurate system-level simulation.

# CHAPTER 4

# CHAPTER 4: SYSTEM ANALYSIS

## 4.0 Introduction

In this chapter, it will include all the function involved within in this scope of project. STIM shall implement addressing, interface data transport and triggering. There will a brief explanation on each sub modules, data transport and triggering function. In this project there are two channel involved which are sensor and actuator. Each channel may implement two TEDS that are Meta TEDS and Channel TEDS function.

## 4.1 Addressing

Addressing is used in conjunction with the interface data transport. A full address specifies whether data is being read or written, to which function, and to which STIM channel Functional and channel addresses are logical addresses. The mapping of functional or channel addresses to physical addresses shall be accomplished within the STIM

- Address Structured

A full address shall be 2 bytes long and structured. For convenience, the most significant byte is called the functional address and the least significant byte is called the channel address

| Functional address<br>Most significant byte | | | | | | | | Channel address<br>Least significant byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| r/w | Function code | | | | | | | Channel number | | | | | | | |
| msb | | | | | | | lsb | msb | | | | | | | lsb |

Figure 4.1: Address layouts

❖ Functional Address

The msb of the functional address is used to specify the direction of data communication over the interface, according to table below. The remaining bits of the functional address represent the function selected.

Table 4.1: Direction bit values

| Value | Communication direction |
|---|---|
| 0 | Write to the STIM |
| 1 | Read from the STIM |

❖ Channel Address

Each transducer in a STIM shall be assigned a channel number. A STIM may have up to 255 channels. The number of implemented channels can be determined by reading the Meta-TEDS. Implemented channels shall be numbered consecutively starting from one. Every channel number between one and the highest implemented channel number shall address an implemented channel. Channel address zero has special meaning and is referred to throughout the standard as CHANNEL_ZERO. When CHANNEL_ZERO is used, the function shall refer to the STIM as a whole or

to all channels collectively. The word *global* or the prefix *meta-* is used to modify the functional address name. CHANNEL_ZERO will not include in this project because this project only consider two channel, sensor and actuator as mentioned early in this chapter.

❖ Function Selected

Table 4.2: The commonly used channel functional address

| Address | Function |
|---|---|
| 0 | Write channel transducer data |
| 1 | Write channel control command |
| 3 | Write triggered channel address |
| 5 | Write channel standard interrupt mask |
| 128 | Read channel transducer data |
| 130 | Read channel standard status |
| 160 | Read Meta TEDS |
| 161 | Read Channel TEDS |

## 4.2 Interface data transport

❖ Function

The data transport shall be supported by a group of signal lines in the physical interface. This service conveys addressing to the STIM and the data associated with the address between the STIM and the NCAP. The data transport role

interacts with the trigger function. The data transport function shall be inactivated before the trigger is asserted.

A data transport frame shall begin by the NCAP sending an address to the STIM. The complete address specifies whether data shall be written to or read from the STIM, and which channel and function are involved. Means shall be provided on the physical interface for the NCAP to signal when the data transport is active and to delimit data transport frames. It shall also be provided for the STIM to acknowledge its readiness for data transport.

Data transport is also used to read data sheet information such as the TEDS and to read status and write control commands to channels.

❖ Data Format

Data shall always consist of an integral number of bytes. When data transport involves multiple byte numeric representations, the most significant byte shall be sent first. For N-byte integer data representation that are not a multiple of 8 bits, pad bits shall be added above the most significant bit to bring the total to a multiple of eight. For N-byte fractional data representations that are not a multiple of 8 bits, pad bits shall be added below the least significant bit to bring the total to a multiple of eight. The physical interface shall transport each data byte in bit-serial form, most significant bit first.

❖ Data Transport Rate

Data rates shall be controlled by the NCAP. All STIMs shall support the common data flow rate. The rate may be changed to a higher rate based on information available in the TEDS. Means shall be provided for both the STIM and the NCAP to regulate the flow of data bytes within a frame. This is referred to as pacing.

❖ Transducer Data

Data transport is most frequently used to read data from sensor and to write data to actuator and event sequence channels. Writing data to a sensor shall have no effect. Reading from a sensor, without triggering shall return the same data as when last read. Reading data from any sensor after an aborted trigger cycle may produce unpredictable results.

Reading data from an actuator shall return the latest data written to it. Reading data from an actuator after initialization, but before writing data to it, shall return the initial state of the actuator.

### 4.2.1 Write Control Command

The control function allows commands to be sent to the STIM to each channel which affects their state or operation. It shall be accessed by writing to the functional address write channel control command for a specified channel.

Control commands shall be 1 bytes only. 4.2.2 explained a complete description of this bit.

Table 4.3: Standard control commands

| Value | Individual channel definition |
|---|---|
| 0 | No operation |
| 1 | Reset channel |
| 2 | Initiate channel self-test |
| 3 | Calibrate channel |
| 4 | Zero channel |
| 5 | Enable event sequence sensor |
| 6 | Disable event sequence sensor |
| 7 | Set event sequence sensor to the configuration mode |
| 8 | Reserved |
| 9 | Enable data sequence or buffered data sequence sensors |
| 10 | Disable data sequence or buffered data sequence sensors |
| 11-255 | Reserved |

### 4.2.2 Read Status Channel

The status function allows the NCAP to determine the state of the STIM of individual channels. Each bit in a specific status register represents the presence or absence of a particular condition. The presence of a condition shall be represented by a one in the appropriate bit position. The standard status register shall be accessed by reading from the functional address read channel standard

status for the channel. The status function is also used in conjunction with interrupts to indicate that the STIM is requesting service and for any purpose.

Table 4.4: Standard status bits

| Bit | Individual channel definition |
|---|---|
| msb | Open to industry |
| - | Open to industry |
| - | Open to industry |
| - | Open to industry |
| - | Reserved |
| - | Reserved |
| - | Reserved |
| - | Channel operational bit |
| - | Channel hardware error bit |
| - | Channel data /event bit |
| - | Channel missed data or event bit |
| - | Channel auxiliary status available bit |
| - | Reserved |
| - | Channel has been reset bit |
| - | Channel trigger acknowledged bit |
| lsb | Channel service request bit |

### 4.2.3 Read Data Sheet Information (TEDS)

The TEDS contains fields that fully describe the type, operation, and attributes of the transducer. There is a setting limit in the TEDS for all data transfer and triggering sequence between NCAP and STIM. It shall be accessed by reading

from the functional address read Meta-TEDS and read Channel TEDS for a specified channel.

Table 4.5: Data transfer timing parameter

| No field | Description | No of bytes |
|---|---|---|
| 19 | **STIM Handshake Timing**<br>Maximum time the STIM requires to negate acknowledge after the data transport end. | 4 |
| 20 | **End-Of-Frame Detection Latency**<br>Maximum time the STIM requires to detect the end of the a data transport. The STIM should be ready to start for the new transaction. | 4 |
| 21 | **TEDS Hold-Off Time**<br>Maximum time the STIM requires to acknowledge the transfer of a single byte. | 4 |
| 22 | **Operational Hold-Off Time**<br>Maximum time the STIM requires to acknowledge the data transfer addressed to operational function. | 4 |

Table 4.6 Triggering sequences timing parameter

| No field | Description | No of bytes |
|---|---|---|
| 19 | **STIM Handshake Timing**<br>Maximum time the STIM requires to negate acknowledge after the trigger sequence end. | 4 |
| 22 | **Channel Write Setup Time**<br>Maximum time required by the STIM after a write transaction but before a trigger. | 4 |
| 23 | **Channel Read Setup Time**<br>Minimum time required by STIM after a trigger but before a read transaction. | 4 |

## 4.3 Triggering

Signal lines in the physical interface shall support triggering. The triggering function provides means for an NCAP to send to a STIM a command for an action to take place (the trigger signal), and for the STIM to signal the time when the action occurred (trigger acknowledgment). Trigger acknowledges is also a response to the NCAP confirming that the action requested by the trigger did occur. Each transducer channel type differs from another chiefly in the way it responds to triggering. The timing of trigger acknowledges depends on the transducer type of the triggered channel. The actions of each transducer channel type in response to triggering are described in 4.3.1 and 4.3.2.

The triggered channel address specifies the channel to which the trigger applies. If the triggered channel address is within the range of implemented channels, then all triggering shall be directed at that channel alone.

Triggering shall only apply to a single channel either sensor or actuator. The channel to which the trigger applies is selected by the triggered channel address. The state diagram in Figure 4.1 illustrates the behavior of the triggering system from the point of view of the STIM. The action initiated by normal triggering belongs to a separate, channel-type-dependent, concurrent process. The timing specifications referred to in the transition from invalid to valid data is described further for individual channel types.

### 4.3.1 Trigger sensor

The trigger signal shall cause a sensor channel to acquire new data or a new data set. For the simplest sensors, an analog-to-digital converter begins a conversion. For a sensor channel, the STIM shall send a trigger acknowledgment coincident with the sample time. Subsequent to this trigger acknowledge and the additional duration specified by the Channel Read Setup Time, the data shall be available to the NCAP. Irrespective of the time needed to read the transducer data, the NCAP shall wait for at least the duration of the Channel Sampling Period between successive triggers. Figure 4.3 illustrate sensor activity concurrent with the quiescent and triggered state

Table 4.7: Sensor states

| State | Description |
|---|---|
| Acquire sample | The sample is being acquired. The action is complete when the sample acquisition is complete, irrespective of any further digitization. |
| Convert data | The STIM channel is digitizing the sample and moving it to the transducer data buffer. The STIM leaves this state when all conditions for valid data are met. The Channel Read Setup Time is the time spent in this state. |
| Valid data | Valid data from the triggered channel is available to be read. |

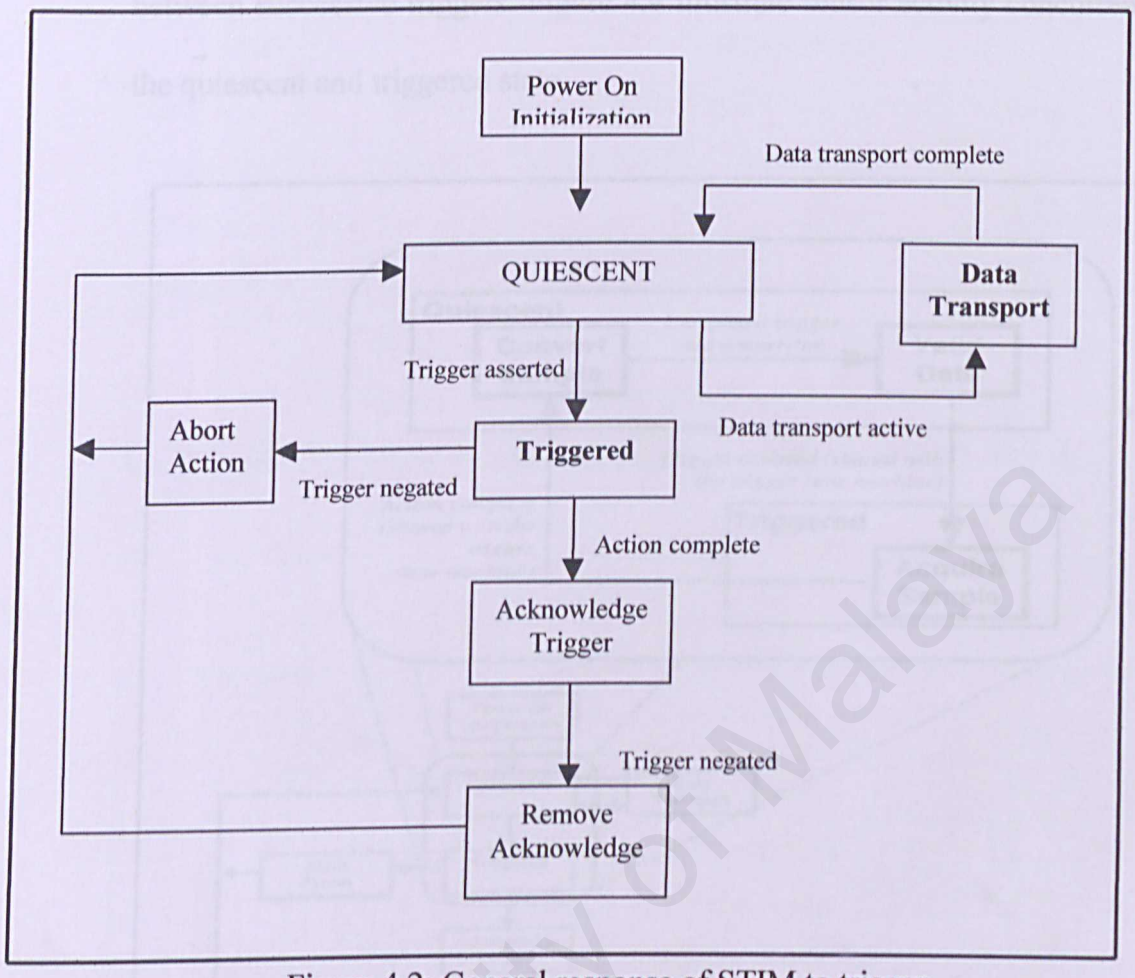



Figure 4.2: General response of STIM to trigger

### 4.3.2 Triggering actuators

The trigger signal shall cause an actuator channel to assume a new state or to step through a set of states. The data set associated with the new state shall have been written to the actuator channel previous to the trigger. Trigger shall not occur until the Channel Write Setup Time has passed following the time the new data is written. Irrespective of the time necessary to write new data to the actuator, the NCAP shall wait for at least the duration of the Channel Sampling Period

between successive triggers. Figure 4.4 illustrate sensor activity concurrent with the quiescent and triggered state.

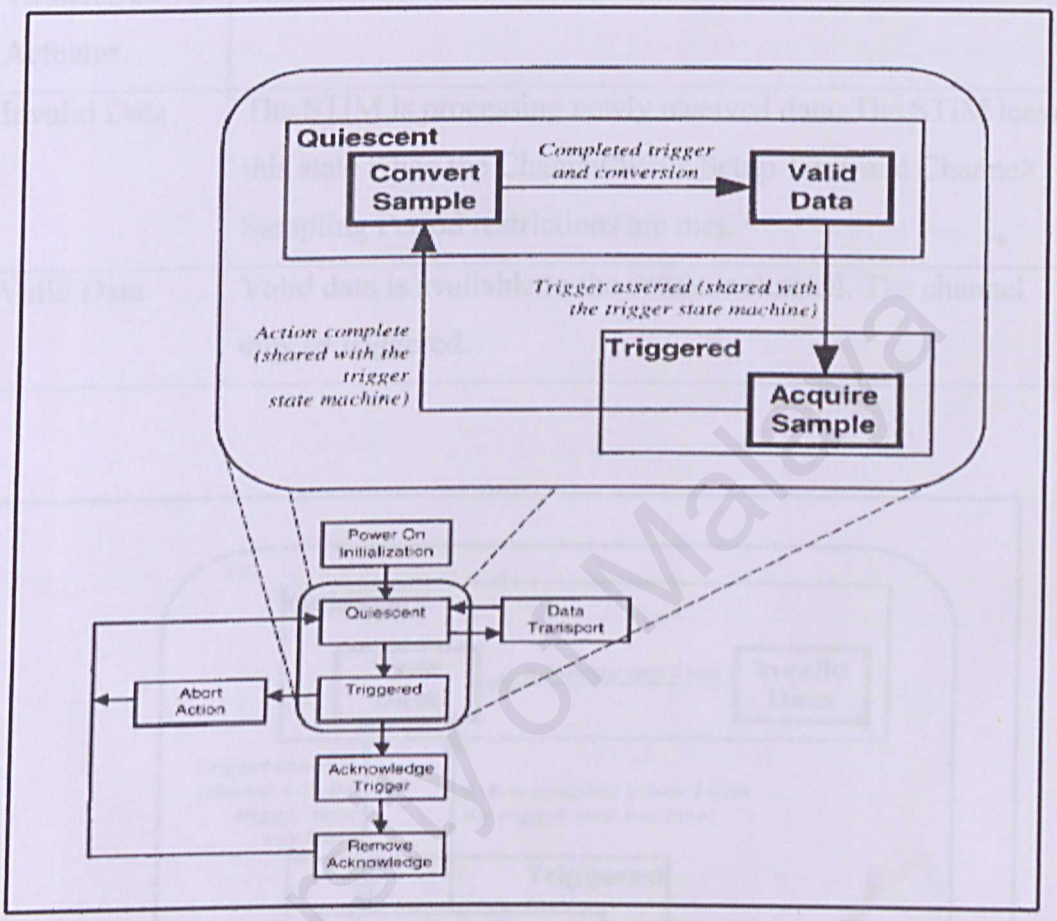



Figure 4.3: Sensor activity

Table 4.8: Actuator states

| State | Description |
|---|---|
| Data Transferred To Actuator | The last data written to the actuator channel is being transferred to the actuator output buffer. |
| Invalid Data | The STIM is processing newly received data. The STIM leaves this state when the Channel Write Setup Time and Channel Sampling Period restrictions are met. |
| Valid Data | Valid data is available to the actuator channel. The channel may be triggered. |

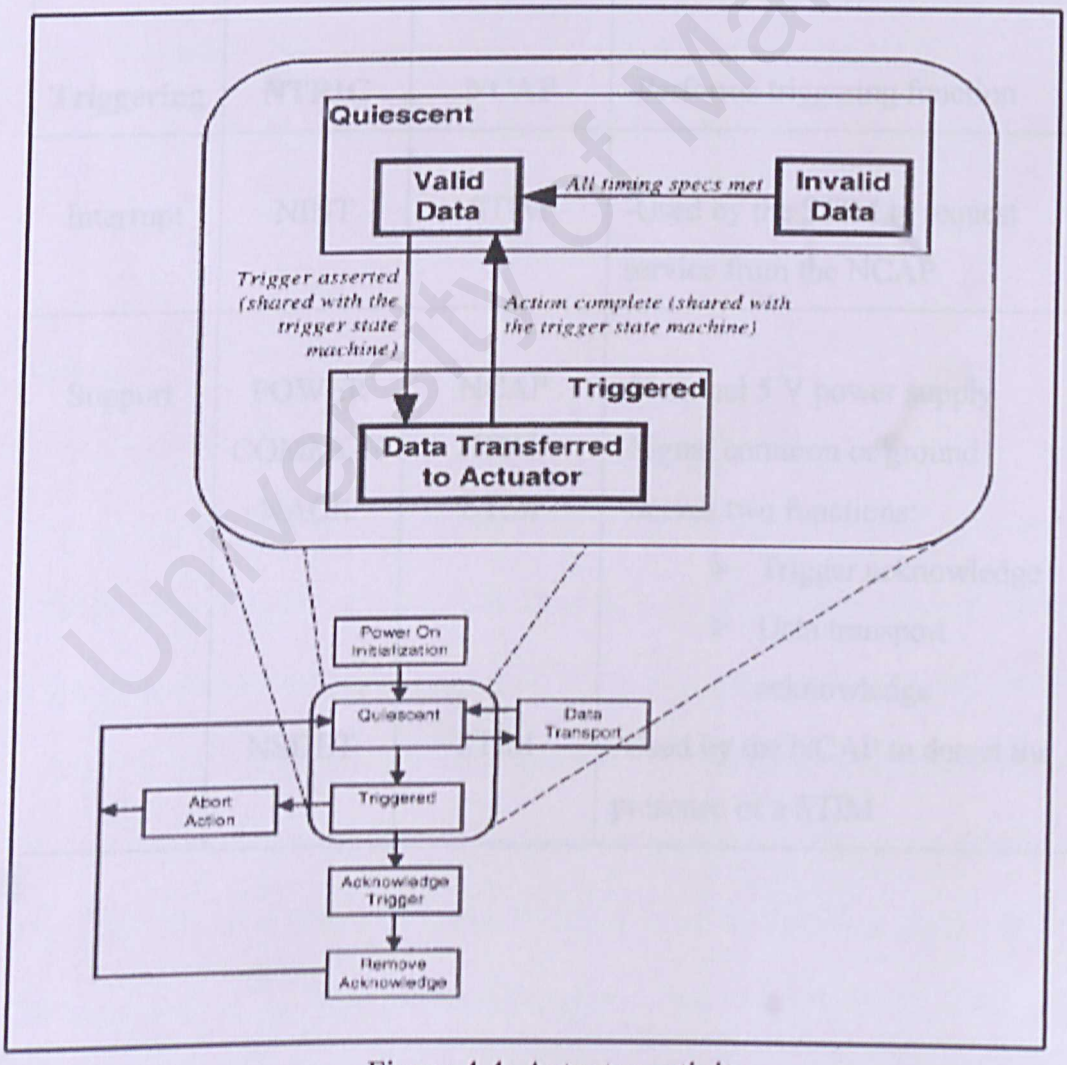



Figure 4.4: Actuator activity

## 4.4 Interface signal lines

| Group | Line | Driven by | Function |
|---|---|---|---|
| **Data** | **DOUT** | **STIM** | -Data transport from STIM to NCAP |
| | **DIN** | **NCAP** | -Address and data transport from NCAP to STIM. |
| | **DCLK** | **NCAP** | -Positive-going edge latches data on both DIN and DOUT. |
| | **NIOE** | **NCAP** | -Signals that the data transport is active and delimits data transport framing. |
| **Triggering** | **NTRIG** | **NCAP** | -Performs triggering function |
| Interrupt | NINT | STIM | -Used by the STIM to request service from the NCAP |
| Support | POWER | NCAP | -Nominal 5 V power supply |
| | COMMON | NCAP | -Signal common or ground |
| | NACK | STIM | -Serves two functions:<br>➢ Trigger acknowledge<br>➢ Data transport acknowledge |
| | NSDET | STIM | -Used by the NCAP to detect the presence of a STIM |

### 4.4.1 Bit Transfer

Data shall be transferred in bit-serial form from the NCAP to the STIM via DIN and from the STIM to the NCAP via DOUT. The transfer shall be controlled by the DCLK line.

- DCLK idles high
- On the first falling edge of DCLK, the first bit to be transferred is asserted by the sender (the NCAP on DIN and the STIM on DOUT).
- On the subsequent rising edge of DCLK, the bit is latched by the receiver (the STIM on DIN and the NCAP on DOUT).
- Subsequent bits are transferred by repetitions of steps one and two.

### 4.4.2 Timing

This timing is for the data transport and trigger signal. All NCAPS and all STIMS shall support a common data rate of 6000 bits/s. The STIM shall support a maximum data rate that is at least this fast; the NCAP shall support a data rate at least this slow. When an NCAP frst communicates with a STIM it shall use a data rate (fclk) less than or equal to 6000 bits/s. After the NCAP reads the Meta-TEDS, it may switch to a data rate less than or equal to the maximum data rate specified in the Meta-TEDSTiming of the data transport is complicated by the fact that the

NCAP is allowed to change the data rate. In order to ensure reliable data transport at the selected data rate, most data-transport-related timing parameters, such as setup and hold times, are based on one of the following parameters:

## 4.5 A Brief History Of VHDL

VHDL that stands for VHSIC Hardware Description Language was developed in the early 1980s as a spin-off of a high-speed integrated circuit research project funded by the U.S. Department of Defense. During the VHSIC program, researchers were confronted with the daunting task of describing circuits of enormous scale (for their time) and of managing very large circuit design problems that involved multiple teams of engineers. With only gate-level design tools available, it soon became clear that better, more structured design methods and tools would need to be developed.

To meet this challenge, a team of engineers from three companies -- IBM, Texas Instruments and Intermetrics -- were contracted by the Department of Defense to complete the specification and implementation of a new, language-based design description method. The first publicly available version of VHDL, version 7.2, was made available in 1985.

IEEE 1076-1987, is the basis for virtually every simulation and synthesis product sold today. An enhanced and updated version of the language, IEEE 1076-1993, was released in 1994, and VHDL tool vendors have been responding by adding these new language features to their products. IEEE 1076-1987 was adopted, simulator companies began enhancing VHDL with new signal types

(typically through the use of syntactically legal, but non-standard enumerated types) to allow their customers to accurately simulate complex electronic circuits.The IEEE 1076-1987 and IEEE 1164 standards together form the complete VHDL standard in widest use today. (IEEE 1076-1993 is slowly working its way into the VHDL mainstream, but does not add significant new features for synthesis users.)

## 4.6 What is VHDL?

VHDL is a programming language that has been designed and optimized for describing the behavior of digital circuits and systems. As such, VHDL combines features of the following:

- A Simulation Modeling Language
- A Design Entry Language
- A Test Language
- A Netlist Language
- A Standard Language

## 4.7 Why choose to use VHDL design?

VHDL (like a structured software design language) is most beneficial when use a structured, top-down approach to design. Real increases in productivity will come later, when you have climbed higher on the VHDL learning curve and have accumulated a library of reusable VHDL components.

Productivity increases will also occur when you begin to use VHDL to enhance communication between team members and when you take advantage of the more powerful tools for simulation and design verification that are available. In addition, VHDL allows you to design at a more abstract level. Instead of focusing on a gate-level implementation, you can address the behavioral function of the design.

VHDL increases can productivity. By making it easy to build and use libraries of commonly used VHDL modules. VHDL makes design reuse feel natural. As you discover the benefits of reusable code, you will soon find yourself thinking of ways to write your VHDL statements in ways that make them general purpose. Writing portable code will become an automatic reflex.

Another important reason to use VHDL is the rapid pace of development in electronic design automation (EDA) tools and in target technologies. Using a standard language such as VHDL can greatly improve your chances of moving

into more advanced tools (for example, from a basic low-cost simulator to a more advanced one) without having to re-enter your circuit descriptions. Your ability to retarget circuits to new types of device targets (for example, ASICs, FPGAs, and complex PLDs) will also be improved by using a standard design entry method.

## 4.8 Basic VHDL Terminology

- *Entity*

All designs are expressed in terms of entities. An entity is the most basic building block in a design. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will have lower-level descriptions contained in it. These lower-level descriptions will be lower-level entities contained in the top-level entity description.

```
entity fulladder is
        port (X: in bit;
              Y: in bit;
             Cin: in bit;
            Cout: out bit;
            Sum: out bit);
end fulladder;
```

- *Architecture.*

All entities that can be simulated have an architecture description. The architecture describes the behavior of the entity. A single entity can have multiple architectures. One architecture might be behavioral, while another might be a structural description of the design.

```
architecture concurrent of fulladder is
begin
        Sum <= X xor Y xor Cin;
        Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end concurrent;
```

- *Configuration.*

A configuration statement is used to bind a component instance to an entity-architecture pair. A configuration can be considered as a parts list for a design. It describes which behavior to use for each entity, much like a parts list describes which part to use for each part in the design.

```
configuration this_build of rcomp is
  for structure
    for COMP1: compare use entity work.compare(compare1);
      for ROT1: rotate use entity work.rotate(rotate1);
end for;
end this_build;
```

- *Package.*

A package is a collection of commonly used data types and subprograms used in the design. Think of a package as a toolbox that contains tools used to build

designs. If the package contains declarations of subprograms (functions or procedures) or defines one or more deferred constants (constants whose value is not immediately given), then a *package body* is required in addition to the package declaration. A package body (which is specified using the package body keyword combination) must have the same name as its corresponding package declaration, but it can be located anywhere in the design, in the same or a different source file.

- *Attribute.*

An attribute is data that is attached to VHDL objects or predefined data about VHDL objects. Examples are the current drive capability of a buffer or the maximum operating temperature of the device.

- *Generic.*

A generic is VHDL's term for a parameter that passes information to an entity. For instance, if an entity is a gate level model with a rise and a fall delay, values for the rise and fall delays could be passed into the entity with generics.

- *Process.*

A process is the basic unit of execution in VHDL. All operations that are performed in a simulation of a VHDL description are broken into single or multiple processes.

```
architecture behavior of dff is
begin
   process (Rst, Clk)
   begin
      if Rst = '1' then
         Q <= "00000000";
      elsif Clk = '1' and Clk'event then
         Q <= D;
      end if;
   end process;
end behavior;
```

## 4.9 Objects, Data Types and Operators

VHDL includes a number of language elements, collectively called objects, that can be used to represent and store data in the system being described. The three basic types of objects that you will use when entering a design description for synthesis or creating functional tests (in the form of a test bench) are signals, variables and constants. Each object that you declare has a specific data type (such as bit or integer) and a unique set of possible values.

The values that an object can take will depend on the definition of the type used for that object. For example, an object of type bit has only two possible values, '0' and '1', while an object of type real has many possible

values (floating point numbers within a precision and range defined by the VHDL standard and by the specific simulator you are using).When an explicit value is specified (such as when you are assigning a value to a signal or variable, or when you are passing a value as a parameter to a subprogram), that value is represented in the form of a literal.

| Data Type | Values |
|---|---|
| Bit | '1', '0' |
| Bit_vector | (array of bits) |
| Boolean | True, False |
| Integer | -2, -1, 0, 1, 2, 3, 4 . . . |
| Real | 1.0, -1.0E5 |
| Time | 1 ua, 7 ns, 100 ps |
| Character | 'a', 'b', '2, '$', etc. |
| String | (Array of characters) |

### 4.9.1 Using Signals

Signals are objects that are used to connect concurrent elements (such as components, processes and concurrent assignments), similar to the way that wires are used to connect components on a circuit board or in a schematic. Signals can be declared globally in an external package or locally within architecture, block or other declarative region.

### 4.9.2 Using Variables

Variables are objects used to store intermediate values between sequential VHDL statements. Variables are only allowed in processes, procedures and functions, and they are always local to those functions.

### 4.9.3 Using Constants and Literals

Constants are objects that are assigned a value once, when declared, and do not change their value during simulation. Constants are useful for creating more readable design descriptions, and they make it easier to change the design at a later time. Explicit data values that are assigned to objects or used within expressions are called *literals*. Literals represent specific values, but they do not always have an explicit type.

## 4.10 Types and Subtypes

Four classes of data types:

- **Scalar types** represent a single numeric value or, in the case of enumerated types, an enumeration value. The standard types that fall into this class are integer, real (floating point), physical, and enumerated types. All of these basic types can be thought of as numeric values.

- **Composite types** represent a collection of values. There are two classes of composite types: arrays containing elements of the same type, and records containing elements of different types.

- **Access types** provide references to objects in much the same way that pointer types are used to reference data in software programming languages.

- **File types** reference objects (typically disk files) that contain a sequence of Values.

## 4.11 VHDL Testbench and Verification

VHDL verification is performed using a set of modules and stimuli called VHDL testbench. The purpose of a testbench is to verify the functionality of a developed model or package. A testbench should be a distinct design unit separated from the model or package to be verified, placed in a design library separate from the model itself. The purpose of the verification is to verify that the developed model is correct, with few or no errors being found. It should not be a means to locate errors in the VHDL code in order to patch them. If the testbench incorporates models of components surrounding the model to be tested, they need only to incorporate functions and interfaces required to properly operate with the model under test; it is not necessary to develop complete VHDL models of them. If external stimuli or configuration data is required, it should be implemented by reading an ASCII file in order to ensure portability. The verification should be performed by someone not involved in the creation of that model or package, to avoid that the same misunderstanding in the verification masks a misunderstanding of the functionality.

## 4.12 VHDL Modeling

VHDL Modeling is performed in various details such as component model, board-level model, and system model. Every model is subject to verification. The main purpose of a model for component simulation is to be used for verification of a component under development, before proceeding with the manufacture. This implies that the model should exactly reflect the structure and functions of the underlying hardware; accuracy being more important than simulation speed. The model shall have correct timing characteristics, at least using estimated values for timing parameters. The main purpose of a model for board-level simulation is to be used for the verification of a board using the component, normally together with several other components. This can be seen as the simulation version of bread boarding. This implies that the model must have acceptable simulation speed, but only need to model the functionality possibly affecting the board and other models. The model should be on the Register Transfer level or higher. The model need necessarily not reflect the actual internal structure of the component

# CHAPTER 5

# 5.0 SYSTEM DESIGN

## 5.1 INTRODUCTION

In this chapter, each module is seen at its block diagram level and at its input and output. We will define the function for each sub module in this project scope. These functions are also in their own respective sub modules. Before we begin, here is a look at our general model of IEEE1451 smart sensor. We will see the design from the top down, then work it way up again for RTL level design. After that, we will see how it all works by looking at the main state machine block diagram with a detailed description for each sub module.

## 5.2 IEEE1451 SMART SENSOR

An overview of a STIM and how it associates through the TII to an NCAP is specifies. The physical connection between the NCAP and STIM is via the 10-pin Transducer Independent Interface (TII). Several dedicated lines have been added to provide power, ground and special purpose control lines. Each line will be driven either by STIM or NCAP.

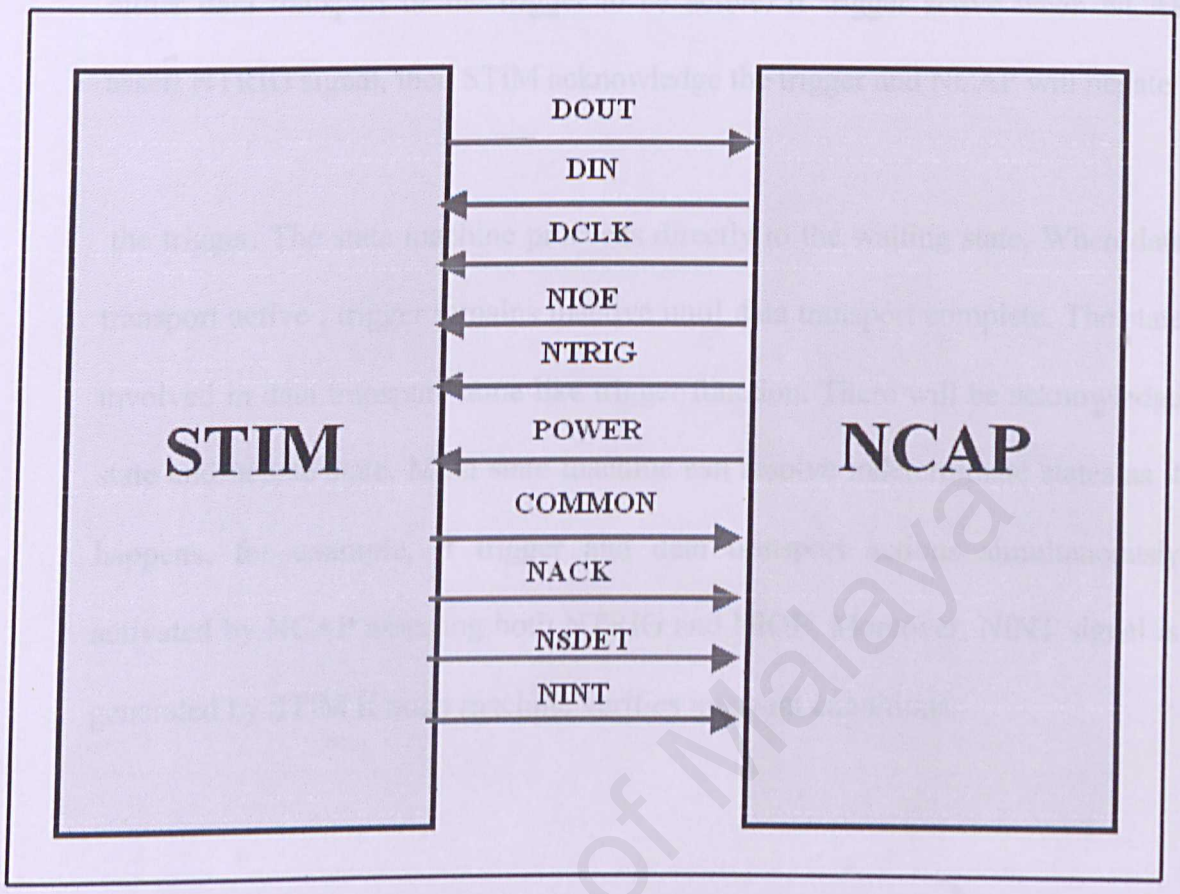



Figure 5.1 General model of IEEE1451 smart sensor

## 5.3 STATE MACHINE

Figure 5.2 represent the state machine of STIM, where there are data transport state and trigger state in this STIM. From the state machine we can see that data transport function and trigger function always interact each other. The trigger function is applies to the selected address channel either sensor or actuator. There will be a Quiescent or waiting state after start state, where STIM is waiting for

either data transport or the trigger to be active. If trigger active when NCAP assert NTRIG signal, then STIM acknowledge the trigger and NCAP will negate

the trigger. The state machine proceeds directly to the waiting state. When data transport active , trigger remains inactive until data transport complete. The state involved in data transport same like trigger function. There will be acknowledge state and negate state. Main state machine can resolve indeterminate states as it happens, for example, if trigger and data transport actions simultaneously activated by NCAP asserting both NTRIG and NIOE. Moreover, NINT signal is generated by STIM if main machine verifies interrupt conditions.

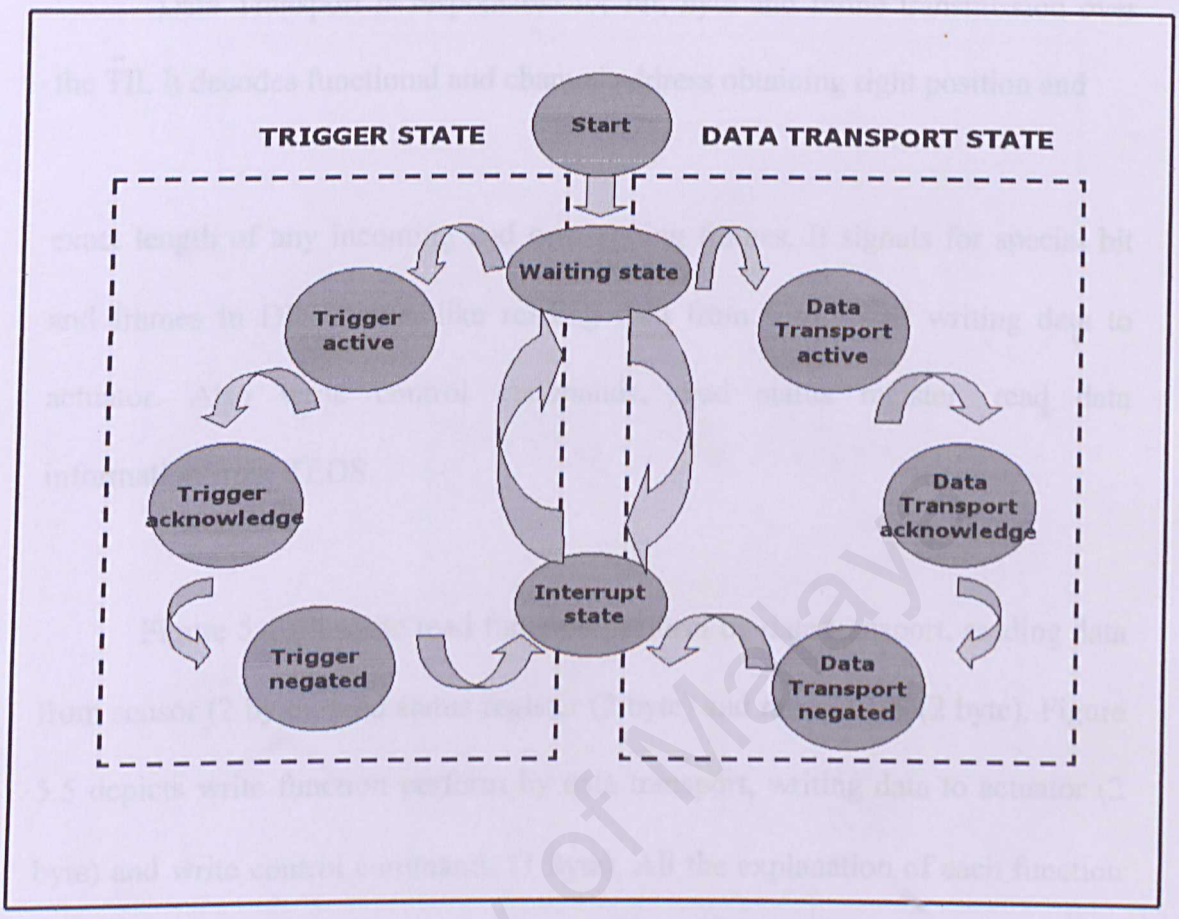



Figure 5.2: State machine of STIM

## 5.4 BLOCK DIAGRAM OF MAIN STATE MACHINE

Main state machine manages all the primary operation of the STIM coordinating both Data Transport and Trigger machines; it controls key signal of STIM (NIOE, NTRIG, NACK) to decide what and whenever an action must be performed, activating the corresponding STIM component. Figure 5.3 depicts block diagram state machine of STIM and all function involved.

Data Transport is responsible for bit, byte and frame transmission over the TII. It decodes functional and channel address obtaining right position and exact length of any incoming and or outgoing frames. It signals for special bit and frames in DIN signal, like reading data from sensor and writing data to actuator. Also write control commands, read status register, read data information from TEDS.

Figure 5.4 illustrate read function perform by data transport, reading data from sensor (2 byte), read status register (2 byte) and read TEDS (2 byte). Figure 5.5 depicts write function perform by data transport, writing data to actuator (2 byte) and write control commands (1 byte). All the explanation of each function has been describe in the previous chapter.

Smart sensor triggering is operates if a trigger signal for the channel (sensor or actuator) is applied, otherwise they are Quiescent. In other words, a sensor takes data only when triggered by the NCAP and an actuator updates its output only when triggered by the NCAP. Figure 5.6 depicts triggering function perform by trigger.

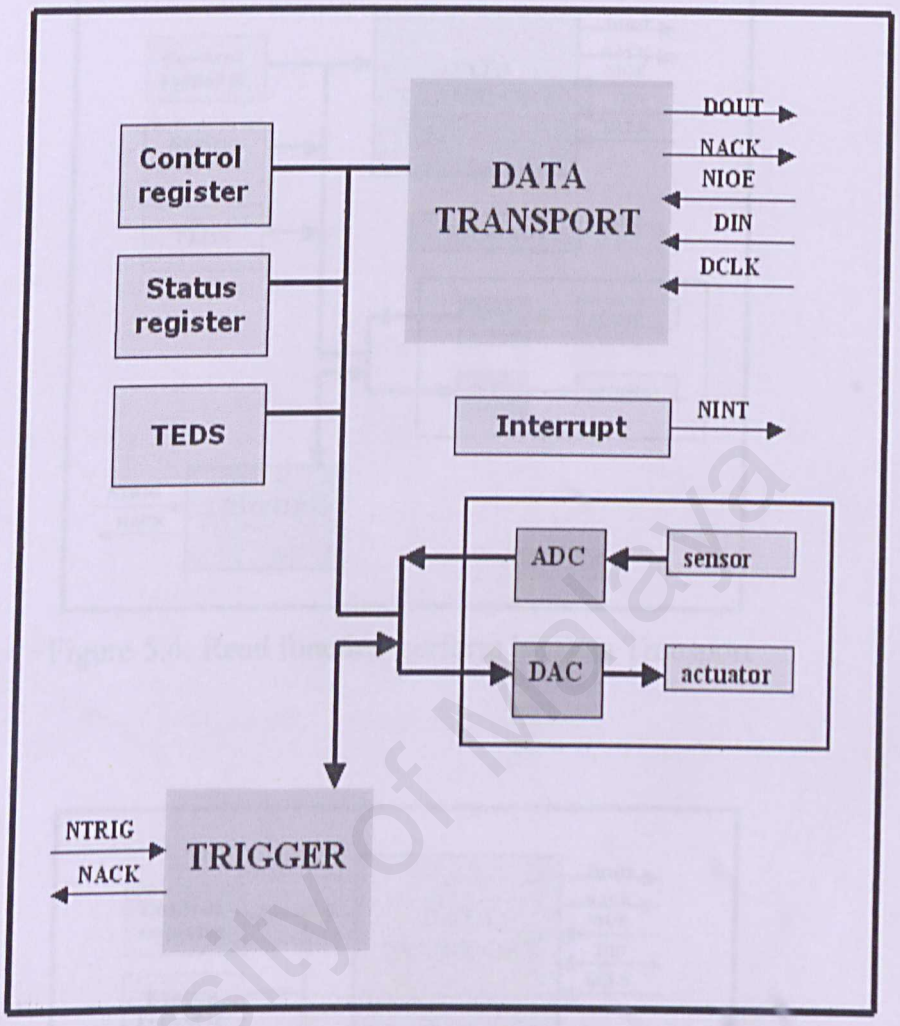



Figure 5.3: Block diagram state machine of STIM

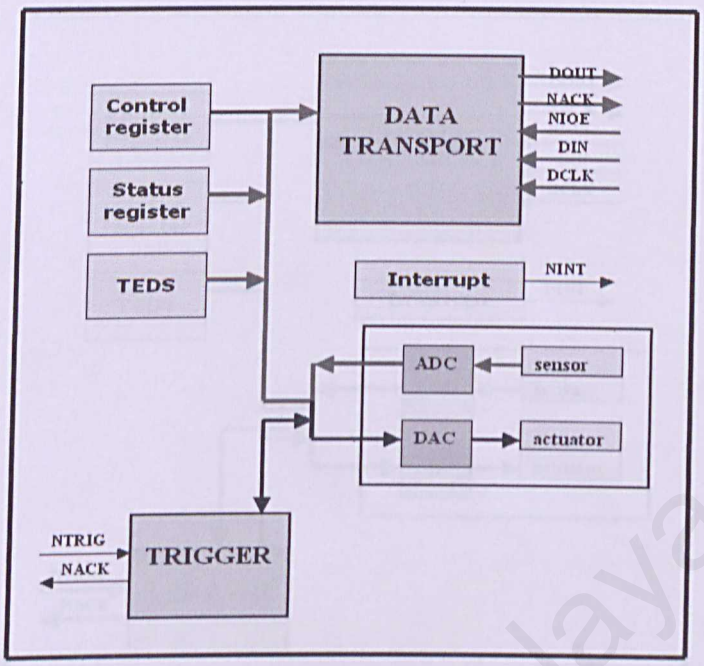



Figure 5.4: Read function perform by Data Transport

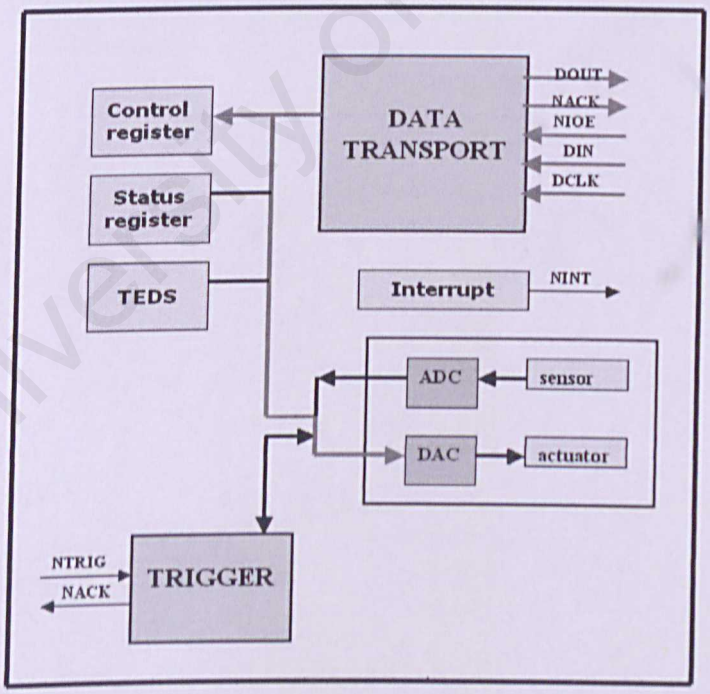



Figure 5.5: Write function perform by Data Transport

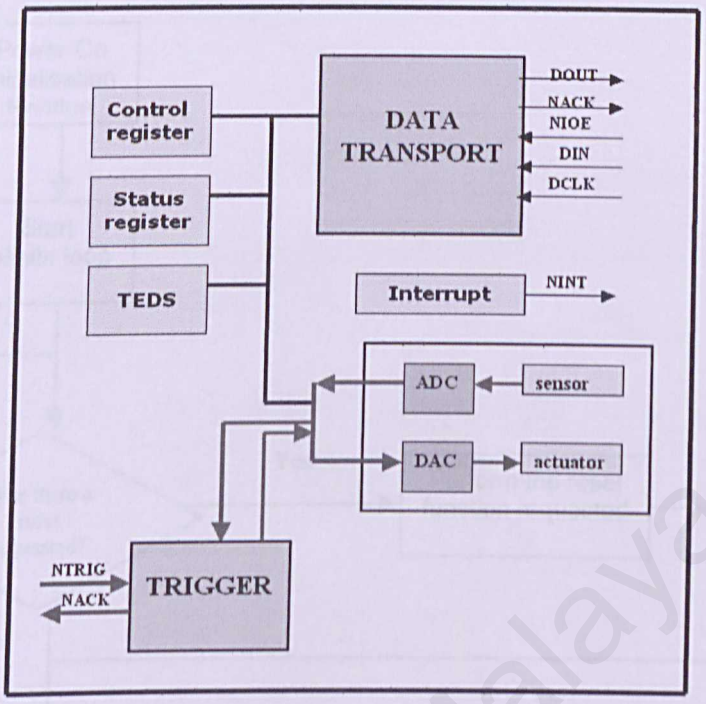



Figure 5.6: Triggering function

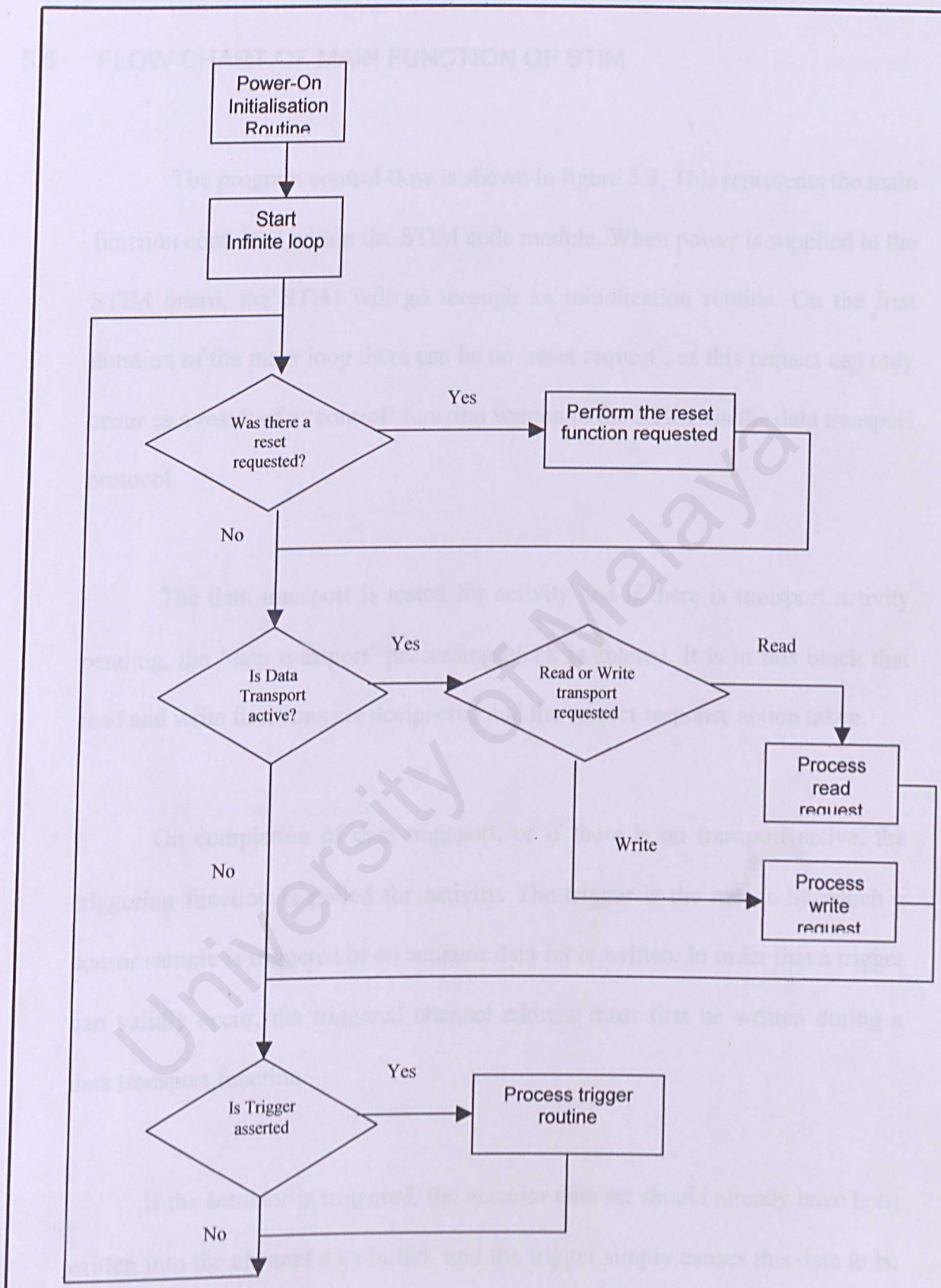



Figure 5.7 The main program control flow chart

## 5.5 FLOW CHART OF MAIN FUNCTION OF STIM

The program control flow is shown in figure 5.3. This represents the main function contained within the STIM code module. When power is supplied to the STIM board, the STIM will go through an initialization routine. On the first iteration of the main loop there can be no 'reset request', as this request can only occur as a result of a 'control' function written to the STIM via the data transport protocol.

The data transport is tested for activity and if there is transport activity pending, the 'data transport' processing block is entered. It is in this block that read and write functions are deciphered and the correct response action taken.

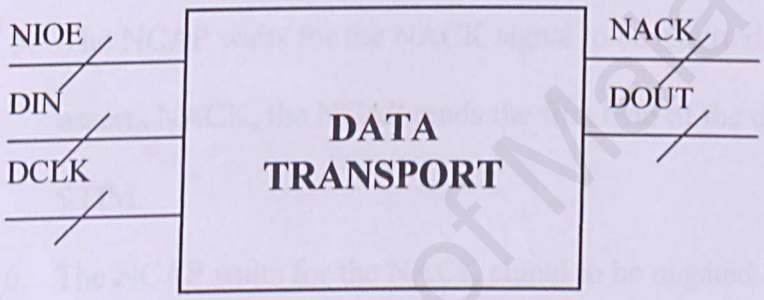
On completion of data transport, or if there is no transport active, the triggering function is polled for activity. The trigger is the means by which a sensor sample is triggered or an actuator data set is written. In order that a trigger can validly occur, the triggered channel address must first be written during a data transport function.

If the actuator is triggered, the actuator data set should already have been written into the channel data buffer, and the trigger simply causes this data to be sent to the actuator from that buffer. Conversely, if the sensor channel has been triggered, the sample is acquired and stored into the channel data buffer at trigger

time. The sensor data will only be returned to the NCAP if it is subsequently requested during a data transport frame.

## 5.6 BLOCK DIAGRAM OF SUB MODULES

### 5.6.1 Data Transport




The parameters of data transport are

Input pin : 1) DCLK (2 byte)
2) DIN (2 byte)
3) NIOE (2 byte)

Output pin : 1) NACK (2 byte)
2) DOUT (2 byte)

- Read channel

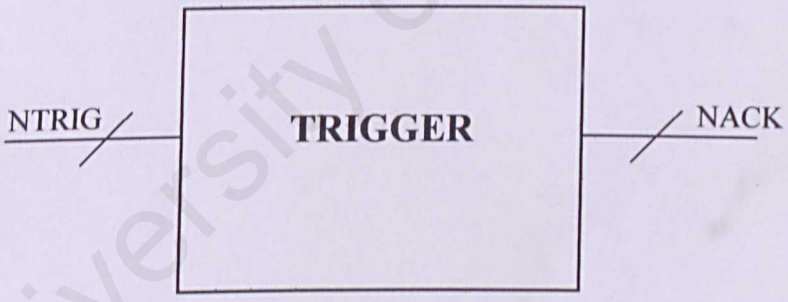
1. Data transfer shall be controlled by DCLK. NCAP keep clocking DCLK and looks for the data on DOUT.
2. The NIOE signal is asserted by the NCAP to tell the STIM to get ready for data transfer.
3. The NCAP waits for the STIM to assert the NACK signal.
4. The STIM asserts the NACK signal and the NCAP sends the function address to the STIM.
5. The NCAP waits for the NACK signal to be asserted. When the STIM asserts NACK, the NCAP reads the first byte of the data from the STIM.
6. The NCAP waits for the NACK signal to be negated and then reads the second byte of data from the STIM.
7. The NCAP negates NIOE and waits for the STIM to negate NACK.

- Write channel

1. Data transfer shall be controlled by DCLK. NCAP keeps clocking DCLK and places the data on DIN.
2. The NIOE signal is asserted by the NCAP to tell the STIM to get ready for data transfer.
3. The NCAP waits for the STIM to assert the NACK signal.

4. The STIM asserts the NACK signal and the NCAP sends the function address to the STIM.
5. The NCAP waits for the NACK signal to be asserted. When the STIM asserts NACK, the NCAP write the first byte of the data to the STIM.
6. The NCAP waits for the NACK signal to be negated and then write the second byte of data to the STIM.
7. The NCAP negates NIOE and waits for the STIM to negate NACK.

### 5.6.2 Trigger




The parameters of trigger are

Input pin : NTRIG (2 byte)

Output pin : NACK (2 byte)

- Triggering

Triggering is normally used before reading a sensor or after writing to an actuator.

1. NCAP waits for the duration of Channel Write Setup Time.
2. NCAP asserts NTRIG.
3. STIM asserts NACK.
4. NCAP negates NTRIG.
5. STIM negates NACK.
6. NCAP waits for the duration of the Channel Read Setup Time.

# CHAPTER 6

# CHAPTER 6: TOOLS AND DESIGN IMPLEMENTATION

## 6.1 HOW TO APPLY Peak FPGA Designer Suite

The steps using Peak FPGA Designer Suite

Step 1: Create the project file

Step 2: Create a new VHDL module

Step 3: Add functionality to the new module

Step 4: Compile the VHDL module

Step 5: Create a test bench

Step 6: Simulate the design

Step 7: Synthesize the design

Step 1: Create the project file

To create a new project file we select ***New Project*** from the ***File*** menu (or select the ***New Project*** icon, which appears on the far left of the toolbar). An empty, untitled Hierarchy Browser window appears as shown below. To give our new project a name and determine where its associated files are to be stored, we then save the project file selecting the ***Save Project As*** item from the ***File*** menu.

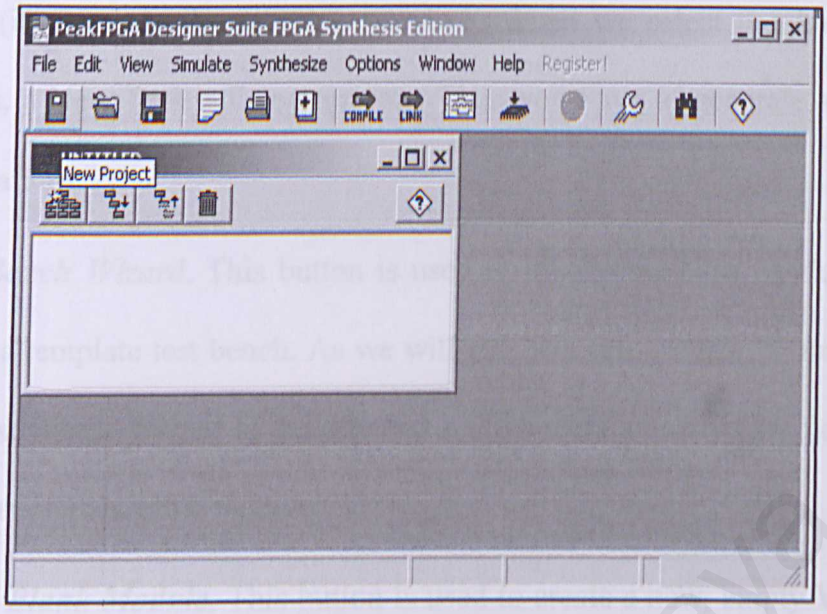



Figure 6.1: Create the project file

There are two ways to add VHDL files to a project: import them from outside the project, or create them in Peak FPGA. To import a file, you simply use the ***Add Module*** button or right click in Hierarchy Browser window then select ***Add Module***.

Step 2: Create a new VHDL module

To create a new VHDL module within Peak FPGA, we will select the ***Create New Module*** toolbar button as shown below. When this button is selected, the ***New Module*** dialog appears. This dialog has three large buttons labeled as follows:

***Module Wizard***. This button is used to invoke the New Module Wizard to create a template synthesizable module. This Wizard is intended to help you create a VHDL module that will have some functionality that you wish to implement in

logic (in our case, in an FPGA device). When we select the ***Module Wizard*** button, a New Entity dialog appears. Then you have to generate all the needed declarations.

***Test Bench Wizard***. This button is used to invoke the New Module Wizard to create a template test bench. As we will see, you can use this Wizard after using the Test Bench Wizard to quickly and easily create a nearly complete test bench for your synthesizable module.

***Create Blank Module***. This button is used to create a new, empty VHDL source file and add the module to the project. This is useful if you want to create a new VHDL file without the Wizard (for example, of you are creating a module that contains only a package and no other functionality, or if do not want to use the template created by the Module Wizard).

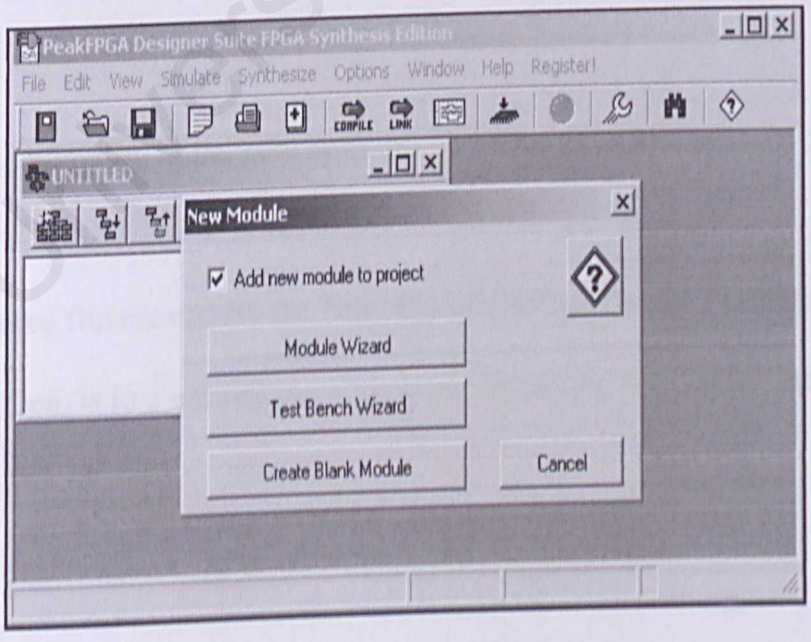



Figure 6.2: Create new VHDL module

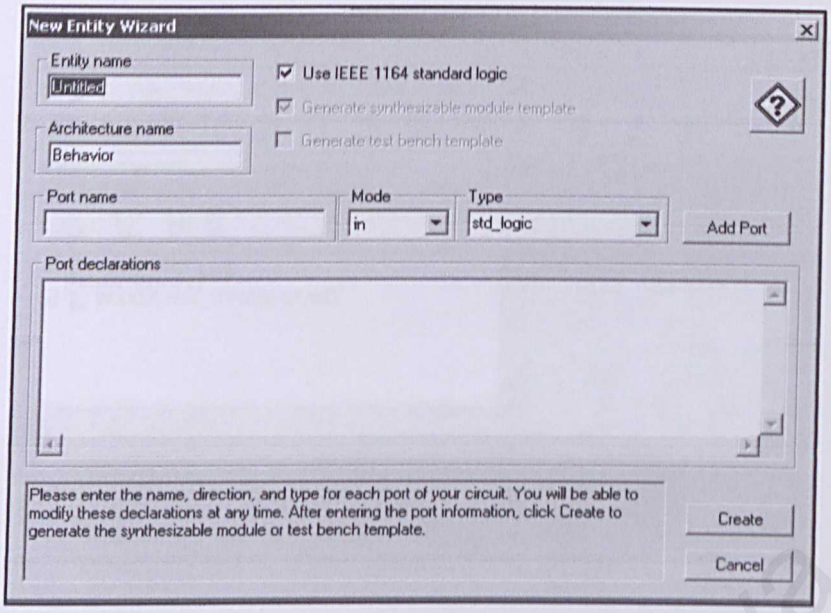



Figure 6.3: The declaration of entity and port name

In the preceding screen image, notice that the module name appears in the Hierarchy Browser, but there is no plus sign next to the name, indicating no hierarchy information. Then click on the ***Rebuild Hierarchy*** button to control when hierarchy information is updated in the Browser window.

Step 3: Add functionality to the new module

The VHDL source file created by the New Module Wizard is not a complete VHDL file. The next step, then, is to complete the source file by adding the needed functionality.

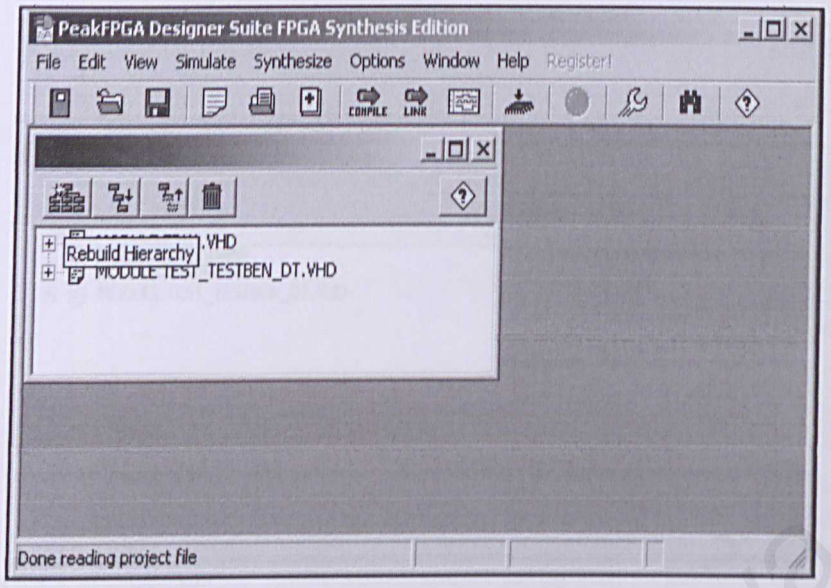



Figure 6.4: To rebuild hierarchy

Step 4: Compile the VHDL module

Once we have entered the VHDL code and modified the template to our liking, we can check our work by invoking the simulation compiler. To invoke the compiler, make sure the appropriate module (at this point there is only one) is selected in the Hierarchy Browser and select the ***Compile*** button from the toolbar.

Notice that the compiler has reported an error, then click Jump to line button to solve the problem. If it will compile without error as shown in the following transcript

*"Compilation is complete, all selected object files are up to date".*

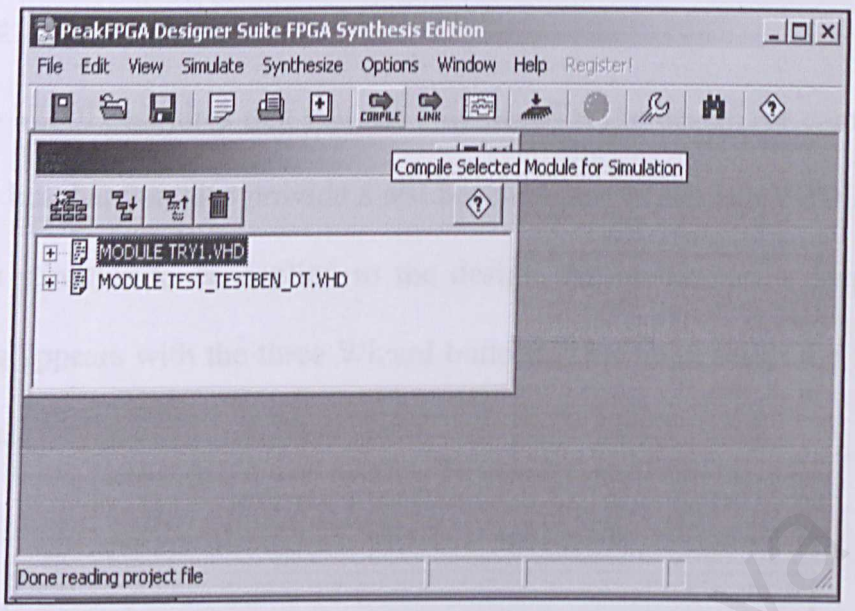



Figure 6.5: Compile the for each module

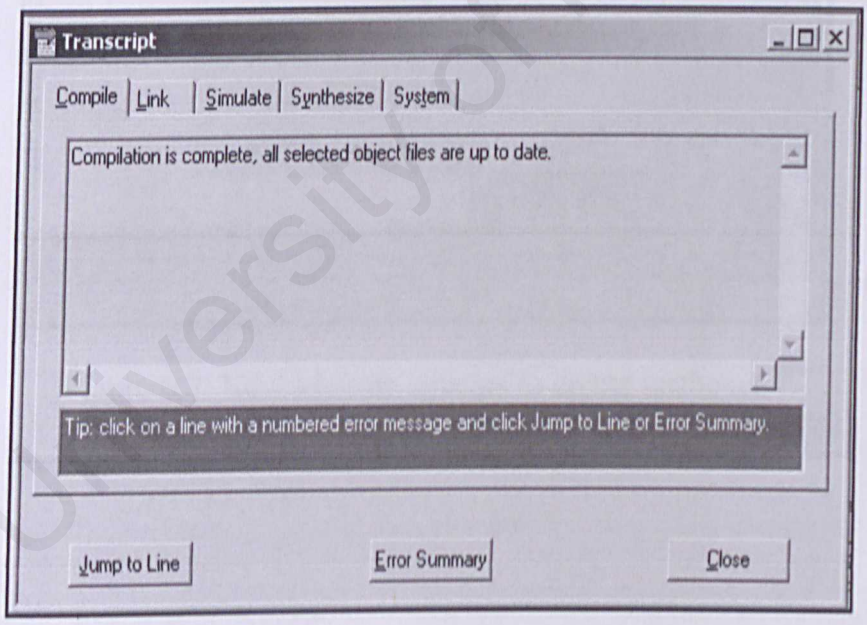



Figure 6.6: Prompt that the modules is successfully

Step 5: Create a test bench

Simulation in VHDL, requires that you not only describe the design (or component of a design) itself, but that you also provide a test bench. A test bench is a VHDL source file that describes stimulus to be applied to the design. As in the earlier step, the New Module dialog appears with the three Wizard buttons. This time, select the ***Test Bench Wizard*** button.

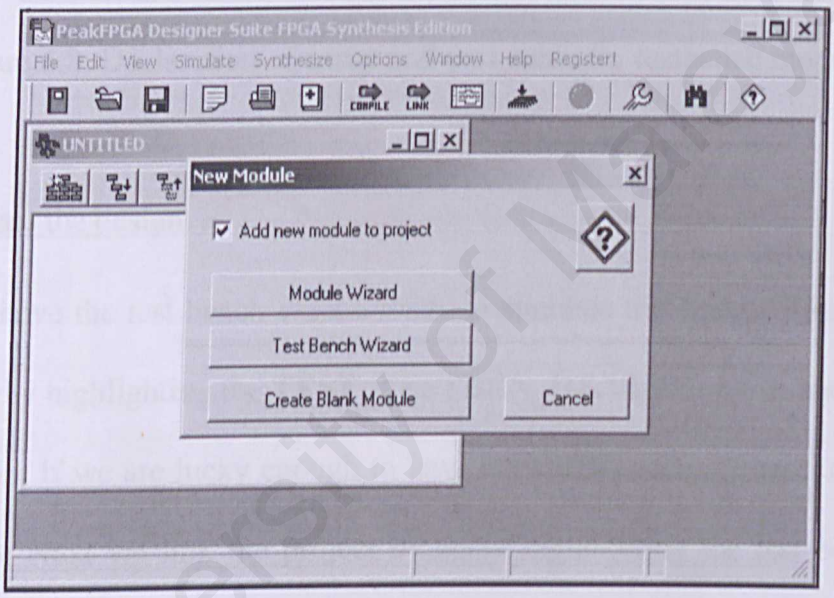



Figure 6.7: New Module appears to create testbench

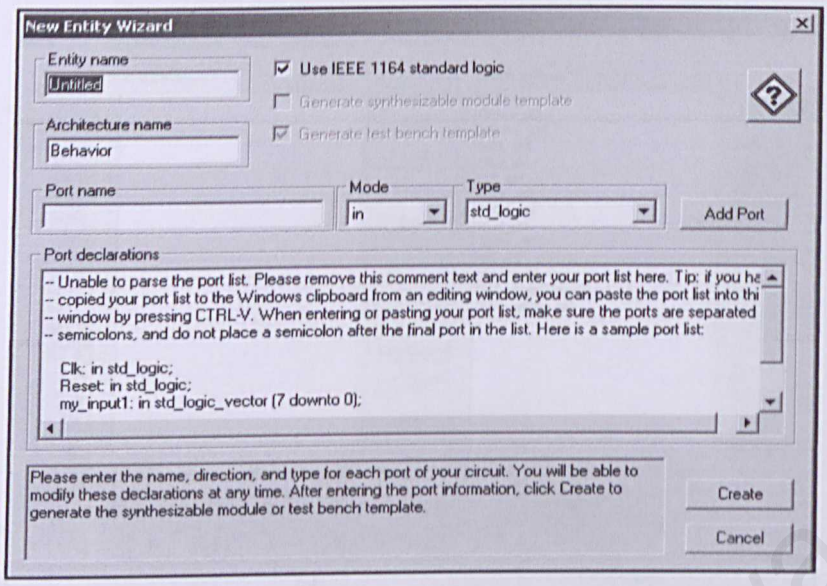



Figure 6.8: Declaration of entity and port name for testbench modules

Step 6: Simulate the design

Now that we have the test bench we are ready to simulate the design. First we compile the test bench by highlighting the **TEST_TESTBEN_DT.VHD** module and clicking the ***Compile*** button. If we are lucky enough to have no VHDL coding errors, our transcript looks like this: Next we link the project by again highlighting the test bench module (**TEST_TESTBEN_DT.VHD**) and clicking the ***Link*** button. After clicking the ***Close*** button to dismiss the Options dialog, we can select the **Load selected simulation** button (first making sure that the test bench is the highlighted module) to start the simulator:

We will simply use the ***Add Primaries*** button to make all top-level signals in the design available for display. When the signals are selected and ordered to our liking, we can click the ***Close*** button to dismiss the selection dialog.

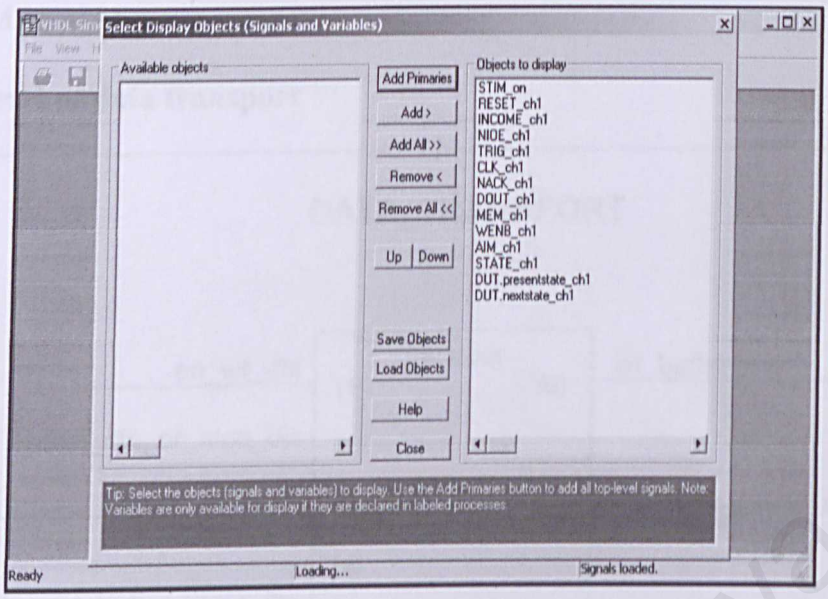
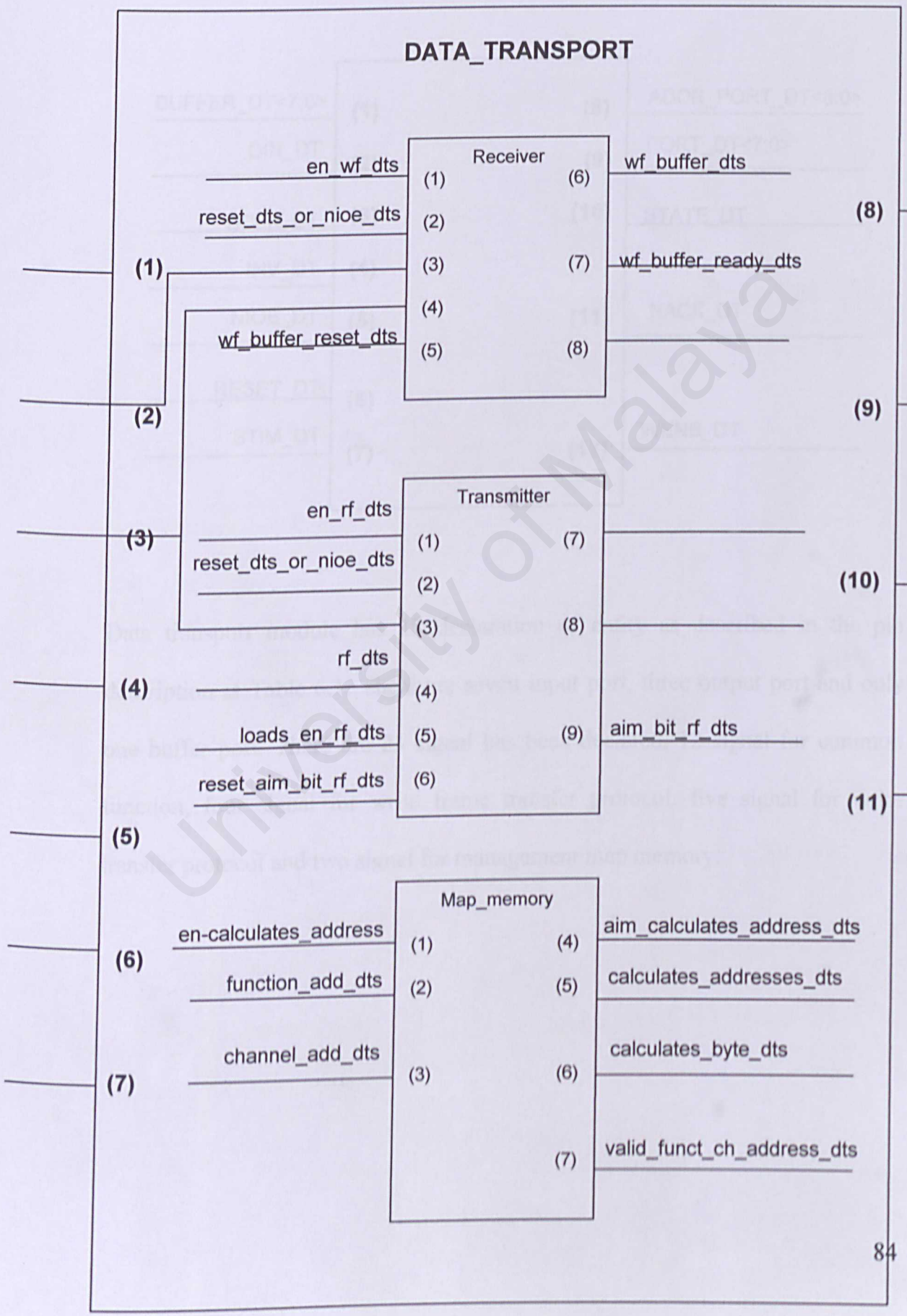



Figure 6.9: Selection of objects for VHDL simulation

Click the ***Go*** button to run the simulation. Notice that the simulation results show us that the shifter appears to be working properly, wrapping the left-most or right-most bit (depending on direction) to the opposite side of the collection of bits while shifting them accordingly:

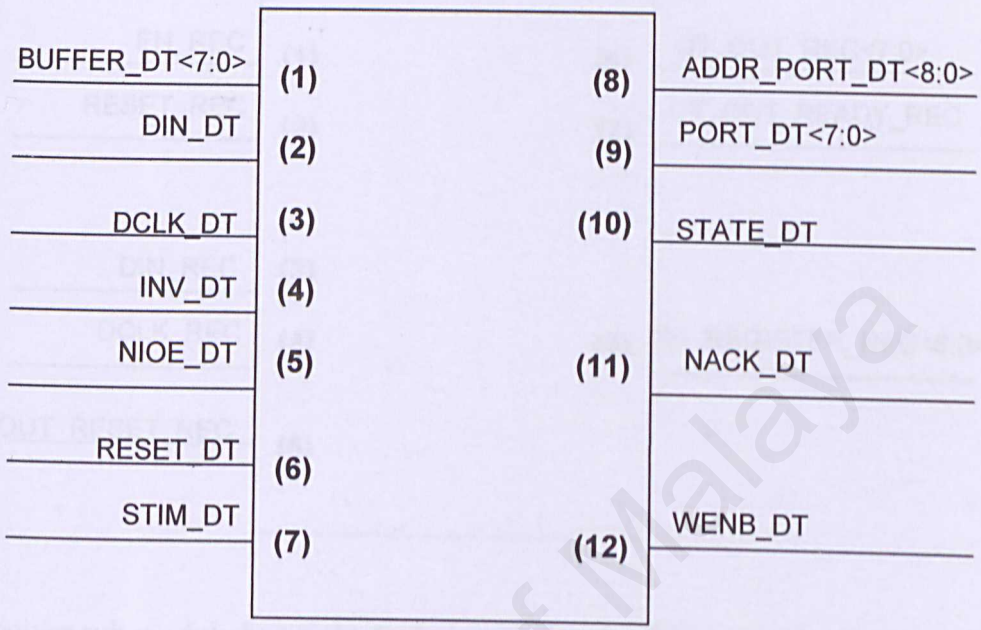
Step 7: Synthesize the design

### 6.2.1 Top level of data transport



**Black box of module**

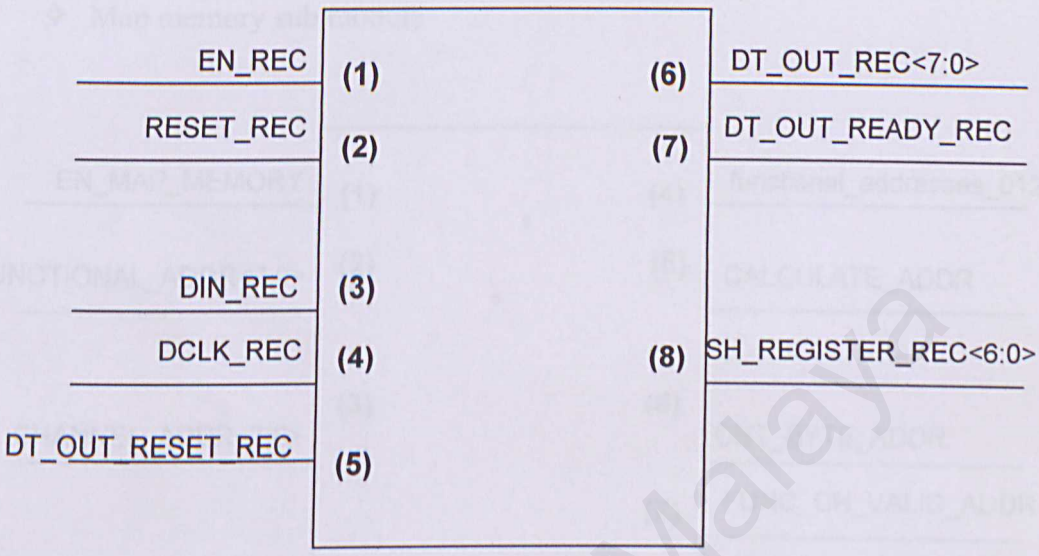
✧ DATA TRANSPORT MODULE




Data transport module has 12 declaration of entity as described in the pin description at Table 6.1. There are seven input port, three output port and only one buffer port. There are 23 signal has been declared, 12 signal for common function, four signal for write frame transfer protocol, five signal for write transfer protocol and two signal for management map memory.

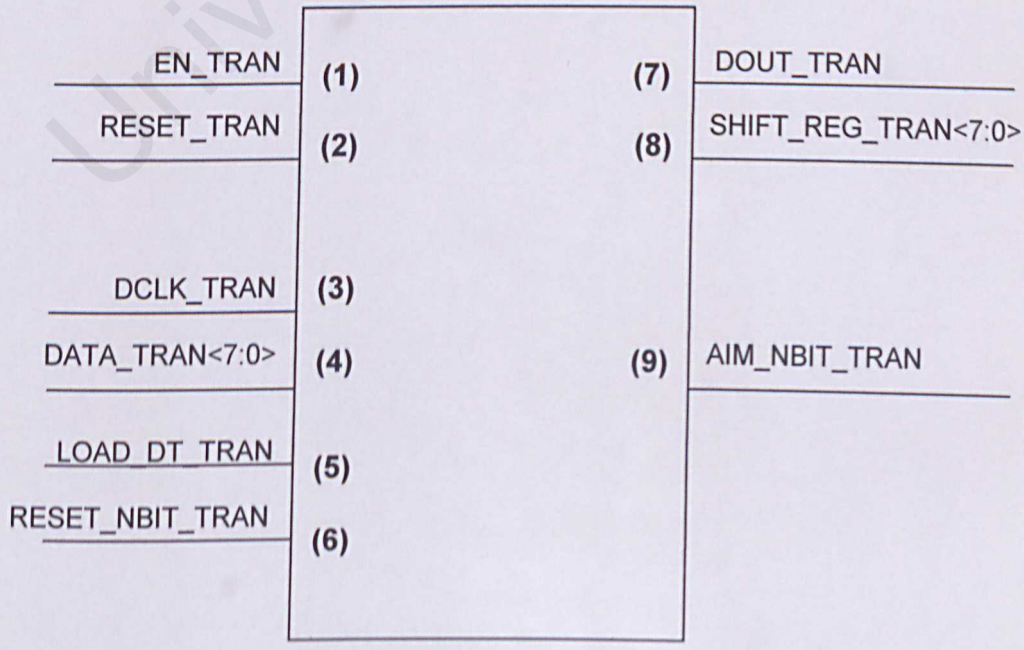
**Black box of sub module**

✧ Receiver sub module



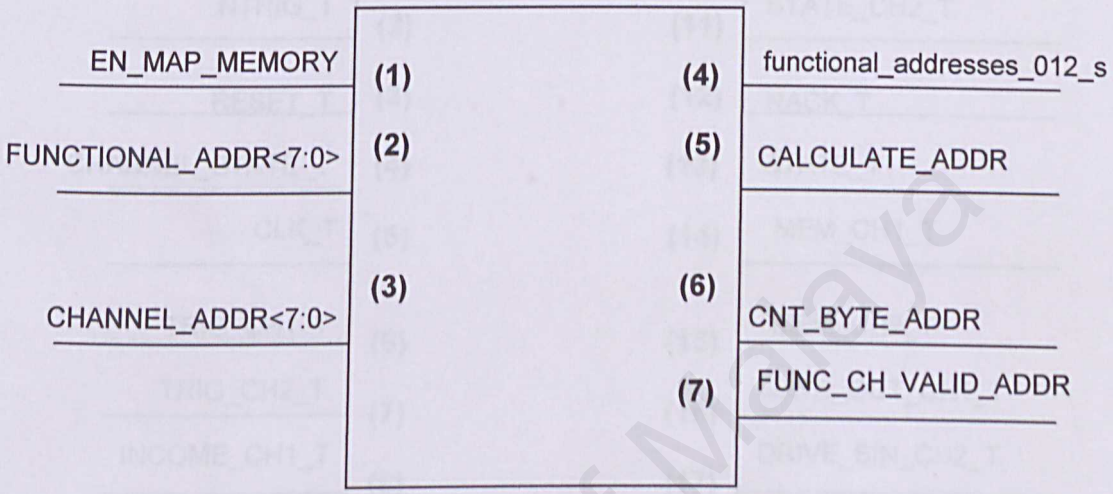

Receiver sub module has eight declaration of entity. There are five input port and three buffer port.

✧ Transmitter sub module



Transmitter sub module has nine declaration of entity. There are six input port and one for out, inout and buffer port.

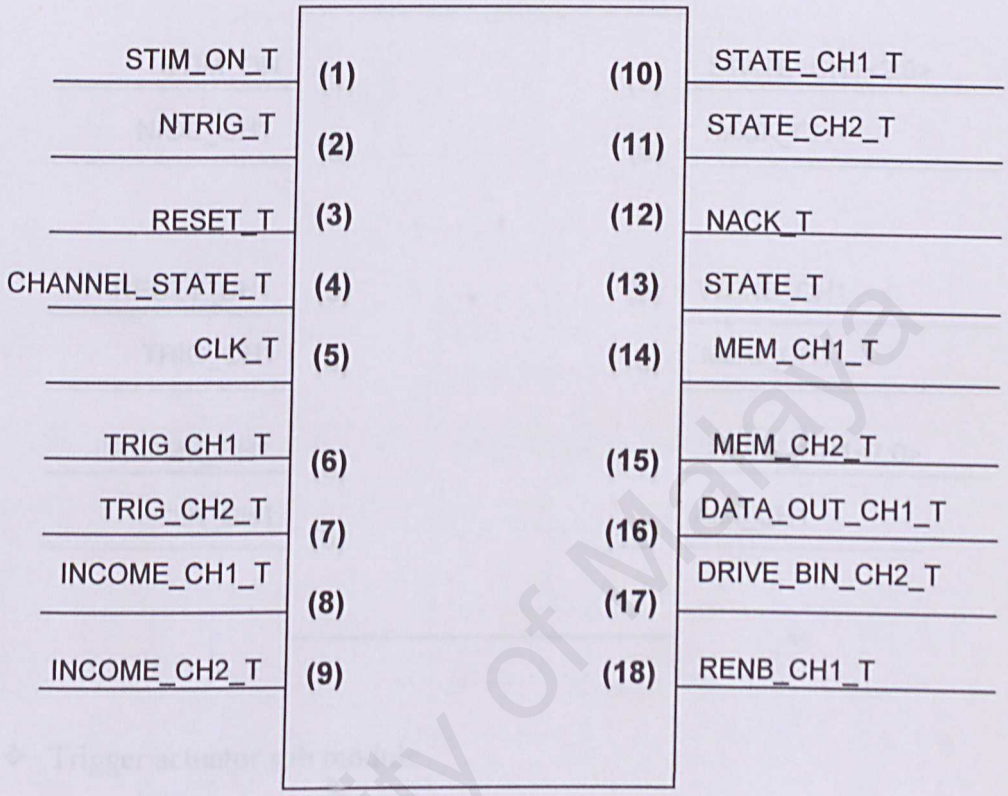
✧ Map memory sub module




Map memory sub module has seven declaration of entity. There are three input port and three for buffer port.

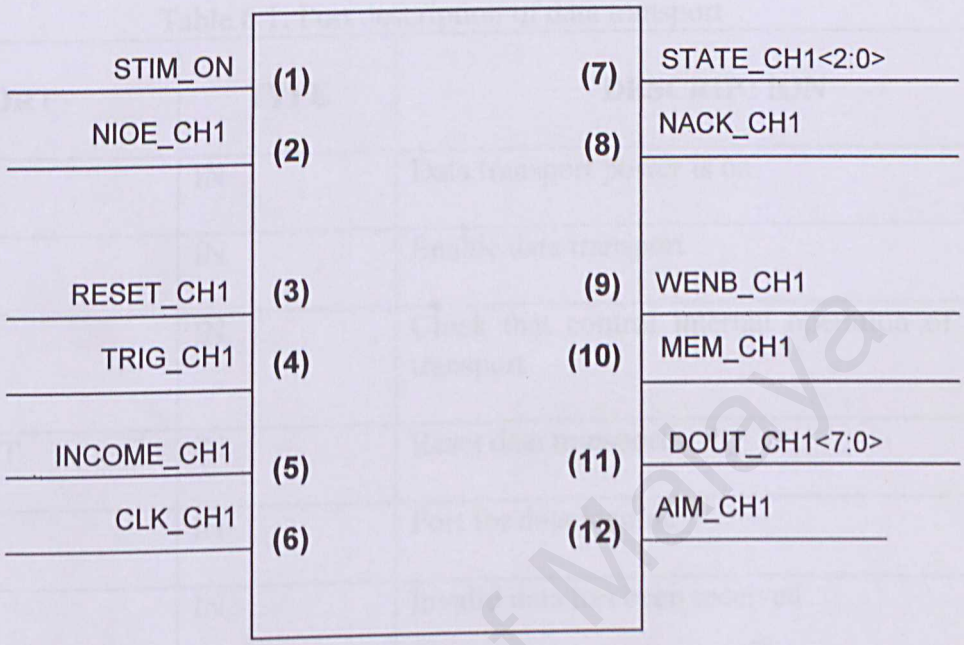
### 6.2.2 Top level of trigger

✧ TRIGGER MODULE

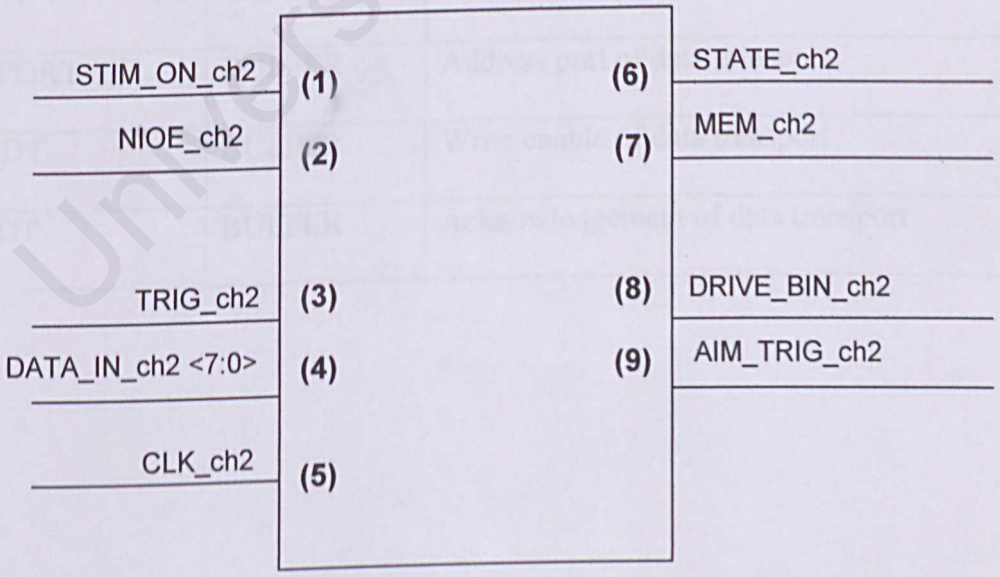


**Black box of sub modules**

✧ Trigger sensor sub module




✧ Trigger actuator sub module



### 6.2.3 Port and Signal Description

✧ **PORT DESCRIPTION (DATA TRANSPORT)**

Table 6.1: Port description of data transport

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| STIM_DT | IN | Data transport power is on |
| NIOE_DT | IN | Enable data transport |
| CLK_DT | IN | Clock that control internal operation of data transport |
| RESET_DT | IN | Reset data transport |
| DIN_DT | IN | Port for data income |
| INV_DT | IN | Invalid data has been received |
| BUFFER_DT | IN | Buffer of data transport |
| STATE_DT | OUT | State of data transport |
| PORT_DT | BUFFER | Value of port for address |
| ADDR_PORT_DT | BUFFER | Address port of data transport |
| WENA_DT | BUFFER | Write enable of data transport |
| NACK_DT | BUFFER | Acknowledgement of data transport |

✧ **PORT DESCRIPTION (RECEIVER)**

Table 6.2: Port description of receiver

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| EN_REC | IN | Enable the receiver to be active |
| RESET_REC | IN | Reset the receiver |
| DCLK_REC | IN | Clock that control the internal operation of data receive |
| DIN_REC | IN | Income data to enters in a circuit of single channel |
| DT_OUT_RESET_REC | IN | Reset of data receive by the register |
| SH_REGISTER_REC | BUFFER | Shift register address to load the data |
| DT_OUT_REC | BUFFER | Support register where to unload the data received |
| DT_OUT_READY_REC | BUFFER | It marks the receiver of ready data in the support registry |

✧ **SIGNAL DESCRIPTION (RECEIVER)**

Table 6.3: Signal description of receiver

| SIGNAL | DESCRIPTION |
|---|---|
| din_rec_s | To filtered the income data from single channel |
| dclk_rec_s | Clock of receiver |
| cnt_rec_s | Counter of n bit of receive data |

✧ **PORT DESCRIPTION (TRANSMITTER)**

Table 6.4: Port description of transmitter

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| EN_TRAN | IN | Enable the transmitter to be active for transmission of data |
| RESET_TRAN | IN | Reset of the transmission |
| DCLK_TRAN | IN | Clock that control the internal operation of transmitter |
| DATA_TRAN | IN | Loading data in the shift register |
| LOAD_DATA_TRAN | IN | It marks them of data loading in the shift register |
| RESET_NBIT_TRAN | IN | Reset the transmission of n bit |
| SHIFT_REG_TRAN | INOUT | Shift register for transmission of data |
| AIM_NBIT_TRAN | BUFFER | Aim for transmission of n bit |
| DOUT_TRAN | BUFFER | MSB data after have been shifted |

✧ **SIGNAL DESCRIPTION (TRANSMITTER)**

Table 6.5: Signal description of transmitter

| SIGNAL | DESCRIPTION |
|---|---|
| cnt_tran_s | Counter of n bit or the amount of transmit bit of data |

✧ **PORT DESCRIPTION (MAP_MEMORY)**

Table 6.6: Port description of map memory

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| EN_MAP_MEMORY_ADDR | IN | Enable the map memory address to be active |
| FUNCTIONAL_ADDR | IN | One byte of functional address |
| CHANNEL_ADDR | IN | One byte of channel address |
| CNT_BYTE_ADDR | BUFFER | Counter of byte in the address |
| functional_addresses_012_s | BUFFER | The last three bits in the functional address |
| FUNCT_CH_VALID_ADDR | BUFFER | Check whether the address valid or not |
| CALCULATE_ADDR | BUFFER | Calculation of address |

✧ **SIGNAL DESCRIPTION (MAP_MEMORY)**

Table 6.7: Port description of map memory

| SIGNAL | DESCRIPTION |
|---|---|
| error_addr_s | Check whether the address error has any error or not |
| base_addr_s | The address in map memory that based on functional address |

✧ **PORT DESCRIPTION (TRIGGER_SENSOR)**

Table 6.8: Port description of trigger sensor

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| STIM_on | IN | The STIM power is on |
| NIOE_ch1 | IN | Enable channel sensor to be active |
| RESET_ch1 | IN | Reset the channel sensor |
| TRIG_ch1 | IN | Triggering to be perform to channel sensor |
| INCOME_ch1 | IN | The first bit to be sent |
| CLK_ch1 | IN | Clock of trigger sensor to control the internal operation |
| STATE_ch1 | BUFFER | The value of state in triggering process |
| NACK_ch1 | OUT | Acknowledgement of triggering channel sensor |
| RENB_ch1 | BUFFER | Enable read process to channel sensor |
| MEM_ch1 | BUFFER | It marks that memory has been use |
| DOUT_ch1 | BUFFER | Data that has been read from sensor |
| AIM_ch1 | BUFFER | Aim for triggering of channel sensor to take place |

**SIGNAL DESCRIPTION**

Table 6.9: Signal description of trigger sensor

| SIGNAL | DESCRIPTION |
|---|---|
| presentstate_ch1 | Present state of channel sensor |
| nextstate_ch1 | Next state of channel sensor |

✧ **PORT DESCRIPTION (TRIGGER_ACT)**

Table 6.10: Port description of trigger actuator

| PORT | TYPE | DESCRIPTION |
|---|---|---|
| STIM_ON_ch2 | IN | The STIM power is on |
| NIOE_ch2 | IN | Enable channel actuator to be active |
| TRIG_ch2 | IN | Triggering to be perform to channel actuator |
| DATA_IN_ch2 | IN | Port for incoming data |
| CLK_ch2 | IN | Clock of trigger actuator to control the internal operation |
| STATE_ch2 | BUFFER | The value of state in triggering process |
| MEM_ch2 | BUFFER | It marks that memory has been use |
| DRIVE_BIN_ch2 | BUFFER | To drive the binary to actuator |
| AIM_TRIG_ch2 | BUFFER | Aim for triggering of channel actuator to take place |

**SIGNAL DESCRIPTION**

Table 6.11: Signal description of trigger actuator

| SIGNAL | DESCRIPTION |
|---|---|
| presentstate_ch1 | Present state of channel sensor |
| nextstate_ch1 | Next state of channel sensor |

# CHAPTER 7

# CHAPTER 7: SYSTEM IMPLEMENTATION AND TESTING

## 7.1 INTRODUCTION

Before the implementation of the coding, a diagram of finite state machine and the flow chart is drawn as a guide. This is to make sure, really understanding on how data transport and trigger module function in the STIM. It is easier done the coding of component first, in the STIM. After the component successfully implement, and then proceed with the coding of top level module linking the portmap of component. VHDL codes must easy to understand to make sure testbench can be created easily.

## 7.2 CODING APPROACH

Most of the sub modules and module have the same coding style. This is to avoid any misunderstanding of the port name and signal name. For the port name the next letter after input and output name is underscore followed by the name of entity. Most of the port name is described in a capitals letter. While the signal is in lower case followed by underscore s, as a signal. Another thing is, the coding provide a comment at the end of the statement. Beside, this is to avoid forgotten whenever there is a changes for each statement. This is because this software can't undo too many times, it only undo the last changes that have been done.

## 7.3 CODING IMPLEMENTATION

For this project, three component of data transport, which are receiver, transmitter and map memory has been done first, then proceed with trigger component, there are trigger sensor and trigger actuator. Then it keeps on by implement trigger top level coding.

## 7.4 CODING EXPLANATION

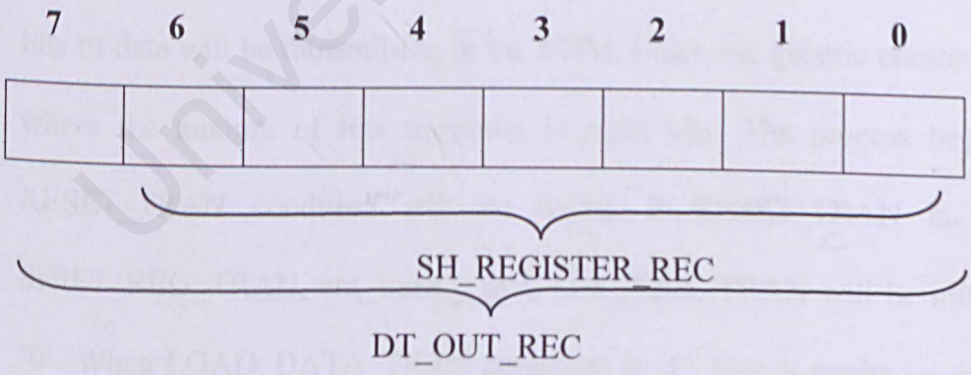
### 7.4.1 Receiver

Receiver is one of the components in the top-level data transport. The entity name mentioned that, this sub module would receive a data. The generic constant n as an integer is the constant value to specify how many bits in each data will be receives by STIM. 8 bits of data will be received. All the description for each port and signal has been discuss in the port description and signal description under user manual part. For each component in the top-level data transport, they provide package declaration where each component includes the set of declaration needed to model a data transport design. The important thing is that they can be collected or linked together into a separate design and therefore data transport can be worked on independently. The identifier provides a name for the package, so that we can use elsewhere in data transport model to refer the package. The behavior of receiver is implemented by the process reception. Whenever RESET_REC is '1', then all port will be initialized.

```
if (RESET_REC='1') then
      SH_REGISTER_REC <= (others => '0');
      DT_OUT_REC <= (others => '0');
   cnt_rec_s <= 0;
   DT_OUT_READY_REC <= '0';
```

To express the behavior, whenever clk_rec_s is at rising edge then it need to evaluate a number of different conditions and complete a different sequence of statement for each case. If EN_REC equal to '0', then it will go to for loop statement.

```
elsif (rising_edge(dclk_rec)) then
      if (EN_REC='0') then
   for i in n-2 downto 1 loop
         SH_REGISTER_REC(i) <= SH_REGISTER_REC(i-1);
   end loop;
```




The sequence of statement in for loop body indicate the receiving bit from 6 downto 1 only because SH_REGISTER_REC(0) will be din_rec. Within for loop

statement cnt_rec_s will be evaluated. If the counter signals less then 7 then it will incremented 1 by 1until cnt_rec_s larger than 7, then another for loop will carry out. The SH_REGISTER_REC will become DT_OUT_REC at the end of loop statement and cnt_rec_s will stop increment. The signal becomes '0' again waiting other data to receive. The shifted shows on how the bit will be receive bit by bit.

```
if (cnt_rec_s<n-1) then
    cnt_rec_s <= cnt_rec_s+1;
else
   for i in n-2 downto 0 loop
        DT_OUT_REC(i+1) <= SH_REGISTER_REC(i);
    end loop;
```

### 7.4.2 Transmitter

Transmitter is another component in the top-level data transport. The behavior of the transmitter is implemented by the process transmission; where each data or bits of data will be transmitting in the STIM. Likewise, generic constant is eight, where the amount of bits transmits is eight bits. The process begins when RESET_TRAN condition will be tested. If RESET_TRAN is '1', then SHIFT_REG_TRAN, cnt_trans_s and AIM_NBIT_TRAN will be initialized as '0'. When LOAD_DATA_TRAN condition is '1', then it marks income data to be load into shift register.

```
if (RESET_TRAN ='1') then
      SHIFT_REG_TRAN <= (others => '0');
      cnt_tran_s <= 0;
      AIM_NBIT_TRAN <= '0';
elsif (LOAD_DATA_TRAN='1') then
      SHIFT_REG_TRAN <= DATA_TRAN;
```

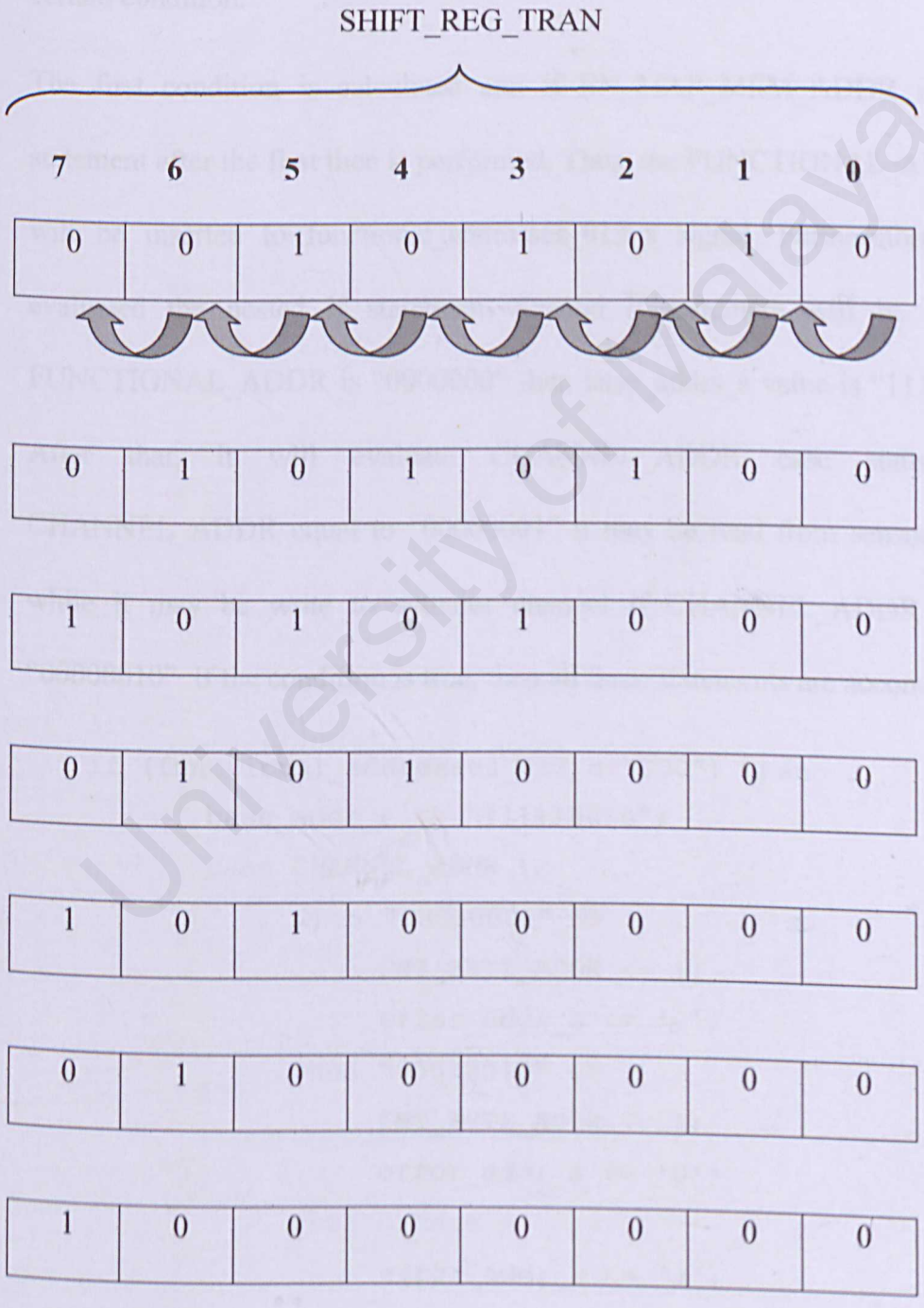
Whenever there is a falling edge of DCLK_TRAN, then EN_TRAN condition will be evaluated. If EN_TRAN is set to '0', then it will go to next statement. This for loop statement includes a specification of seven times the body of the loop to be executed. The sequence of statement in for loop body shows on how transmission of bit. The MSB will be transmit first because MSB will specify the direction of data communication whether write to STIM or read from STIM. DOUT_TRAN is the MSB that will be transfer first. Then the data will shift to the left until the end of loop.

```
elsif (falling_edge(DCLK_TRAN)) then
      if (EN_TRAN='0') then
      for i in n-1 downto 1 loop
         DOUT_TRAN <= SHIFT_REG_TRAN(n-1);
         SHIFT_REG_TRAN(i) <= SHIFT_REG_TRAN(i-1);
         SHIFT_REG_TRAN(0) <= '0';
      end loop;
       cnt_tran_s <= cnt_tran_s+1;
```

Within the sequence in for loop, the counter will be incremented whenever one bit has been shifted to the left for transmission. The counter plays with signals

that has been declare cnt_tran_s. if cnt_tran_s value is seven, then it is the end of bit transmission and the signal will be set as '0' again.

Example of shifted data



### 7.4.3 Map Memory

The behavioral architecture for map memory contains calculate_address process. This each channel depends on functional address and channel address. The functional address and channel address will be test to make sure it operates in certain condition.

The first condition is calculated and if EN_MAP_MEM_ADDR is '0' the statement after the first then is performed. Thus, the FUNCTIONAL_ADDR port will be inserted to functional_addresses_012_s signal. Furthermore, it will evaluated the nested if statement. Second if statement will be tested, if FUNCTIONAL_ADDR is "0000000" then base_addrs_s value is "111110010". After that, it will evaluate CHANNEL_ADDR case statement. If CHANNEL_ADDR equal to "00000001" it may be read from sensor channel, while it may be write to actuator channel if CHANNEL_ADDR equal to "00000010". If the condition is true, then all three statements are accomplished.

```
if (functional_addresses_012_s="000") then
      base_addr_s <= "111110010";
      case CHANNEL_ADDR is
            when "00000001" =>
                  CNT_BYTE_ADDR <= 1;
                  error_addr_s <= '0';
            when "00000010" =>
                  CNT_BYTE_ADDR <= 1;
                  error_addr_s <= '0';
            when others =>
                  error_addr_s <= '1';
```

If condition of functional_addresses_012_s in the second if statement is false then functional_addresses_012_s case statement will be check. If the value selection expression matches to any of value in the range, the statement will be carrying out. Functional_addresses_012_s will select the function to be performing to each channel, whether it performs write process or perform read process. Below is a part of functional_addresses_012_s to be evaluated.

```
case CHANNEL_ADDR is
        when "00000001" =>
             base_addr_s <= "000100000"; --32 ch1
        when "00000010" =>
             base_addr_s <= "001000000"; --64 ch2
        when others =>
             error_addr_s <= '1';
        end case;

case functional_addresses_012_s is
        when "001" =>
           CNT_BYTE_ADDR <= 2;
           error_addr_s <= '0;
        when "010" =>
           CNT_BYTE_ADDR <= 2;
           error_addr_s <= '0';
        when "011" =>
           CNT_BYTE_ADDR <= 1;
           error_addr_s <= '0';
        when "100" =>
           CNT_BYTE_ADDR <= 2;
      when others =>
           error_addr_s <= '1';
end case;
```

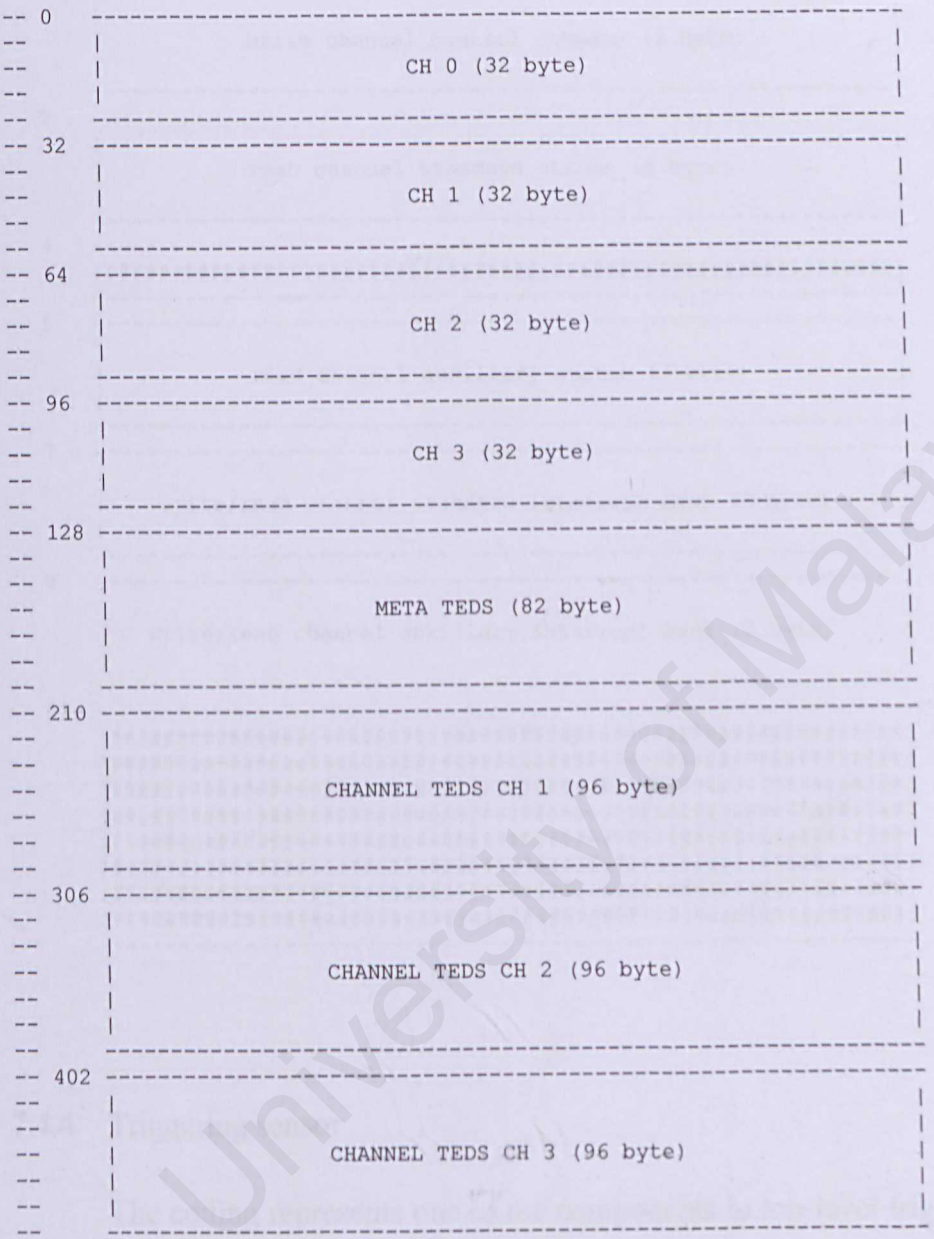
The final evaluation is to test whether FUNCTIONAL_ADDR value is "1010000", if the condition evaluates to true, the TEDS statement is implemented. It performs selected condition in the case statement.

```
elsif (FUNCTIONAL_ADDR="10100000") then
    base_addr_s <= "010000000";
    case CHANNEL_ADDR is
      when "00000001" =>
         CNT_BYTE_ADDR <= 96;
         error_addr_s <= '0';
       when "00000010" =>
         CNT_BYTE_ADDR <= 96;
         error_addr_s <= '0';
       when others =>
       error_addr_s <= '1';
    end case;
```

The end of the process, if EN_MAP_MEM_ADDR is '1' then FUNC_CH_VALID_ADDR set as '0', this indicates that the address is unvalid and no need to calculate the address.

```
elsif (EN_MAP_MEMORY_ADDR='1') then          --en=1
      FUNCT_CH_VALID_ADDR <= '0';
   CALCULATE_ADDR <= '0';
```

### 7.4.3.1 Map memory



7.4.3.2 Map memory for channel 1 and channel 2

```
-- 0   ------------------------------------------------------------
--     |                                                          |
--     |          write channel control command (2 byte)          |
--     |                                                          |
--     ------------------------------------------------------------
-- 2   ------------------------------------------------------------
--     |                                                          |
--     |          read channel standard status (2 byte)           |
--     |                                                          |
--     ------------------------------------------------------------
-- 4   ------------------------------------------------------------
--     |##########################################################|
--     ------------------------------------------------------------
-- 5   ------------------------------------------------------------
--     |                                                          |
--     |          read channel auxiliary status (2 byte)          |
--     |                                                          |
--     ------------------------------------------------------------
-- 7   ------------------------------------------------------------
--     |                                                          |
--     |    write/read channel standard interrupt mask (2 byte)   |
--     |                                                          |
--     ------------------------------------------------------------
-- 9   ------------------------------------------------------------
--     |                                                          |
--     |   write/read channel auxiliary interrupt mask (2 byte)   |
--     |                                                          |
--     ------------------------------------------------------------
-- 11  ------------------------------------------------------------
--     |##########################################################|
--     |##########################################################|
--     |##########################################################|
--     |##########################################################|
--     |##########################################################|
--     |##########################################################|
--     |##########################################################|
--     ------------------------------------------------------------
```

## 7.4.4 Triggering sensor

The coding represents one of the components in top-level trigger. This triggering function provides means for all NCAP to send STIM a command for an action to take place. Trigger shall be applying to a sensor. Trigger sensor behavior will be described in a finite state machine. The coding shows on how triggering sensor change from one state to another state. There are six state has been described in the coding. Figure 7.1 depicts the flow chart of state in triggering sensor.

A behavioral architecture body for trigger_sensor is shown. The process trigger_sensor implement the behavior. When STIM power is on then it will perform for loop statement where DOUT_ch1 will became "00000000" the end of loop statement. It is because assumption the data that will be read form sensor is "00000000'.Then it will go through presentstate_ch1 case statement for selective next state of trigger sensor. All the output port will be initialized.

```
if ( STIM_on='0' or RESET_ch1='1') then
      NACK_ch1 <= '1';
      DOUT_ch1(0) <= INCOME_ch1;
      MEM_ch1 <= '0';
      RENB_ch1 <= '0';
      AIM_ch1 <= '1';
       nextstate_ch1 <= quiescent_ch1;
else
      for i in 7 downto 1 loop
      DOUT_ch1(i) <= '0';
      end loop;
  case presentstate_ch1 is
```

The first state is quiescent_ch1 state. The trigger sensor does nothing while waiting for trigger on action take place to sensor. All ports will be initaliazed. The state of quiescent_ch1 state is "000" because STATE_ch1 port using std_logic_vector to identified each state in trigger_sensor process. If NIOE_ch1 is '1' then the next state will be triggering_ch1.

```
when quiescent_ch1 =>
      NACK_ch1 <= '1';
```

```
    DOUT_ch1(0) <= INCOME_ch1;
    MEM_ch1 <= '0';
    RENB_ch1 <= '0';
    AIM_ch1 <= '1';
    STATE_ch1 <= "000"; --0
if (NIOE_ch1='1') then
      nextstate_ch1 <= quiescent_ch1;
elsif (NIOE_ch1='0') then
      nextstate_ch1 <= triggering_ch1;
end if;
```

During triggering state, each port will be initialized based on the value that have been set up in the process and it depends on their function. For AIM_ch1 is active in the present state. This buffer port, aims to perform triggering function. If NIOE_ch1 is '0', triggering_ch1 state changes state to demanduse_mem_ch1 state. Triggering state is "001".

```
when triggering_ch1 =>
    DOUT_ch1(0) <= INCOME_ch1;
    NACK_ch1 <= '0';
    MEM_ch1 <= '0';
    RENB_ch1 <= '0';
    AIM_ch1 <= '0';
    STATE_ch1 <= "001";
    if (NIOE_ch1='1') then
       nextstate_ch1 <= quiescent_ch1;
    else
      nextstate_ch1<= demanduse_mem_ch1;
    end if;
```

The port MEM_ch1 will be set as '1' in demanduse_mem_ch1 state. If else statement will check the condition of TRIG_ch1. If TRIG_ch1 equal to '0', then the next state is write_mem_ch1. While, if TRIG_ch1 is '1', it indicates that the next state still demanduse_mem_ch1 state.

```
when demanduse_mem_ch1=>
      NACK_ch1 <= '0';
      DOUT_ch1(0) <= INCOME_ch1;
      MEM_ch1 <= '1';
      RENB_ch1 <= '0';
      AIM_ch1 <= '0';
      STATE_ch1 <= "010";
   if (NIOE_ch1='1') then
         nextstate_ch1 <= quiescent_ch1;
   elsif (TRIG_ch1='1') then
         nextstate_ch1<= demanduse_mem_ch1;
   elsif (TRIG_ch1='0') then
         nextstate_ch1<= read_mem_ch1;
     end if;
```

read_mem_ch1 state is the read from memory process; therefore MEM_ch1 and RENB_ch1 will be set as '1'. STATE_ch1 is "011". The proceeding state will be the set_aim_ch1.

```
when read_mem_ch1 =>
      NACK_ch1 <= '0';
      DOUT_ch1(0) <= INCOME_ch1;
      MEM_ch1 <= '1';
      RENB_ch1 <= '1';
      AIM_ch1 <= '0';
      STATE_ch1 <= "011"; --3
      Nextstate_ch1 <= set_aim_ch1
```

In set_aim_ch1, port MEM_ch1 still set as '1' because this process still using a memory and AIM_ch1 will be set as'1'. The next state is inactivate_ch1.

```
when set_aim_ch1 =>
      NACK_ch1 <= '0';
      DOUT_ch1(0) <= INCOME_ch1;
      MEM_ch1 <= '1';
      RENB_ch1 <= '0';
      AIM_ch1 <= '1';
      STATE_ch1 <= "100"; --4
     nextstate_ch1 <= inactivate_ch1;
```

While in the final state, trigger_sensor action became inactivate. All triggering action is disable. If NIOE_ch1 is '0', trigger_sensor still in the inactivate_ch1 state. From all the process that has been go through, if NIOE_ch1 condition is '1', the next state will be quiescent_ch1 state.

Another process is trigger_sensor_clk process. Whenever there are rising edge in the clock cycle then nextstate_ch1 will became presentstate_ch1. This is to show how one state change to another state depends on the clock cycle.

```
trigger_sensor_clk: process(CLK_ch1)
begin
   if (rising_edge(CLK_ch1)) then
          presentstate_ch1 <= nextstate_ch1;
   end if;
end process trigger_sensor_clk;
```

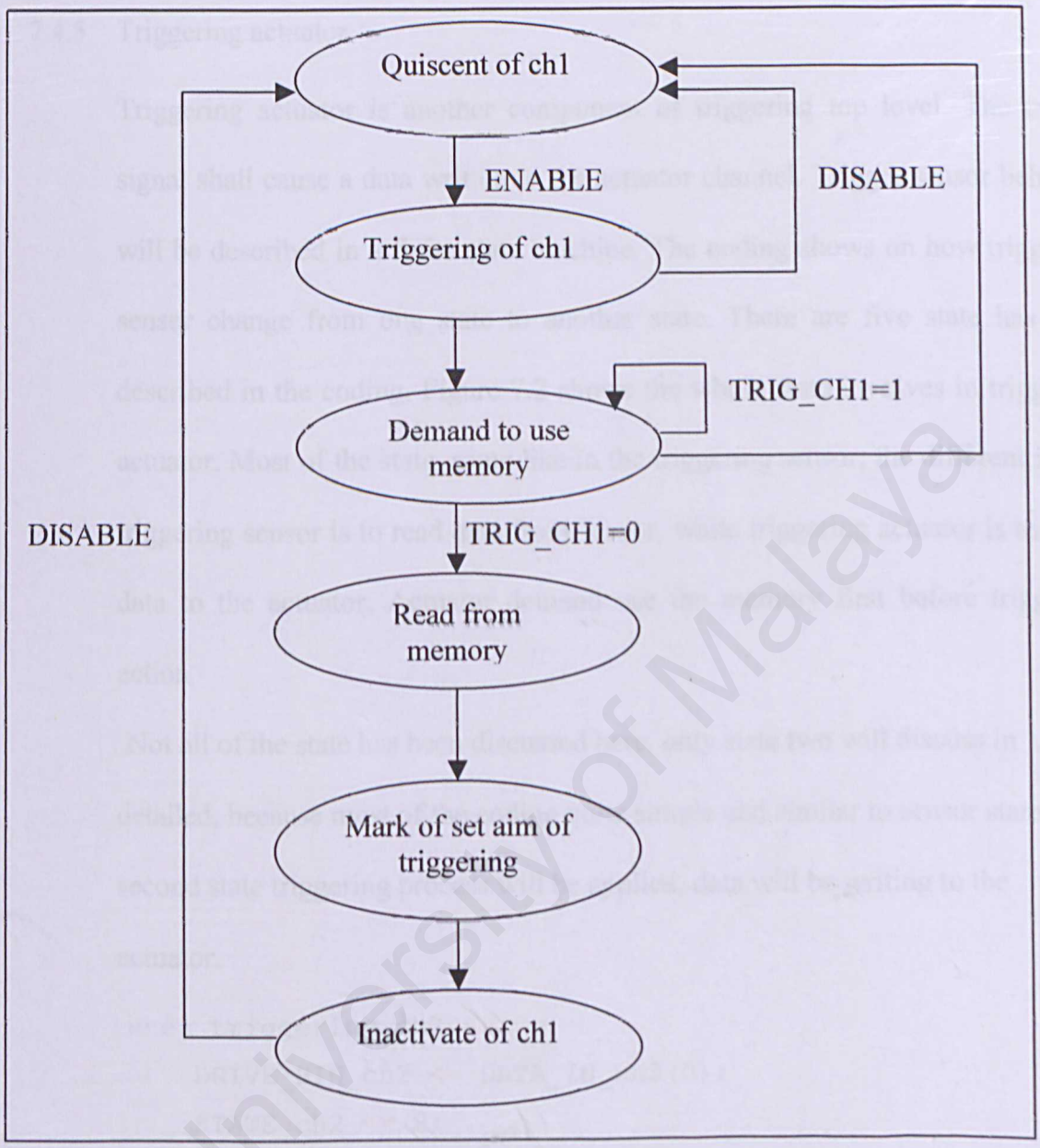



Figure 7.1: Flow chart for triggering sensor

### 7.4.5 Triggering actuator

Triggering actuator is another component of triggering top level. The trigger signal shall cause a data written to the actuator channel. Trigger sensor behavior will be described in a finite state machine. The coding shows on how triggering sensor change from one state to another state. There are five state has been described in the coding. Figure 7.2 shows the whole state involves in triggering actuator. Most of the state, same like in the triggering sensor, the different is that triggering sensor is to read data from sensor, while triggering actuator is to write data to the actuator. Actuator demand use the memory first before triggering action

Not all of the state has been discussed here, only state two will discuss in detailed, because most of the coding quite simple and similar to sensor state. In second state triggering process will be applied, data will be writing to the actuator.

```
when triggering_ch2 =>
    DRIVE_BIN_ch2 <= DATA_IN_ch2(0);
    STATE_ch2 <= 2;
    nextstate_ch2 <= set_aim_ch2;
```

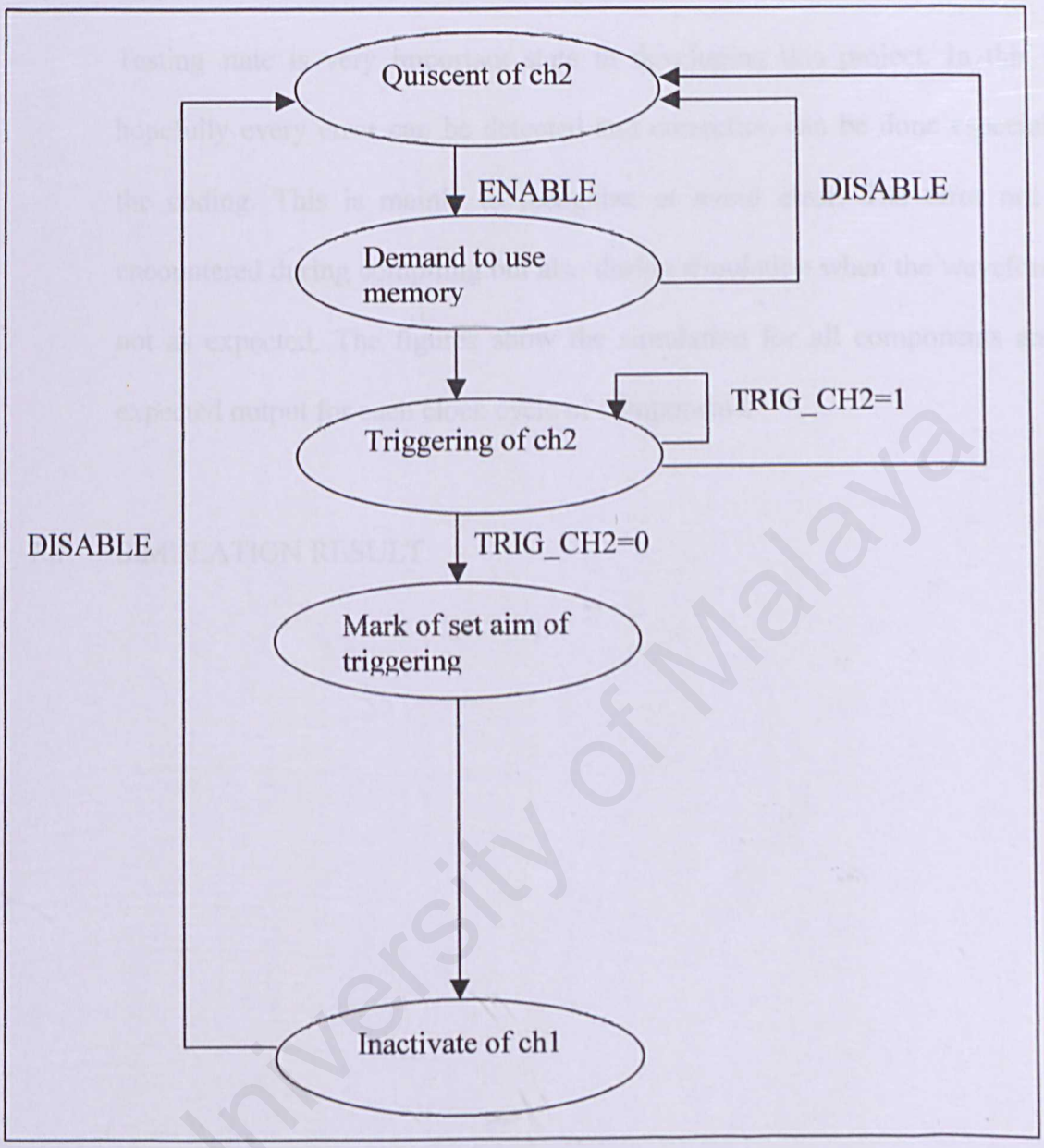



Figure 7.2: Flow chart for triggering actuator

## 7.5 TESTBENCH

Testing state is very important state in developing this project. In this stage hopefully every error can be detected and correction can be done especially in the coding. This is mainly to recognize or avoid error. The error not only encountered during compiling but also during simulation when the waveform are not as expected. The figures show the simulation for all components and the expected output for each clock cycle of components.

## 7.6 SIMULATION RESULT

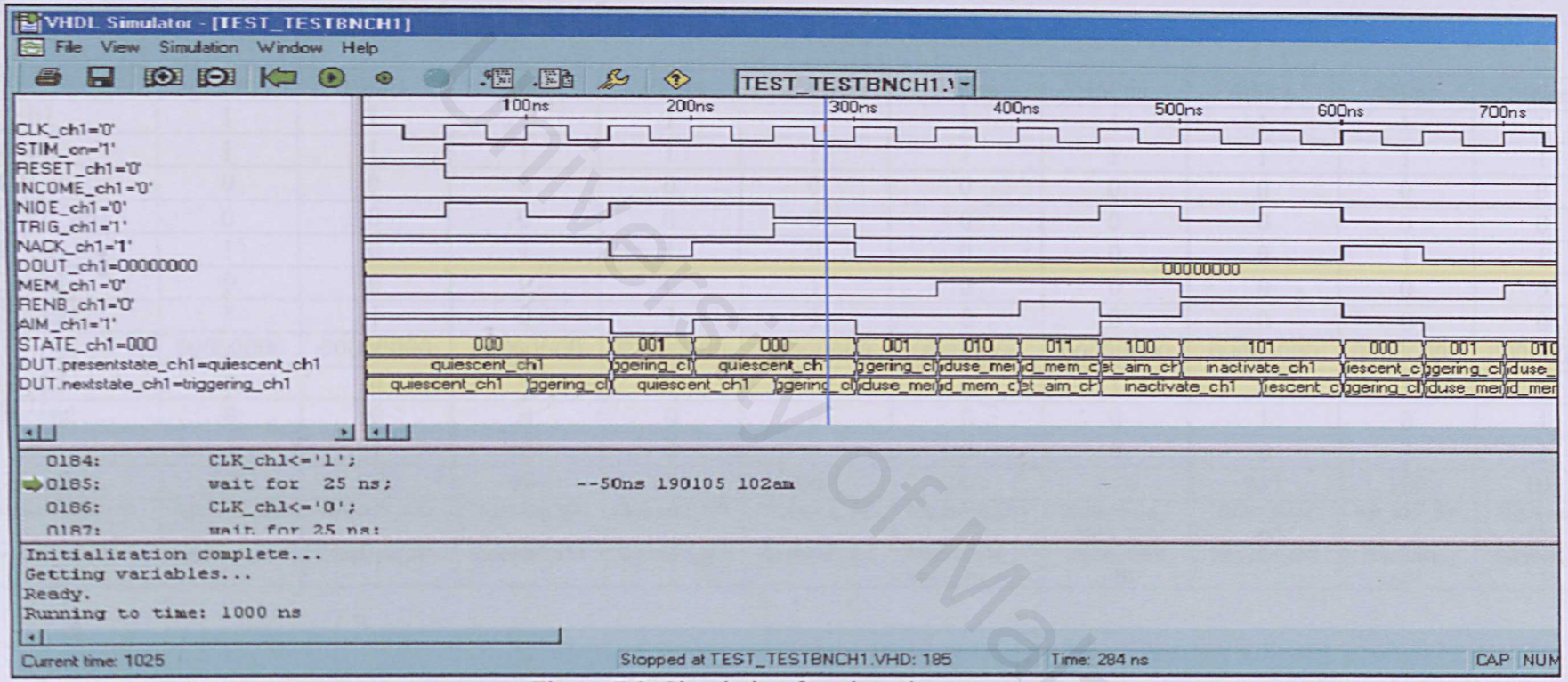



Figure 7.3: Simulation for triggering sensor

Table 7.1: Expected output for triggering sensor

| PORT | 50ns | 100ns | 150ns | 200ns | 250ns | 300ns | 350ns | 400ns | 450ns | 500ns |
|---|---|---|---|---|---|---|---|---|---|---|
| CLK_ch1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STIM_on | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| RESET_ch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| INCOME_ch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| NIOE_ch1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| TRIG_ch1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| NACK_ch1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| DOUT_ch1 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| MEM_ch1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| RENB_ch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| AIM_ch1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| STATE_ch1 | 000 | 000 | 001 | 000 | 000 | 001 | 010 | 011 | 100 | 101 |
| DUT.presentstate_ch1 | Quiscent_ch1 | Quiscent_ch1 | Triggering_ch1 | Quiscent_ch1 | Triggering_ch1 | Triggering_ch1 | Demanduse_ mem_ch1 | Read_mem _ch1 | Set_aim_ch1 | Inactivate_ ch1 |
| DUT.nexttstate_ch1 | Quiscent_ch1 | Triggering_ch1 | Quiscent_ch1 | Quiscent_ch1 | Quiscent_ch1 | Demanduse_ mem_ch1 | Read_mem _ch1 | Set_aim_ch1 | Inactivate_ ch1 | Quiscent_ch 1 |

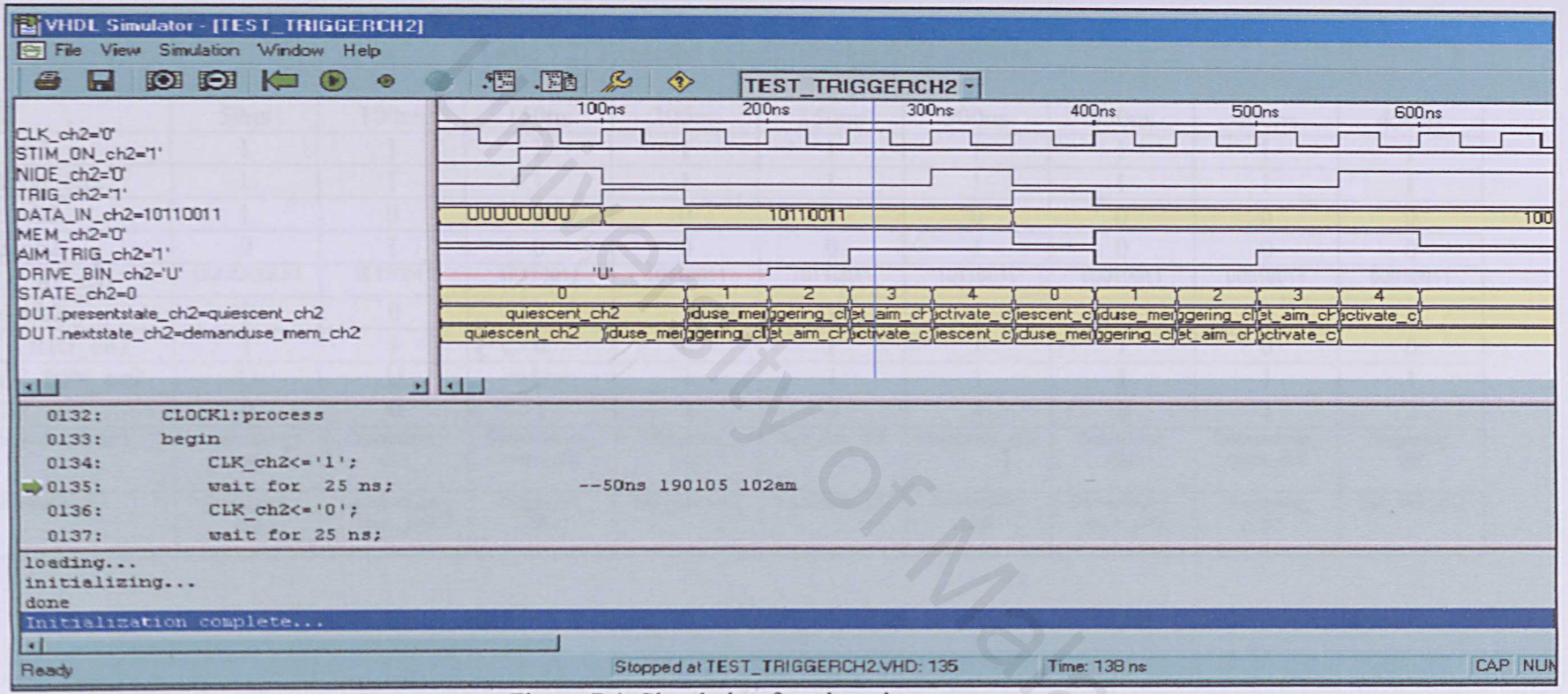



Figure 7.4: Simulation for triggering actuator

Table 7.2: Expected output for triggering actuator

| PORT | 50ns | 100ns | 150ns | 200ns | 250ns | 300ns | 350ns | 400ns | 450ns |
|---|---|---|---|---|---|---|---|---|---|
| CLK_ch2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STIM_on_ch2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| NIOE_ch2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| TRIG_ch2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| DATA_IN_ch1 | UUUUUUUU | 10110011 | 10110011 | 10110011 | 10110011 | 10110011 | 00010011 | 00010011 | 00010011 |
| MEM_ch2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| AIM_TRIG_ch2 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| DRIVE_BIN_ch2 | U | U | U | 1 | 1 | 1 | 1 | 1 | 1 |
| STATE_ch2 | 0 | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 |
| DUT.presentstate_ch1 | Quiescent_ ch2 | Quiescent_ ch2 | Demanduse_ mem_ch2 | Triggering_ ch2 | Set_aim_ch2 | Inactivate_ch2 | Quiescent_ ch2 | Demanduse_ mem_ch2 | Triggering_ ch2 |
| DUT.nexttstate_ch1 | Quiescent_ ch2 | Demanduse_ mem_ch2 | Triggering_ ch2 | Set_aim_ch2 | Inactivate_ch2 | Quiescent_ ch2 | Demanduse_ mem_ch2 | Triggering_ ch2 | Set_aim_ch2 |

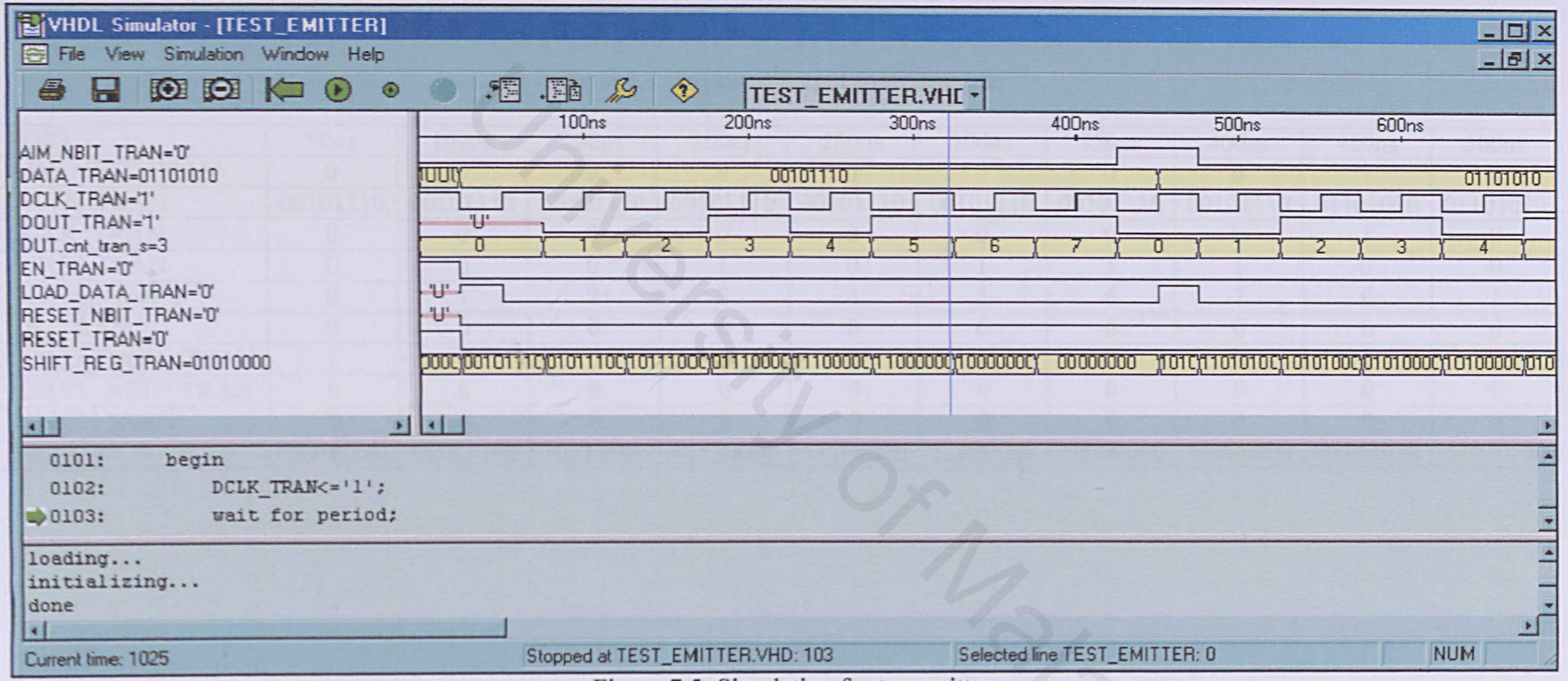



Figure 7.5: Simulation for transmitter

Table 7.1: Expected output for transmitter

| PORT | 50ns | 100ns | 150ns | 200ns | 250ns | 300ns | 350ns | 400ns | 450ns | 500ns |
|---|---|---|---|---|---|---|---|---|---|---|
| AIM_NBIT_TRAN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| DATA_TRAN | 00101110 | 00101110 | 00101110 | 00101110 | 00101110 | 00101110 | 00101110 | 00101110 | 01101010 | 01101010 |
| DCLK_TRAN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| DOUT_TRAN | U | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| DUT.cnt_tran_s | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 |
| EN_TRAN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LOAD_DATA_TRAN | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| RESET_NBIT_TRAN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RESET_TRAN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SHIFT_REG_TRAN | 00101110 | 01011100 | 10111000 | 01110000 | 11100000 | 11000000 | 10000000 | 00000000 | 01101010 | 11010100 |

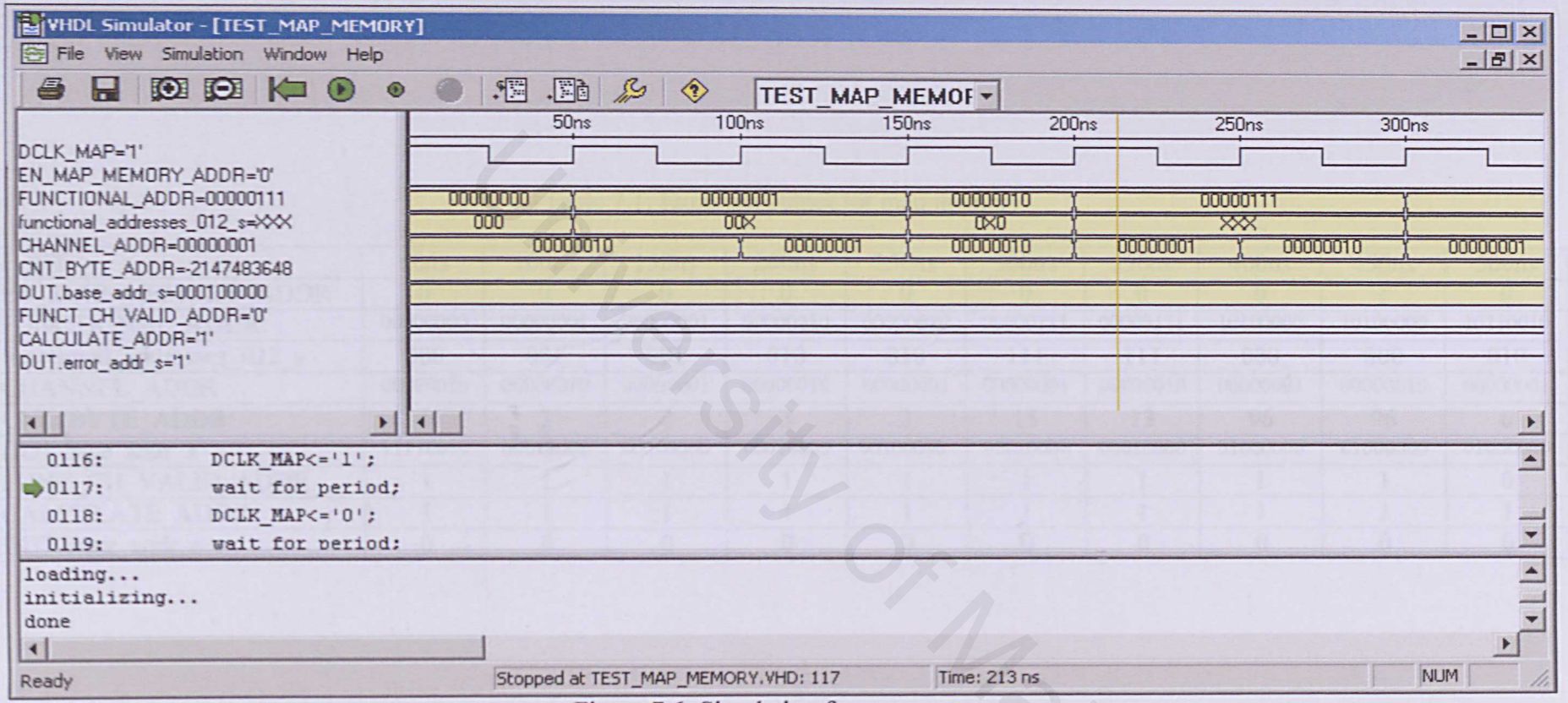



Figure 7.6: Simulation for map memory

Table 7.1: Expected output for map memory

| PORT | 50ns | 100ns | 150ns | 200ns | 250ns | 300ns | 350ns | 400ns | 450ns | 500ns |
|---|---|---|---|---|---|---|---|---|---|---|
| EN_MAP_MEMORY_ADDR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| FUNCTIONAL_ADDR | 00000000 | 00000001 | 00000001 | 00000010 | 00000010 | 00000111 | 00000111 | 10100000 | 10100000 | 10110010 |
| functional_addresses_012_s | 000 | 001 | 001 | 010 | 010 | 111 | 111 | 000 | 000 | 010 |
| CHANNEL_ADDR | 00000010 | 00000010 | 00000001 | 00000010 | 00000001 | 00000001 | 00000010 | 00000001 | 00000010 | 00000001 |
| CNT_BYTE_ADDR | 1 | 2 | 2 | 3 | 3 | 13 | 13 | 96 | 96 | u |
| DUT.base_addr_s | 11110010 | 00010000 | 00100000 | 00010000 | 00100000 | 00010000 | 00010000 | 01000000 | 01000000 | 01000000 |
| FUNC_CH_VALID_ADDR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| CALCULATE_ADDR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| DUT.error_addr_s | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# CHAPTER 8

# CHAPTER 8 : DISCUSSION

## 8.1 INTRODUCTION

This chapter discuss mostly on system evaluation is a process happened continuously from the beginning until the end of the project implementation. Within the duration, generally many technical and non technical problems encountered during the development coding. Although, most of the problems were detected and resolved, but some of the error are not. This chapter concentrate more on the problem encountered writing the codes and during the simulation. Beside that, system strength and system weakness have been discuss in detailed. So, every weaknesses of the simulation can be improved in the future enhancement the end of this chapter.

## 8.2 SYSTEM STRENGTH

VHDL is designed to fill a number of needs in the design process. It allows how it is decomposed into subsystem and how the subsystems are interconnected. Second, it allows the specification of function of system using familiar programming language forms, and then it allows the design of a system to be simulated before being manufactured. The data transport and trigger designed does not actually have certain advantages, it does a basic function. The simulation results of triggering sensor, transmitter, receiver and map memory show that the function works properly.

## 8.3 SYSTEM WEAKNESSES

Even though, during the design of the STIM, it is made of two modules data transport and triggering of sensor, only triggering of sensor successfully implemented. The data transport contains three sub modules and this entire sub modules could generates the waveforms, but not all perform as expected output. Thus, the portmap coding of data transport design could not be implemented, due to the lack of time.

## 8.4 FUTURE ENHANCEMENT

Here are some of the suggestion and possible future enhancement that could be considered on improving the STIM design. Because the coding only covers the behavior of sub modules of data transport and trigger, future could implement the top level of data transport and trigger function, then simulation of each. Then the expected output of finite state machine of data transport can be shown.

Due to the complexity of STIM design, the coding only shows a few modules the data transport and trigger module. In the real STIM design, there is another module such as controller and interrupt module that should be implementing. For each module, there still having other sub modules to be implement. If the entire module successfully develops, then real STIM have been implemented can be called as working chip.

Beside that, after using PeakFPGA as a software to implement VHDL coding, I found that a lot of limitation in the software. Although, it is easy to use but this software lack of functionality compared then Xilinx software. Future design should be done in using Xilinx software because Xilinx have a lot of functionality and Xilinx can zoom in into the STIM and all can be shown.

## 8.4 PROBLEM SOLVING

Problem encountered since in the beginning until the end of the project. The problem occur when writing the codes and during simulation. This is because of the complexity of STIM. Before writing coding for the top level, I have to zoom in into data transport module, which component occupies. Then try to write coding for the component first. After that, proceed with the top-level module coding. Since there are five sub modules have to complete first, only trigger module able to finish on time, but data transport still in development. This is due to the lack of time and lack of experience in VHDL language.

- ✧ Time constraint

I'm not only took this course in this final semester. I have another four courses that I have taken along with this project. Most of the course needs more concentration because each course has an individual and group assignment to be complete in a given time as well as this project. Of course I have to spent more time study each course, to make sure I can score in the quiz and final exam for this final semester.

To solve the problem, I have to manage and schedule my time properly than before. One month before viva, I spent most of my time in Jawinet lab because most of the component still not completes that time. This is to ensure the project can be settle down when viva week around the corner.

✧ Lack of experience in language

Although I have learned VHDL before but I still lack of experience. A lot of thing to learn for more detail, because there are difficult to develop VHDL codes and testbench. Problems arise frequently during writing coding and simulation. A lot of time to spent in order to get exactly like expected output. To overcome this problem, I always refer several of examples from Internet, reference book and also having discussion with supervisor and moderator.

# CHAPTER 9

# CHAPTER 9 : CONCLUSION

## 9.1 INTRODUCTION

Design a STIM, need a lot of understanding on how the STIM function. STIM contains a lot of component to be a working chip. Only highly experienced people can design such complexity circuit. The above factor also, means that the time taken to complete the whole function in a STIM is not within two or three month. As an experienced person also, they need five years to complete this project. Therefore this project not achieved working chip. This project implements the coding of component for top-level data transport and top level trigger. Trigger top level also has been created but has not been fully achieved because trigger sensor and trigger actuator portmap are not connected properly. This is due to lack of time.

This proposed design, actually covers only the basic functionality from what I have understand after study the STIM. So, this design maybe and may not achieved 100% as a real STIM. It might be achieved certain coding. I have try my best to totally understand on VHDL programming language, by refers various source from reference book, internet and having a discussion with supervisor and moderator to share but, I still lack of experience and need more time to really master in this hardware language. The problem encountered has been resolved, but one of the simulations not fulfilled as expected output. If I get

a chance and a time to spend, study back and modify the coding, maybe the simulation will achieve the expected output.

## 9.2 EXPERIENCED AND KNOWLEDGE

Throughout the duration of system development a lot of invaluable experience has been obtained. The most important is the experience of developing hardware coding. Develop top level of STIM in VHDL coding is indeed challenging and exciting experienced. All theory that I have learned in my course last year has been applied. During the system implementation and testing, I had through my hard time especially during testing stages. A lot of error encountered and sometimes there is no error but the simulation still does not prove as the expected output. Sometimes, I have to repeat same process and work a few times to get any solution for the problem. Even though I have to repeat again and again, at the end, I feel happy because the simulation accomplishment. From that, I have learned what my mistake and how to troubleshoot, and when the same problem arise again, I can solve it immediately, faster than the day before.

Beside that, by developing the project, personally, I feel that I have learned a lot of knowledge on VHDL programming language, which I never realized and thought before this. VHDL subject that I have learned during my studies, just a basic theory and easy to understand, we never encountered any significant problem while using Peak FPGA. But in the real world development, this programming language is complicated, a lot of process to be implements in

order to reach working chip. What I have learned before is not enough for this project; we have to get more information and knowledge from revision book, Internet and other findings to gain more example. From the examples I can get an indication on how to create the coding, it give a lot of hint that I never done before. Another thing is, to success on study we must have our own responsibility to ask a person who are master in those language. No need to shy asking someone who has more experienced, because we must learn from our own mistake. With the help of certain individual especially my supervisor, moderator, and senior a part of the problem was overcome.

**REFERENCES**


Stephen R. Schach, (2002), *Object Oriented and Classical Software Engineering*. 6th ed. New York: McGrawHill.

Ilene J. Bush-Vishniac, (1999), *Electromechanical sensors and actuator*. 1st ed. New York: Springer.

K. Brindley, (1998). *Sensor and Transducer*. 1st ed. London: Heinemann Professional Publishers.

Pferrari, A. Flammini, D. Marioli, A. Torani, *A low cost Internet-enable smart sensor*, Proc. On IEEE Sensors 2002, 12-14 June 2002, Orlando, USA.

Institute of Electrical and Electronics Engineers, *IEEE Standard for a Smart Transducer Interface for Sensor and Actuator-Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Format*, IEEE Std. 1451.2-1997.

Peter J. Ashenden, (1996). *The Designer's Guide to VHDL*. 1st ed. San Franscisco, California: Morgan kaufmann Publishers, Inc.

From Internet

-(URL-http://www.**smart**sensor.com/data/tmi2-sa.pdf/)

-(URL-http:// ieee1451.nist.gov/senoct11-2col.pdf/)

-(URL-http:// kistler.com/web/article.nsf/.../000-276/$File/000-276e-06.01.pdf)

-(URL-http:// www.**sensor**smag.com/articles/0600/83/main.shtml)

-(URL-http:// standards.ieee.org/bearer/sba/03-25-04.html)

-(URL-http:// ece.osu.edu/ie/main/Publications/wireless_sensor/deepika_thesis.pdf)

-(URL-http:// www.vlsibank.com/sessionspage.asp?titl_id=3050)

-(URL-http://www.sensorsportal.com/HTML/SENSORS/TEDS_Sensors.htm)

-(URL-http://www.doulos.com/)

-(URL-http://www.safoo.com/)

# APPENDIX

# IEEE-P1451.2 Smart Transducer Interface Module


| | |
|---|---|
| Stan P. Woods | Hewlett-Packard Company |
| Janusz Bryzek, Ph.D. | Intelligent MicroSensor Technology |
| Steven Chen | Aeptec Microsystems |
| Jeff Cranmer | Lucas Control Systems Products |
| Edwin Vivian El-Kareh | AB Networks |
| Mike Geipel | Eurotherm Controls |
| Fernando Gen-Kuong | Endevco |
| John Houldsworth | Eurotherm Controls |
| Norm LeComte | Texas Instruments |
| Kang Lee | National Institute of Standards and Technology |
| Michael F. Mattes | SSI-Controls Technology |
| David E. Rasmussen | Hewlett-Packard Company |



## Abstract

This paper provides a technical overview of the smart transducer interface module (STIM), the key element of the proposed IEEE-P1451.2 Draft Standard for Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheets (TEDS) Formats. The draft standard is released for balloting as of August, 1996. Objectives and genealogy of this standard are summarized. Key technical innovations such as the TEDS, representation of physical units, general calibration model, triggering of sensors and actuators, variable transfer rate between a host and the STIM, and support for multivariable transducers are briefly discussed. Detailed descriptions of the STIM, TEDS, digital interface, and plug-and-play operation are also provided. The specifics of physical units encoding, an example of a TEDS, and an example of timing requirements for taking a sensor reading are also included to aid the overview.


## 1. Objectives of IEEE-P1451

The Draft Standard for A Smart Transducer Interface for Sensors and Actuators, IEEE-P1451, aims at simplifying transducer connectivity to existing networks. IEEE-P1451 consists of two parts:

- P1451.1, developing a network independent common object model for smart transducers.
- P1451.2, enabling connection of transducers to network microprocessors.

## 2. Introduction

Imagine that you can:

- Select the transducer best suited to solve the measurement or control problem independently of the selected control network.
- Use the same transducers on multiple control networks.
- Select the control network best suited for the application without transducer compatibility constraints.
- Achieve automatic self-configuration when a transducer is connected to a network microprocessor.

These are goals the IEEE-P1451.2 Smart Transducer Interface[1] is helping to meet.

This paper describes the progress made to date by the IEEE-P1451.2 Transducer to Microprocessor Working Group to facilitate the ease of connecting transducers to microprocessors. At the core of this effort is a transducer electronic data sheet (TEDS), which is a data structure stored in a small amount of nonvolatile memory, physically associated with the transducer. The TEDS is used to store parameters which describe the transducer to the network capable application processor (NCAP), making self-identification of the transducer to a system possible.

The working group has defined the contents of the TEDS and a digital hardware interface to access the TEDS, read sensors, and set actuators. The resulting hardware partition encapsulates the measurement aspects[2] in a smart transducer interface module (STIM) on one side of

---

[1] These goals are being met in combination with IEEE-1451.1 which is not covered in this paper.

[2] Covered by P1451.2.

ıe digital interface, and the application related aspects[3] ı the NCAP.

his paper describes the hardware block diagram of the TIM, including the TEDS and the digital interface.

## 3. Background

Control networks provide many benefits for transducers,[4] uch as:

- Significant reduction of installation costs by eliminating long and large numbers of analog wires.
- Acceleration of control loop design cycles, reduction of commissioning time, and reduction of downtime.
- Dynamic configuration of measurement and control loops via software.
- Addition of intelligence by leveraging the micro-processors used for digital communication.

One major problem for analog transducer manufacturers is the large number of networks on the market today. It is currently too costly for many transducer manufacturers to make unique smart transducers tailored for each network on the market.

In September 1993, the proposal of developing a smart sensor communication interface standard was accepted by IEEE-TC9.[5] In March, 1994, the National Institute of Standards and Technology (NIST) and the Institute of Electrical and Electronics Engineers (IEEE), hosted a first workshop to discuss smart sensor interfaces and the possibility of developing a standard interface that would simplify connectivity of smart transducers to networks. Since then, a series of four more workshops have been held and two technical working groups formed in February, 1995:

- The P1451.1 working group concentrating on a common object model for smart transducers along with interface specifications to the model [1] [2] [3].
- The P1451.2 working group concentrating on defining the TEDS, the STIM, and the digital interface including connector pin allocation and a com-munication protocol between the STIM and the NCAP [1] [4].

[3] Covered by P1451.1.

[4] "Transducer" is defined by IEEE-P1451 working groups as either a sensor or an actuator.

[5] Technical Committee 9 (TC-9) on Sensor Technology of the Instrumentation and Measurement Society

## 4. Key technical features

Figure 1A. depicts a STIM and the associated digital interface as described in the P1451.2 draft. The STIM is shown here under the control of a network-connected microprocessor. In addition to their use in control networks, STIMs can be used with microprocessors in a variety of applications such as portable instruments and data acquisition cards as shown in Figure 1B.

The STIM embodies specific unique features of this proposed standard, which are briefly described below.

### 4.1 Single general purpose TEDS

The TEDS as presently defined supports a wide variety of transducers with a single general purpose TEDS structure.[6] This approach makes the rest of the system easier to implement and the implementation scaleable. If specific fields are not required for a given transducer, these fields have zero width, saving the required memory.

### 4.2 Representation of physical units

The P1451.2 draft adopts a general method for describing physical units sensed or actuated by a transducer. The method, described in the table in Appendix A, employs a binary sequence of ten bytes to encode physical units. A unit is represented as a product of the seven SI[7] base units and the two SI supplementary units, each raised to a rational power. This structure encodes only the exponents; the product is implicit. Appendix A contains examples for distance, pressure, acceleration, and strain.

The U/U forms (enumerations one and three in Appendix A) are for expressing "dimensionless" units such as strain (meters per meter) and concentration (moles per mole). The numerator and denominator units are identical, each being specified by subfields two through ten [5].

[6] As opposed to creating a unique TEDS structure for each kind of transducer: for example, one TEDS structure for temperature sensors and another TEDS structure for servo actuators, each structure with unique required fields.

[7] Le Système International d'Unités

the digital interface, and the application related aspects[3] on the NCAP.

This paper describes the hardware block diagram of the STIM, including the TEDS and the digital interface.

## 3. Background

Control networks provide many benefits for transducers,[4] such as:

- Significant reduction of installation costs by eliminating long and large numbers of analog wires.
- Acceleration of control loop design cycles, reduction of commissioning time, and reduction of downtime.
- Dynamic configuration of measurement and control loops via software.
- Addition of intelligence by leveraging the micro-processors used for digital communication.

One major problem for analog transducer manufacturers is the large number of networks on the market today. It is currently too costly for many transducer manufacturers to make unique smart transducers tailored for each network on the market.

In September 1993, the proposal of developing a smart sensor communication interface standard was accepted by IEEE-TC9.[5] In March, 1994, the National Institute of Standards and Technology (NIST) and the Institute of Electrical and Electronics Engineers (IEEE), hosted a first workshop to discuss smart sensor interfaces and the possibility of developing a standard interface that would simplify connectivity of smart transducers to networks. Since then, a series of four more workshops have been held and two technical working groups formed in February, 1995:

- The P1451.1 working group concentrating on a common object model for smart transducers along with interface specifications to the model [1] [2] [3].
- The P1451.2 working group concentrating on defining the TEDS, the STIM, and the digital interface including connector pin allocation and a com-munication protocol between the STIM and the NCAP [1] [4].

[3] Covered by P1451.1.
[4] "Transducer" is defined by IEEE-P1451 working groups as either a sensor or an actuator.
[5] Technical Committee 9 (TC-9) on Sensor Technology of the Instrumentation and Measurement Society

## 4. Key technical features

Figure 1A. depicts a STIM and the associated digital interface as described in the P1451.2 draft. The STIM is shown here under the control of a network-connected microprocessor. In addition to their use in control networks, STIMs can be used with microprocessors in a variety of applications such as portable instruments and data acquisition cards as shown in Figure 1B.

The STIM embodies specific unique features of this proposed standard, which are briefly described below.

### 4.1 Single general purpose TEDS

The TEDS as presently defined supports a wide variety of transducers with a single general purpose TEDS structure.[6] This approach makes the rest of the system easier to implement and the implementation scaleable. If specific fields are not required for a given transducer, these fields have zero width, saving the required memory.

### 4.2 Representation of physical units

The P1451.2 draft adopts a general method for describing physical units sensed or actuated by a transducer. The method, described in the table in Appendix A, employs a binary sequence of ten bytes to encode physical units. A unit is represented as a product of the seven SI[7] base units and the two SI supplementary units, each raised to a rational power. This structure encodes only the exponents; the product is implicit. Appendix A contains examples for distance, pressure, acceleration, and strain.

The U/U forms (enumerations one and three in Appendix A) are for expressing "dimensionless" units such as strain (meters per meter) and concentration (moles per mole). The numerator and denominator units are identical, each being specified by subfields two through ten [5].

[6] As opposed to creating a unique TEDS structure for each kind of transducer: for example, one TEDS structure for temperature sensors and another TEDS structure for servo actuators, each structure with unique required fields.
[7] Le Système International d'Unités

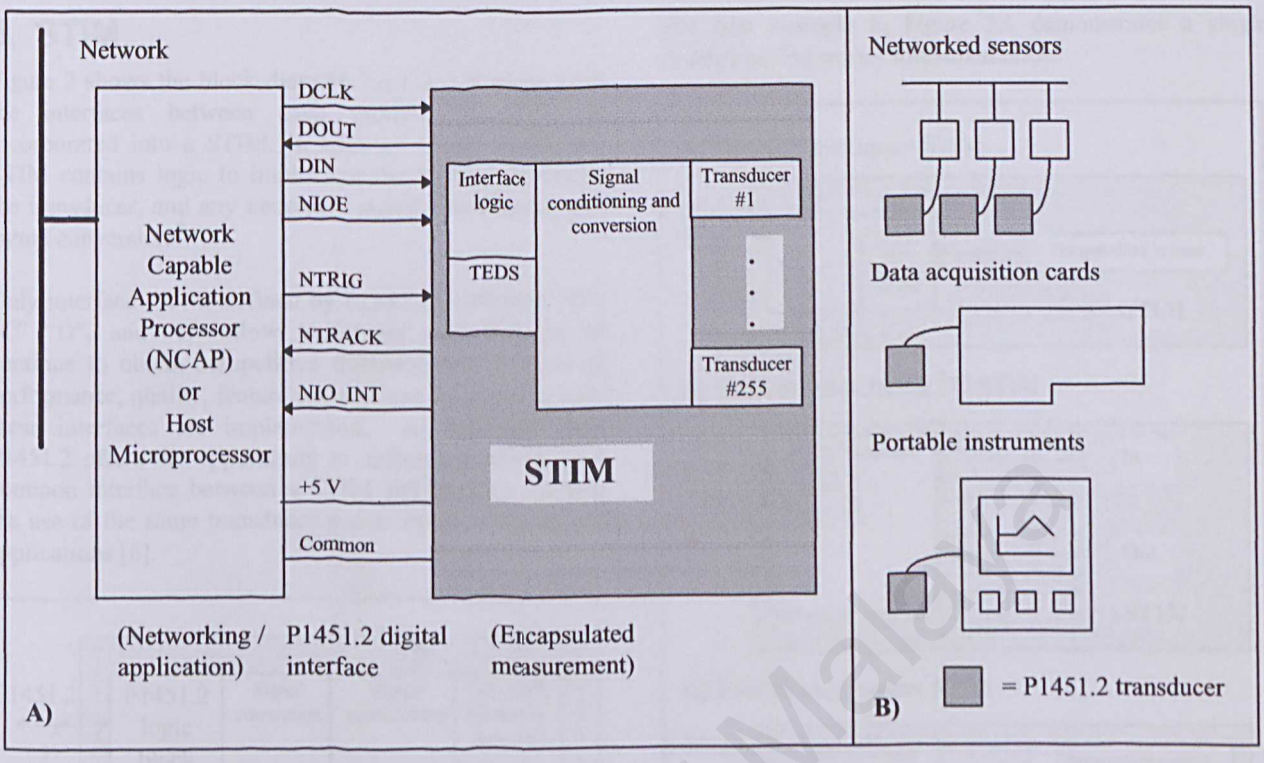



**Figure 1: Hardware partition proposed by P1451.2 and possible uses for the interface**

### 4.3 General calibration model

The P1451.2 draft provides a general model to optionally specify the transducer calibration. It is very flexible yet can collapse to an acceptable size for a simple linear relationship. The scheme supports a multi-variable, piece-wise polynomial with variable segment widths and variable segment offsets.

### 4.4 Triggering of sensors and actuators

The proposed digital interface has hardware trigger lines to allow the NCAP to initiate sensor measurements and actuator actions, and to allow the STIM to report the completion of the requested actions. The NCAP can trigger an individual channel, or all transducer channels at once. In the latter case, there are TEDS fields provided to specify timing offsets between the STIM's channels and to determine when each measurement or actuation has occurred relative to the single trigger acknowledgment. The draft proposes that the slowest channel be the reference channel and that all the offsets be specified relative to this channel.

### 4.5 Variable transfer rate between host and STIM

The hardware data clock line is driven by the NCAP. There is a field in the TEDS which specifies the maximum data transport rate that the STIM can support. This provides a flexible mechanism to match NCAPs and STIMs.

### 4.6 Support for multi-variable transducers

P1451.2 includes support for multi-variable transducers in a single STIM. A STIM may have up to 255 inputs or outputs allowing the creation of multi-variable sensors, actuators, or combinations thereof. Several multi-variable STIM examples are shown in Figure 3.

## 5. STIM

Figure 2 shows the block diagram for a STIM, along with the interfaces between each module. A TEDS is incorporated into a STIM. In addition to the TEDS, the STIM contains logic to implement the P1451.2 interface, the transducer, and any necessary signal conditioning and signal conversion.[8]

Only interface "A" is defined by P1451.2. Interfaces "B", "C" "D", and "E" allow transducer manufacturers to continue to obtain competitive differentiation in areas of performance, quality, feature set, and cost by choosing how these interfaces are implemented. At the same time P1451.2 offers the opportunity to design transducers to a common interface between a STIM and NCAPs enabling the use of the same transducer across many networks and applications [6].

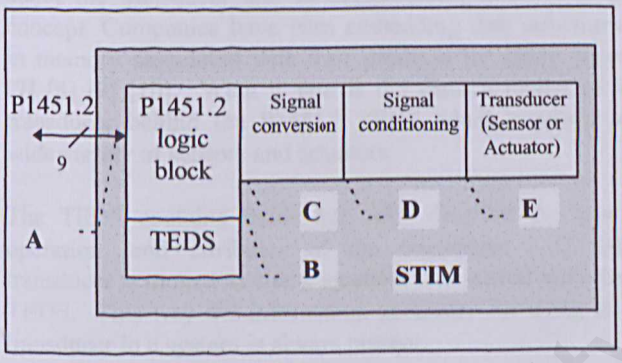



**Figure 2: STIM block diagram**

The P1451.2 logic block shown in Figure 2 may be implemented in several ways. The working group has now implemented STIMs using a field programmable gate array (FPGA) and a low-cost microcontroller to serve as the logic block. These methods demonstrate that P1451.2 STIMs can be built today using off-the-shelf parts. The microcontroller option provides the additional advantage of potentially combining all the logic, TEDS, and signal conversion into one integrated circuit, where the P1451.2 logic block is implemented using microcontroller firmware.

Figure 3 shows four examples of STIM configurations using a low-cost microcontroller. These examples demonstrate the flexibility in STIM design provided by P1451.2.

[8] Signal conditioning and signal conversion are not covered by P1451.2.

The first example in Figure 3A demonstrates a single channel analog sensor implementation.

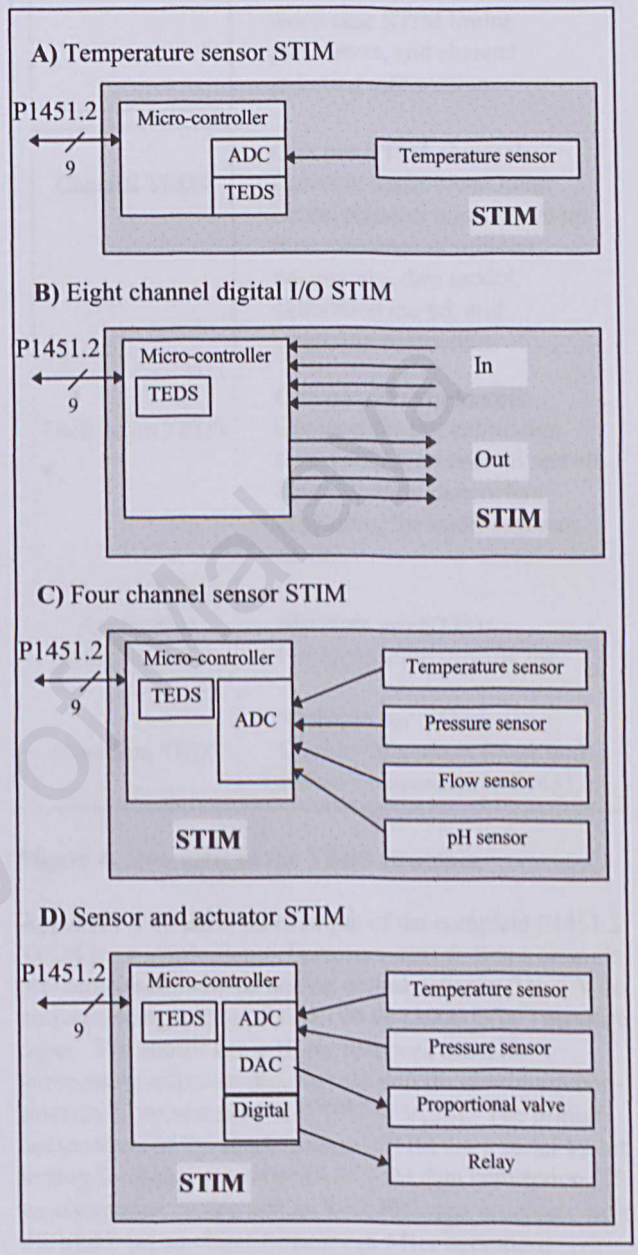



**Figure 3: STIM examples**

The second example in Figure 3B demonstrates the implementation of a digital input/output (I/O) module with four digital inputs and four digital outputs. The TEDS model in P1451.2 allows this STIM to be described as an eight-channel STIM or alternatively it could be described as a two-channel STIM with one input channel and one output channel each with a length of four. This flexibility

in the model allows digital I/O modules with thousands of inputs/outputs to be implemented if such a product were needed.

The third example in Figure 3C shows a STIM with multiple analog sensors. These four sensors could be measuring a process liquid.

Figure 3D illustrates that combinations of sensors and actuators can be combined into one STIM to support all the transducers used in control system solutions. The code implementing the control loop could reside either in the NCAP or the microcontroller used to implement the P1451.2 interface.

## 6. TEDS

The TEDS is one of the main technical innovations introduced in P1451.2. A TEDS, which carries information about the transducer and its performance, is not a new concept. Companies have been embedding data structures in memory associated with their products for many years [7] [8] [9] [10]. What is new is the general model of a transducer behind the P1451.2 TEDS which supports a wide variety of sensors and actuators.

The TEDS contains fields that fully describe the type, operation, and attributes of the transducer. If the transducer is moved to a new location, it is moved with the TEDS. This way the information necessary for using the transducer in a system is always present.

Figure 4 shows the main addressable sections of the TEDS along with examples of the content for each segment. The sections shown with dotted lines (calibration-TEDS, application specific-TEDS and extension-TEDS) are optional.

The calibration specification in the TEDS permits the sensor manufacturer to describe a multi-dimensional calibration for each channel. To eliminate high order polynomials it is possible to specify a segmented calibration where each segment can have a variable width and offset. It is expected that a general correction engine will be present in the NCAP that understands this calibration scheme so that it can be run "blindly" no matter which transducer is attached. An example of a multi-segment calibration curve with simple linear segments is shown in Figure 5.

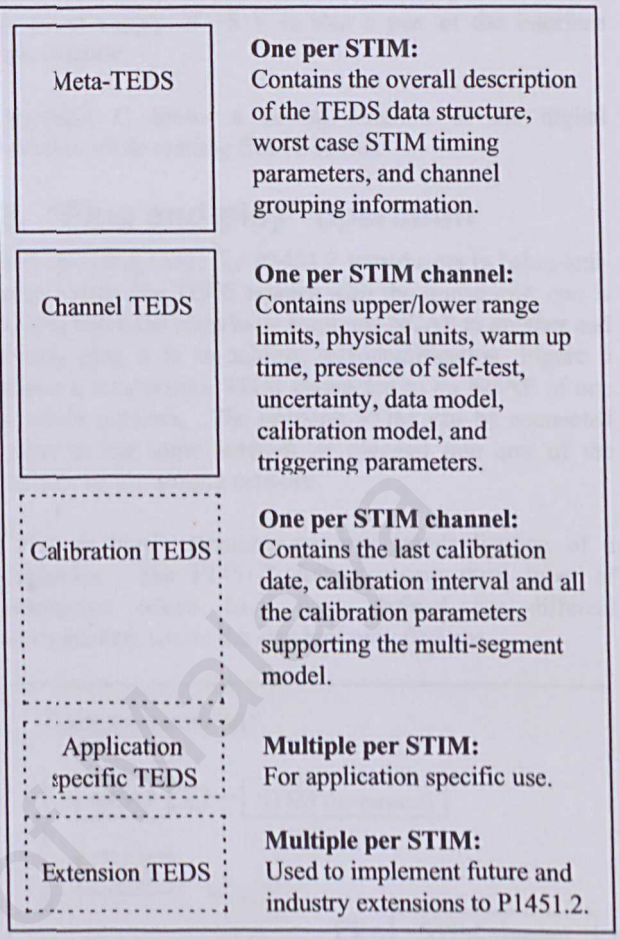



**Figure 4: Overview of the TEDS structure**

Appendix B contains an example of the complete P1451.2 TEDS for a single channel pressure sensor. It is a ceramic pressure sensor with an analog output between 0 to 5 V dc corresponding to 0 to 20,684,190 Pa (3000 lb/in$^2$) pressure input. The sensor has a 10 ms response time, no appreciable warm-up requirement and the maximum non-linearity is measured to be 0.56% of $V_{supply}$. The primary components of the single channel STIM are a serial 12-bit analog-to-digital converter (ADC) for data conversion (75 µs conversion cycle), and an 8-bit PIC-type processor with 4K by 12-bit on-chip EEPROM (8 MHz operation). Calibration is fixed and is specified using five equal segments with non-zero offsets for each segment. This allows first order calibration functions to be used to reduce the non-linearity in the analog output (0.56% reduced to 0.03%).

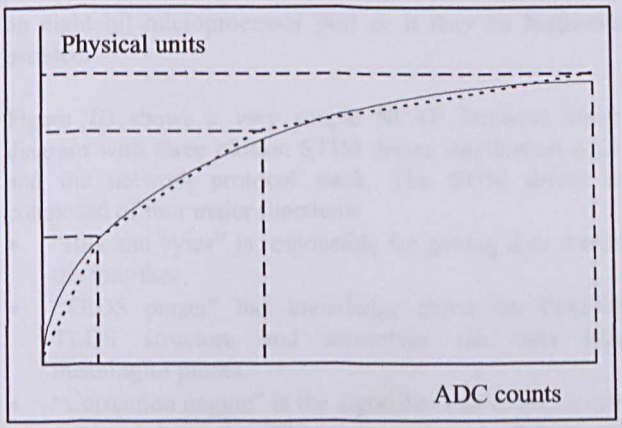



**Figure 5: Multi-segment calibration curve**

This specific TEDS implementation in Appendix B required the following amount of memory:

| | |
|---|---|
| Meta TEDS | 366 bytes |
| Channel TEDS | 96 bytes |
| Calibration TEDS | 103 bytes |
| ***Total*** | ***565 bytes*** |

Out of the 565 bytes, the manufacturer identification consumed 55 bytes, and product description consumed 205 bytes, for a total of 260 bytes or 46% of the TEDS.

The TEDS in Appendix B was created by Texas Instruments to demonstrate how a real transducer could be described by P1451.2. This does not imply that Texas Instruments will be offering this as a product.

## 7. Digital interface

Figure 1A shows the digital lines specified in the digital interface. Basic communications between the NCAP and STIM require four lines (DCLK, DOUT, DIN, and NIOE). DCLK is driven by the NCAP. The data transfers are based on SPI-like (serial peripheral interface), bit-transfer protocol.

The NCAP drives the NTRIG line to initiate a measurement or action, and the STIM uses the NTRACK line to acknowledge that the requested function has been performed. The STIM can notify the NCAP of any exception conditions by use of the NIO_INT line.

The P1451.2 draft defines a set of status registers to support notification of standard exceptions such as hardware errors, busy channels, STIM power cycle, calibration failure, and self-test failure.

A power supply of +5 V is also a part of the interface specification.

Appendix C shows a timing diagram of the digital interface while reading from a sensor.

## 8. "Plug and play" operation

The operating mode for P1451.2 transducers is "plug-and-play." Since the TEDS resides with the transducer, one is able to move the transducer from one NCAP to another and simply plug it in to achieve self-identification. Figure 6 shows a temperature STIM connected to an NCAP of one vendor's network. The pressure STIM can be connected either to the same network or plugged into any of the NCAPs on the second network.

"Plug & play" operation requires standardization of a connector. The P1451.2 draft proposes three types of connectors which have been defined for different environments where the standard may find use.

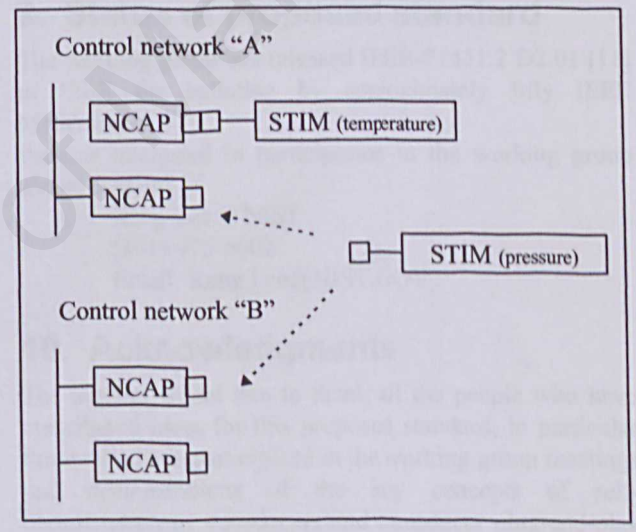



**Figure 6: "Plug-and-play" model of use**

It is important to note that P1451.2 may be implemented as a single row of pins or wires connecting two circuit boards (an NCAP and a STIM) together. This implementation could be used internally on a product where the separation between NCAP and STIM is not visible to the user. A company may want to build several networked transducers and use a common internal interface (P1451.2) to join the transducer portion with the network/application portion of the networked smart transducer.

Figure 7A shows the NCAP hardware support required to drive a STIM. This may be as simple as firmware driving

an eight bit microprocessor port or it may be hardware assisted.

Figure 7B shows a very simple NCAP firmware block diagram with three blocks: STIM driver, application code and the network protocol stack. The STIM driver is composed of four major functions:

- "Bits and bytes" is responsible for getting data across the interface.
- "TEDS parser" has knowledge about the P1451.2 TEDS structure and assembles the data into meaningful pieces.
- "Correction engine" is the algorithm that converts raw readings from the STIM into units specified in the TEDS for sensors or units specified in the TEDS into STIM settings for actuators.
- "P1451.2 application programming interface (API) driver" provides access to TEDS blocks, sensor readings, actuator control, triggers and interrupt requests.

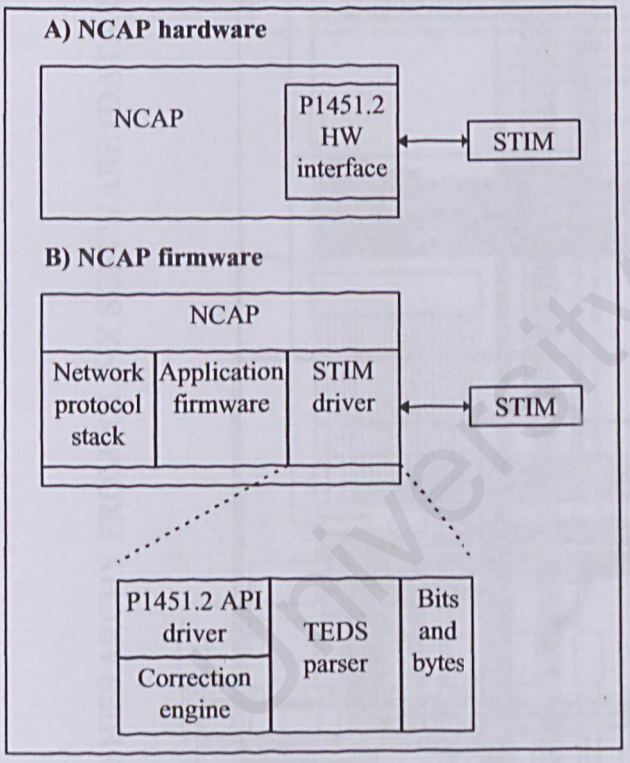



**Figure 7: NCAP support required for P1451.2**

Most of the TEDS' contents need not be parsed and kept on the NCAP. This will depend on the measurement or control problem being solved and differentiation desired in NCAP products. If the application needs the contents of the TEDS for use in another part of the system, then the TEDS may be read from the STIM and sent out over the network with very little parsing. If the NCAP will be making sensor measurements and sending sensor readings in engineering units, then the calibration factors must be parsed and kept on the NCAP for use by a correction engine.

In principle only one STIM driver for each kind of NCAP is required to support P1451.2. For each microprocessor family supporting this proposed standard there could be a single software driver for the P1451.2 interface, a single TEDS parser, and a single conversion engine.

It is expected that the drivers for the popular networks will be developed by the network providers. In past demo implementations, three network providers developed IEEE-1451.2 drivers for their network's NCAPs: Allen-Bradley Company (DeviceNet™), Echelon Corporation (LonWorks™) and Honeywell Micro Switch Division (Smart Distributed System™).**

## 9. Status of proposed standard

The working group has released IEEE-P1451.2 D2.01 [11] to IEEE for balloting by approximately fifty IEEE Members.

Persons interested in participation in the working group please contact:

Kang Lee at NIST
(301)-975-6602
Email: Kang.Lee@NIST.GOV

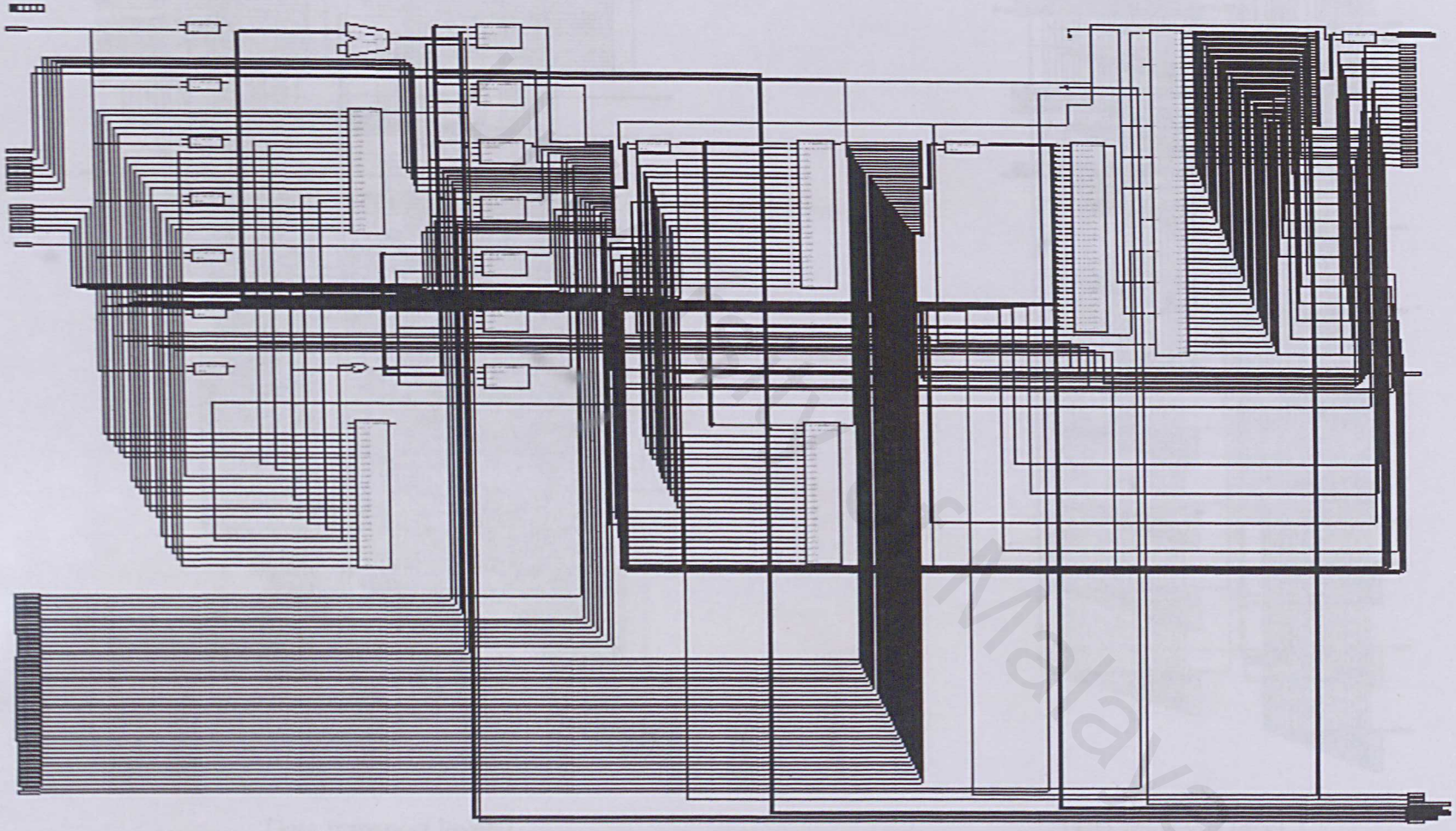
## 10. Acknowledgments

The authors would like to thank all the people who have contributed ideas for this proposed standard, in particular those who have participated in the working group meetings and demonstrations of the key concepts of self-identification of transducers and transducer plug-and-play operation on multiple networks.

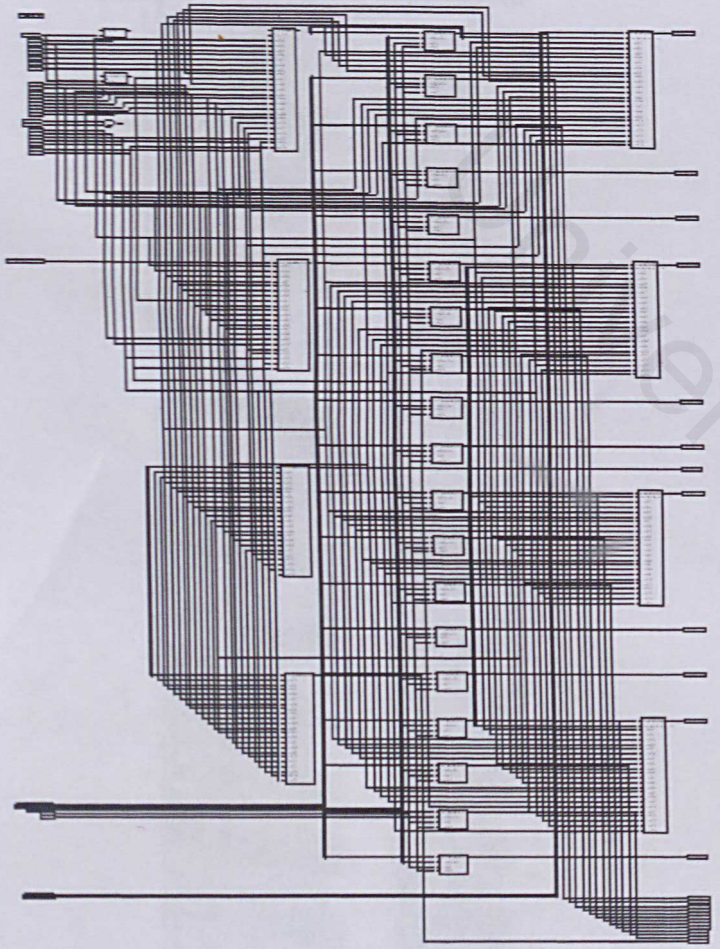
## 11. Summary

We have presented an overview of the technical aspects of IEEE-P1451.2. In addition, some of the practical considerations for implementation and support of the standard have been discussed. The working group has made progress and has reached the first official IEEE ballot milestone permitting a wider audience to obtain a copy of the working group's efforts.

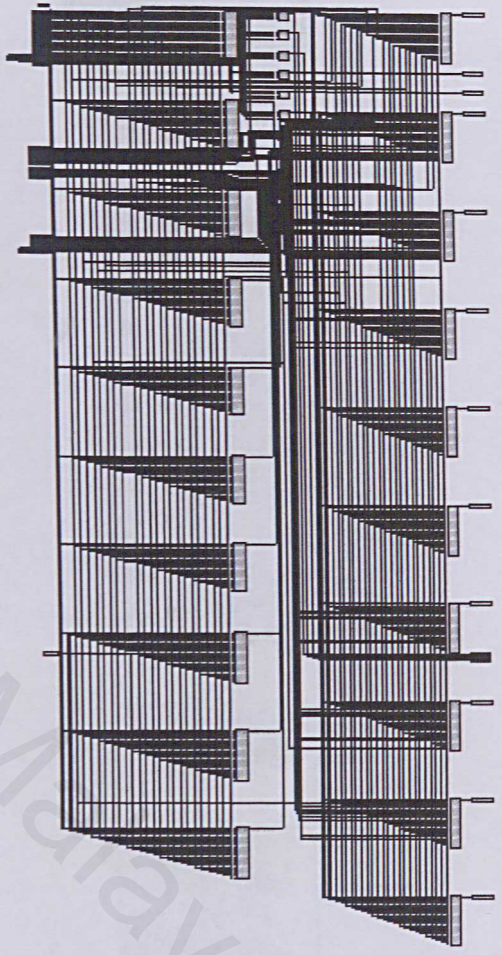
RTL DESIGN HIERARCHY FROM XILINX SOFTWARE (DATA TRANSPORT)



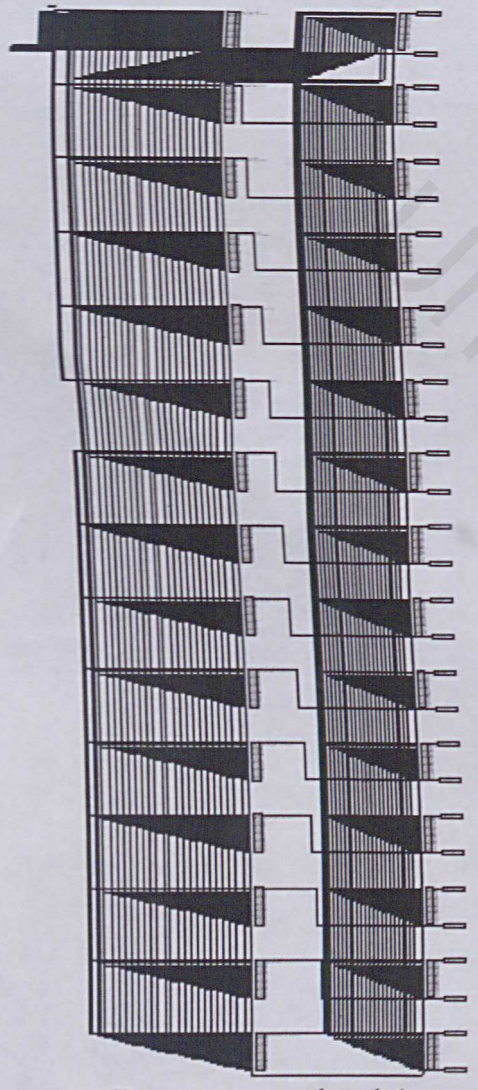
Data transport level 1



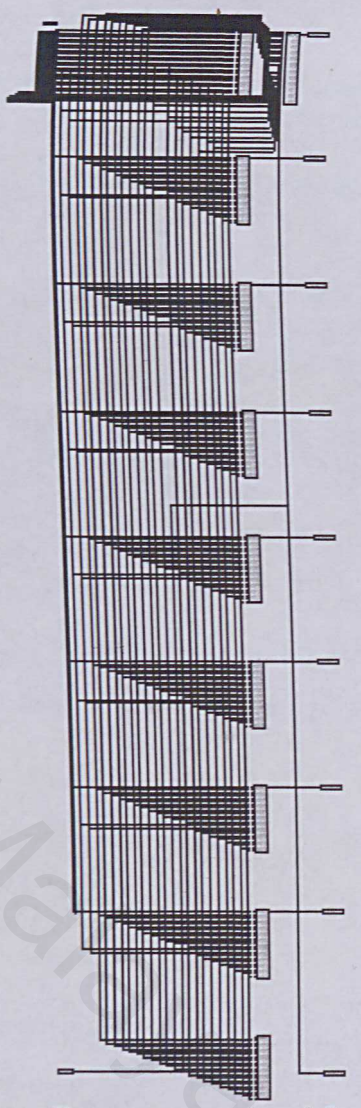
Data transport level 2



Data transport level 3



Data transport level 4



Data transport level 5