AN FPGA-BASED DIGITAL TRIGGERING SYSTEM WITH MODEL-INTEGRATED CONFIGURATION ENVIRONMENT FOR THE CONTROL OF NIM ELECTRONICS

by
Lerato Jerfree Mohapi

A dissertation submitted to the Department of Electrical Engineering
in fulfilment of the requirements for the degree of
MSC (ENG) ELECTRICAL ENGINEERING
at the
UNIVERSITY OF CAPE TOWN

Supervisors : Dr. Simon Winberg (UCT) and Mr. Sean Murray (iThemba LABS)

August 2012

Declaration

I know the meaning of plagiarism and declare that all the work in this dissertation, save for that which is properly acknowledged and referenced, is my own. It is being submitted for the degree of Master of Science in Electrical Engineering at the University of Cape Town. This work has not been submitted before for any other degree or examination in any other university.

Signature of Author: .....
University of Cape Town

Cape Town
August 2012

ABSTRACT

Nuclear Instrumentation Module (NIM) is a standard that defines mechanical and electrical specifications for electronic modules used in experimental particle and nuclear physics. Together with other standard analogue electronics such as pre-amplifiers, NIM electronics are used to acquire the electrical charge pulses generated by detectors, extract the quantities of interest and convert them into a digital format for processing. Detectors normally generate large volumes of data thus making it difficult for these electronic modules to cope in processing of this data. Although general purpose computers are becoming more parallel and their processing throughput continues to increase, these systems remain unsuited to the high processing rates and large data volumes for real-time processing of nuclear physics experiments.

This dissertation presents a project to develop a real-time trigger that is hardware reconfigurable, triggering on user specified events, and captures data for permanent storage and later processing. This triggering platform is planned to replace previous analogue triggering systems that involve time-consuming manual tasks of connecting analogue electronics with NIM components. These manual tasks involve a multitude of wiring connections and their timings are error prone. Multiple on-going experiments could not time-share the expensive NIM electronics, implying lengthy waits between experiments and inefficient resource usage. The new triggering platform provides significant time saving for physicists setting up experiments, together with a model-based system that speeds up the design and setting-up of experiments, while also reducing the wear and tear of dismantling and connecting equipment.

The research methodology involved studying manual processes used by physicists and engineers to set up experiments. The new triggering platform, which automates parts of these manual processes, was tested experimentally using radioactive sources and its results compared the same experiment using the old system. Experiments to test the new system used ^{22}Na radioactive sources, and the electronics involved were represented as blocks in the Scilab model. On average, the VHDL code was generated in $50ms$, synthesized in $16s$, installed on the FPGA in less than $5s$. Initial experiments showed timing problems, particularly long latencies and jitter. Solutions for these problems are briefly described in this dissertation. The synthesis results for iThemba LABS AFRODITE experimental model demonstrated space optimized VHDL code generation by occupying 10% of total FPGA LEs. These results showed that the new triggering platform allows time-sharing of multiple electronics in small to large nuclear experiments. Therefore user requirements were met in this project.

ACKNOWLEDGEMENTS

I would like to gladly express my gratitude to the following people who assisted towards successful completion of this research project:

- **Dr. Simon Winberg**, my UCT supervisor, for his guidance, and encouragement during the entire research project
- **Sean Murray**, my iThemba LABS supervisor, for his assistance, zeal to see a finished product, and motivations to his management for my financial assistance
- **National Research Fund (NRF)**, through **iThemba LABS**, for financial assistance
- **'Makhoarai**, **'Malerato**, and **Khoarai Mohapi**, my family, for their support, always being there for me in times of need and proof reading this thesis
- **Professor Mike Inggs**, my co-supervisor at UCT for acceptance into this course and the RADAR and Remote Sensing group, for his guidance in the beginning of this project even towards the end of the successful completion of this project
- **Colleagues**, J. Mira, R. Neveling, and experimental nuclear physicists at iThemba LABS for their guidance in this project
- **Colleagues**, M. Senekane, and EIT group at iThemba LABS for their guidance and proof-reading this dissertation

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- **AFRODITE** – AFRican Omnipurpose Detector for Innovative Techniques and Experiments
- **ADC** – Analogue to Digital Converter
- **CFD** – Constant Fraction Discriminator
- **COIN** – Coincidence Logic
- **CTP** – Central Trigger Processor
- **DCFD** – Digital Constant Fraction Discriminator
- **DCPT** – Digital Central Trigger Processor
- **DPP** – Digital Pulse Processing
- **ECL** – Emitter-Coupled Logic
- **FPGA** – Field Programmable Gate Array
- **FSM** – Finite State Machine
- **GUI** – Graphical User Interface
- **HDL** – Hardware Description Language
- **HPGe** – High-Purity Germanium Detectors
- **I/O** – Input/Output
- **LABS** – Laboratory for Accelerator Based Sciences
- **LAN** – Local Area Network
- **LVDS** – Low-Voltage Differential Signalling
- **NIM** – Nuclear Instrumentation Modules
- **PDE** – Programmable Delay Elements

- **PDL** – Programmable Delay Line
- **PLL** – Phase-Locked Loop
- **RCP** – Rapid Control Prototyping
- **TAC** – Time-to-Amplitude Converter
- **TTL** – Transistor-Transistor Logic
- **VMAGIC** – VHDL Manipulation and Generation Interface
- **VME** – Versa Module Eurocard
- **VHDL** – VHSIC Hardware Description Language

NOMENCLATURE

- **Bridge FPGA** A V1495 FPGA used for the VME interface and for the connection between the VME and the 2nd FPGA (FPGA "User") through a proprietary local bus.
- **CAEN** – A leading company in the design and manufacture of sophisticated electronic instrumentation for Subnuclear, Nuclear, and Astroparticle Physics.
- **Digital Pulse Processing** - signal processing technique in which detector signals are directly digitized and processed to extract quantities of interest.
- **Field Programmable Gate Array** - An integrated circuit that is designed to provide flexibility by allowing the user to reconfigure it for desired application.
- **Gamma rays** – Electromagnetic radiation, made up of extremely high frequency waves, and carry a large amount of energy.
- **Isotopes** – Atoms of the same element with different numbers of neutrons.
- **Phase-Looked Loop(PLL)** – A frequency control mechanism configured as multipliers, demodulators, tracking generators, or clock recovery circuits.
- **Positrons** – positively charged subatomic particle having the same mass and magnitude of charge as the electron and constituting the antiparticle of a negative electron.
- **true-DSP Synthesis** – A design methodology which addresses the problem by automating the generation of an RTL implementation from a high level description of DSP algorithms.
- **User FPGA** – A second V1495 FPGA used for managing the front panel I/O channels and user desired logic function.
- **V1495** – a VME 6U board, 1U wide, suitable for various digital Gate/Trigger/Translator/Buffer/Test applications, which can be directly customised by the User, and whose management is handled by two FPGAs.
- **Versa Module Eurocard (VME)** – A computing systems architecture that consists of electrical specifications for data bus and mechanical specifictions for describing the backplane, bus connector board sizes and enclosers.

CHAPTER 1

INTRODUCTION

This dissertation focuses on the development of a real-time reconfigurable hardware triggering platform for use in nuclear physics experiments for capturing and storing data for later processing. This triggering platform uses synthesizable Scilab models that run on a field programmable gate array (FPGA) platform to provide a faster method for setting up NIM-based digital triggers. Although computers are becoming more parallel and their effective data processing throughput continues to increase, they are also not suited to the large data volumes processing rates needed for real-time processing for nuclear physics experiments. The real-time trigger setup presented in this dissertation is introduced for extraction of the quantities of interest from data generated by detectors for permanent storage and discarding unwanted data events. The previous real-time trigger assists analogue acquisition chains to cope with the processing of large data volumes in nuclear physics experiments. This previous trigger setup involves time-consuming manual tasks of connecting analogue electronics with standards such as Nuclear Instrumentation Module (NIM). NIM is a mechanical and electrical specification for electronic modules used in nuclear and experimental physics.

iThemba Laboratory for Accelerator Based Sciences (iThemba LABS) is the group of multidisciplinary laboratories administered by the National Research Foundation of South Africa, based in Gauteng and Western Cape provinces. At iThemba LABS, NIM is inherently present in most electronic modules used in experimental physics trigger setups. In spite of the advantages provided by NIM such as flexibility and interchangeability to iThemba LABS, major drawbacks include manual tasks of wiring-up connections which are also error prone.

This chapter gives an outline of the background information of the technologies used in this project. The next section to follow will be the problem description. Objectives and the overview of methodology used in this project are the next sections discussed. The other section included in this chapter is the one which discusses the scope and limitations of this project, and this will be followed by the dissemination strategy. The last section of this chapter provides an overview of this dissertation.

1.1 BACKGROUND

When NIM electronics are used in experimental physics and other medical applications, an optimum system can be configured for a particular application and later its instruments can easily be restructured as required for different experiments or measurements. Users can also update an existing system with a few new modules, thereby augmenting the value of instrumentation on hand [14]. As shown in [15], NIM electronics are created using different mechanical coverings (housing) with physical knobs and electronic circuits inside. These NIM electronics are connected together using wires in experimental and nuclear physics applications to setup a real-time trigger. Therefore, a real-time trigger is composed of traditional analogue NIM electronics chains.

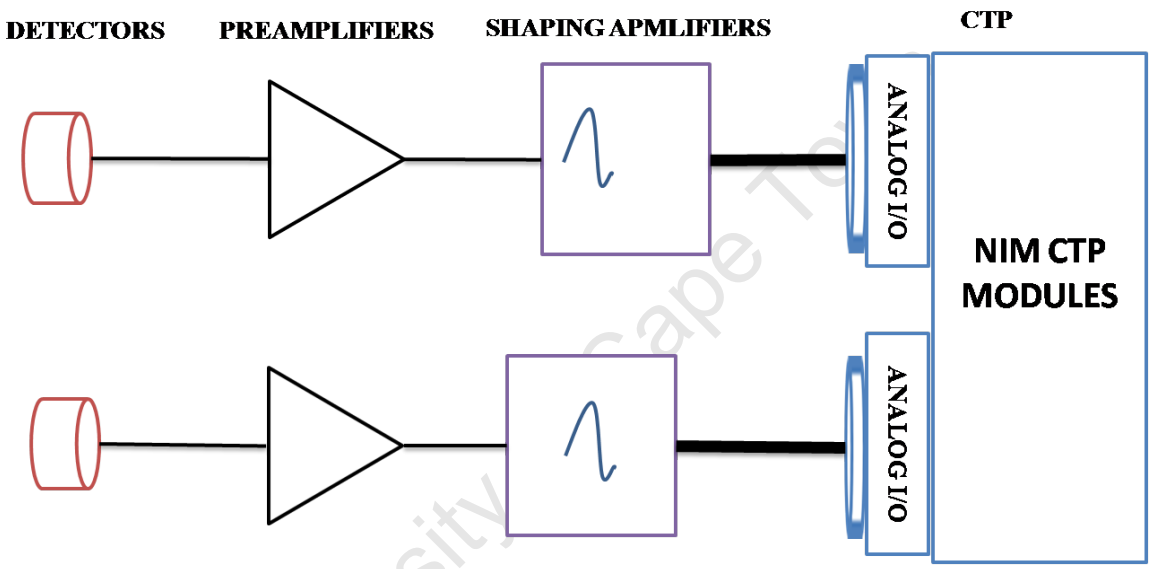


Figure 1.1: Overview of standard analogue NIM electronics trigger setup

The elements of the previous real-time triggering setup are depicted in Figure 1.1. The trigger setup is composed of signal sources as detectors (e.g. High Purity Germanium detectors). After the amplification of the signal by the pre-amplifiers, most designs will use the shaping amplifier to shape, filter and amplify the signal. The peak sensing ADC of the central trigger processing (CTP) takes over. The CTP is composed of several NIM electronics housed in one or more electronic crates. In most cases, the constant fraction discriminators (CFD) act as the Peak sensing ADC and the first modules closer to the shaping amplifiers. The CFDs produce accurate timing information from analogue signals of varying heights but the same rise time. Everything after the CFD performs conditioning of the signal and necessary operations to determine the trigger output. The trigger output is simply to determine whether there is a valid user specified physics event.

In some cases, the triggering system provides a Yes/No response to a trigger pattern; For these situations memory is adequate. However, storing and synchronizing large data streams becomes prohibitively cumbersome for even small size multi-parameter experiments [3]. The FPGAs become a solution to overcome this throughput limitation [4]. These high density chip architectures combine logic cells with storage cells that can be used for high-speed interconnections of

several Digital Pulse Processing (DPP) algorithms inside the same logic chip. DPP is a signal processing technique in which detector signals are directly digitized and processed to extract quantities of interest.

1.2 PROBLEM DESCRIPTION

For a complete nuclear physics experiment to be performed successfully at iThemba LABS, there is currently a need for repeatable trigger setups. These trigger setups involve time-consuming manual processes of connecting standard analogue NIM electronics. These manual tasks involve a multitude of wiring connections and their timings are error prone. Multiple on-going experiments can not time-share the expensive NIM electronics, implying lengthy waits between experiments and inefficient resource usage.

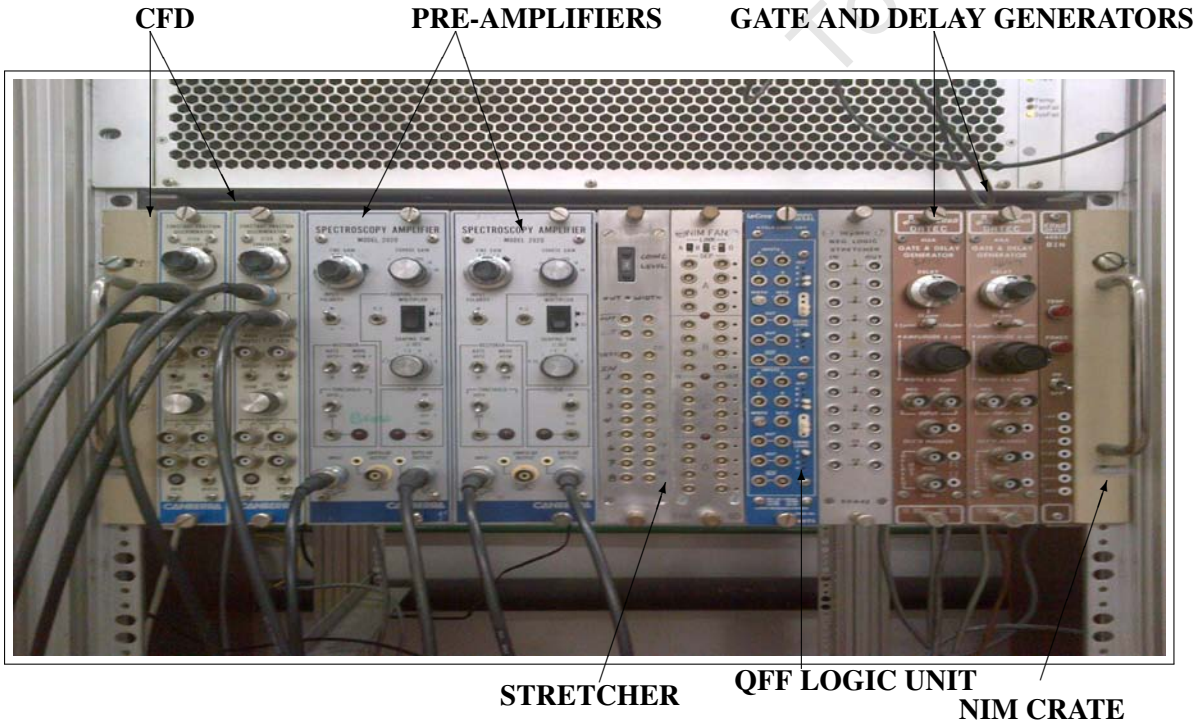


Figure 1.2: Standard analogue NIM electronics plugged in a Crate

The NIM electronics used in a trigger system are plugged into a NIM Crate or NIM Bin. As depicted in Figure 1.2 the setup occupies a large area of space and there is multiple wirings. Although companies such as CAEN, WEINER, etc. offer a range of digitizers running DPP firmware for physics applications, these digitizers are expensive. These commercial digitizers require deep knowledge of the digital algorithms (DPP firmware) and the relevant parameters, VHDL knowledge, and extra support from companies such as CAEN or WEINER, depending on where the digitizer used was manufactured.

1.3 FOCUS

This project focuses on the implementation of the central trigger processor (CTP) which include the sampling ADC, the timing adjustments to condition detector signals, CTP decision-making, and the readout modules using a general purpose trigger board named V1495. V1495 is a general purpose Versa-Module Europa (VME) trigger board which can be directly customised by the user, and whose management is handled by two FPGAs, User and Bridge FPGAs. The V1495 is manufactured by CAEN and is suitable for various digital applications such as triggers, translators, buffer, etc. in experimental physics.

Furthermore, the use of a rapid control prototyping environment, Scilab, will be used to implement models of the CTP modules. The advantages of this is to hide the underlying trigger configuration technicalities while providing physicists with the user paradigm very close to their old NIM electronics and wiring. Therefore modelling of NIM electronics and automatic hardware description language (HDL) code generation of the modelled trigger logic are also the center of interest in this research.

1.4 OBJECTIVES

The main objective of this project is to provide significant time saving for physicists setting up experiments, together with a model-based system that speeds up the design and setting-up of experiments, while also reducing the wear and tear of dismantling and connecting equipment. Therefore this project will go through the following steps towards achieving the major goals:

- Review and implementation of FPGA-Friendly algorithms of the logic circuits residing in the NIM electronics at iThemba LABS
- Design and implementation of NIM electronic models with HDL internals
- Development of a trigger GUI control of logic analyser for trigger on-chip debugging
- The implemented triggering system evaluation by functional verification and experimental results analysis

1.5 HYPOTHESIS

The use of DPP modules is under rapid development to replace modular analogue electronics. DPPs are used because of their ability to allow implementation of signal filtering functions that are not possible through traditional analogue signal processing. DPPs also provide less overall processing time with higher throughput without significant resolution degradation and stability. The companies such as CAEN [34] and WIENER [35], use FPGA technology to produce digitizers running DPP firmwares. These digitizers accept signals directly from preamplifiers or photomultipliers and implement a digital replacement of several analogue modules [1].

The digitizers have great stability and reproducibility as well as the ability to reprogram and tailor the DPP algorithms in each module. This technique therefore allows one digitizer board to

do the jobs of several analogue modules thus reducing the cost of implementation of data acquisition systems in experimental physics and other medical applications. One other board CAEN implemented is the general purpose VME board, V1495 [16], suitable for implementation of digital triggering and variable input/output channels that can be used in this project.

Even though there are not many digital, rather HDL synthesizable versions of NIM logic circuits residing in most of analogue NIM electronics at iThemba LABS that can be programmed into the V1495, this board can be used to replace most of these modules and perform basic triggering. The major question is whether a complete trigger model HDL code can be generated within an acceptable time, synthesized and successfully fit into the fabric with acceptable timing analysis? In addition, can trigger timing adjustments be made accurately to assure that the correct data is recorded and low jitter operable trigger logic is implemented successfully?

1.6 Methodology Overview

The research methodology involves studying manual processes used by physicists and engineers to setup physics experiments. The basic idea is to present previous manual trigger operations in a codified executable form while providing physicists with the user paradigm very close to their old NIM electronics and wiring. The iterative spiral [19][20] model of software develop-

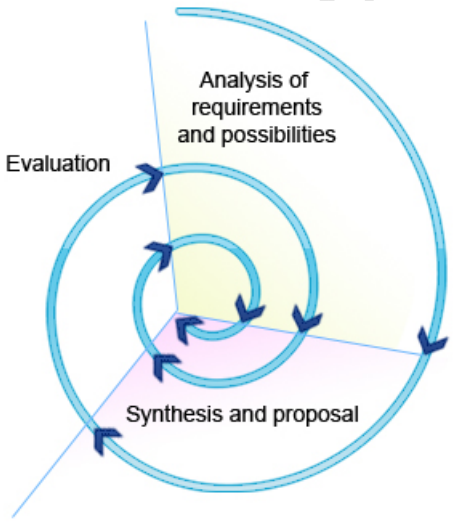


Figure 1.3: Overview of research methodology (Image based on [20])

ment depicted in Figure 1.3 will be followed to implement the new triggering platform with NIM components. This model will permit iterative triggering platform development and cyclic progress assessment. The triggering platform will be slice partitioned into each individual NIM component development process outlined below:

- Analysis of requirements and possibilities: Investigate on functionality by studying each standard analogue NIM electronic component and investigate on whether it can be implemented using HDL.
- Synthesis and proposal: Propose a solution and investigate whether the implemented model can be synthesized.

- Evaluation: Compare delays of the digital NIM component with standard NIM electronic component.

When each NIM component is implemented in a Hardware Description Language (HDL), the Scilab model of that component will be implemented as well. The model will then be integrated with the synthesizable HDL code using the chosen HDL code library. Thus providing a complete synthesizable NIM model. These steps are repeated until all NIM models with underlying HDL code are implemented. The complete testing is carried out in each of the iterations. The testing involves a model comprising of the NIM block connected to input and output blocks, and generating the code for them. This iterative process facilitates better monitoring, data mining and corrective actions in the early hours and better requirements management in an incremental mode [20]. For the entire project, the requirements analysis, design process, HDL coding, modelling and testing will be undertaken in a rigid loop.

1.7 SCOPE AND LIMITATIONS

This project will only provide NIM electronics models and DSP-based NIM electronic (NIM DPP firmware) components that can be used in any FPGA-based trigger board. However, the top-level entity will depend on the trigger board chosen. For instance, in this project the top-level entity was designed for use with the V1495 module. The V1495 general purpose VME trigger board [16] will be used as it fully complies with the required digital input/output (I/O) channels such as NIM, Transistor-Transistor Logic (TTL), Low-Voltage Differential Signalling (LVDS) and Emitter-Coupled Logic (ECL). The V1495 trigger board uses Altera Cyclone EP1C20 as User FPGA, implying that, the system will have to use the Altera Quartus II FPGA design software to synthesize the HDL Code generated by the implemented triggering platform model interpreter tool with automatic code generator. The insitu[1] NIM testing will have to be implemented such that benefits provided by NIM electronics such as flexibility and interchangeability are preserved.

1.8 DISSEMINATION STRATEGY

This section shows the plan to disseminate the work done in this dissertation, especially in academic forums such as conferences, journals, etc. An abstract has been submitted and accepted for presentation in the 4^{th} International Conference in Science and Technology. The strategy involved a first paper which was submitted to the Reconfig 2011 conference. This first paper was submitted in the early months (May 2011) of this project and was not mature enough to get accepted. In preparing for presentations in the ICSTIE 2012 conference, a paper presentation to the Radar and Remote Sensing group (RRSG) will be done. The final phase of this project will also be the training to increase the spread of the research and knowledge of the new design criteria to the physicists and researchers at iThemba LABS. Thus a seminar will be organised at iThemba LABS with audience as physicists and engineers.

[1] Occurs without interrupting the normal state of a system - this case state of NIM

1.9 THESIS OVERVIEW

This dissertation comprises six chapters. Chapter 1 starts off with outlining the background information followed by the problem statement. Then it goes on describing the project objectives and hypothesis. The methodology overview and dissemination strategy were also covered. The remaining 5 chapters are as follows:

Chapter 2 reviews the critical points of the trigger concepts including substantive digitizers running DPP firmware, as well as theoretical and methodological contributions of Rapid Control Prototyping to FPGA based system and NIM electronics.

Chapter 3 entails mainly guidelines for identifying the major requirements of the triggering system and how to go about designing it as well as specifying the evaluation methodology.

Chapter 4 outlines a fully-functioning working overall system architecture design and implementation as well as showing detailed schematics and important design parameters. Code snippets and algorithms are discussed for clarification.

Chapter 5 describes how the specified evaluation methodology was used for checking that the implemented triggering platform user requirements and specifications were met, and that it fulfils its intended purpose in the trigger design and implementation. This chapter verifies the timing safety of the generated very high scale integrated circuit HDL (VHDL) code beatstream and shows how the time to reach the trigger decision was manipulated to assure accurate data pattern recording. Results analysis is also done in this chapter.

Chapter 6 summarizes the findings as well as describing possible extensions to the triggering platform project.

CHAPTER 2

LITERATURE REVIEW

This chapter introduces key trigger theory concepts relevant to experimental physics trigger setups. Section 2.1 gives an overview of the triggering systems concepts. Section 2.2, gives a review of substantive digitizers running DPP firmware. This chapter contains Section 2.3 as the last section that discusses the concepts and choice of RPC environment to be used.

2.1 CONCEPTS OF TRIGGER SYSTEMS

A real-time trigger system is introduced in nuclear experiments to select the interesting interactions for permanent storage and to discard the background [2]. The trigger system key concepts include

- Timing: Trigger systems consist of more than one detector, hence time to reach trigger decision is dependent on the path taken by the detectors signals to reach the CTP decision making modules. Hence timing of signals using delays, stretchers and gate and delay generator must be performed.
- Clocked Readout: Well defined in [2], and also used in the AFRODITE experiment that is depicted in Appendix I.1 and explained in [33][42]. Clocked readout uses radio frequency to perform timed readout of trigger patterns or values of interest, by providing start and stop signals to a Time-to-Amplitude Converter readout module (see Figure I.1).
- Event-Trigger Readout: These modules include bit pattern registers that record a given input pattern given a valid user event.
- Dead-time: Finite reaction time of NIM electronic components due to the discharging of capacitors or the re-establishment of the initial conditions after processing event [2].
- CTP Decision-Makers: Trigger systems often consist of more than one detectors, the decision to determine a valid user event is done by performing the AND operations. Reference [2] defines other CTP decision making modules as follows:
  - AND and OR of two or more signals;
  - Majority logic: (n out of m detectors have responded, with $n \leq m$);

  - Threshold: n lines encode a digital value, for example energy E, and the condition $E \succ E_{thr}$ is used to trigger;
  - Topological patterns with a well-known signal profile.

- Buffering: Bit patterns of interest in a triggering system need to be buffered to avoid capturing data even when an invalid event occurred. The buffer therefore allows the trigger and readout path to cope with the large ensuing instantaneous rate variations of signals coming from detectors [2].

Some or all of these concepts are used in triggering systems for provision of experimental constraints and definition of the trigger timing adjustments, with the goal of selecting interactions of interest and reducing background. For the combination of trigger signals from different detectors a central component, often called central trigger processor, is needed. Figure 2.1

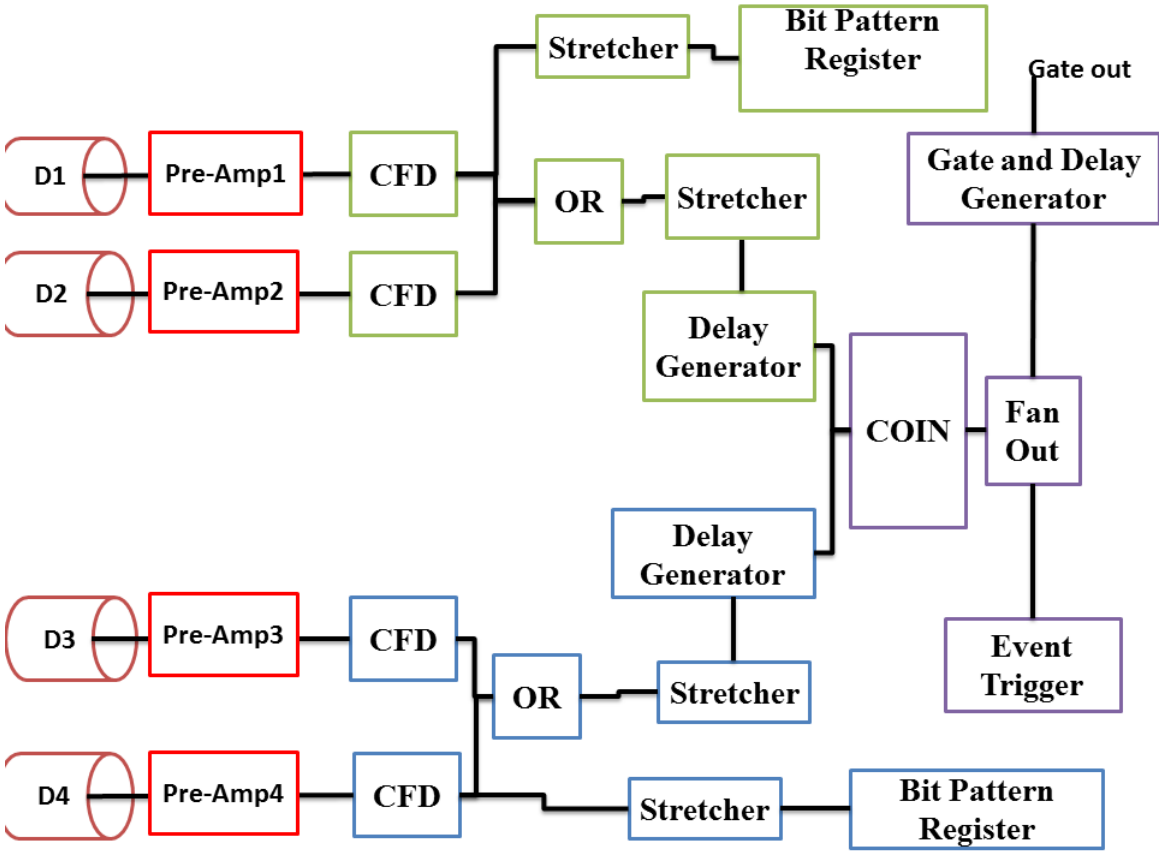


Figure 2.1: Overview of trigger electronics setup

depicts a more practical view of triggering systems NIM components illustrated in Figure 1.1. As depicted in Figure 2.1, the trigger is composed of four detectors, partitioned into pairs thus forming two paths (First path along blue components and second path along green components) in the CTP. Due to ensuing instantaneous rate variations of signals coming from detectors, cable delays and some delays from CFD (Sampling ADC) operations, the CTP timing modules, pulse stretchers and delay generators, are used to ensure accuracy in trigger decision making. Event

trigger based readout, the BPR, buffers the bit-pattern of interest until the next rising-edge of event-trigger module output.

The detectors, D1 to D4, depicted in Figure 2.1, normally detect some γ-rays with energy E_x, hence are called energy dispersive. Energy dispersive detectors include High Purity Germanium (HPGe), Silicon (Li), Mercuric Idiode (HgI_2), etc. Before being processed for triggering, signals from energy dispersive detectors are pre-amplified using charge sensitive pre-amplifiers [43][37].

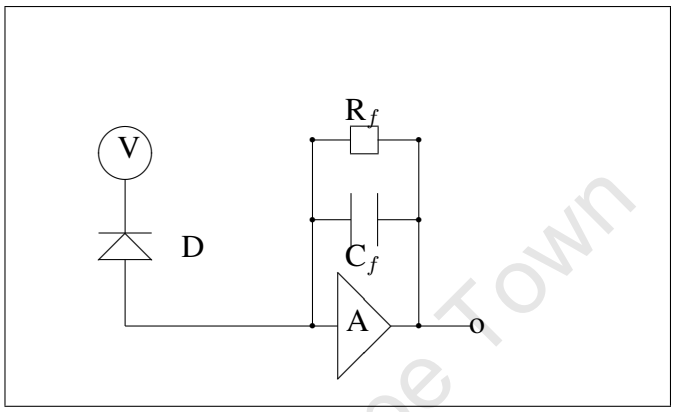


Figure 2.2: Charge sensitive pre-amplifier with RC feedback [37]

Assuming a detector D, as illustrated in Figure 2.2, biased with voltage source V and connected to the input of pre-amplifier A, which has feedback capacitor C_f and feedback resistor R_f. When the γ-ray is absorbed in the detector material it releases an electric charge $Q_x = E_x/\epsilon$, where ϵ is a material constant. Q_x is integrated onto C_f, to produce the voltage $V_x = Q_x/C_f = E_x/(\epsilon C_f)$. Measuring the energy E_x of the γ-ray therefore requires a measurement of the voltage step V_x in the presence of the amplifier noise σ. Reducing noise in an electrical measurement is accomplished by filtering. Traditional analogue filters use combinations of a differentiation stage and multiple integration stages to convert the pre-amplifiers output steps into either triangular or semi-Gaussian pulses whose amplitudes (with respect to their baselines) are then proportional to V_x and thus to the γ-rays energy [37]. These filters reside in the shaping amplifier as depicted in Figure 1.1, but Figure 2.1 only shows use of pre-amplifiers just after the detectors, this is because modern pre-amplifiers perform both amplification and shaping of the signals.

The shaped output pulses are then sent to the peak sensing analog-to-digital converter (ADC) of the CTP. The peak sensing ADC are a type of specialized ADC that sample only around signal peaks, outputting values based on the peak amplitude, to account for the high speed of the incoming signals. The CFD is often used as the sampling ADC in a trigger system. The CFD circuit provides amplitude invariant, and in some cases, rise-time invariant timing. In a standard CFD, an attenuated version of the input signal is compared to the delayed version of the same input signal[64]. This creates a bipolar timing signal with time invariant zero crossing. This means that a timing mark is developed by determining when these two signal are equal. These timing mark is equivalent to determining when the difference of these two signal is zero. As

visually illustrated in Figure 2.3 a standard CFD circuit consists of a delay generator, attenuator, two inverters as comparators, and an AND gate to determine the CFD output. As depicted in

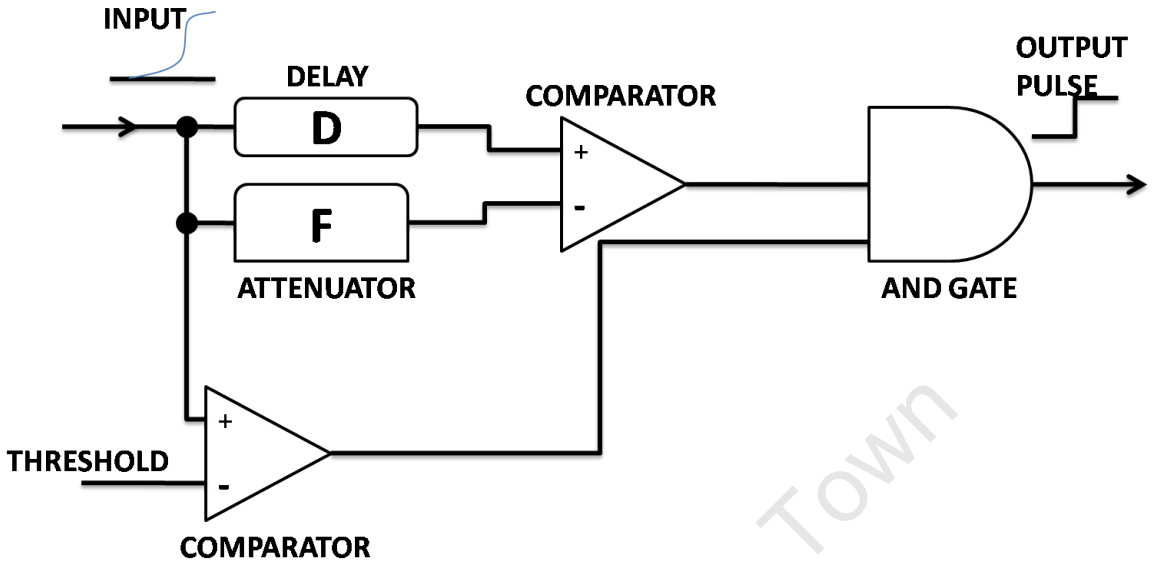


Figure 2.3: A standard Constant Fraction Discriminator circuit diagram [64][65][66]

Figure 2.3, the input signal is split into the delayed, attenuated (multiplied by some fraction f) and original version. An AND operation is performed on the output of the comparison of the delayed signal with the attenuated signal, and the original signal with the threshold value. The resulting output pulse is called the CFD output. In the single level trigger systems, this CFD output can sometimes be used as the overall trigger output. In multi-level triggers such as the ones explained in [33] and [39], often when the signals from the detectors are sampled and digitized, further trigger processing include signal conditioning, which often implements trigger timing adjustments using delay generators and pulse stretchers. Combining both delay generators and pulse stretcher forms a gate and delay generator, which when given an asserted pulse signal, generates a gate of a given length, but delayed with a given delay. Trigger decision making can be made by coincidence levels, or logic gates (AND, OR, etc.), thus giving an output of logic 1 when a valid user-specific physics event occurs and logic 0 otherwise.

In most cases, trigger system are used to trigger some readout to some storage or other processing systems. To manage this, during a trigger processing, signals are buffered on a bit-pattern-register (BPR) at every rising-edge of the valid trigger event. The BPR values can either be over-written at the next event trigger rising-edge or wait for a veto signal to allow new value to overwrite the old one soon as they have been read and stored. A BPR implementation choice is normally made per experimental requirements.

2.2 Digital Pulse Processing Systems

Digital pulse processing (DPP) systems implemented in an FPGA shorten the path from the detectors to actual CTP. The detector signal form is digitized with a sampling (or digitizing) ADC immediately after the preamplifier [3][36]. The digitized signal pulse is then shaped digitally and the pulse height is extracted. The digital processor is the key element doing this operation

and either a FPGA or a DSP can be employed. As depicted in Figure 2.4, DPP based trigger-

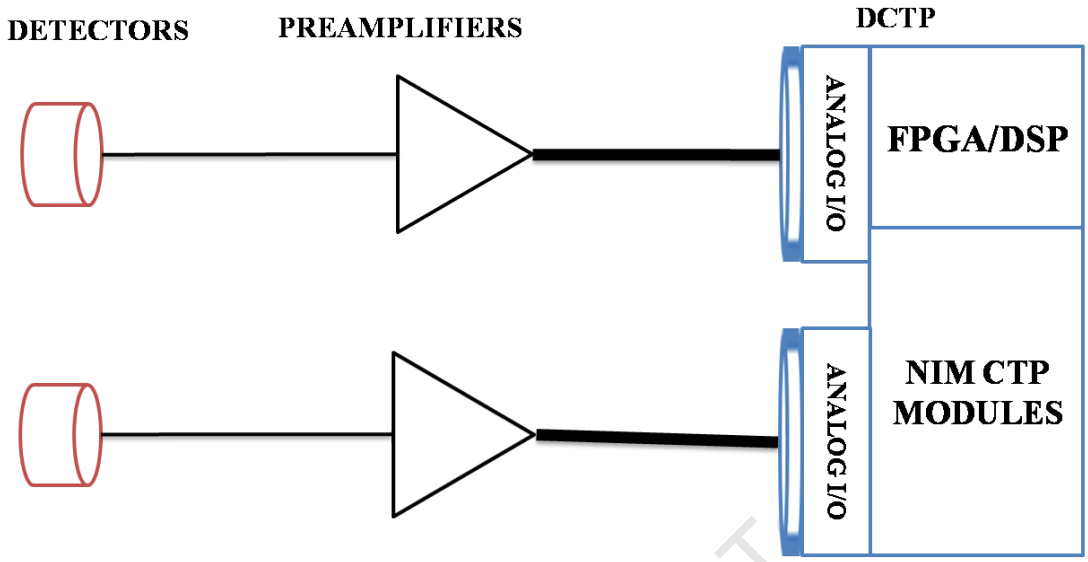


Figure 2.4: Digital trigger systems using DPP firmware overview

ing systems are composed of CTP components programmed into an FPGA. Main advantages include low cost and reliability, reproducibility, changeability and adaptation of algorithms, and register programming instead of manual regulations and trigger timing adjustments.

In the digital domain, easy to implement filters such as the finite impulse response filters (FIR) are used to directly shape and digitize pre-amplifier signals. The FPGA-friendly algorithms presented in [40] of digital filters are often used as sampling ADCs of the pre-amplifiers output. In trigger systems used in nuclear physics applications, the Trapezoidal version of the FIR filter is often considered best for it typically covers the rise time of the incoming signal and thus makes the pulse height measurement less sensitive to variations of the signal shape on the rising edge [37]. As shown in [37] and [29], the trapezoidal filter could be implemented in the FPGA by using the CFD algorithms presented in [31] and [29]. Equation (2.1) was studied in this project which when implemented successfully digitally approximates the standard analogue sampling ADC module, the CFD.

$$CFDTrace[k] = \sum_{i=1}^{L} F \times Trace[k-i] - Trace[k-i-D] \tag{2.1}$$

Where:

- $CFDTrace$ - A CFD value that will compared to threshold value set by user
- $Trace$ - A data series
- k - Point where CFD value is calculated.
- i - Iterator from 0-L, representing ith value of the data series

- L - Running average length
- F - Fraction of original Trace
- D - Delay

Equation (2.1) illustrate the principle of CFD with resemblance to the standard CFD circuit discussed in the previous section, Section 2.1. The pre-amplifier output ($Trace[k-i]$) is multiplied by the fraction F that is to correspond to the intended fraction of full amplitude. This multiplication of the input trace value with the fraction is similar to the attenuator operation in the standard circuit that is depicted in Figure 2.3. The pre-amplifier output signal ($Trace[k-i]$) is inverted and delayed ($-(Trace[k-i-D])$) for a time greater than the rise time. This operation is performed for the entire pre-amplifier data series length, and results are summed. The sum of these values ($FxTrace[k-i] - Trace[k-i-D]$) for the data series is compared with the Threshold value defined by the user. When the $CFDTrace[k]$ value is greater than or equals to the threshold, the CFD output is asserted, otherwise the CFD output is set to logic 0.

[34] and [35] provide commercially available digitizers used in experimental nuclear physics. They both use FPGAs running DPP firmware to convert analogue signals using sampling ADCs and further process the data. The NIMBox [35] from Weiner is defined in [35] and has similar properties as the project discussed in this dissertation but use National Instruments LabView VIs for modelling. [34] shows that disadvantages of using this commercial products include, deep knowledge of the digital algorithms and the relevant parameters, VHDL knowledge, and/or CAEN or Wiener support.

2.3 GRAPHICAL SYSTEM MODELLING

The triggering system discussed will have a generalised software abstraction of NIM-based trigger electronic components. A modelling environment that can automatically generate a HDL code is required. This will simplify design of trigger systems by allowing physicists to follow true DSP synthesis methodology [22], thus enabling synthesis capability plus a well defined abstraction layer for library of NIM electronics. There are two open source modelling environments suitable for researchers, Ptolemy II and Scilab. Scilab is normally shipped out with Xcos which is the graphical editor to design hybrid dynamical systems models. Models can be designed, loaded, saved, compiled and simulated just like in the Ptolemy II environment, which focuses mainly in supporting heterogeneous, concurrent modeling and design.

2.3.1 Modelling and HDL Code Generation with Scilab Xcos

Scilab includes hundreds of mathematical functions. It has a high level programming language that allows access to advanced data structures, 2-D and 3-D graphical functions [24][8]. It is more like Matlab in the scientific computation and simulation in graphical modelling. Matlab has Simulink and Scilab has Xcos as graphical editors.

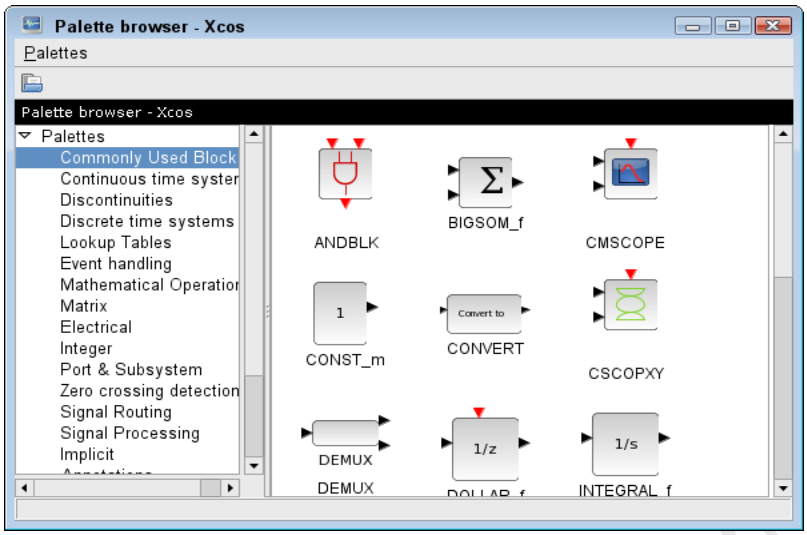


Figure 2.5: XCos Palettes

Scilab is considered in this project because of its Xcos modelling and simulation. Xcos offers blocks of electronic devices and connectors such that a complete electronic circuit can be designed, drawn and exported to the specimen. Xcos provides functionalities for modelling of mechanical systems, hydraulic circuits, control systems, etc. These functions are organised in palettes which include Signal processing, Logical blocks, Thermo-hydraulic blocks, Mathematical operations, Discrete and continuous system blocks, Electrical, User defined blocks, and Annotations such as text and LaTeX/MathML [24]. The Palette Browser is shown in Figure 2.5. Xcos graphical user interface (GUI) also includes an editor, as shown in Figure 2.6, where a block can be dragged on from the palette browser in order to model a given circuit diagram. There are several programming languages (C, Fortran, Modelica, and Java) used in Scilab, and Scilab has its own interpreter for programming in Scilab. This is a form of scripting language. Scilab is fast enough to act on its own as a programming environment but due to lack of other functionalities, it provides a way of adding user-defined blocks as well as creating new palettes.

Adding automatic HDL code generation in Scilab will provide software-like development process to complete complex hardware design projects successfully. This will allow an easier and more intuitive method for system description using the high level system design which is of high level of abstraction biased on hardware models. FPGA-friendly algorithms can be rapidly iterated to the best design to achieve the best implementation that meets the system requirements. There can also be some exploration of design alternatives, elaboration of these models into fixed-point representations, and generation of readable HDL code that can be synthesized by industry-standard synthesis tools.

Code generation from within the Scilab and Ptolemy II environments is done for C programming language. So far only one attempt has been made to provide HDL code generation from within an older Scilab project named Scicos-HDL [26]. This project is not maintained and is only supported in the Scicoslab project (the gtk version of Scilab). The latest version which supported Scicos-HDL was ScicosLab 4.3 with the current stable release 4.4 not verified yet.

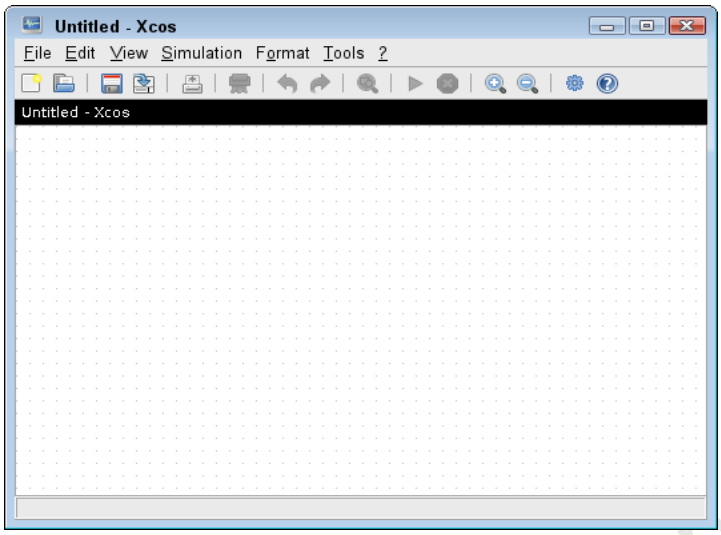


Figure 2.6: XCos Editor

The source package of this version of ScicosLab does not compile successfully in the current versions of Linux (Ubuntu 10.04.3 LTS and SL 6) and still used the older scilab 4.x blocks. An attempt was made, in this project, to port the Scicos-HDL to the new Xcos palette browser but was unsuccessful due to lack of backward compatibility in the new Scilab/Xcos project.

2.3.2 Modelling and HDL Code Generation with Ptolemy II Vergil

Ptolemy II is a Java based environment for simulation of heterogeneous models with Vergil graphical system modeller. Vergil is composed of Actors which communicate with each other through the use of ports [25]. Actors are software components that execute concurrently and communicate through messages sent via interconnected ports. A model is a hierarchical interconnection of actors in the Ptolemy II environment. In Ptolemy II, the semantics of a model is not determined by the framework, but rather by a software component in the model called a director, which implements a model of computation. As depicted in Figure 2.7, Ptolemy II vergil graphical modeller is composed of directors , supporting process networks (PN), discrete-events (DE), dataflow (SDF), synchronous/reactive(SR), rendezvous-based models, 3-D visualization, and continuous-time models [25]. The focus is on assembly of concurrent components. The key underlying principle in the Ptolemy II project is the use of well-defined models of computation that govern the interaction between components. Ptolemy II therefore does not provide a different GUI from the computation algorithms, everything is well defined in the Java programming language and thus makes it better designed for the open source community as it provides a standard way (Actor Oriented design) of adding new Actors while easily maintaining the whole system architecture and design.

Research has been done on Ptolemy II HDL Code Generation. Nothing concrete so far but student projects most of which require future investigations and implementations. Ptolemy II is shipped out with Java and C code generation, covering some of the actors. The code generation framework is under active development and the code can now be generated for several models. Code generation for all the directors is not complete due to the lack of helpers for the Actors

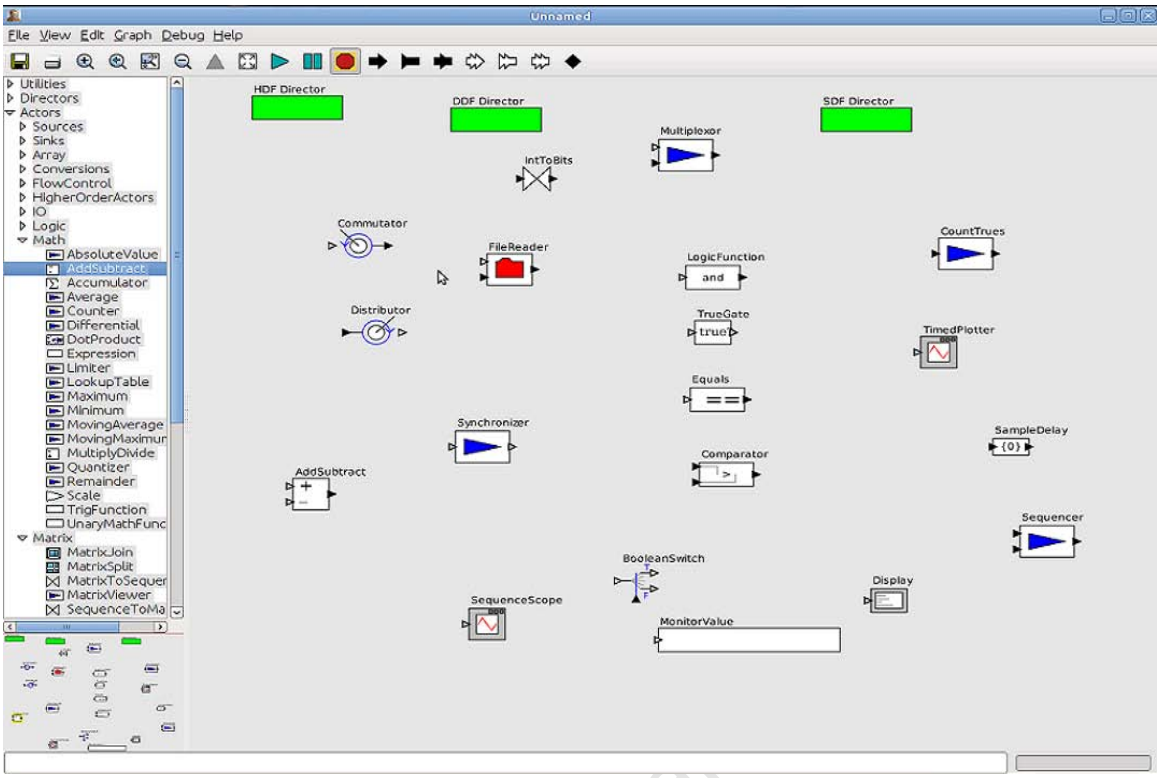


Figure 2.7: Vergil GUI for Ptolemy II With Directors and Actors

contained in these models, the lack of codegen support for the data types used in the models amongst others. The VHDL code generation [27] work has been performed at University of California at Berkeley. Code generation in Ptolemy II supports both Java and C for most of the available Actors. Some work with Java HDL (JHDL) and Ptolemy II in the Ptolemy/Copernicus directory and a rough draft of a start of a prototype of VHDL code generation was done but not completed [27].

2.3.3 Choice of Modelling Environment

Both Scilab and Ptolemy II were found to include models for most of the existing electronic modules. The following points were considered in choosing the graphical system modeller to be used:

- Complexity in adding the HDL code generator
- Industrial adaptation
- Simplicity of user interface for final users

Adding HDL Code Generator

It is not necessarily the objective of this project to add a code generator to the modelling environments but to find and use a system modeller with a code generator. None of the free open

source system modellers already has automatic HDL code generation. In the great tradition of open-source, the VHDL Manipulation and Generation Interface (vMAGIC) project that fulfils this hole practically, was found. vMAGIC is a Java library to read, manipulate, and write VHDL code. The basic functionality as well as the designflow is described in [8], stressing the advantages when designing with vMAGIC. Since none of this Graphical System Modellers (Ptolemy II and Scilab) has a HDL code generator, VHDL Manipulation and Generation Interface seems to be the only available project that is general enough to provide the basis of code generators within a JAVA programming language, with three basic tasks that include:

- Reading of existing code,
- Manipulation of existing code, generation of new code,
- Writing of manipulated and/or generated code.

vMAGIC is not a code generator by itself, but a homogeneous framework for implementing code generators [8], therefore it is a suitable candidate for this project. It is critical that a choice to be made between Ptolemy II and Scilab, provides an interface to code generation with less complexity and a well defined design methodology.

Industrial Adaptation

Ptolemy-II, a more radical component-based approach, supports several domains, each of which is based on a particular model of computation and may be combined with others to build a system model [32]. Industrial adoption has been limited, possibly because of the extent to which it represents a departure from current design practice, leading to it being regarded as high risk for adoption. Older Scilab versions contains two toolboxes for modelling and simulation:Scicos and the OpenModelica Toolbox [8]. Scicos is the counterpart of Simulink but is limited to block diagrams only, so is its successor XCos, which is the recent graphical modelling editor for Scilab and no conclusion can be made about its industrial adoption. Xcos is a new departure for Scilab. The Xcos toolbox has not reached a sufficient level of maturity yet for use in an industrial setting. However Matlab/Simulink is almost analogous to Scilab/Xcos, thus preferred more than Ptolemy II.

Simplicity of user interface for final users

The GUI editors shipped out with both Scilab and Ptolemy II, XCos and Vergil respectively, are easy to understand and use. They both classify the GUI blocks into different palettes and allow drag and drop to the Block editor. With Scilab, the new Xcos GUI Editor provides the docking mechanism whereby one can avoid using multiple windows for a single project as it was the case with Scicos. But with Ptolemy II, there are still multiple windows used for each single project carried out which can be annoying at times, particularly when a user enjoys a needy working space. An elegant design for this project was found to be Xcos. Thus making Scilab/Xcos the preferred choice as an RCP environment for the triggering platform discussed in this dissertation.

CHAPTER 3

METHODOLOGY

This chapter provides detailed guidelines of the triggering platform design process. There were four phases in this research project. Since a trigger system consists of multiple NIM electronics components pipelined together to perform triggering, each NIM component goes through a spiral software methodology [52][51][53][54] until it is successfully software abstracted into synthesisable graphical system models. It is therefore important to note that this research project is an iterative and incremental approach to design and testing of the system under discussion. The four main phases of this methodology are:

Phase 1: Requirements gathering and constraints identification
Phase 2: Design specification
Phase 3: Design of hardware and software - With more software design than hardware
Phase 4: Evaluation process

This chapter elaborates on these phases by starting off with section 3.1, explaining the user requirements and constraints. This chapter proceeds with design specification in section 3.2, followed by the design in section 3.3. The evaluation method for this research project, which focuses on assessing safety and performance of the triggering system, is presented in section 3.4 of this chapter.

3.1 USER REQUIREMENTS AND CONSTRAINTS

This section explains the process by which requirements were established, which were then used to guide the design of the triggering platform. Section 3.1.1 outlines user requirements whereas section 3.1.2 discusses design constraints.

3.1.1 User Requirements

By studying how physicists at iThemba LABS setup standard analogue triggers, reading available documentation manuals[43][37][45][44] for available analogue electronic NIM modules, and thesis [42][33] on experimental nuclear physics research done at iThemba LABS, user requirements were gathered. These user requirements of the triggering platform presented are classified into functional and non-functional.

Functional Requirements

The main requirement for this research project is modelling of NIM electronics and automation of synthesizable HDL code. The automatically generated HDL code must be synthesisable and timing-safe, making sure all the timing related errors, such as latency and jitter, are well documented. A summary of the functional requirements with regard to the triggering platform aspect of the system is as follows:

- Graphical System Modeller
- HDL code generator
- Digital CTP HDL codes
- Trigger Board with FPGA and NIM I/O channels
- CTP Control GUI

The graphical system modeller is required to hide all the underlying complexities of designing FPGA applications for triggering. The available V1495 general purpose trigger board must be used in this research to avoid additional costs for this project. This board is preferred at iThemba LABS because of available digital I/O channels used in nuclear physics applications. It also has a User FPGA for control of I/O channels and user specified logic. The V1495 has been used by [56], [55], [57] and [58] for nuclear physics trigger concepts such as event-trigger-based readout, CTP decision making (coincidence), clocked readout, and dead time logic respectively. In this project, it is required that some or all of the nuclear physics triggering concepts, explained in section 2, be fitted in the V1495 board User FPGA for multi-level trigger processing. The last requirement was the control of the on-chip triggering system. Therefore, when a trigger design at hand has been modelled, automatically a GUI must be generated that will provide an interface for trigger timing adjustments and readout via registers on the FPGA.

Non-Functional Requirements

In addition to an implementation of a low cost triggering system, this triggering platform must assist physicists in setting up low error experiments by automating manual configurations of standard trigger setups. The non-functional requirements are outlined as follows:

- Trigger design requirements: low error and less tiring trigger setup processes, as well as repeatable and reproducible trigger design
- Cost requirement: low cost trigger design and implementation.
- Timing-safe HDL code generation: timing-safe trigger HDL code generation
- FPGA Logic Elements: VHDL space optimization of trigger models
- Performance: The model-to-VHDL interpreter should be able to generate HDL code at a reasonable speed. However, this is not a mandatory requirement because it is assumed that computers will continue to become faster (based on Moores law).

- Insitu NIM triggering: hiding trigger design and control complexities while maintaining user paradigm of old NIM electronics

While providing an easier and more intuitive method of trigger system design based on NIM electronics graphical models, automatically generated HDL code must fit into the User FPGA of the V1495 board. This requires investigations on space optimization in FPGA applications.

3.1.2 Design Constraints

The triggering platform should be created within the following constraints:

- Graphical system modeller: Design and development will be based on existing open source RCP softwares such as Scilab and Ptolemy II. The choice of such RCP software will be based on industrial adaptation and ability to automatically generate HDL codes. Additionally, the RCP environment used must be platform independent, for the target users are multi-platform based, using Linux, and Windows operating systems (OS).
- Trigger board: The use of V1495 for generated and synthesized CTP HDL implementation.
- Integrated software environment (ISE): Altera[1] Quartus II ISE will be used because V1495 contains Cyclone EPC20 FPGA
- Trigger control GUI: The use of the VMIC77 restricts the RCP Server with VME driver to run on a Linux operating system. The RCP client will be multi-platform based.
- Digital I/O channels: the V1495 has only the NIM, ECL, LVDS and TTL I/O input slots that are software programmable

These constraints depend on the approach used to achieve most of the objectives of this project. It is recognised that some of these constraints can be removed provided a different trigger board was used in this project.

3.2 Design Specification

The specifications for the triggering system design were based on the following sources of information: 1) research theses by postgraduate physics students; 2) equipment manuals; and 3) observations of a selection of physics experiments perform by researchers at iThemba LABS. The NIM I/O channels are the most preferred. Support for other digital I/O levels such as TTL, LVDS and ECL will be considered as they are used in experimental physics applications such as the ones presented in [41][56]. These I/O standards will act as input to CTP modules such as pulse stretchers, delay generators, gate and delay generators, etc. The CTP receives as inputs, pre-amplified analogue output signals from charge sensitive detectors [42][33][43]. Based on these user requirements and constraints, the following triggering platform specifications are outlined:

[1] Altera is one of the worlds largest supplier of programmable logic devices

- Digital NIM CTP modules: Timing modules, CTP decision-makers, readout modules, and sampling ADCs.
- Registers: Register VME addresses and 32-Bit read and/or write registers
- HDL code: VHDL code generation library
- Graphical System modeller
  - Automatic Timing-safe VHDL code generation
  - Reproducible and repeatable triggers
  - Automatic trigger design documentation
- V1495 Trigger Board
  - VME 6U board, 1U wide
  - I/O Channels: support for NIM, TTL, ECL, and LVDS.
  - User and Bridge FPGAs
  - User FPGA with maximum 20,000 logic elements (LE)
  - 64 inputs, expandable to 162 (with 32 outputs)
  - 32 outputs, expandable to 130 (with 64 inputs)
  - 405 MHz maximum frequency support for PLL synthesis
  - I/O delay smaller than $15ns$ (in Buffer Mode)
  - Remote trigger on-chip timing adjustments
  - On-chip scope interface
- Trigger control GUI

When a synthesized trigger model is successfully programmed into the FPGA, an on-chip scope interface used to dynamically select output signals to the Tektronix scope for debugging has to be implemented. Using registers and finite state machines as shown in the CAEN Mod. V1495 examples [16], the trigger implementation on an FPGA must be accessible via a VME interface using address mapping. This mechanism is used to allow settings of on-chip trigger variables. Each NIM component that requires real-time parameters adjustments or readout without disconnecting the trigger setup, must have a register settable via a VME driver application. Modules such as gate and delay generator, may need delay value adjustments, so they must have registers accessible remotely. In addition to automatically generating the timing-safe and synthesisable VHDL code, the graphical system modeller must support use of each NIM component more than once in one trigger model, hence providing reproducibility and repeatability in trigger designs. Each one of these models in a trigger must also be debugged remotely using the Tektronix scope connected to a LAN. The trigger control GUI must also be included for remote interface to the trigger variables such as registers, delays, stretch lengths and readout values from pattern registers.

3.3 Design of Hardware and Software

In this section, the design processes of both software and hardware are discussed. The software design process is defined first as it covers more than half the work done in this research project. The hardware design process will then follow after software design processes.

3.3.1 Software Design process

This process was started off with choosing a graphical system modeller. This step is followed by designing the NIM components HDL codes, which will give rise to a model-to-VHDL interpreter. The model-to-VHDL interpreter design focuses mainly in adding the code generator to the chosen graphical system modeller. The design of the VHDL codes for NIM components must be done first. This way it would be easier to design the model interpreter when all NIM HDL codes parameters are known. The model-to-VHDL interpreter will then be designed after the NIM components VHDL codes are implemented and tested. These tests will be iterated until all NIM models VHDL are abstracted and tested. The software design process is visually illustrated in Figure 3.1.

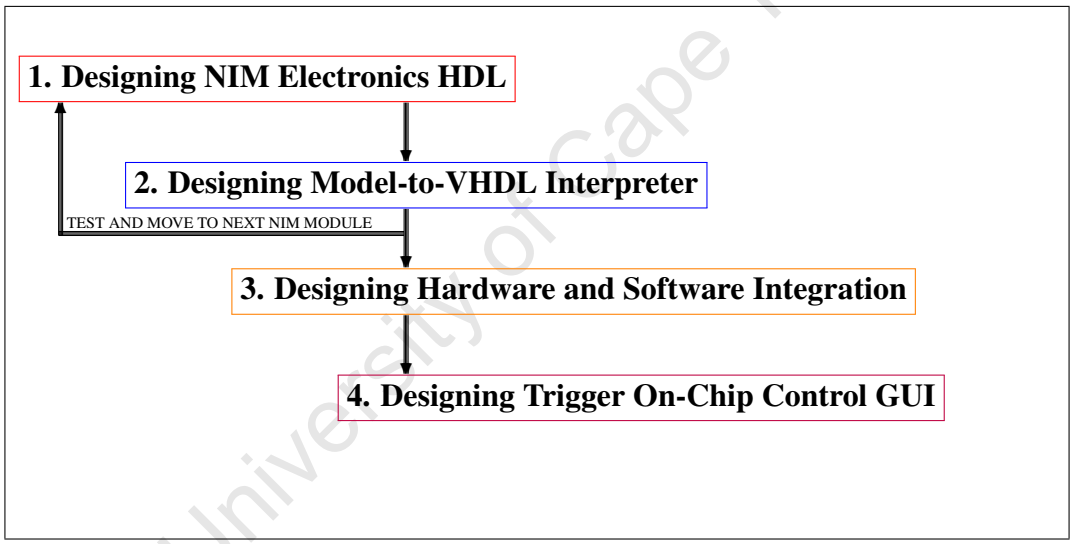


Figure 3.1: Software Design Process Overview

NIM Electronics HDL Design

The design of digital NIM electronics using HDL basics such as AND, OR, FIFO, Flip-Flops, latches, finite state machines, etc. [59][60][61] was performed. The idea behind the HDL is to describe NIM electronic circuits residing in most standard analogue NIM modules used at iThemba LABS. These NIM electronics systems can be designed using different levels of abstractions [62]. The behavioral level, also known as algorithmic, will be used to abstract this NIM electronics systems using constructs such as loops and processes. In addition, the HDL components designs must support code reuse such that the modular design criteria used in standard analogue NIM electronics is preserved. The design of NIM electronics HDL should therefore use entity, components, and signals definitions. The HDL language was chosen to be VHDL since it covers most of the required HDL behavioural level components that can be

used. Also there are enough VHDL examples from [16] showing how the V1495 trigger board applications can be programmed.

Model-to-VHDL Interpreter Design

The design of model-to-VHDL interpreter is more an iterative and incremental approach than any other process in the design of this triggering platform. For each NIM electronics model, corresponding VHDL code component instantiations, signal declarations, and addition to Quartus II project file will be codified in Java programming language that is used in programming of the Scilab-Xcos [8] graphical modeller. These will be done by method invocations of vMAGIC library classes. The NIM components VHDL files such as stretchers, delay generators, coincidence, logic gates, etc. will be stored in a library folder, and will be copied to the project directory every time the corresponding model is used in a trigger model. This will optimise performance of code generator and component based VHDL design will be supported, so is VHDL code reuse.

Systems Integration Design

When all NIM components are presented in a codified form, simpler ways of placing them in a trigger board with an FPGA must be created. As gathered from the requirements of this project, the use of the V1495 general purpose trigger board with an Altera cyclone User FPGA will be used. The Altera Quartus II ISE will therefore be used to synthesise, fit, assemble, and timing analyse the NIM electronics VHDL codes. The use of the Quartus II command-line scripting [63] will be used for integrating the model interpreter will the Quartus II synthesis, fitting, assembling and timing analysis tools. These Quartus II command-line executables reduce the amount of memory required during each step in the trigger systems design flow. [63] states that, because each executable targets only one step in the design flow, the executables themselves are relatively compact, both in file size and the amount of memory used during processing. This memory usage reduction improves performance of the model interpreter when synthesizing, fitting, assembling and timing analysing the trigger models. The Quartus II command-line tools such as $quartus\_map$ (synthesising), $quartus\_fit$ (fitting), $quartus\_asm$ (assembling), $quartus\_sta$ (timing analysing), and $quartus\_rtl$ (RTL viewer) will therefore be called in a Java program that implements the shell scripting processes. This will allow integration of the model interpreter with Altera Quartus II HDL design compiling tools, thus providing an easy to use trigger systems design environment. The SSH command line tool will also be used to remotely program the V1495 trigger board.

Trigger On-chip Control GUI Design

In the successful interpretation and code generation of the trigger model, the model-to-VHDL interpreter program will determine which registers were assigned to which NIM component in that model, as well as the assigned VME addresses, then use Java XML marshalling to create a configuration file for the trigger control GUI. This way each control GUI will only contain interfaces to the implemented trigger registers, hence avoids confusing the user when controlling the trigger. The RPC server with VME driver will be implemented for testing purposes for remote trigger control. VHDL was used for designing digital NIM electronics because enough examples are given in the CAEN V1495 modules website [16]. This will therefore guide the research

in how the board can be programmed. The automation of manual process in trigger design also include automatic GUI control for trigger parameters, thus a dynamically customisable GUI design process is needed.

3.3.2 Hardware Design process

The hardware design process focuses on the review of the V1495 board design as well as designing analogue trigger setups used for comparison with the implemented digital trigger versions. This process is done so that most of the trigger timing related errors maybe documented and eliminated. The two main tasks performed in the hardware design are:

- Studying analogue NIM electronics and setting up a standard analogue trigger with NIM electronics
- Studying different methods of using V1495 trigger board to setup a digital trigger system

The study of analogue NIM electronics will guide this research with setting an optimal analogue trigger with detectors, pre-amplifiers, CFDs and CTP modules. It was realised that the analogue CFDs are still required in the design of digital CTP using the V1495 trigger board. The electronics hardware of the experimental setups involved in the evaluation of this project are listed as follows:

- **Signal Sources:** A detection device of some kind producing a signal.
- **Preamplifiers:** The signals amplifier.
- **Sampling ADC:** CFD devices to digitise pre-amplifier signals.
- **Central Trigger Processor:** CTP modules for signal conditioning and trigger decision making
- **On-Chip Debugging:** An interface to trigger debugging

The signals from the two detectors are amplified with Canberra Mod. 2020 Spectroscopy Amplifiers. This is another form of charge sensitive pre-amplifiers discussed in section 2.1. After the signals are amplified, the Canberra Mod. 2128 CFDs are used to produce accurate timing information from pre-amplifier output signals. From the CFDs the pulses produced are sent to the CTP inside the V1495 board and the analogue CTP electronics as NIM inputs. Using VME and a Local Bus Register Access (LRA) interface, an on-chip debugging module that dynamically selects outputs to display in the four channels of the Tektronix scope is going to be designed. This method will also provide an interface to remotely adjust trigger timing modules.

3.4 EVALUATION PROCESS

The evaluation process used in this research project involved five main steps. These steps are outlined by the points below. Each step is then elaborated together with an explanation of

how the processes used were both guided by the literature and adapted to fit the context of this project.

Step 1. Model NIM-based triggers, and measure code generation and synthesis times
Step 2. Model a full experimental physics trigger and document usability of triggering platform
Step 3. Program the trigger board and perform on-chip debugging
Step 4. Remotely adjust timing parameters and perform readout using trigger control GUI
Step 5. Analyse the results gathered in steps 1 to 4

The first step involves modelling each NIM component by connecting it to I/O ports representing NIM I/O channels on the V1495 trigger board. The code generation and synthesis times will then be recorded. These were done to determine by how much the design cycles were reduced by automatically generating code from NIM models. This technique of reducing design cycles by code generation from models is presented by (M.S Moore,M. Brooks,G. Willden, 2004) [46]. To demonstrate how FPGA space optimized VHDL was automatically generated, the first step also includes modelling a trigger with multiple NIM components and recording code generation and synthesis times, while gradually incrementing the number of models. For the second step of the evaluation process, a full experimental CTP comparative to the AFRODITE experiment will be implemented. The explanation of this trigger was done by [33][42]. The AFRODITE covers most of the trigger concepts defined in section 2.1, such as, clocked readout, signal conditioning, CTP decision making, etc.

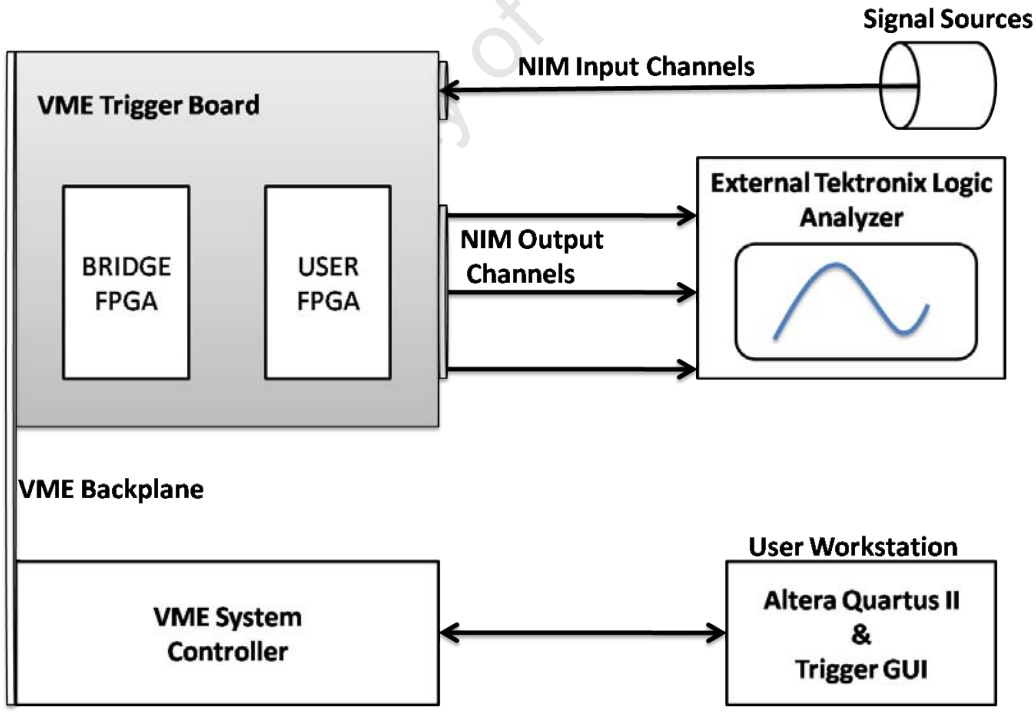


Figure 3.2: Overview of the proposed debugging and experimental test setup

The third step includes programming the synthesized trigger model into an FPGA and observing the trigger on-chip performance measurements as compared to standard analogue NIM-based

triggers using the setup in Figure 3.2. The Tektronix scope will be used for debugging the triggers in this step. This method was also used by MIPS technologies by debugging Xilinx FPGA devices with with Tektronix logic analysers and mixed signal scopes [47]. As depicted in Figure 3.2, similar setup as [47], and Altera Quartus II handbook method [48], but with NIM output channels to the scope, was used in this project to provide real-time trigger debugging. As the forth step is used to perform trigger timing adjustments that will affect trigger behaviour, the Tektronix scope debugging shows changes in real time. In the forth step, tests involved establishing whether the trigger GUI could effectively control trigger on-chip remotely from working station (user PC). These tests involve adjusting timing modules by setting implemented 32-bit read and/or write registers in the ToC through the VME address mapping as done in examples presented by CAEN [16]. Successful timing-adjustments and scope output configurations will affect the scope display accordingly. Success of these configuration tests imply that the RPC client communicates well with remote trigger board.

The fifth step involves the discussion on results extracted from the model interpreter and ToC performance tests. The demonstration of how the system will be used to model the nuclear physics experimental circuits is discussed.

CHAPTER 4

TRIGGERING SYSTEM DESIGN

This chapter outlines how subsystems for the triggering platform under discussion are interconnected. An appropriate design of the triggering platform will provide smooth transition from current experimental triggering design process to true-DSP[22] synthesis methodology.

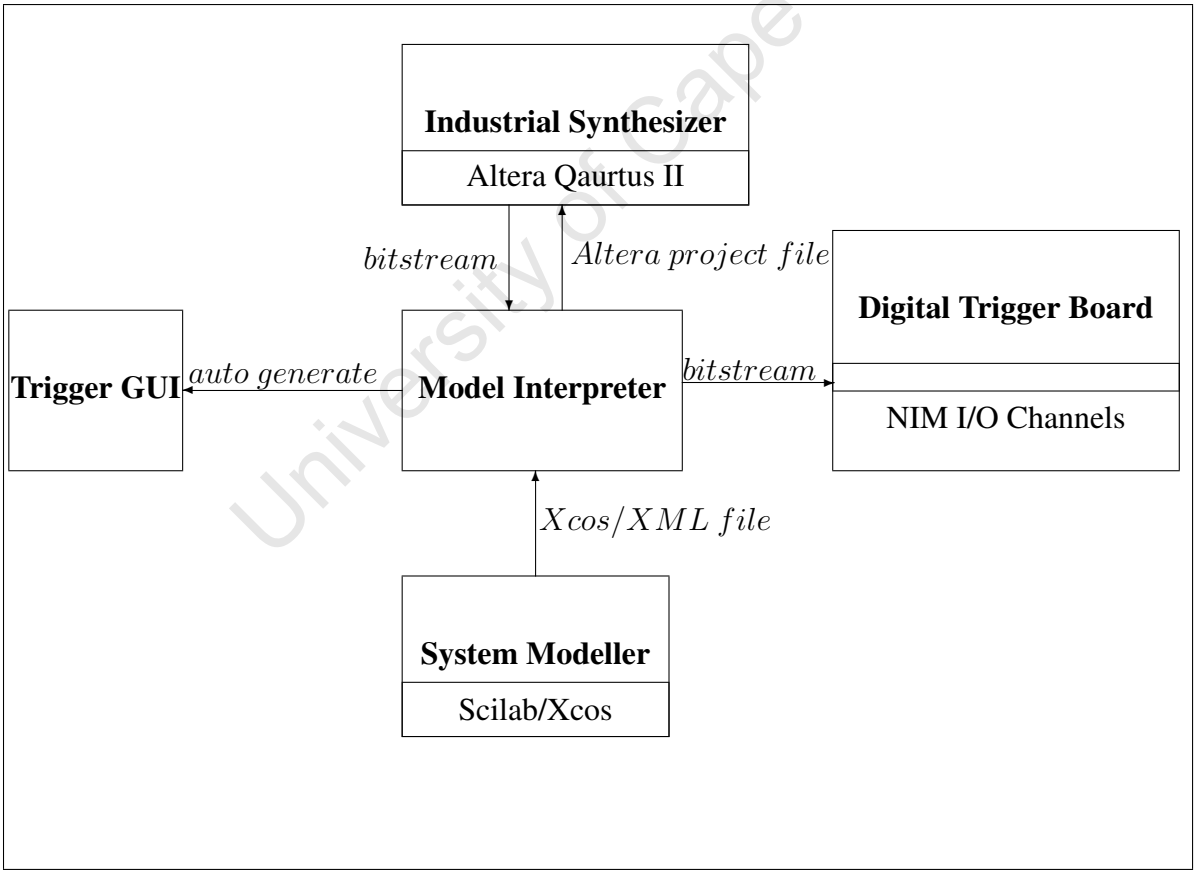


Figure 4.1: System Design Overview

As depicted in Figure 4.1, the triggering platform consists of a model-to-VHDL interpreter, central component to this project, for automatic HDL code generation, synthesis and program-

ming of bitstream to user FPGA. The model interpreter takes, as input, the XML file generated by the graphical system modeller, Xcos, a Scilab graphical model editor. The XML file is interpreted by the model interpreter, which then generates Altera Quartus II ISE project files, VHDL codes and trigger GUI XML configurations accordingly. The trigger GUI is automatically generated on Xcos file arrival whereas the bistream/programming-files is generated by the Altera Quartus II after project synthesis and fitting, this is called assembling the project. After a successful synthesis, fitting, assembling and timing analysis of the project, the model interpreter will remotely program the user FPGA in the trigger board on-the-fly.

4.1 MODEL INTERPRETER TOOL-FLOW DESIGN

The model interpreter is a tool to provide graphical interface above the VHDL internals so that users (physicists) can use the graphical system modeller to produce a diagram for their experimental physics trigger logic and automatically generate code to be placed into an FPGA. Figure 4.2 shows the tool-flow design of the model interpreter and how subsystems interconnect to perform modelling, automatic VHDL code generation and synthesis, and logic placement into the fabric.

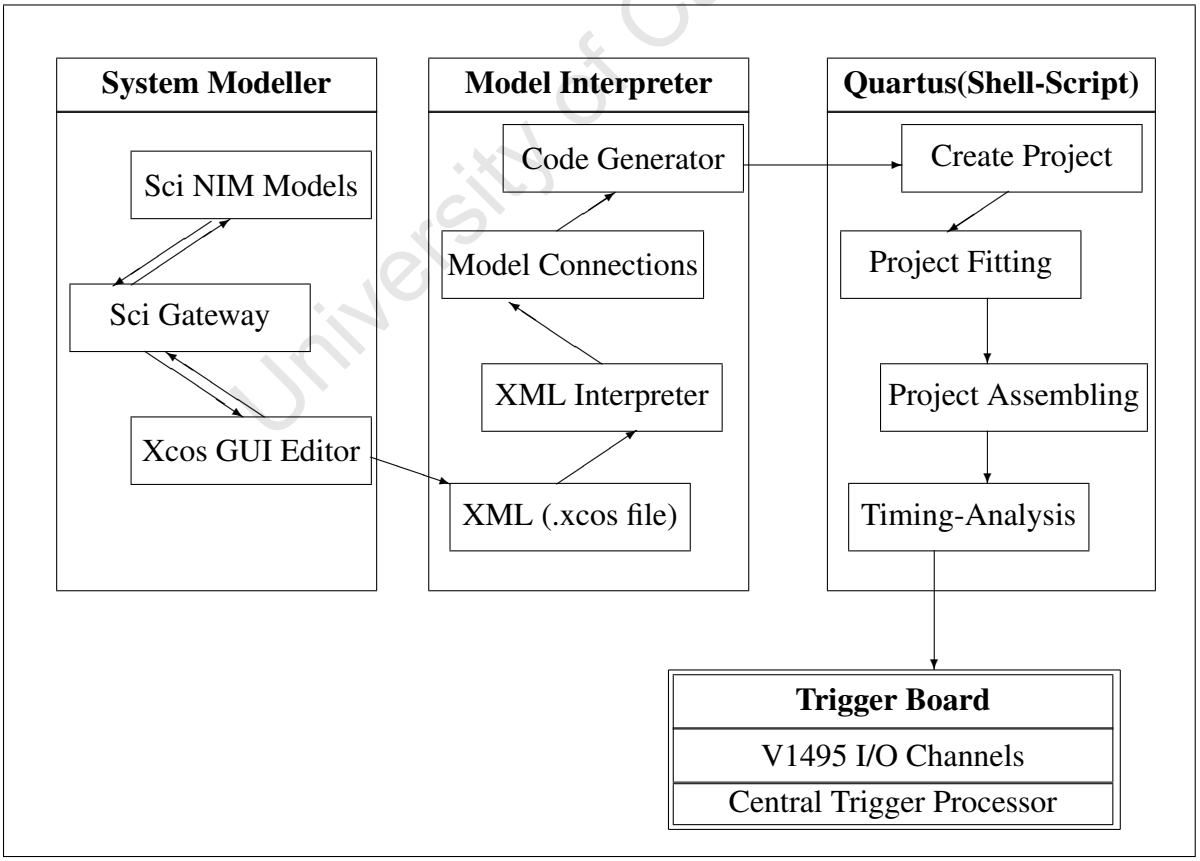


Figure 4.2: Scilab-Based Model-to-VHDL interpreter tool-flow design showing interaction of subsystems

4.1.1 Graphical System Modeller

To model NIM electronics, the Scilab scripts that include both the interface and the computation functions [8] will be programmed and loaded into the palettes browser of the Xcos graphical editor. The interface functions are programmed in Scilab scripting language whereas the computation functions in C programming language. This section focuses on the interface functions of the NIM components required for the graphical modelling of NIM-based triggers in the Xcos editor. In this graphical editor, users will model their experiments with NIM components that are under the Palette category named NIM blocks, created specifically for this project. The NIM blocks implemented include CFD as sampling ADC of the CTP. Code Listing 4.1 shows a sample interface function for the CFD.

Listing 4.1: Simple definition of CFD Scilab interface function with comments after double forward slash

```
//Create functin named CFD.
function [x, y, typ]=CFD(job, arg1, arg2)
  x=[];y=[];typ=[]; // Initialise x,y,type to empty vector
  select job // Implement a case statement with argument job
    case 'set' then // If selected operation is 'set'
      x=arg1; // Assign X to the configuration argument 1
      messagebox('No settings for a CFD block type');//Display message
    case 'define' then       // for case define,
      model = scicos_model();// then define model
      model.sim = list('CFD',4);// with block type set to
      model.blocktype = 'c';// C computation function
      model.out = [1;1;1];  // Number of output ports=3
      model.in = 1;         // Number of input port=1
      x=standard_define([2 2],model,[],[])// Assign model dimensions to x
  end
endfunction
```

As depicted in code Listing 4.1, a CFD model is made up of 1 and 3 input and output ports respectively. The CFD block defined in code Listing 4.1 is a simple block with static ports definitions. The timing modules consists of pulse stretchers, gate and delay generators and timers. The timing NIM models provide configurations for signal timing adjustments and conditioning.

Listing 4.2: Multiple I/O Stretcher Scilab interface function with comments after double forward slashes

```
//Create a simple custom block.
function [x,y,typ]=MultBitStretcher(job,arg1,arg2)
  x=[];y=[];typ=[];
  select job
    case 'set' then
      x=arg1;
      messagebox('No settings for a Stretcher block type');
    case 'define' then
      model = scicos_model();
      model.sim = list('Stretcher',4);
      model.blocktype = 'c';
      model.out = -[1:32]';// Number of output ports=32
      model.in  = -[1:32]';// Number of input port=32
```

```
        x=standard_define([2 2],model,[],[])
    end
endfunction
```

As illustrated in code Listing 4.2, multiple I/O ports are defined. This is the case with other timing modules. Multiple I/O ports in NIM models support repeatable trigger setups and allow one NIM component to process multiple signals without using multiple of these components. Thus saving the model graphical editor working area so that even for bigger experiments, all the trigger model blocks can be on view, thus avoiding a need for graphical editor scroll bars.

Dynamic reconfiguration of ports required in NIM components models was also considered. This will facilitate usage of one model for diverse I/O channels requirements without reimplementation of each model. The Scope model, one of the models for on-chip trigger (ToC) debugging, uses this functionality to dynamically assign number of input ports for the model. Figure 4.3 illustrates this functionality as well the NIM palettes browser.

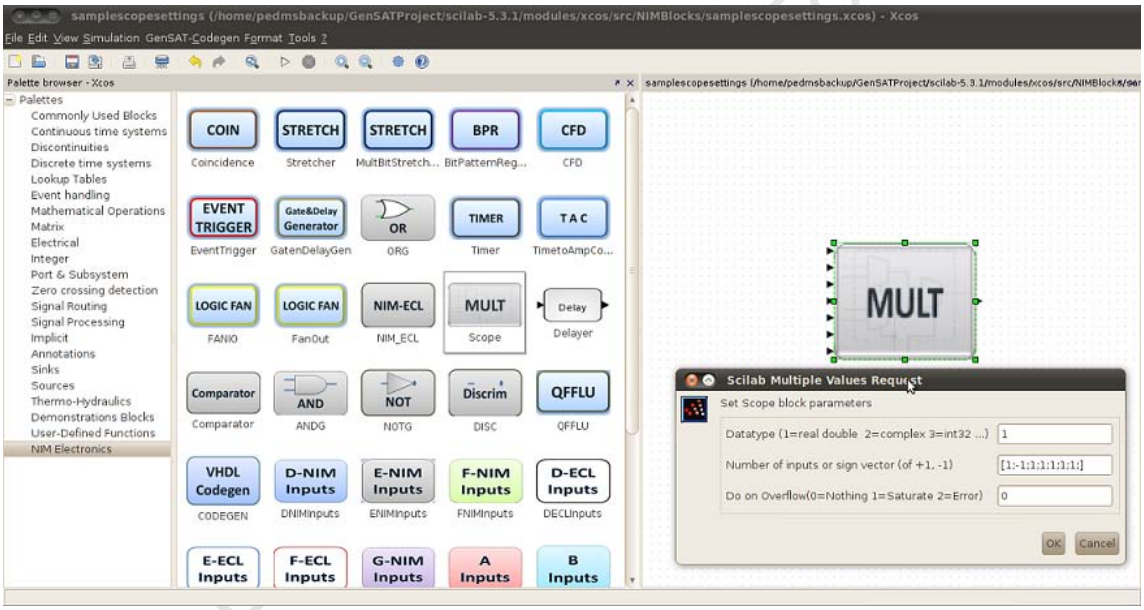


Figure 4.3: Graphical system modeller with NIM electronics palettes browser and Scope module multiple I/O ports settings in view

As depicted in Figure 4.3, the Xcos consists of the palettes browser, and the graphical editor where models can be dragged and dropped on to model systems, this case triggering systems for experimental nuclear Physics. On the graphical editor is the Scope model with a popped-up dialog window for setting number of inputs to the model.

The trigger modelling consists of connected NIM componets using Scilab explicit and implicit links for control and normal I/O ports respectively. When the user saves trigger models, an XML file is created (With an extension .xcos, top of Figure 4.3 show a link to such file, in front of close, minimize, and maximize window controls). Appendix E, code Listing E.1, is an implementation of a script that loads NIM models into the NIM palettes during Scilab start

up.

4.1.2 Model-to-VHDL Interpreter Design

The Xcos file generated is interpreted and analysed by model interpreter, which will automatically generate the corresponding VHDL code. The model-to-VHDL interpretation is made up of three parts, namely the XML interpretation, VHDL code generation and Quartus II shell scripting. The XML interpreter and vMAGIC-based VHDL code generation classes interact to perform the automated code generation of FPGA-friendly triggering system algorithms. The Quartus II scripting synthesizes, fits, assembles and timing analyses the created triggering system project. Figure 4.2 shows interactions of model interpreter classes.

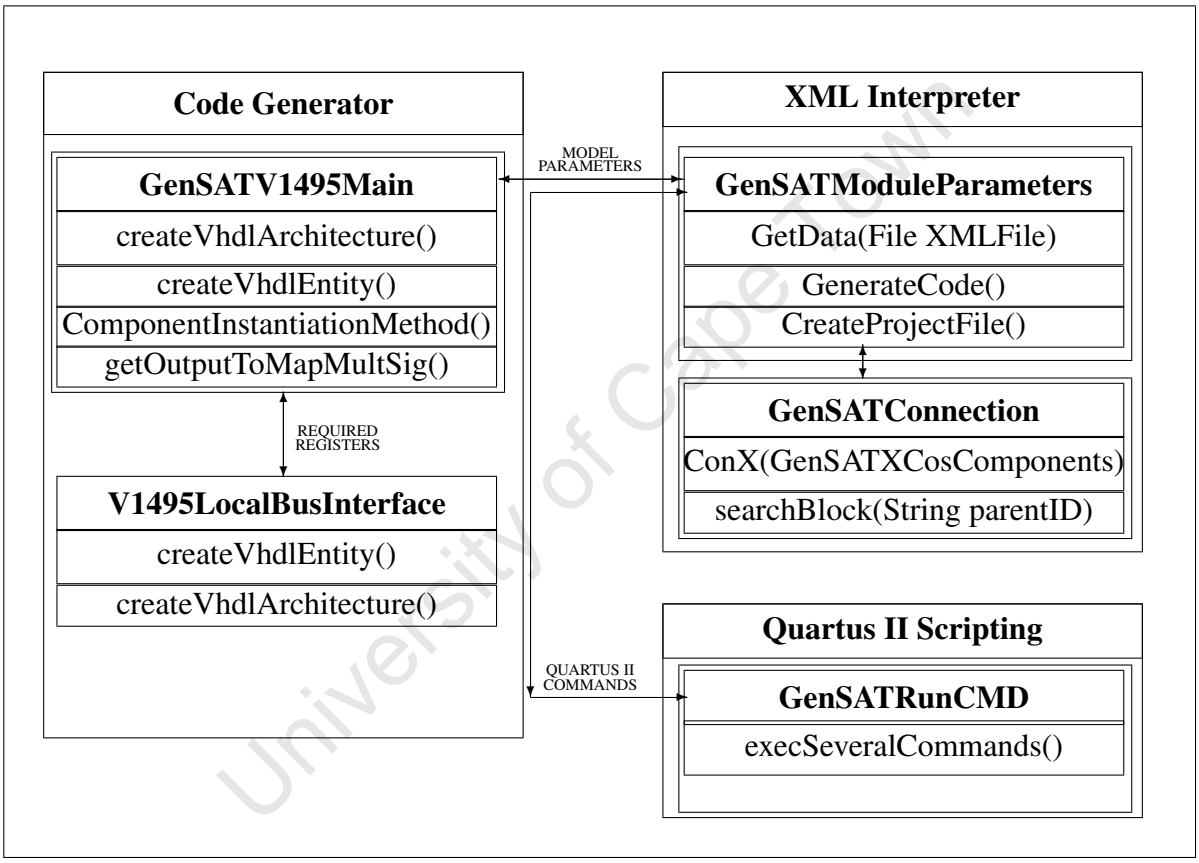


Figure 4.4: Scilab-Based Model-to-VHDL interpreter tool-flow design showing interaction of classes

As depicted in Figure 4.4, the GenSATModuleParameters class receives and analyses the XML file (.xcos file). The GetData() method of the GenSATModuleParameters traverses throughout the XML file to determine NIM components links and connections. The way in which the blocks are connected is stored in the list data structure class of type GenSATConnection. The GenSATConnection type has the block name, block parent id, other blocks connected to it, block I/O connections and links. When the connections are created successfully, the GenSATV1495Main class uses them to generate the VHDL code accordingly. The connections variables are called in the GenSATV1495Main with code generation methods. For each NIM

block, a mapping to the corresponding VHDL Code class (GenSATBitPatternRegister, GenSATStretcher, etc.) is done. Each block represents a VHDL entity that will be instantiated in the main-routine (VHDL project top-level file). When the VHDL Codes are created successfully, the Quartus II command-line scripting class is used to execute the Quartus II commands $quartus\_map$, $quartus\_fit$, $quartus\_asm$, and $quartus\_sta$ for synthesis, fitting, generation of programming files and timing analysis for the trigger model respectively. Then the FPGA can be programmed with the generated bitstream remotely using secure shell (SSH) and bash scripting. This model-to-VHDL interpreter design is aimed at generating a well partitioned VHDL code with multiple dynamic port mappings and signal assignments. The automatically generated VHDL code top-level entity is composed of the fundamental units visually illustrated in Figure 4.5.

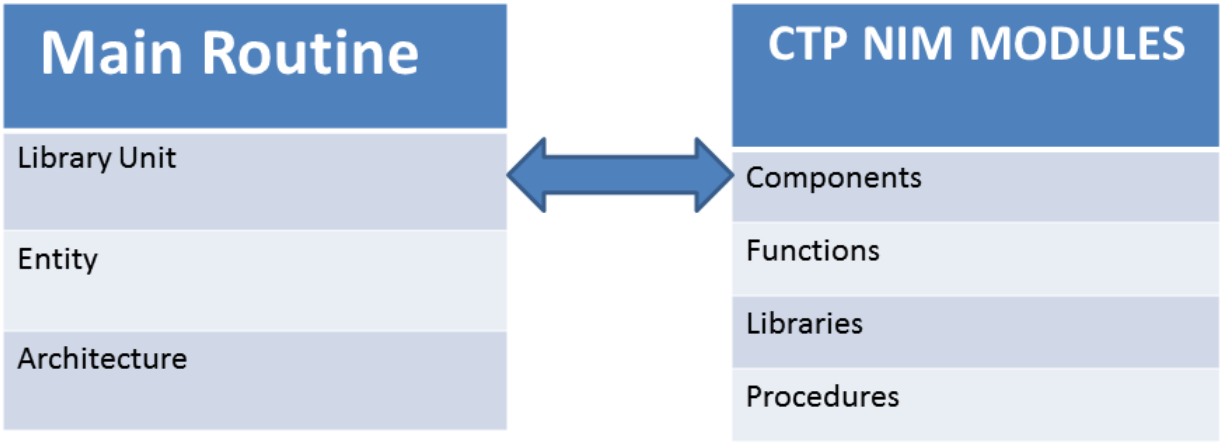


Figure 4.5: Fundamental Generated VHDL Code Units by Model Interpreter

As illustrated in Figure 4.5, the top-level entity, main routine, will be composed of library inclusions, top-level entity, and the architectural behaviour which will include signal declarations and port assignments. The CTP NIM components will be individual entities which will dynamically be instantiated in the top-level entity, main-routine. The code Listing F.1 illustrates the VHDL template snippet for the top-level entity for every triggering systems designed and implemented using the triggering platform under discussion. But the top-level entity is synthesizable only to the V1495 trigger board used in this project. Appendix F, Code Listing F.1, shows the top-level VHDL entity that defines the V1495 ports. The corresponding architectural behaviour starts off with internal signal declarations, followed by port assignment and module entity/ports instantiations. The Quartus II project file template with pin assignments, triggering systems packages, and comments showing where NIM components VHDL files will be placed, is depicted in code Listing G.1, and Appendix G.

4.2 Central Trigger Processor Design

In this section, designs of FPGA-friendly algorithms of digital NIM modules are discussed. These algorithms form the CTP NIM components which are abstracted by the NIM models discussed in section 4.1.1. The CTP NIM modules designs preserve the same modular design for NIM based analogue trigger electronics. This implies managing several individual components

of the experimental trigger physics at hand. The main advantage though, is easier debugging and stand alone verification of CTP components, for a trigger system is a mission-critical system. The CTP modules design inherently use the hierarchical design methodology and assign each designs into logical partitions that are functionally independent. The CTP components are partitioned as follows:

- Sampling ADC: a digital CFD module
- Signal conditioners: Mainly timing modules
- CTP decision makers: Event-Trigger, Coincidences, etc.
- Local Register Access: composed of LUT, Register and a state machine

4.2.1 Sampling ADC Design

The digital CFD was not tested in this project due to lack of analogue I/O channels in the trigger board available for testing. Nevertheless a standard CFD module was used to digitize the pre-amplified detector signals. The CANBERRA Mod. 2128 Constant Fraction Discriminator [67] is a single CFD module operating at $200MHz$ frequency. This module has got the 40% fraction F. The delay value D is provided externally by the proper length of a cable across the delay connectors. For experimental setups in this project the 50 $\mho$ cable of $1m$ length was used. This will create a delay D of $6ns$ calculated using the equation of propagation delay (D=$1m$/0.55xC for C being the speed of light). The threshold is set to $700mV$ to avoid low-energy $\gamma - rays$. The CFD module provides two NIM output signals (fast negative timing output signals) and two TTL output signals (fast positive timing output signals).

The digital CFD designs considered in this project for future reference make use of trapezoidal filter algorithms. Two algorithms from [31] and [29] of digital CFD were considered in this project. When designing with those equations, an assumption is made that, there is a data series, Trace[i] for i=0,1,2,..., which in the VHDL implementation of this project is a FIFO memory.

$$CFDTrace[k] = \sum_{i=1}^{L} F \times Trace[k-i] - Trace[k-i-D] \tag{4.1}$$

$$FF[i] = \sum_{j=i-(FL-1)}^{i} Trace[j] - \sum_{j=i-(2\times FL+FG-1)}^{i-(FL+FG)} Trace[j] \tag{4.2}$$

$$CFD[i+D] = FF[i+D] - FF[i]/2^{(W+1)} \tag{4.3}$$

In equation (4.1) above, D is the delay, F is the fraction of the original trace and L is the average length for noise reduction. For equation (4.2), FL is called the fast length and FG is called the fast gap of the digital trapezoidal filter. D is called the CFD delay length and W is called the CFD scaling factor. Equation (4.2) was selected and implemented in VHDL for this project. The equation (4.2) was partitioned into two entities, *GenSATaCFD* and *GenSATbCFD*, with

architectural behaviours made up of accumulators to compute the first and second summation operations (For $j = i-(FL-i)$ to $j = i$, and for $j = i-(2xFL+FG-1)$ to $j = i-(FL+FG)$) respectively.

Equation (4.3) computes the CFD value which when above the threshold, a logic value 1 is output, but logic 0 others wise. Implementation of this equation was done by instantiating both *GenSATaCFD* and *GenSATbCFD* and subtracting their outputs. This however was done twice for $FF[i + D]$ and $FF[i]$. The output for the $FF[i]$ part had to be divided by $2^{(W+1)}$ and this was implemented using a shift register to shift ($W + 1$ steps to the right) the output 32-bits vector from the $FF[i]$.

4.2.2 Timing Modules Design

Good care has to be taken when implementing these timing CTP modules as latency (The time to reach the trigger decision) might hinder us from producing a perfect trigger design. Memories were discovered to be essential in holding data during the trigger and careful timing to avoid missing of important events. But these are complex considerations since memories takes up more space in an FPGA and timing-safe coding implies a lot of verification. The trigger timing adjustments accuracy and a low jitter operable experimental trigger are of utmost importance in this project. For the trigger to function with less timing errors, timing algorithms such as delay generators, pulse stretchers, etc. operate in parallel to find the best possible timing settings. These timing algorithms must have less clock skews and gated clocks [10] that can introduce glitches and register incorrect data. The timing safe property of a trigger implies the accurate trigger timing adjustments and low jitter operable trigger logic.

PLL and CRC based Timing Modules Design

The first design technique for the timing components of the CTP includes internal FPGA clock frequency synthesis using on-chip Phase Locked Loops (PLL). This technique increases timing precision upto 2.5ns per clock cycle in the V1495 User FPGA. As depicted by Figure 4.6, the PLL clock output is distributed throughout the CTP. The PLL clock can then be used by different

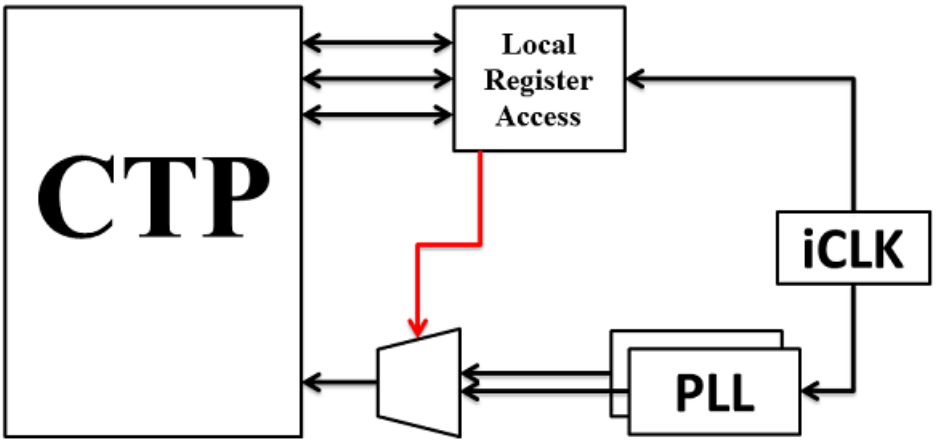


Figure 4.6: Synthesized Internal Clock with PLL

timing modules such as delay generators, stretchers, etc. with the preferred precision. Usage of

on-chip PLL implementation is motivated by capability of dynamic reconfiguration of the PLL settings. This capability allows switching the clock frequency during operation. Increasing the timing precision this way becomes advantageous for trigger processing as sampling is done accurately within a short period of time on-the-fly, thus missing of important trigger events is avoided. PLL instantiations for the clocks with 100MHz frequency (10*ns* per clock cycle) was implemented in this project using Altera mega function wizard. This technique is explained in [9] and shows that the clock can be synthesized with fine resolutions such as 2.5ns, 10ns.

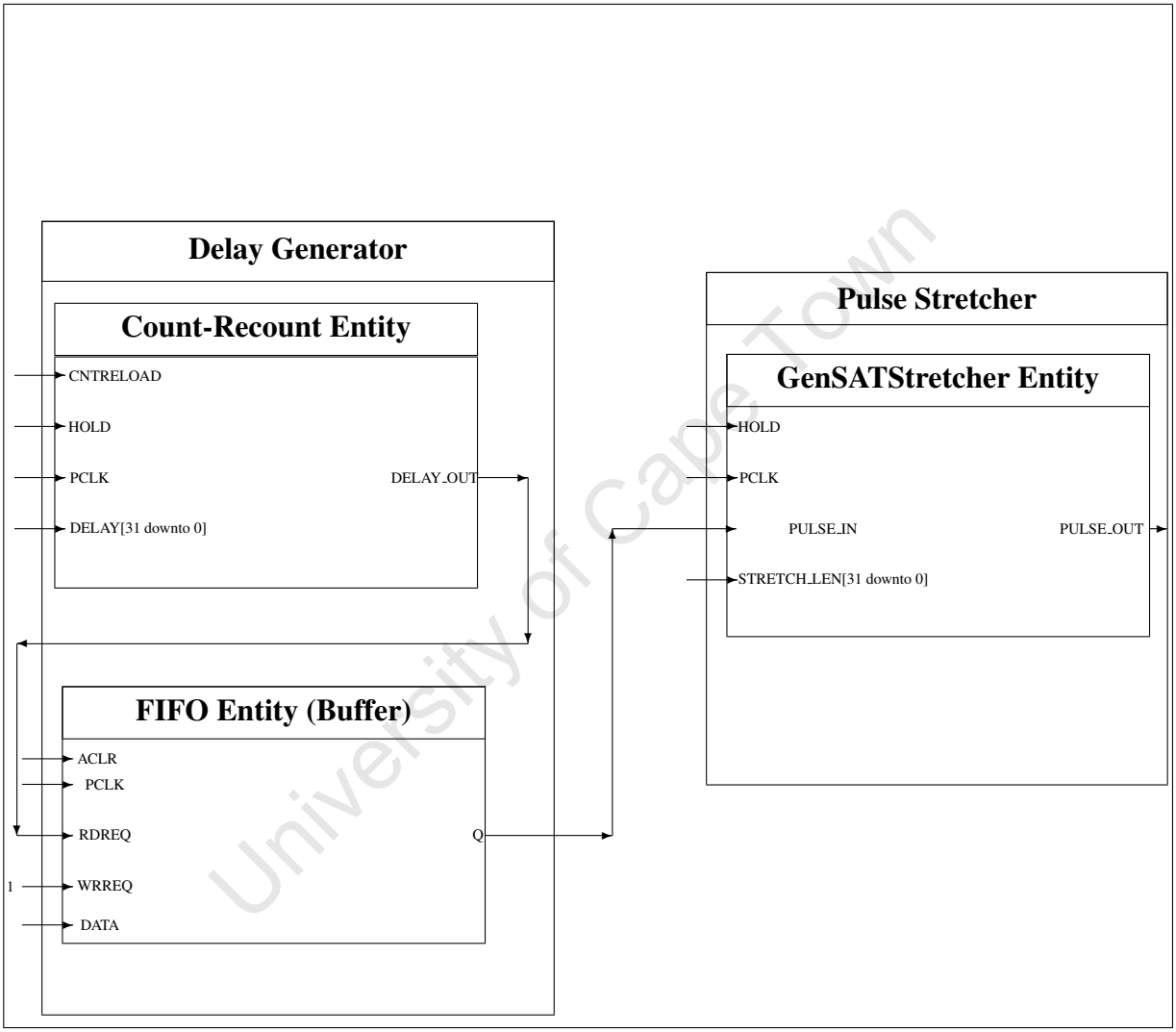


Figure 4.7: Timing modules design: PLL and CRC based GDG timing module consisting of pulse stretcher and delay generator

Figure 4.7 illustrates usage of the combination of PLL clock and a counter-re-counter (CRC) model in gate and delay generator module. The choice of GDG to demonstrate the design of timing modules is because of its composition of a pulse stretcher and delay generator. As depicted in Figure 4.7, a signal delay generator consists of a CRC with PLL clock as input and FIFO to buffer signals to be delayed. On the rising-edge of the PLL clock, an increment to an index is performed by the CRC process. When this index is equivalent to the delay length, a

read request is set to active high. When the read request is active high, the output from the FIFO is enabled. Therefore an input to the stretcher is initiated. The index can be reset to zero when a synchronous clear signal is set to logic value 1 or delay register value changes. Then recounting (incrementing the index value again) to delay the incoming signals. Hence why the technique is called PLL and CRC based timing design.

The delay value must be less than the FIFO length, this case 4096 words, to avoid missing of valid trigger events in triggering systems designed using the triggering platform discussed in this thesis. The maximum of 40.96μs could be implemented in the V1495 User FPGA. Figure 4.8 demonstrates why the maximum delay value must be equal to the FIFO length.

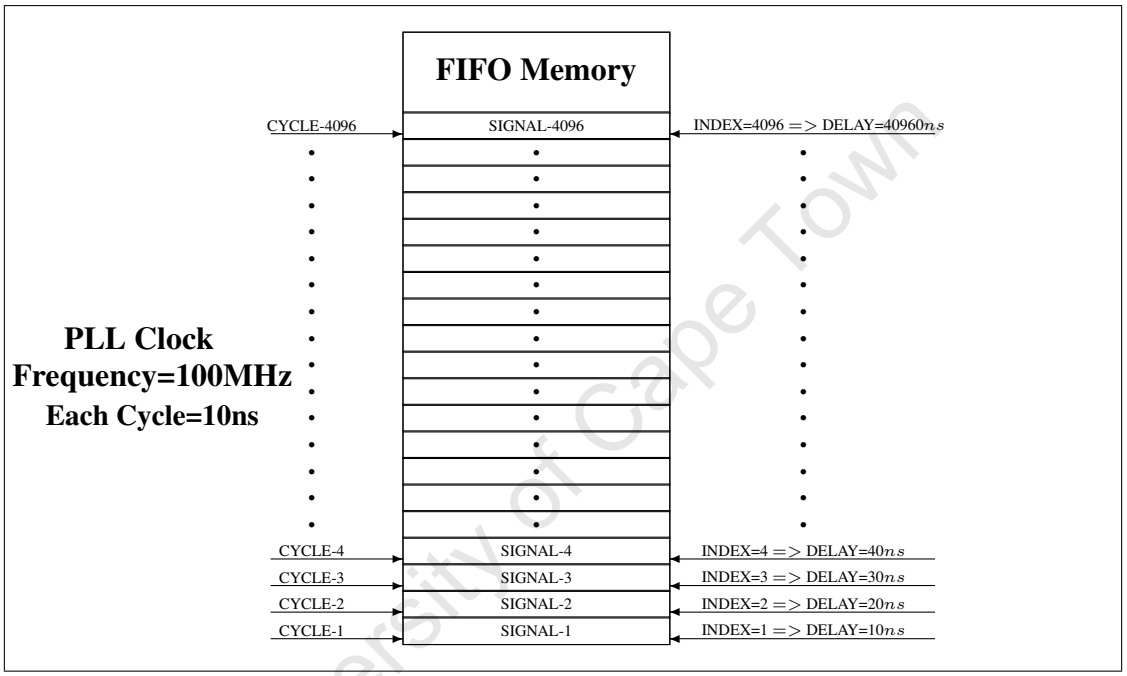


Figure 4.8: Demonstration of FIFO length and maximum delay settable on the Delay generator module implemented

As depicted in Figure 4.8, since the counter-re-counter and FIFO entities are synchronized to the same clock (100MHz PLL clock, $PCLK$ as depicted in Figure 4.7), at every signal input to the FIFO memory, an increment is performed on the INDEX variable of the counter-re-counter entity, which happens every clock cycle (every 10ns). When the FIFO gets full before the $rdreq$ signal is asserted to request a read operation, the $full$ signal will be asserted and no more write operations will be performed. Therefore important detector signals of interest maybe missed. To avoid this, Figure 4.8 demonstrates how the delay length was constraint to values between 1 and 4096. The actual implementation of the registers were restricted to the values up 2047 words (i.e DELAY=REG_RW1(10 downto 0)), implying that the actual maximum delay is 2047x10ns (i.e 20.47μs), since every clock cycle of the 100MHz PLL clock is 10ns as illustrated in Figure 4.8.

Assuming that the delay register is set to hexadecimal value 4, Figure 4.8 suggests that at the first cycle (CYCLE-1), the first input signal(SIGNAL-1) will be inserted into the FIFO,

followed by the second signal (SIGNAL-2) at clock cycle two (CYCLE-2). The third signal (SIGNAL-3) is inserted at the third cycle (CYCLE-3) followed by the forth signal (SIGNAL-4) at the forth cycle (CYCLE-4). At every clock cycle (CYCLE 1-4), the CRC module increments the index value until its equivalent to the delay value in the delay register, during which the delay output will be asserted, so is the $rdreq$ signal to request read operation from the FIFO. When the read operation is requested, the first signal gets output from the FIFO, delayed by current index value (or delay register value) multiplied by $10ns$, this case being 4x$10ns$, which is $40ns$ delay. Since the delay length is restricted to be less than 4096, the delay, and gate and delay generators were considered timing-safe.

Similar operations are made in the pulse stretcher. The slight difference is that the same signal is displayed for the duration of the time until the index is equals to the stretch length. When the index is equal to the stretch length, this index is reset to zero and incremented until it is equivalent to stretch length again. This is repeated over and over during the trigger window.

PDL Based Timing Modules Design

Some programmable FPGA development boards include programmable delay elements (PDL) to reduce glitch power and improve circuit performance via clock skew as explained in [11]. PDL devices were used in this project to allow usage of 1ns precision in delay modules. The gate and delay generator is the perfect example for timing with 1ns precision. Since there are only four (2 synchronous and 2 asynchronous PDL) PDL devices available in the V1495 trigger board used in this project, investigations on how to use them concurrently in more than three timing modules in one experiment were made. Reusing the PDE devices in multiple timing NIM components required implementation of a sequential algorithm that functions as a wait and use technique. For this reason, much time was not spent in using programmable delay lines technique explained in [16]. However a simple gate and delay generator example found in [16] was used to study how PDLs work.

4.2.3 CTP Decision-makers Design

The CTP decision maker determines a valid physics event based on user defined NIM inputs. These input signals are conditioned using the timing modules discussed in the previous section to ensure timing-safety. When latency is well minimized, a coincidence module comes into play.

Listing 4.3: The Coincidence module VHDL code showing how a switch and inputs are combined to make a coincidence checker

```
1. --Library Definition
2.
3. ENTITY GenSATMultBitCoin IS PORT
4. (
5.    BITS     : IN STD_LOGIC_VECTOR(63 downto 0);
6.    SW       : IN STD_LOGIC_VECTOR(63 downto 0);
7.    BIT_OUT    : OUT   STD_LOGIC
8.  );
9. END GenSATMultBitCoin;
10.
11. ARCHITECTURE rtl OF GenSATMultBitCoin IS
```

```
12. BEGIN
13. -- The output BIT_OUT will give a pulse for each coincidence of
    defined simultaneous inputs
14. -- Variable SW represent 64-bits vector of switches set dynamically
15. -- by model-to-VHDL interpreter. If one of these switches is set to
    active
16. -- high then the signal from the corresponding input port will not be
     ignored.
17. -- then a coincidence is expected at that point with another port
    with its
18. -- switch set to active high.
19.
20. BIT_OUT <= (BITS(0) or NOT(SW(0))) and (BITS(1) or NOT(SW(1))) and
    ... (BITS(63) or NOT(SW(63)));
21.
22. END rtl;
```

The coincidence module outputs an active high signal when the specified user input signals are all active high. It is designed using a multiple (64-Bits) inputs AND gate with a control 64-bits switch dynamically set using the model-to-vhdl code generator. The user can specify the inputs by connecting ports of interest to the COIN model. For every port connected to the COIN model, the corresponding switch value is set. Code Listing 4.3 shows the code snipped of the coincidence module in VHDL. To make code Listing 4.3 readable, line 20 was reduced, and full code snipped of coincidence module is shown in appendix C, code Listing C.5. As depicted by code Listing 4.3, when a switch value is set, then the corresponding incoming signal will be checked for coincidence with other bits whose switches are enabled. For instance, if SW(0) and SW(1) are set to active high, then the values for BITS(0) and BITS(1) will be checked for coincidence. The values from BITS(2) to BITS(63) will be ignored. Assuming X=($BITS(0)\ OR\ NOT(SW(0))$) and Y=($BITS(1)\ OR\ NOT(SW(1))$), the truth Table 4.1 shows how values of BIT_OUT interchanges.

Table 4.1: Truth table to show how coincidence operation is tested for coincidence level set to two: BITS(0) and BITS(1)

SW(63-0)	NOT(SW(63-0))	BITS(0)	X	BITS(1)	Y	BIT_OUT
X"00...03"	FF...FC	0	0	1	1	0
X"00...03"	FF...FC	1	1	1	1	1
X"00...03"	FF...FC	1	1	0	0	0
X"00...03"	FF...FC	0	0	0	0	0

Table 4.1 shows that the switch values, SW(0) and SW(1), are asserted, thus the switch hexadecimal value is X"000...03", for values at 63 down to 0. This implies that all other input values, BITS(63 downto 2) will be ignored in coincidence checking. Only BITS(0) and BITS(1) will be checked for coincidence. As shown in Table 4.1, when both BITS(0) and BITS(1) are asserted, the X and Y values are asserted as well, so is the coincidence module output, BIT_OUT. Thus BITS(0) and BITS(1) are in coincidence with each other.

4.2.4 Trigger On-chip Debugging Module Design

A trigger is a complex system and must function predictably under all experimental and user defined conditions, therefore the technical monitoring and trigger path verification are essential. It was realized that the ability to access and check performance of the trigger data must be designed and implemented. So an on-chip debugging (ToC Debugging module) using timing-safe LUT made up of Finite State Machine (FSM) and the 32-bit registers (adapted from [16] examples) was designed.

Local Register Access

Figure 4.9 shows the FSM for the implemented Local Register Access (LRA) module. The

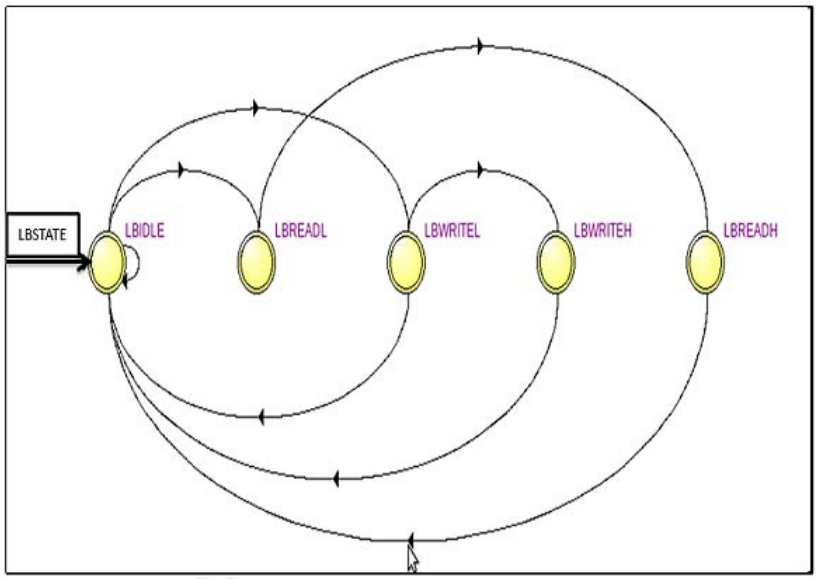


Figure 4.9: FSM for the Local Register Access

LBSTATE represents the LRA states LBREADL, LBREADH, LBWRITEL, and LBWRITEH for Reading lower bits(15 downto 0) of the 32 bit register, reading Higher bits (31 downto 16), writing lower bits to the 32 bit register, and writing higher bits to the 32 bit register respectively. During the LBIDLE (idle state) state, register address sampling is performed to check whether there is write or read access to the register and then set LBSTATE to LBWRITEL or LBREADL accordingly. In the LBWRITEL state, the lower 16-bits are temporarily stored and the next state is the LBWRITEH where the higher 16-bits are concatenated with the lower 16 bits. Using the case statements, a look-up of the address that maps to the registers is made and then a write of the input data into the 32-bit register that corresponds with the VME address is performed. Some registers are read only and some are read-write registers. The read only registers are never under the LBWRITEx (for x=L or x=H) states but only under the LBREADL state for they can only be read, while the read-write register can be accessed in both write(LBWRITEH) and read(LBREADL) states.

When LBSTATE is set to LBREADL (read access), checking of whether the registers are ready to be read is done first, then the look-up of the corresponding register address using case state-

ments follows, hence why called LUT based LRA. The LBREADH state then gets enabled concurrently with saving the lower 16-bits of the register. When all is done, the LRA state transition to an idle state (LBIDLE) and the registers become ready to be accessed again. Table H.1, lists the implemented registers with their VME address, showing also the operations that can be performed on them. When a write operation is performed on each register, a corresponding reload signal is asserted to reset trigger on-chip configurations. The **V1495LocalBusInterface** class will only declare, initialise, and implement the required LRA registers. This was implemented to optimise the generated VHDL code. This Java class works hand-in-hand with the **GenSATV1495Main** class to automatically generate an optimised VHDL code as illustrated in Figure 4.2.

ToC Debugging Module

The architecture of the ToC module including the debugging processor as depicted in Figure 4.10.

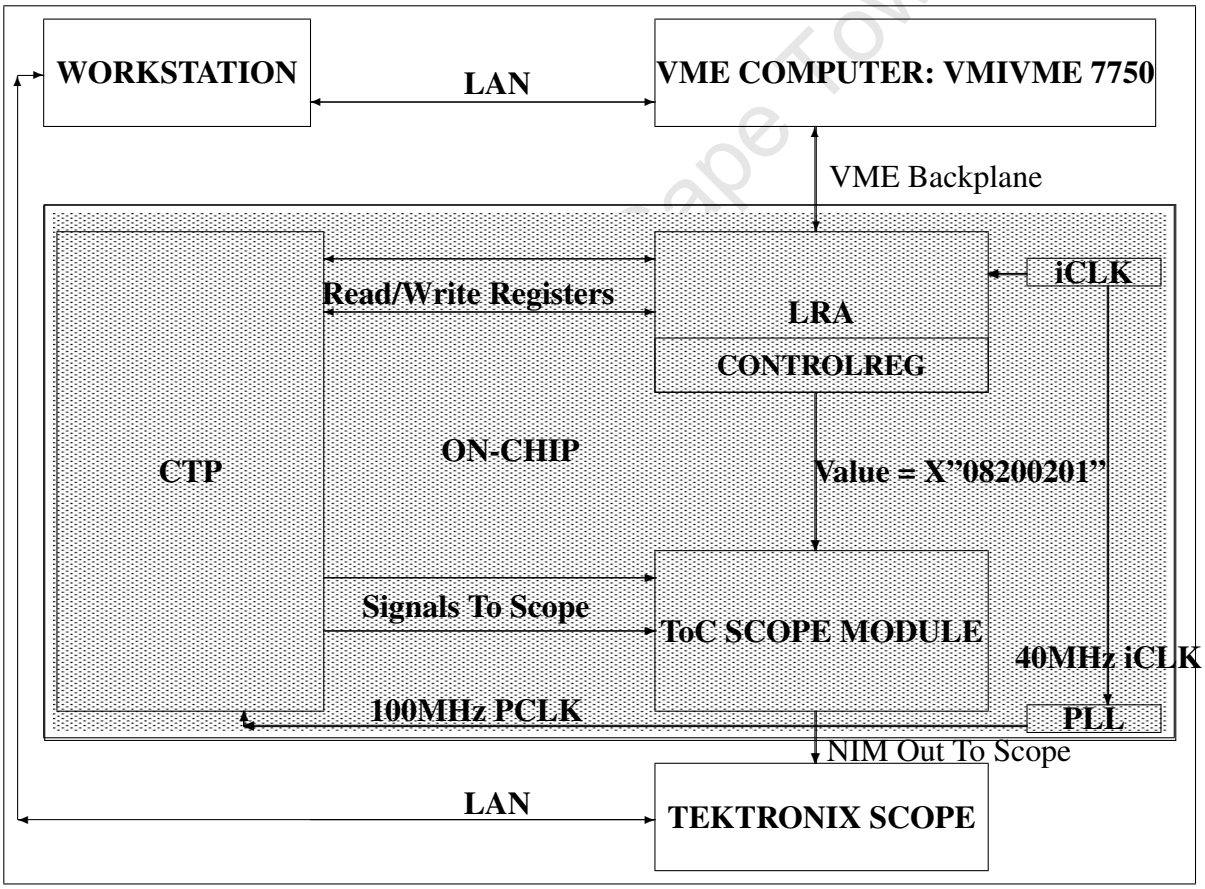


Figure 4.10: Digital Trigger On-Chip Debugging Architectural Overview

Figure 4.10 visually illustrates a single board VME computer, the VMIVME 7750, which is connected to a VME crate backplane on the system-controller/master slot. This computer is running the RCP server application with a VME driver. The V1495 VME trigger board is connected to the VME backplane on the slave slot. The single board VME computer communicates

with the V1495 board via the VME backplane. The ToC scope module, in the V1495 board User FPGA, connects directly to the 4 NIM I/O channels connecting directly to the scope, Tektronix scope for this setup. The scope can then be remotely accessed, on a LAN, using either the implemented RPC GUI or the Tektronix scope web interface using a given IP address, this case 192.21.126.45.

The RPC GUI on the workstation also provides an interface for remote ToC timing-adjustments. Therefore trigger registers can be accessed via the GUI remotely. As depicted by Figure 4.10, the ToC module is controlled by the CONTROLREG register adjusted/set via LRA. This ToC control register uses segmented control register technique to select which input to the ToC Modules can or cannot be displayed on the scope for debugging purposes. The scope module takes as inputs 8 NIM signals and output the selected 4 output signals out of the 32. This is because most of the available scopes have 4 inputs. A selection can then be made of which input to output at which output port by setting the register with 2^n (for n being the input port number to scope module) at any segment (8 bits) of the 32-bit segmented registers settable via the VME.

For instance, to set input 1 to be output at port 1, the register is set with the Hexadecimal X"00000001", or to set input 1 to be output at port 2, the register is set with Hexadecimal number X"00000100", etc. This means to set any input n (for n=0,1,2,...,7), the register is set with hex number X"2^n" at every segment of the 32-Bit register, CONTROLREG. Sample hexadecimal value of X"08200201", depicted in Figure 4.10, means ports 1, 2, 3 and 4 of the Tektronix scope will display inputs 1,2,5 and 8 of ToC scope modules respectively.

4.3 TRIGGER BOARD DESIGN

For testing purposes and trigger verification processes, the VME 6U V1495 board from CAEN was used. The V1495 is suitable for trigger applications[16]. The V1495[16] general purpose trigger board is visually illustrated in Figure 4.11.

4.3.1 User and Bridge FPGAs

As depicted in Figure 4.11 and as described in [16], the trigger board consists of two FPGAs. The first one is the Bridge FPGA, used for the VME interface and for the connection between the VME and the second FPGA (User FPGA) through a proprietary local bus. The BRIDGE FPGA manages also the programming via VME of the User FPGA. The User FPGA is the Altera Cyclone EP1C20 device. The User FPGA manages the front panel I/O channels and is essentially an empty FPGA[16]. It is available to be programmed by the User according to the desired trigger logic functions. The V1495 provides the ability to tailor and reprogram the User FPGA therefore can be used to replace several analogue and digital trigger modules.

The User FPGA space optimization of the trigger logic must be considered since the Altera Cyclone EP1C20 contains only 20,000 LEs. That is very small compared to the number of LEs available in modern (newer versions) FPGA devices such as Cyclone III, Stratix, Xilinx Vertex 5, etc.

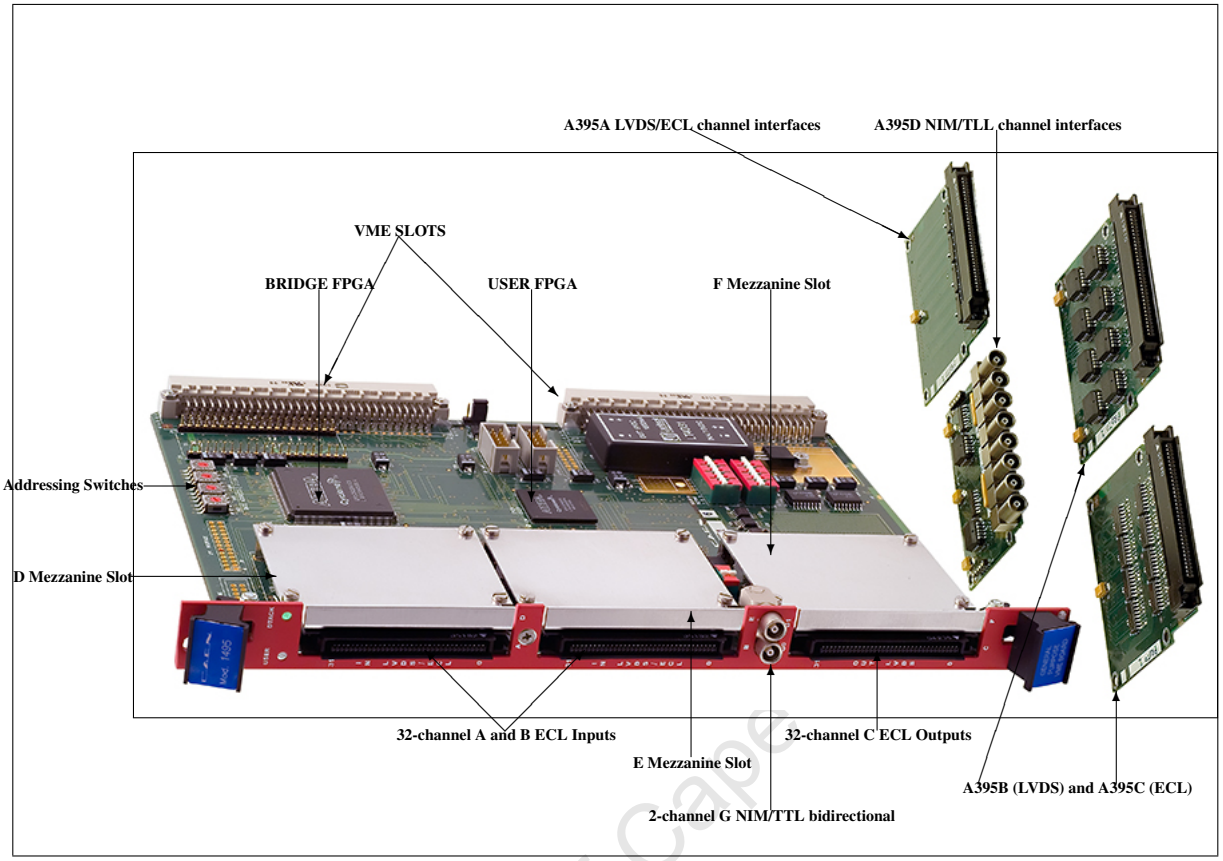


Figure 4.11: V1495 Trigger Board Design

4.3.2 Inputs and Outputs Channels

The I/O channel digital interface is composed by four sections placed on the motherboard. The channel interface can be freely expanded by adding up to three independent mezzanine boards, choosing between the four available types. As depicted in Figure 4.11, the V1495 has two fixed standard ECL/LVDS, A and B, 32-channel input ports, one C 32-channel output port, and two NIM/TTL I/Os (G0, G1). The NIM/TLL channels in the G0 and G1 are bidirectional as well as NIM and TTL level selectable.

There are three other mezzanine expansion slots where extra channels can be fitted to expand the number of I/O channels upto 162. The mezzanine channels interfaces include 32-channel LVDS/ECL/PECL (A395a, A395b and A395c), 8-channel NIM/TTL (A395D), and 8-channel 16-bit resolution analogue (A395E) ports. The LVDS/ECL/PECL and NIM/TTL channels interfaces are bidirectional and are I/O level selectable. As depicted in Figure 4.2 of the graphical system modeller, one can configure usage of this channels accordingly, and the model interpreter will automatically define them in the top-level entity of the triggering system modelled.

4.3.3 VME Addressing and Access

The on-board addressing switches depicted in Figure 4.11 are use to address the board to enable VME addressing. For this project the base address was set to Hexadecimal value 3000. When

the board is connected to a VME crate, the registers can be accessed by setting the address with base address as 30000 (e.g 31030 for register REG_RW2, for 1030 being register address). V1495 comes with a free VME firmware upgrade tool used to send the bitstream to either the Bridge or User FPGA. The Bridge FPGA firmware is proprietary whereas the User FPGA firmware is user-defined. Code Listing 4.4 shows Unix shell commands used to upgrade both the BRIDGE and User FPGA firmware.

Listing 4.4: V1495 firmware upgrade tool Linux commands and VME Universe command

```
# Command for Programming VME FPGA
/root/V1495/V1495Upgrade/V1495Upgrade $IMAGE $ADDRESS vme

#Programming User FPGA
/root/V1495/V1495Upgrade/V1495Upgrade $IMAGE $ADDRESS
# Now tell VME fpga to load the user image into the user fpga.
/root/vmisft-7433-3.6/vme_universe/test/vme_poke -a VME_A32SD -A 0x38016
    -d VME_D32 1
```


4.3.4 Timers for PDLs

Four independent digital programmable (2 synchronous and 2 asynchronous) timers are available in the V1495 board for trigger applications. It is possible to chain them for generating complex Gate/Trigger pulses. For this project, they were used in PDL based timing modules for 1ns precision. [38] demonstrates the usage of these timers.

4.3.5 Trigger Setup Design with V1495

The V1495 trigger designs use the VME protocol. This protocol was used in this project as an interface for setting trigger variables and reading out values of the on-chip registers. The trigger board was fitted with two A395D mezzanine cards for NIM I/O channels. These were selectable and port I/O enabled in the top level entity of the trigger model synthesised. The path for signals from the detectors would therefore go via the pre-amplifiers (Canberra) and analogue CFDs for digitizing before they go into the V1495 board.

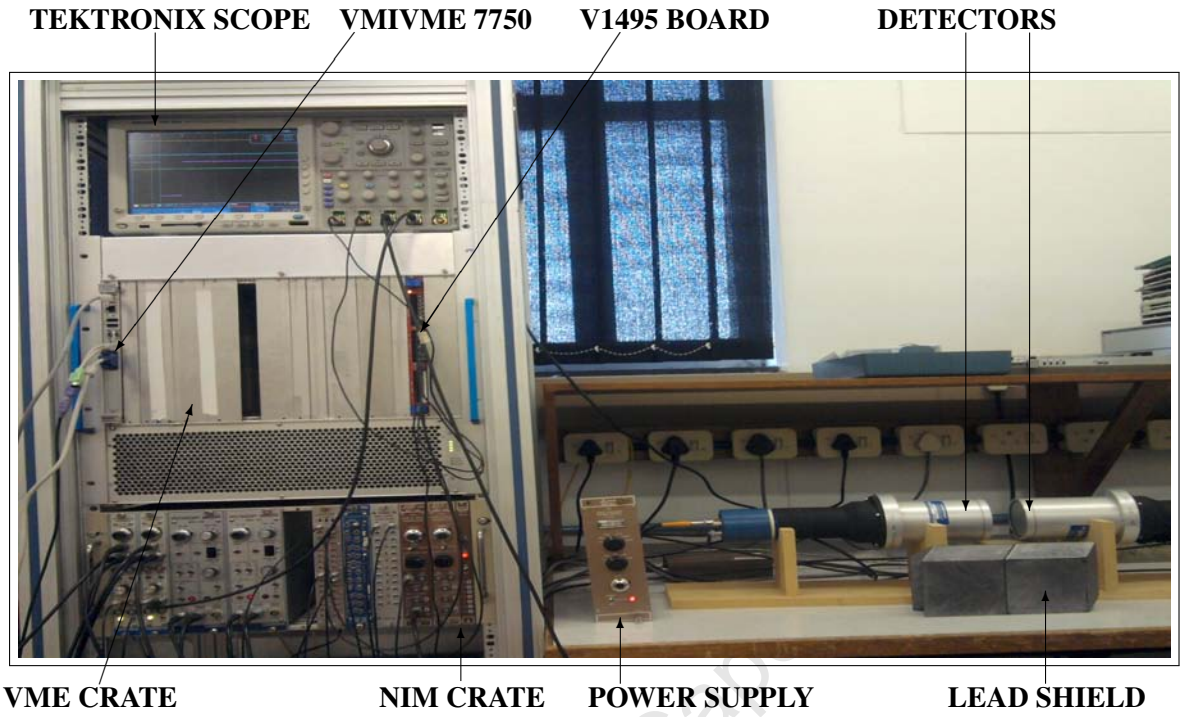


Figure 4.12: Trigger setup design with V1495, VMIVME 7750 VME Single Board Computer, and Tektronix logic analyser and mixed signal scope

The Readout Controller (RC) defined by (W. Parsons, 2010) [41], use the LVDS as inputs to the trigger board. This setup differs from the one used in this project in that, they used the LVDS I/O channels and a signal processing integrated circuit, designed at CERN for the TOTEM experiment, to directly convert the analogue charge deposited on the charge collector into a digital signal. This way the signals paths from detectors are shortened to the V1495 board by directly digitizing them after the detectors.

In this project, however, the ENGELHARD sodium iodide (NaI) scintillation detectors were used as depicted in Figure 4.12. The NaI detectors were power supplied by Ortec High-Voltage power supply. Signals from the NaI detectors were preamplified and digitized using the Canberra pre-amplifiers and CFDs respectively. The digital readout control part of the trigger setup in this project consists of VME crate fitted with the V1495 board and VMIVME-7750 Intel Pentium-III Processor-Based VME single board computer. The V1495 User FPGA is programmed remotely with the CTP bitstream via the VMIVME-7750 computer. In the case of a successful User FPGA programming, the on-chip trigger can be debugged using the Tektronix logic analyser and mixed signals scope. The trigger variable can then be adjusted accordingly via the VMIVME-7750 computer, which is running an open network computing remote procedure call (ONC RPC) server application with a VME driver on a Linux platform. Trigger registers and Tektronix scope displays can be accessed by workstations on a LAN, using the GUI automatically generated after the trigger model VHDL code generation was performed.

4.4 TRIGGER GUI DESIGN

The trigger GUI functions as a remote trigger controller, mainly for timing adjustments and readout of recorded bit patterns. This had to function in real-time, using the monitoring functions similar to the ones used in the Experimental Physics and Industrial Control System (EPICS).

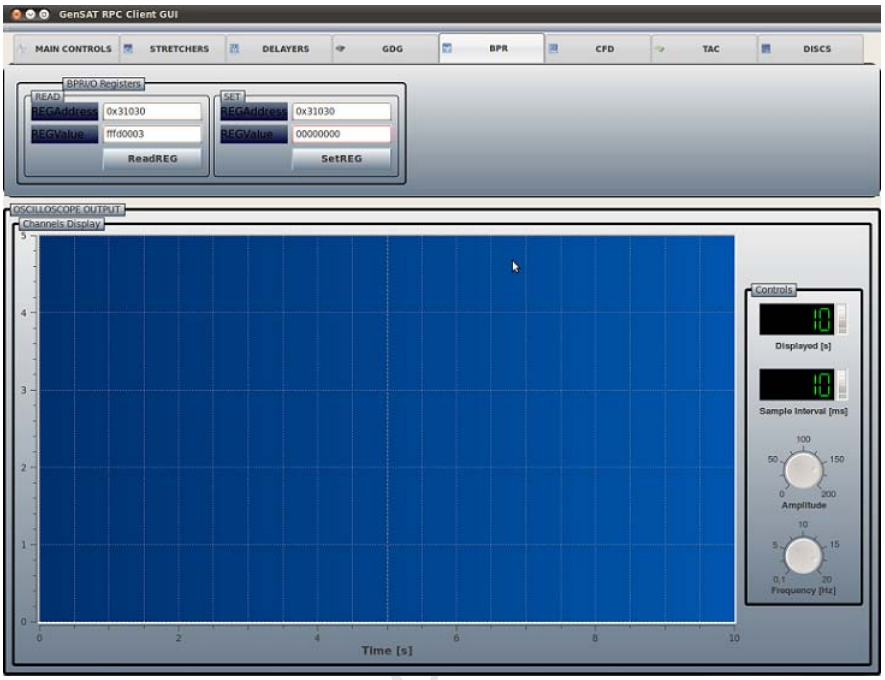


Figure 4.13: Trigger Control GUI Prototype with Register Controls and Scope Display.

EPICS is mainly used at iThemba LABS to monitor, set and read process variables. Unfortunately due to limited time, a simple RPC client and server application was implemented for testing purposes. The RPC Server application makes use of VME driver to implement register address mapping and create a socket that client application can access remotely. The client applications are Qt based (Qt 4.6 QWidgets and QThreads) GUI with XML processing. The XML configuration file is dynamically generated using Java JAXB library as explained earlier in this chapter. The Qt GUI application interprets the XML configuration file and create NIM components tabs that are depicted in Figure 4.13.

4.4.1 Rules of Interface Design

To improve the usability of the trigger control GUI, a well designed interface was implemented. The following eight golden rules of interface design by Shneiderman [28] were a guide to good interactive design with GUI to remotely adjust trigger variables.

- Strive for consistency: Identical terminology of NIM trigger components was used in tabs, prompts, menus, and help screens; and consistent set and read register commands were employed throughout each module tab.

- Shortcuts: The use of tabs catered for reduction of the number of interactions and to increase the pace of interaction by putting front view of every NIM component controls.
- Offering informative feedback: If an error occured or a register is set successfully an appropriate pop-up message is shown. This was implemented using Qt's QMessageBox class.
- Dialog design to yield closure: Sequences of actions (set, and read registers) was organized into each NIM component with each uniquely identified by the register name as defined in the VHDL implementations, i.e If a stretcher uses REG_RW2 as its register, the QGroupBox label will be set to "REG_RW2".
- Offering of simple error handling: using exceptions (simple C++ try and catch statements) the GUI has the ability to detect errors and handle them appropriately.
- Easy reversal of actions: This feature is not implemented yet but must be to allow users to reverse action if need arises.
- Support for internal locus of control: The GUI allows users to initiate read and set of registers by clicking the read and set buttons provided on each register group box.
- Reduction of short-term memory load: The use of tabs to put each NIM component controls visible was used in this case. For each NIM component, register read and write controls are the only ones available to avoid confusion.

CHAPTER 5

SYSTEM EVALUATION AND RESULTS

The sections in this chapter follow the evaluation methodology outlined in chapter 3. The evaluation processes include:

- Modelling and HDL code generation tests
- Space optimization tests
- ToC GUI controller and debugging

The obtained evaluation results are then analysed to determine the usability, effectiveness and efficiency of the triggering platform. This will pave a way to conclusions and further recommendations on the triggering platform discussed. Results that include automatically generated VHDL codes are moved to the appendices to make this dissertation readable. In sections that have such results, the reader will be pointed to the corresponding appendices.

5.1 TRIGGER MODELLING AND CODE GENERATION TESTS

Table 5.1: components with acronyms as shown on blocks and their functional descriptions

NIM Component	Block Name/Acronym	Description
Logic Fans	Fanin,Fanout	Many/Single in/out
Scope	MULT	Selects scope outputs
Gate and Delay Generator	Gate & Delay Generator	Generates delayed gate
Logic Gates (AND and OR)	AND and OR	AND and OR operations
Constant Fraction Discriminator	CFD	Digitizes the input signals
Delay Generator	Delay or Delay Generator	Delay a signal
Stretcher	Stretcher	Widens logic signal
Bit Pattern Register	BPR	Store validity of signal
Coincidence	COIN	Check signals coincidence
Time to Amplitude Converter	TAC	Converts time to amplitude

This section is more about modelling triggering systems. All trigger models presented in this section and the next sections are composed of the NIM components tabulated with their acronyms and functional description as depicted in Table 5.1. The rest of the other modules blocks not listed in Table 5.1 are assumed self explanatory.

5.1.1 Single Model Interpretation and Synthesis

The model interpreter provides an Xcos GUI with a palettes browser. The NIM electronics has its own palette and contains the NIM models abstracting away the VHDL internals. The performance of the model interpreter to automatically generate the code of individual NIM models was tested by connecting the models directly to input and output functions, as shown for the signal delayer model in Figure 5.1. The VHDL code was automatically generated for each NIM model. The delay model in Figure 5.1 was physically replaced, by rewiring

Figure 5.1: Sample Signal Delayer Model

individual NIM models to test timing performance. The generated VHDL code snipped for the delay model in Figure 5.1 is shown in the code Listing 5.1.

Listing 5.1: Delay Model Top Level Entity Automatically Generated VHDL Code Snippet showing only automatic naming and port instantiations of delay generator components

```
library IEEE;
-- List of Library packages included
use work.GenSATPack.all;

entity GenSATMain is
    port (
       -- V1495 Ports Defnitions go here
       -- ...
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
  -- LRA Components Definition
  --Internal Signal Declarations
begin
    -- Settings for Port Levels go here
    --PLL instantiation - Output is 100MHz PCLK
    instance_pll : PLL port map (
            inclk0   => LCLK,
      c0 => PCLK
        );
    -- Dynamic Delay Signal Assignments and instantiations
    E(0) <= DSignal_O_7fd7(0);
    DSignal_I_7fd7(0) <= NOT(D(2));
    DELAY_7fd7 <= conv_integer(REG_RW2);
    --Delay Model-CRC Instantiation with Unique ID as 7fd7
```

```
    GenSATDelayer_7fd7 : GenSATV1495delayer port map (
            HOLD      => HOLD,
      LCLK      => PCLK,
      DELAY     => DELAY_7fd7,
      WRENA     => Dwrreq_7fd7,
      DELAY_OUT             => Drdreq_7fd7
        );
    --Delay Model-FIFO Instantiation with Unique ID as 7fd7
    GenSATDelayerFIFO_7fd7 : gbpr_fifo port map (
            aclr    => Dgbpr_clr_7fd7,
      clock   => PCLK,
      data    => DSignal_I_7fd7,
      rdreq   => Drdreq_7fd7,
      wrreq   => Dwrreq_7fd7,
      empty   => DisEmpty_7fd7,
      full    => DisFull_7fd7,
      q   => DSignal_O_7fd7,
      usedw   => Dusedw_7fd7
        );
        --LRA Instantiation Will Go here
end;
```


The code Listing 5.1 shows the top-level entity named GenSATMain. This entity is followed by the architecture implementation of the GenSATMain entity. In this architecture behaviour, delay model signals are declared. Each signal name contains a general text and the delay model unique identity number. For instance, the input signal name is **DSignal_I_7fd7**. The **DSignal_I** is a general input signal name for the delay model, whereas the **7fd7** uniquely identifies this particular delay model. Following the signal declarations are dynamic instantiations and port mappings.

For the delay model depicted in Figure 5.1, the instantiations include the **GenSATV1495delayer** entity instantiation and the Altera single-clock first-in-first-out (SCFIFO) mega function, **gbpr_fifo** instantiation. The **gbpr_fifo** SCFIFO is used for buffering the incoming signal while counting from zero until the set delay value is reached. The counting is done by the CRC entity, **GenSATV1495delayer**. The delay model instantiations are also uniquely identified by the same model ID as the one appended in signal names.

The last instantiation for the delay model architectural behaviour is the LRA. The LRA is fully discussed in section 4.2.4. Appendix C shows code snipped and models for other NIM models starting off with the full version of code Listing 5.1. Table 5.2 is for single digital NIM model and the time in micro-seconds taken by the model interpreter tool flow to generate and synthesize their VHDL code. From Table 5.2, an average of 43.7ms and standard deviation of 4.7ms were obtained. Therefore the time taken by the model interpreter to generate code for an individual NIM electronics model that is directly connected to the I/O functions is 43.7± 4.7ms.

Table 5.2: Table of NIM Models with Time taken to generate VHDL code

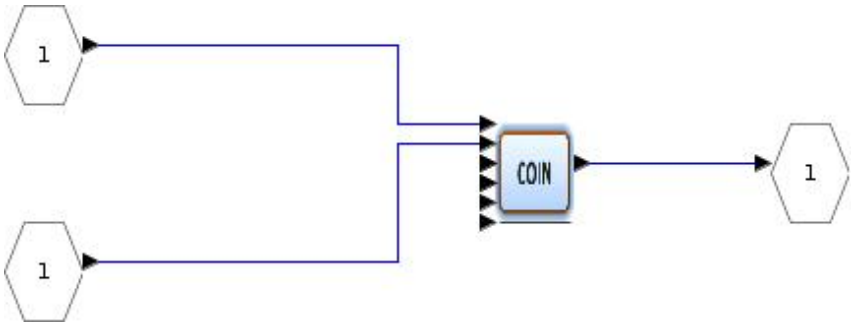
NIM Models	VHDL ACG Time (ms)	VHDL Synthesize Time (ms)
Bit Pattern Register	55	18303
Coincidence Logic	45	16970
Constant Fraction Discriminator	47	18023
Event Trigger	46	17209
Fan In	44	17184
Fan Out	43	17622
Gate and Delay Generator	51	23096
Logic Gates (AND, OR, etc.)	44	16897
Pulse Stretcher	44	17736
Scope	46	15095
Delay Generator	50	23550
Time to Amplitude Converter	48	16213

5.1.2 **Multiple NIM Models Interpretation and Synthesis**

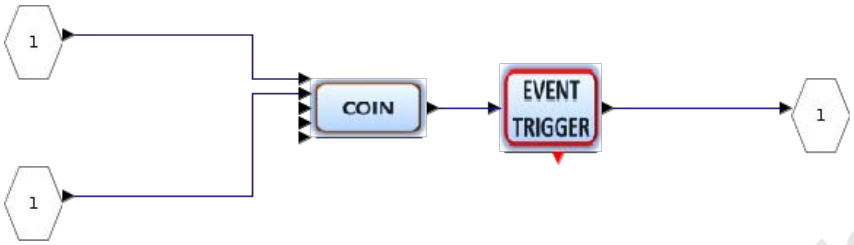
To further provide convincing results of how long it takes for model interpreter to generate the code for the modelled trigger system, NIM models were gradually added to the components of the logic circuit depicted in Figure 5.2a. The model in Figure 5.2a will gradually be build towards a 7-modules trigger model in Figure 5.2c. The trigger models in Figure 5.2 demonstrate examples of the CTP. Initially the experiment is started with the setup in Figure 5.2a of one coincidence module. To improve the functionality, an event trigger module was introduced to the setup. The event-trigger module has both the control (red) and logic display (black) output ports. With the event trigger added to the model, the trigger model will contain 2 modules as depicted in Figure 5.2b. Code generation and synthesis times were measured for each case a model was added to the trigger logic diagram. The values were added to Table 5.3. Table 5.3 led to the creation of a graph of number of modules against time taken to generate and synthesize the VHDL code as illustrated in Figure 5.3.

Table 5.3: Table of Number of NIM models with Time taken

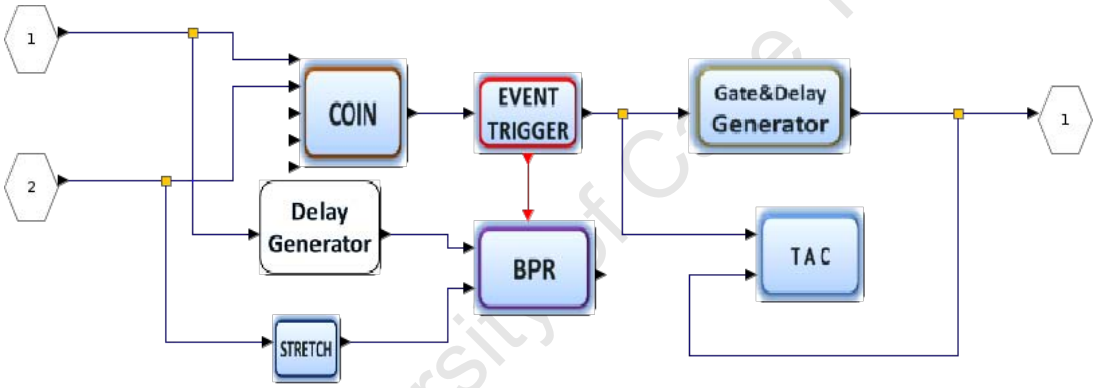
Number of Model	Time to Generate VHDL Code (ms)	Time To Synthesize Model (ms)
1	48	17053
2	53	17077
3	59	17236
4	57	18447
5	57	18550
6	56	18615
7	55	18603



(a) Initial trigger model - Coincidence case



(b) 2 NIM Models in a Trigger Logic



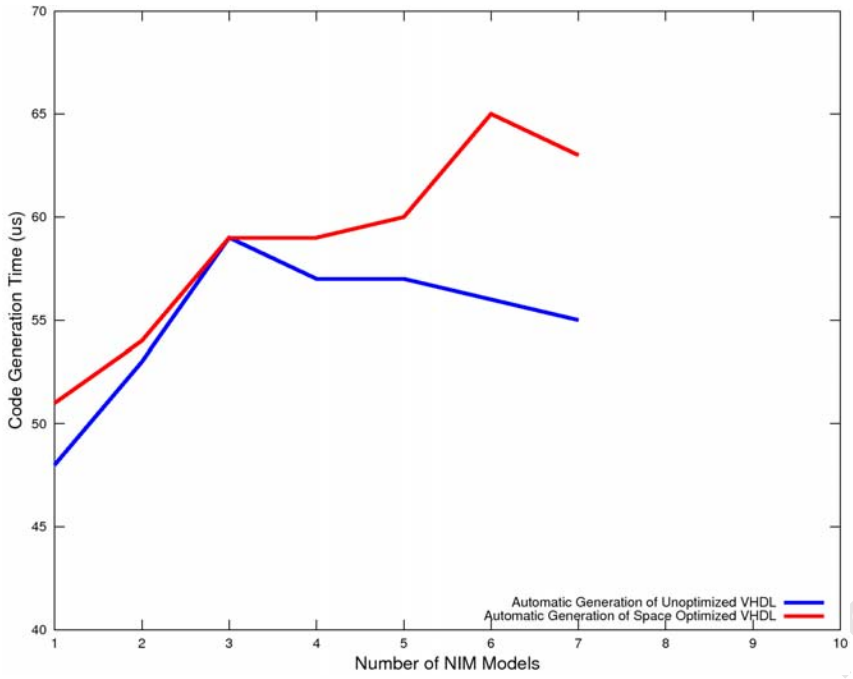
(c) 7 NIM Models in a Trigger Logic

Figure 5.2: Trigger models tested for performance

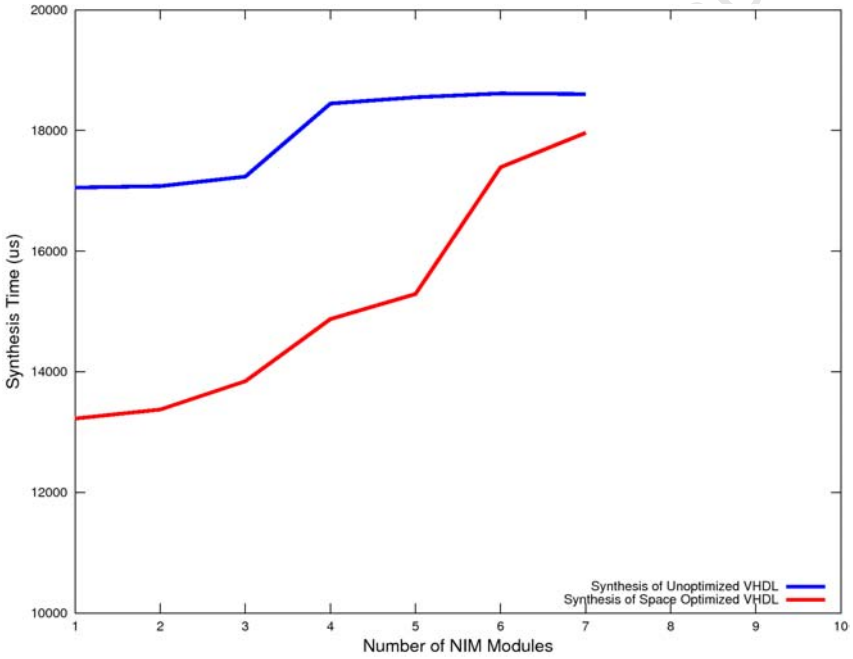
5.1.3 Multiple NIM Models Space Optimization

This section addresses the question of whether small-to-large trigger experiments can be modelled and synthesized to fit into the V1495 trigger board as mentioned in the hypothesis in Section 1.5. Optimizing space occupied by the trigger models is of significance because multiple NIM components can be used repeatedly to produce an optimal trigger at a low cost. The number of logic elements to be programmed into the FPGA increased when NIM models were gradually added to the trigger model in Figure 5.2c, and this consequently increased the risk of running out of logic elements on the User FPGA. This can be decreased by reducing the number of buffers used in the delay modules (delay generator, and gate and delay generators). This will introduce some glitches and wrong data maybe recorded during the valid trigger event, therefore the idea of reducing buffers was rejected.

The LRA implementation discussed in chapter 4 used fixed number of the 32-bit read and/or



(a) Graph of time taken to generate VHDL code for the trigger model against number of NIM models in it



(b) Graph of time taken to synthesize the trigger model against number of NIM models in it

Figure 5.3: Graphs of Trigger Model Code generation and Synthesis times

write registers, whether used in the trigger or not. The LRA, as said before in section 4.2.4, was used to give remote access to the trigger-on-chip (ToC). The LRA allows users to remotely perform on-chip timing adjustments and readout some data for analysis and further processing. The

elegant way to optimize space in the user FPGA was to automatically generate the LRA. The LRA VHDL code generated dynamically contains only the required 32-bit read-write registers and gets rid of unused registers.

The model interpreter tool also constrained the clock asynchronism. The timing modules clock was synthesized using PLL to increase the frequency to 100MHz (10 ns per clock cycle). The LRA clock is set to the V1495 default 40 MHz (25ns per clock cycle) clock which is synchronised with the Bridge FPGA of the V1495 board. This ensured that no race conditions and synchronization problems were encountered. Having optimized the space this way, a table of number of NIM models with time taken to generate and synthesize the models was reconstructed. Reconstruction of Table 5.4 was done by following the same steps as in subsection 5.1.2, where NIM modules were gradually added to the trigger model at hand to improve functionality, then measuring times for synthesis and code generation.

Table 5.4: Table of Number of NIM models with Time taken (Dynamic LRA Code Generation)

Number of Model	Time to Generate VHDL Code (ms)	Time To Synthesize Model (ms)
1	51	13225
2	54	13374
3	59	13843
4	59	14874
5	60	15290
6	65	17391
7	63	17961

The average synthesis time difference was calculated to be -2782ms. The -2782ms time was calculated by taking synthesis times in the first Table 5.3 and subtract from those in the second Table 5.4. The resulting negative value implies that the time taken to synthesize the trigger models was decreased. Table 5.4 and Table 5.3 both indicate a drastic increase on the time to synthesize the trigger model when the delay modules are added. The sudden increase can also be noticed in Figure 5.3b of the graph of synthesis time against number of models, where there is sudden increase in graph steepness. The increase in time is between 3 and 6 models (i.e. when delay modules are added) as depicted in Figure 5.3b for graph of synthesis time against number of models, when the generated code is space optimized (i.e. Less number of LEs), even when it is not. This is due to Altera Single Clock First-In-First-Out (SCFIFO) mega functions used as buffers in most timing models (i.e delay, gate and delay generators, etc.). Although the altera mega functions are well optimized [17], using the SCFIFO in this timing modules introduces extra logic elements in timing models hence increases time taken to synthesize them.

On the contrary, Table 5.4 showed better results in synthesis optimization at an expense of slightly increasing the time to generate the code. This is because the more the number of 32-bit read-write registers in a trigger model, the more the time it takes to dynamically generate VHDL code for LRA. The average code generation time difference was found to be 2.143ms. This value is positive and implies an increase in the code generation time. As depicted in Figure 5.3a for graph of time to generate VHDL code, the red graph line represents time taken

to generate the space-optimized VHDL code whereas the blue line represents the time taken to generate the VHDL code without optimized VHDL code. The red line is slightly above the blue line, which means that, the time taken to generate the VHDL code with only required LRA read-write registers is more than the time taken to generate VHDL code with fixed LRA read-write registers.

This means that dynamically generating the LRA VHDL code is time expensive but optimizes the space occupied by the trigger model to be programmed into the FPGA. This is further proven by the graphs illustrated in Figure 5.3. As depicted in both graphs 5.3a and 5.3b, the red-line is for optimized trigger model VHDL code whereas the blue-line is for unoptimized trigger model VHDL code. This graph-lines overlap in synthesis and code generation graphs, which shows that when space-optimizing a trigger model VHDL code, more time is taken to generate that code and less time is taken to synthesize the model VHDL code.

5.1.4 Modelling Complete Experimental Physics Trigger

This section demonstrate how most of the trigger concepts discussed in chapter 2, section 2.1, can be modelled using the triggering platform under discussion. This will help determine how usable this triggering platform is. The modelling of one of the biggest experiment at iThemba LABS, the AFRODITE was done for this purpose. This however results in a lengthier VHDL code generation and synthesis cycle time, as well as limitations to the range of triggering resolution that the system can provide. The AFRODITE comparative experimental physics trigger was incrementally modelled using stretchers, delays, coincidence levels, logic gates (AND, OR, etc.).

As depicted in Figure 5.4 of the AFRODITE model, the AFRODITE consists of 24 photovoltaic, 32 low energy photon spectrometer (LEPS) and 28 CLOVER detectors energy signals. Therefore the input signals to the AFRODITE CTP are 91 including the 7 BGO shield signals. Modelling the AFRODITE commenced with the signal conditioning of the 24 Photovoltaic energy signals. An assumption is that the incoming 24 timing signals (outputs from spectroscopy pre-amplifiers) are digitized successfully using analogue CFDs. Figure 5.4 depicts that the V1495 Mezzanine slot A is used as 32 ECL input channels to the CTP model. The LVDS and ECL channels support up to 32 bits of signal inputs hence why they were chosen for this model instead of the A395D NIM channels, which are only 8. The digitized 24 photovoltaic signals coming through the A channels are split to both multiple I/O OR gate and pulse stretchers. In the AFRODITE photovoltaic signal conditioning model depicted in Figure 5.4, each block was assigned unique alphanumeric identity. As shown in code Listing Appendix I.1, the last 4 alphanumeric characters of each NIM model ID were appended to the corresponding NIM component instantiation name as expected.

As depicted by Figure 5.4 the digitized Photovoltaic Cells, Clover, and LEPS signals enters the AFRODITE CTP through the A,B,D and E mezzanine slots with ECL input channels respectively. Since the BGO shield digitized input signals are only seven, the E slot ($E-NIMInputs$ block as depicted in Figure 5.4), was fitted with A395D mezzanine card with eight NIM input channels instead of the 32 ECL or LVDS channels. The rest of the NIM blocks such as pulse stretchers, delay generators, logic gates, coincidence levels, etc., are pipelined to perform signal conditioning, readout, buffering, and CTP decision-making accordingly.

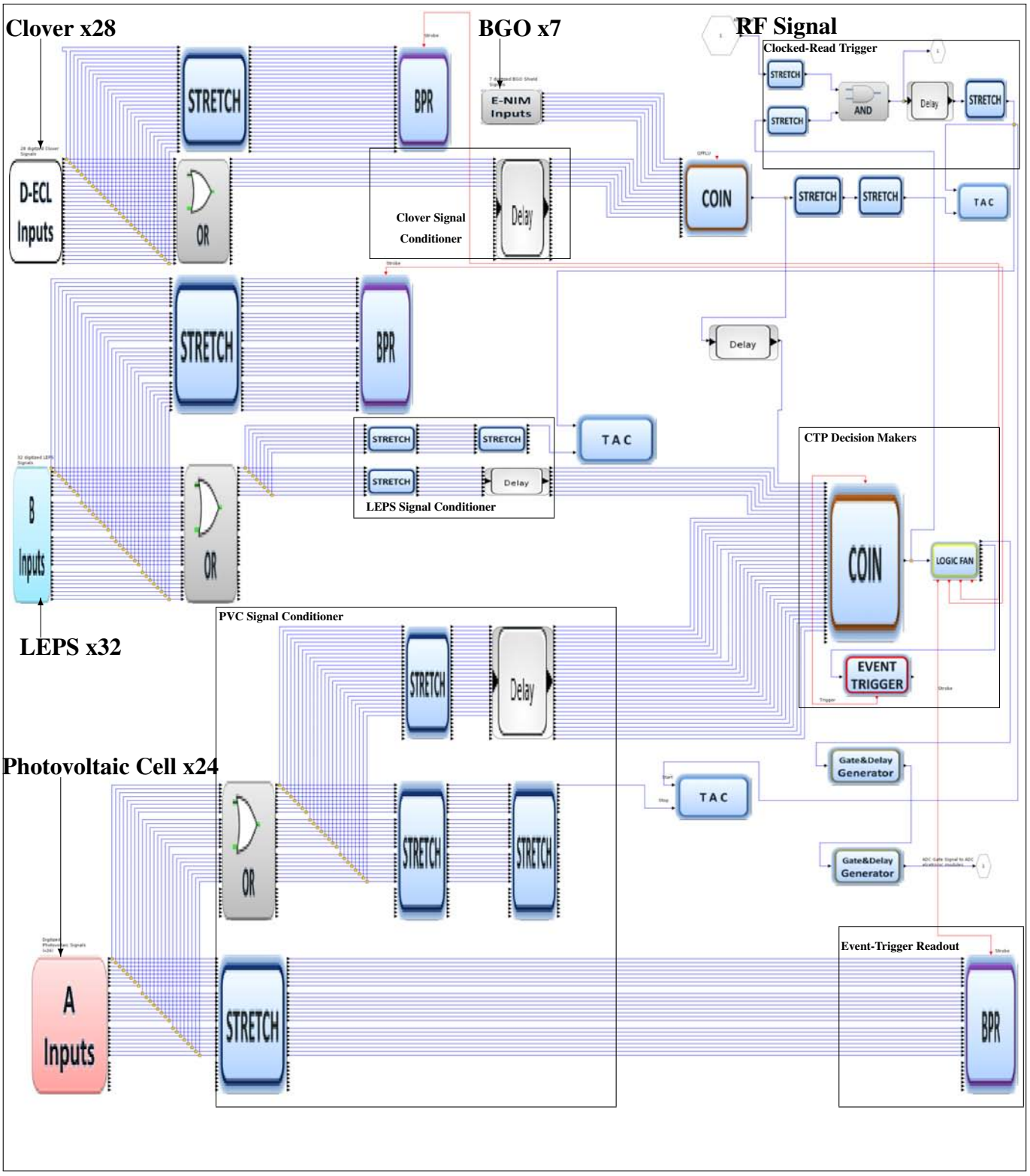


Figure 5.4: AFRODITE Electronics Model highlighting readout modules, signal conditioners and CTP decision-makers

These CTP blocks are graphically represented as their name acronyms in the model in Figure 5.4. Due to multiple I/O channels, some models were stretched so that all the block ports were shown appropriately in the graphical model editor. The VHDL code for the complete AFRODITE experiment was generated in $576ms$ and successfully synthesized in $47s$. The total logic elements for the full AFRODITE comparative trigger model are 2055, that is 10% of the V1495 User FPGA LEs. Although the AFRODITE model depicted in Figure 5.4 provides demonstration of FPGA space optimization and trigger concepts, the comparison to its standard version is scheduled for next year. This is due to the AFRODITE trigger setup being used for other experiments at the moment hence a slot for testing is available next year. However, a medium K600 Experiment was successfully tested and compared to its standard trigger version. The K600 Spectrometer operate at low frequencies of $1KHz$. The K600 Spectrometer model is depicted in Figure 5.5.

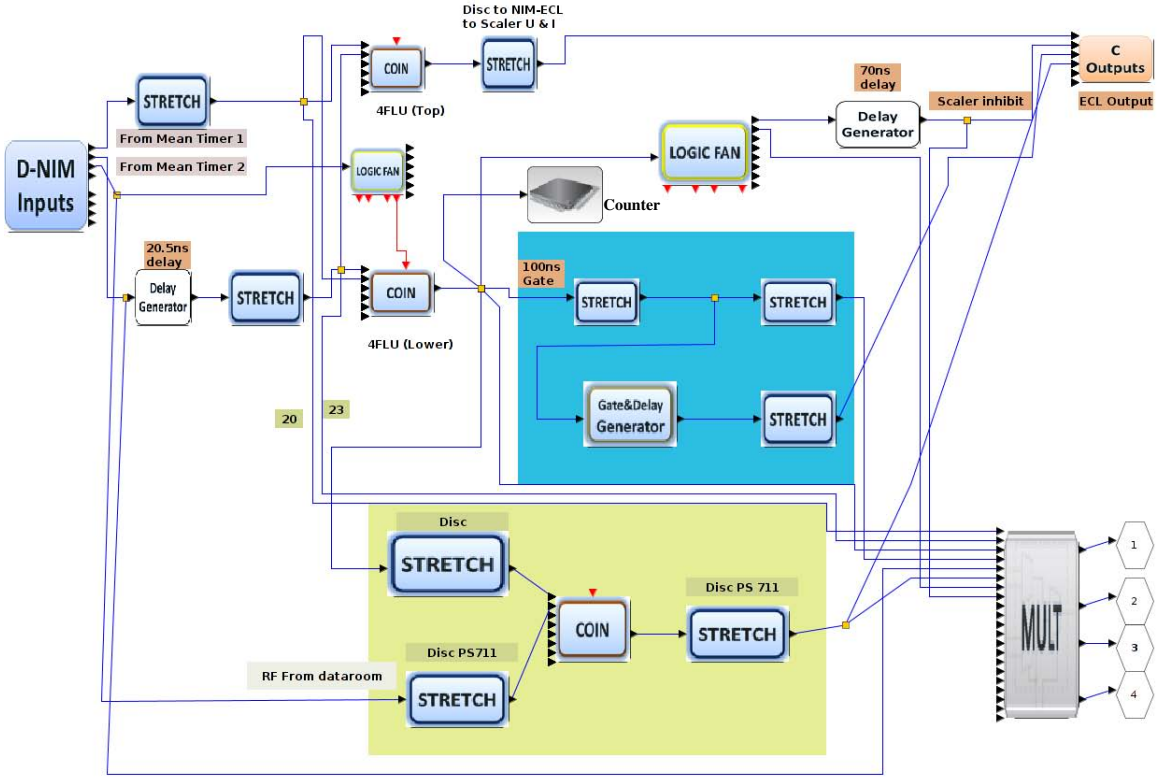


Figure 5.5: The K600 Spectrometer Experiment Model

As depicted in Figure 5.5, there are three NIM input signals. The two signals from the mean timers (Mean Timer 1 and 2) are further conditioned using the timing modules before they are used to determine the trigger decision. These two signals are then sent as input to the CTP decision making modules, which performs an AND operation on both of them. The coincidence modules marked 4FLU (lower) and (top) are the CTP decision makers. A digital counter module is connected to the output of the coincidence module marked 4FLU (Lower) to count the valid trigger events. The value from this counter was compared to values from analogue counters which were both connected to the V1495 trigger board NIM output and the standard K600 Spectrometer experiment trigger output. The duplicate of the input signals from the mean

timers was sent to both the analogue and the digital K600 Spectrometer trigger. Five tests were performed and the counter reading with delays were recorded in Table 5.5. As depicted in

Table 5.5: Table of results obtained

Counter 1 (A)	Counter 2 (D)	Digital counter	Variable Delay (ns)	Jitter (ns)
3,003	3,003	3,003	89.6-93.5	4.5
20,255	20,255	20,255	89.6-93.6	4.6
69,433	69,433	69,433	89.6-93.6	4.6
12 725	12 725	12 725	89.6-93.4	4.4
48 583	48 583	48 583	89.6-93.6	4.6
102 174	102 174	102 174	89.6-93.6	4.6

Table 5.5, the delay of the digital trigger as compared to the standard analogue K600 trigger ranged from 89.6 to 93.6ns. Table 5.5 also show that the trigger output jitter was found to be 4.6ns. During all the five test cases, the digital counter and the analogue counters recorded the same trigger events. Therefore timing related errors such as the 4.6ns jitter and 89.6ns delays were ignored.

5.2 TRIGGER ON-CHIP DEBUGGING

In this section, each NIM model is compared with the corresponding standard analogue NIM electronics in subsection 5.2.1. All tested NIM models and their timing diagrams are shown in Appendix B. Section 5.2.2 demonstrate how a synthesized timing-safe trigger model can be debugged.

5.2.1 Digital NIM Performance Comparison to Standard Analogue NIM

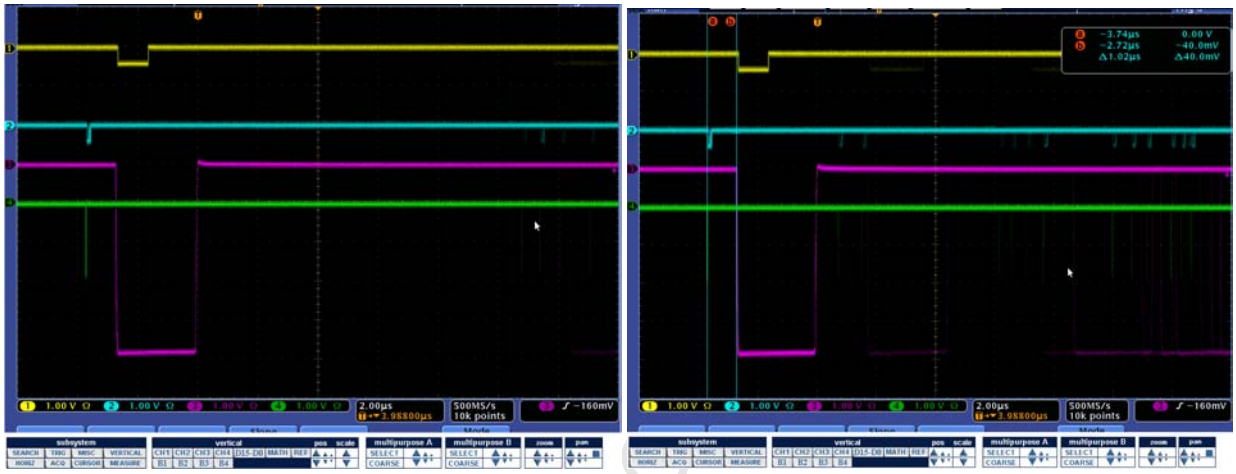
A sample gate and delay generator model with its timing diagram is included in this subsection for demonstration purposes. As depicted in Figure 5.7a, channel 4 (green) is the incoming

(a) Gate and delay generator modelc

Figure 5.6: Gate and delay generator model

signal from the analogue CFD. The analogue CFD is the universal source to both the V1495 board and the analogue gate and delay generator. It is important to note that the CFD output is a NIM signal, thus it is inverted (fast negative output signal). This implies that even the output from the trigger system will be inverted, meaning when a signal is active high, the signals will

be inverted as depicted in Figure 5.7a. When the channel 4 (green) is active high, a gate of a given length is generated at channel 3 (purple). The channel 3 represents a signal from the analogue gate and delay generator. The gate generated at channel 3 is delayed by a manually set delay value. The length of the generated gate is also manually set using adjustable knobs on the standard analogue gate and delay generator. Channel 2 (cyan) in Figure 5.7a represents the output signal from the V1495 without any further processing. This was done to determine the I/O delays of the V1495 board. Channel 1 (yellow) represent an output signal from the digital gate and delay generator programmed into the V1495 board.



(a) Analog and Digital Gate and Delay generator timing diagram

(b) Analog Gate and Delay generator delay measurement

(c) Digital Gate and Delay generator delay measurement

(d) Digital Gate and Delay generator gate measurement

Figure 5.7: Gate and delay generator model timing diagrams extracted from Tektronix scope

Comparison of the standard analogue NIM electronics with the digital NIM model implemented in this project was done by calculating the difference of delays of the reference signal from the CFD and output from either the analogue or digital module. Figure 5.7b shows that the delay measurements were done using the cursors (orange background) a and b of the Tektronix scope.

Figure 5.7b, the analogue GDG generated a gate, channel 3 (cyan) and was delayed by 1.02 μs when the input signal, channel 4 (green) from the CFD is active high. On the other hand, the digital GDG in the V1495 generated a gate and was delayed by 1.06 μs. This value differs from the set value of 1μs delay for both the analogue and digital GDG. For the analogue GDG, an extra delay of 0.02μs was read, while for the digital case the additional delay was 0.06μs. The gate set for the digital GDG was also 1μs. Figure 5.7d shows the generated gate width of the digital GDG of 1.02μs.

Using the same analysis provided for the GDG model in this section, the same procedure was followed for the remaining modules. Therefore a table of performance comparison was constructed as depicted in Table 5.6. Individual NIM models and their timing diagrams are included in Appendix B. From Table 5.6 values, the average 0.016057μs delay was calculated,

Table 5.6: Table of analogue NIM electronics against digital NIM electronics performance

NIM Model Name	analogue delay(us)	digital delay(us)	Difference
Coincidence Logic	0.0307	0.0382	0.0075
I/O Channels	0.0016	0.0037	0.0021
Fan In/Out	0.0304	0.0383	0.0079
Gate and Delay generator	1.02	1.06	0.04
Logic Gates (AND, OR, etc.)	0.0269	0.0328	0.0059
Pulse Stretcher	0.0421	0.0511	0.009
Delay generator	1.02	1.06	0.04

whereas the mean standard deviation was 0.001634057μs. Therefore the average digital NIM modules delay was found to be 16.057$\pm$0.233 ns as compared to the standard analogue NIM components.

5.2.2 Modelling and Debugging Timing-safe Trigger

This section demonstrates how remote ToC debugging and timing adjustments can be performed. This section will also show how the correct data pattern can be recorded in an experimental physics triggering system. The tests were made on a triggering system that consists of the sodium-22 (^{22}Na) source and two detectors as illustrated in Figure 5.9. The tests will demonstrate how a low jitter operable reconfigurable triggering system can be modelled and implemented on an FPGA and use to test coincidence in two detectors with the ^{22}Na as the radio-active source, and show how an optimal pattern recording accuracy can be achieved.

To initiate the experimenting process, similar steps followed in subsection 5.1 will be carried out. The final basic trigger model is depicted in Figure 5.8. The trigger model VHDL code was generated in 13820μs. For this section, the automatically generated VHDL code is listed in Appendix D. This was done so that this section is easy to read. As depicted in Figure 5.8, a coincidence is tested in two γ-ray detectors in coincidence. During radioactive decay of ^{22}Na, two gamma rays are radiated back to back due the annihilation of positron (e^+e-) due to the β^+ decay of the ^{22}Na. The positron annihilates with electron to produce two gamma rays at an angle of 180 degree, to conserve momentum. The setup of two detectors facing each other and

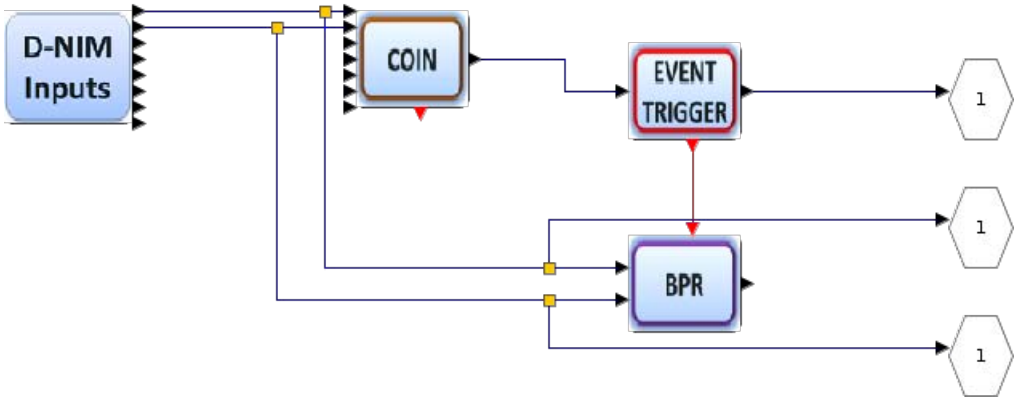


Figure 5.8: Initial basic trigger model

the ^{22}Na source centred as depicted in Figure 5.9 support the 180 degrees angular separation of the gamma rays. This way the chances of the coincidence events are maximized. Coincidence level is set to two since there are only two detectors. The detectors, Detector 1 and Detector 2 in Figure 5.9, signals are sent to the CFD which will then send square pulse signals as input to the V1495 board, through the A395D mezzanine board fitted on slot D, with port level select set to NIM. For safety and reduction of radiation intake, the Lead containers and shield were used as depicted in Figure 5.9.

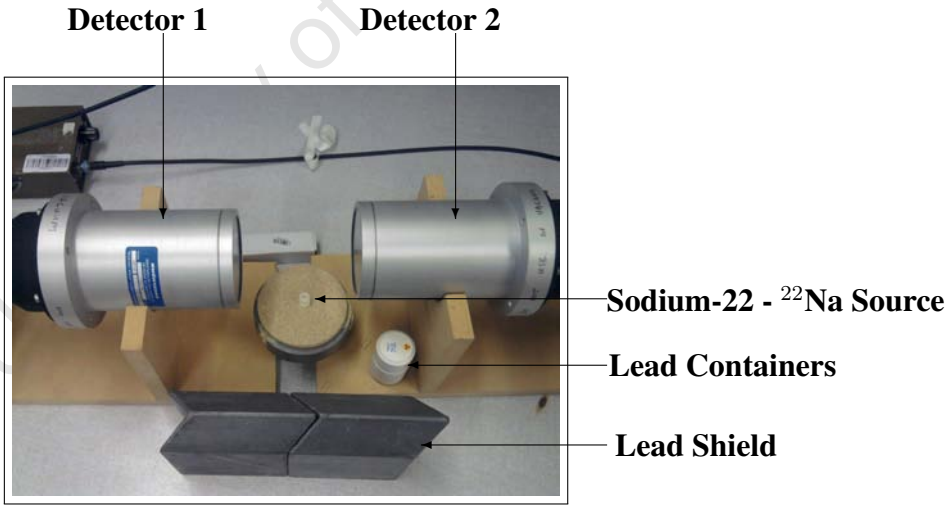


Figure 5.9: Two Detectors and ^{22}Na Source in a 180 Degrees Setup

For the trigger model in Figure 5.10, NIM input signals from CFD take two paths in the V1495 User FPGA with different propagation delays. The trigger decision is solely dependent on the path via the event trigger. If a coincidence occurs, an event trigger is generated. At the positive-edge of the event trigger output, a bit pattern register (BPR) records the two detectors output pattern. The coincidence and event trigger modules act as the CTP decision maker which signals the BPR to temporarily store the events data pattern. The latency caused by, ensuing instantaneous detector variations, CFD to V1495 cable delays, or CFDs, will cause the BPR to

record the wrong bit pattern. For the model shown in Figure 5.8, the output value of the BPR

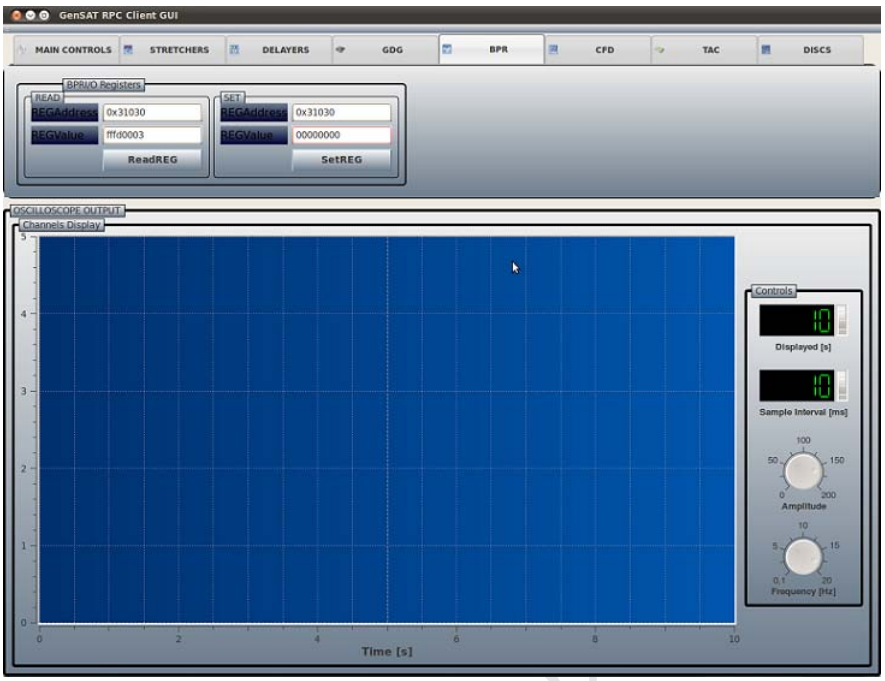


Figure 5.10: GenSAT Multi-threaded RPC Client-Server Readout GUI

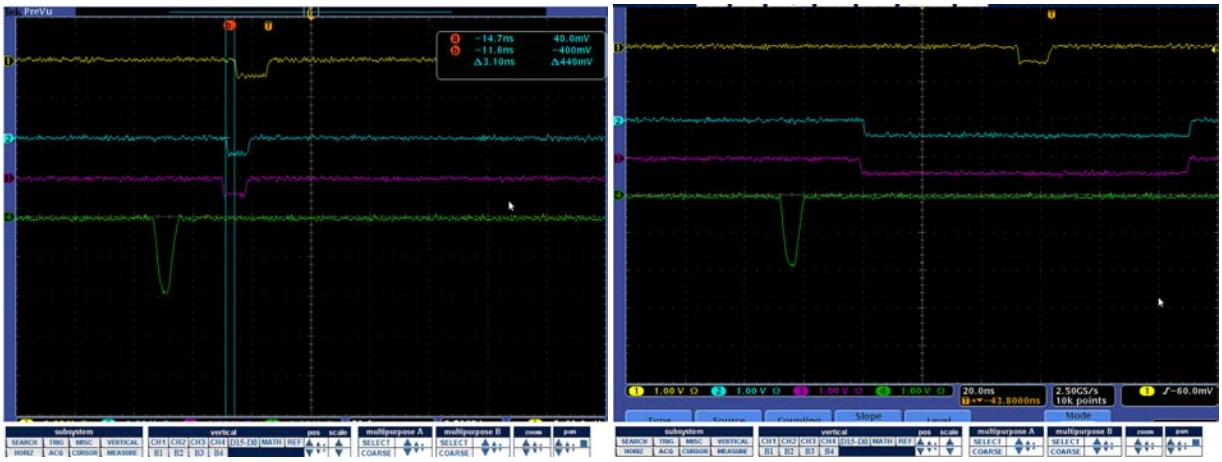
can have values of 0 (00), 1 (01), 2 (10), and 3 (11). Hundred BPR values were read and written to a text file using GenSAT multi-threaded RPC client-server read-out GUI.

Figure 5.10 depicts the GenSAT multi-threaded RPC client-server read-out GUI with BPR tab in view. The BPR output values are variable because by the time the trigger decision is made the detector values had already changed. The timing diagram on the scope, as depicted in Figure 5.11a, shows the detectors outputs (channels 2 and 3) firing at the same time and the event-trigger output (channel 1) becomes active high 3.10ns later. The event trigger timing diagrams shown in Figure 5.11 depicts that the event trigger happens variable nano-seconds after a coincidence occurred. Values read from the scope display shows variable latency values (variable time to reach trigger decision) ranging from 100ps-to-4.10ns.

To ensure recording the correct data, the model was modified by adding the pulse stretchers between the BPR and the detectors to stretch the incoming pulses to the BPR. As depicted in Figure 5.12, the coincidence output was delayed such that the event trigger will be delayed as well. The delay and stretch values were set via the GenSAT GUI depicted in Figure 5.10. The event trigger becomes active high 20ns after the coincidence between the two detectors, the 100ns stretched detector values are still active-high. The timing diagram of the trigger model in Figure 5.12 is depicted in Figure 5.11b.

The modelled trigger system in Figure 5.12 is done to ensure that the detectors correct values are read despite of the jitter[1] present in the event trigger output. However some bit-patterns of interest were still missed in the triggering system illustrated by Figure 5.12. Table 5.7 shows

[1] Variable Latency - Time to reach the trigger decision



(a) Digital event-trigger model delay measurement (b) Modified basic trigger model timing diagram

Figure 5.11: Basic trigger model with its timing diagrams

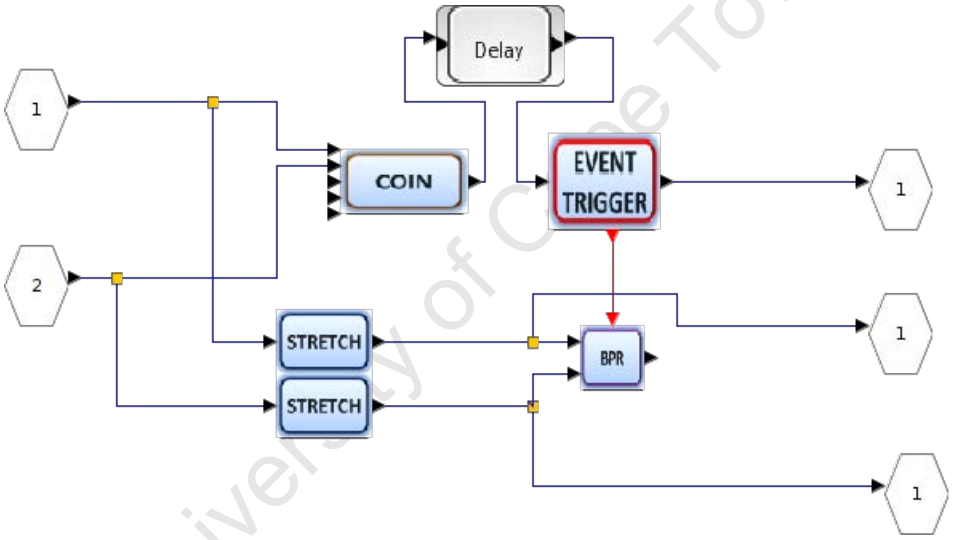


Figure 5.12: Modified basic trigger model

the BPR readout values of the triggering system depicted in Figure 5.12. Table 5.7 still show alternating values of the BPR values as hexadecimal 0,1,2, and 3. The constant hexadecimal **"fffd"** value is appended by the universal VME driver when mapping the VME values so it can be ignored.

To overcome varying BPR readout values, the timing-safe triggering system has to be modelled. The triggering system must record correct data continuously. The correct data for the triggering system consisting of two detectors and 2-input-bits BPR that depends on the positive-edge of the event-trigger is always logic value "11" (decimal and hex value 3). Varying BPR values occurs due to either input signal jitter[2] or latency[3]. But the latency and its jitter has already been covered in trigger model earlier in this section. It was shown that to overcome errors caused by

[2]Variable delay of the incoming signal.
[3]Variable time to reach trigger decision.

Table 5.7: Modified basic trigger model: BP Register read-out values

Values(0-25)	Values(26-50)	Values(51-75)	Values(76-100)
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0003	fffd0000	fffd0000
fffd0003	fffd0001	fffd0000	fffd0000
fffd0003	fffd0001	fffd0000	fffd0000
fffd0003	fffd0001	fffd0000	fffd0000
fffd0003	fffd0001	fffd0000	fffd0000
fffd0003	fffd0001	fffd0000	fffd0000
fffd0003	fffd0000	fffd0000	fffd0000
fffd0003	fffd0000	fffd0000	fffd0000
fffd0003	fffd0000	fffd0000	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0002	fffd0001
fffd0003	fffd0000	fffd0000	fffd0001
fffd0003	fffd0001	fffd0000	fffd0001
fffd0003	fffd0001	fffd0000	fffd0001
fffd0003	fffd0001	fffd0000	fffd0001

latency, signals can be conditioned for alignment by delaying the signals. This requires prior knowledge of propagation delay variations of signals and timing adjusting them to overcome latency. But the jitter for input signals caused by ensuing instantaneous variations caused by

Figure 5.13: GUI View for Setting Scope ToC debugging CONTROLREG with hexadecimal value 32160801 that will display ToC Scope inputs bit 0 and bit 2 at channels 1 and 2 of the Tektronix scope respectively

CFD operations, cable noise, and cable delays was not covered.

Table 5.8: Timing-safe trigger model: BP Register read-out values

Values(0-25)	Values(26-50)	Values(51-75)	Values(76-100)
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003
fffd0003	fffd0003	fffd0003	fffd0003

To compensate for the input signal jitter, a timing-safe trigger model was implemented (modelled and synthesized) by stretching the input signals. Jitter may differ per trigger setup depending on what kind of electronics are used. The timing-safe trigger model logic diagram is illustrated in Figure 5.14. As depicted by Figure 5.14, 2 input pulses are stretched for $1000ns$ using two pulse Stretchers. The stretch lengths were set to the default value of $1000ns$ because the CFDs used were of the same model and the Tektronix scope output showed small but significant amount of jitter. Stretching these signals with the same length fixed timing errors introduced by input jitter. This $1000ns$ stretch length and the ToC scope control register can be adjusted using the GUI depicted in Figure 5.13. After stretching the signal, the coincidence is checked and if it exists the valid event is generated. At the positive-edge of the input trigger event, the BPR records the 11 binary value. The 11 value recorded by the BPR shows that the detectors fired in coincidence relative to the user-defined coincidence window, hence the timing-safe trigger model.

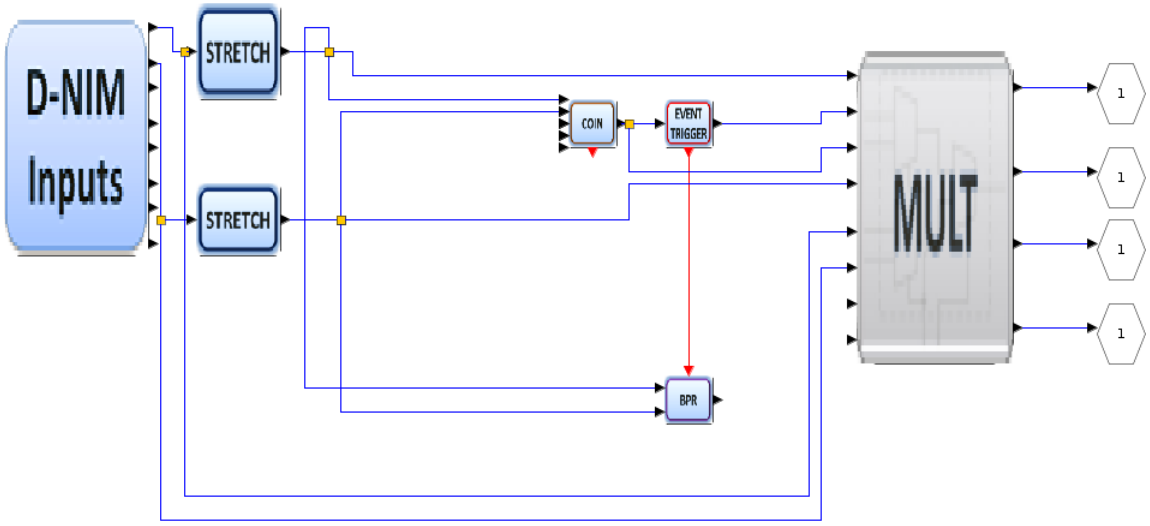


Figure 5.14: Timing-safe trigger model with Scope Module for debugging

Table 5.8 show the hexadecimal value **fffd0003** read 100 times. The values were read by the GenSAT trigger read-out GUI every second by polling the RPC-Server with VME driver for registers address mapping. Figure 5.15 depicts the selected scope display of bits 0 and 2 of the

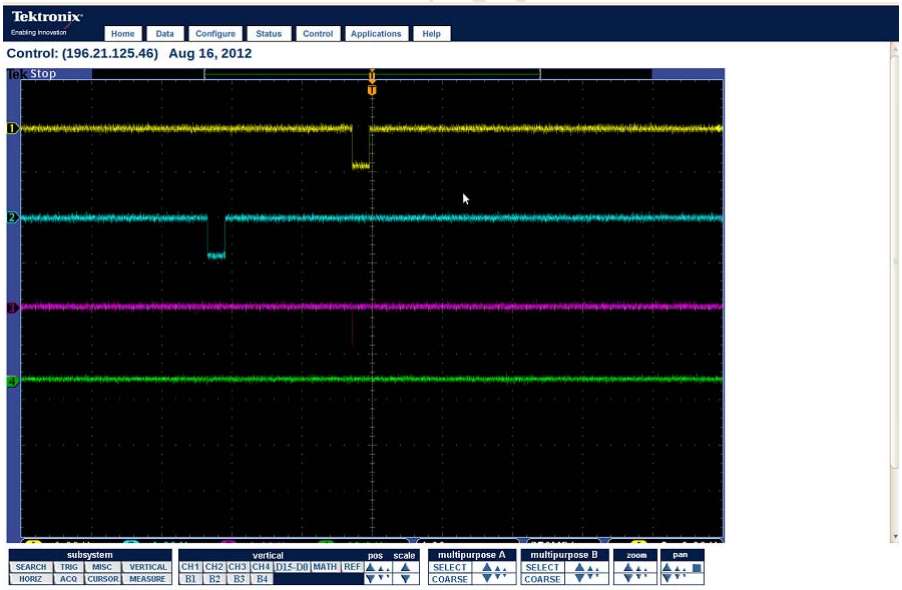


Figure 5.15: The Tektronix Scope Display for CONTROLREG set to hexadecimal value 32160801: Channels 1 (yellow), and 2 (cyan) of Tektronix scope displaying inputs bit 0 (from top stretcher of Figure 5.14), and bit 2 (from bottom stretcher) respectively

Scope model input in Figure 5.14. As shown in Figure 5.13, the scope register value was set to hexadecimal value 32160801, and only valid values are 08 (2^3) and 01 (2^0) at segments 1 and 2 of the 32-Bits registers. Therefore the two output channels 0 and 2 displayed as depicted in Figure 5.15 are correctly mapped to channels 1 and 2 of the Tektronix scope.

5.3 RESULTS SUMMARY

This section provides a summary of the results obtained in this chapter. The results summary covers 1) Model-to-VHDL interpreter performance; 2) FPGA space optimization; 3) Comparison to standard NIM electronics; and 4) Techniques used to implement a timing-safe trigger system.

5.3.1 Model-to-VHDL Interpreter Performance

The speed of the model interpreter code generator is directly proportional to the number of dynamic NIM models present in a trigger model. The more the number of dynamic models with registers to implement in the LRA VHDL code, the more time it takes to generate VHDL code. The speed of the model-synthesis, however, is directly proportional to the number of logic elements present in each model. The more logic elements in a NIM trigger logic, the more time taken to perform model synthesis, assembly, fitting and timing analysis.

5.3.2 FPGA Space Optimization

The VHDL code and FPGA space optimization are also the major factors of synthesis time. Reducing the number of registers used per experimental physics trigger and reducing the FIFO lengths was realised as the best technique for User FPGA space optimization. However, reducing number of FIFOs in a trigger model was not possible for important physics event could be missed. Optimising the space was highly achieved in the AFRODITE model, with 10% of LEs occupying the User FPGA of the V1495. The AFRODITE synthesizable model provided assurance that the triggering platform discussed in this thesis is usable in small to large physics experiments. The AFRODITE Model demonstrated how the trigger concepts such as clocked readout, signal conditioning, CTP decision making, I/O connections and buffering were incorporated in the project under discussion. The AFRODITE FPGA implementation was not functionally verified due to lack of resources such as detectors and sampling ADCs. But another experiment, the K600 Spectrometer experiment, was modelled, synthesized and tested. The results as compared to its standard version is summarised in the next section.

5.3.3 Comparison to Standard NIM Electronics

The comparison of standard NIM analogue chains to the digital NIM components presented in this project, showed that, the digital implementation is $16^{+}_{-}0.233\ ns$ slower, as calculated from Table 5.6. This delay is tolerable because a $16^{+}_{-}0.233\ ns$ delay is small therefore it can be ignored since most experimental frequencies are in a few MHz (e.g 40MHz for the L1 CTP [2]). From the $16^{+}_{-}0.233\ ns$, a $15ns$ I/O delay is already specified for the V1495 trigger board. This was verified by connecting inputs directly to the board outputs. A real-time trigger experiment to detect the ^{22}Na source was performed to show how the triggering platform can be used in real-world radio-active detection. ^{22}Na was detected by checking the Coincidence of two Detectors, and the BPR was signalled correctly to record a pattern. The COIN and event trigger modules act as the CTP decision maker. When a valid trigger event occurs but wrong data is recorded in a given trigger window, timing-adjustment can be made till accuracy is maintained.

The K600 Spectrometer is a medium-sized experiment with four analogue inputs. These input

signals were digitized using the Canberra Mod. 2128 CFDs set with 700mV threshold. The $1m$ cable was used in this experiment too. The $4.6ns$ jitter on the trigger output was measured in this experiment, however the delay as compared to the standard version of the same experiment increased to $89.6ns$. Although this delay will still further be investigated, it was ignorable since the counters which were used to count the valid trigger events (in both the digital trigger and the standard trigger) measured the same values.

5.3.4 **Model-based Synthesis of Timing-Safe Trigger Systems**

Initial experiments showed timing problems, particularly long latencies and jitter. For the trigger to function with less timing errors, timing algorithms such as delay generators, pulse stretchers, etc. operate in parallel to find the best possible timing settings. These timing algorithms have less clock skews and gated clocks to avoid glitches and register incorrect data. These timing modules can be chosen and used appropriately to fit the trigger requirements for timing adjustments. FIFO memories as buffers were carefully implemented to avoid missing of valid user specified trigger events. For instance, in a delay generator HDL abstraction, delay value must be less than the FIFO word length. PLL instantiations for the clocks with 100MHz frequency (10 ns per clock cycle) was implemented in this project using Altera Quartus mega function wizard. The choice of the 100MHz PLL frequency synthesis was due to less jitter which was recently proven to be $4.6ns$ for the K600 Spectrometer experiment performed at iThemba LABS. The same $4.6ns$ jitter was measured using the Tektronix scope for the NaI experiment discussed in Section 5.2.2.

CHAPTER 6

CONCLUSIONS AND FURTHER WORK

6.1 CONCLUSIONS

The model-based triggering platform that was developed in this research project has provided significant time saving for the physicists that use NIM electronics in their experiments at iThemba LABS. For example, the trigger configuration that previously took a day to configure manually took less than half the time with the new system. The software like design processes, provided by the triggering platform, allow NIM components to be used repeatedly in a single trigger model design. Therefore the main goals such as significant time saving, speeding up of the design and setting-up of experiments, as well as reducing the wear and tear of dismantling and connecting equipment, were achieved by implementing a model-based synthesis of timing-safe real-time trigger system.

Significant advantages of this model-based design for triggering systems include the fact that it facilitates rapid design iterations and it moves the verification processes all the way to the beginning of the design cycle. This helps detect triggering system specification related errors, design errors, and implementation errors early. For this project, the counter part played by the model interpreter is providing ways to which the users (Physicists) can reproduce the trigger with out the cost of losing previous configurations of the previously set-up trigger. All timing related errors are already taken care of, hence designing with the triggering platform discussed in this thesis, does not require any specific HDL coding, reconfigurable and FPGA applications development technical skills from Physicists and engineers at iThemba LABS.

The software-based trigger system allows integration with other version-control systems that even with the same change made to a given trigger design at hand, changes may be reverted to previous versions, provided current settings do not suffice requirements of the given trigger circumstances and delay adjustment parameters. With the automated VHDL code generation that support code reuse and code partitioning, trigger design paths are shortened while at the same time, interchangeability and flexibility of the classical NIM based trigger designs is preserved, whereby the user can configure the optimum system for a particular application and later easily restructure the instruments as required for different experiments or measurements. Users can also update or expand an existing system with a few new modules, thereby augmenting the value of instrumentation on hand.

The generated VHDL code by the implemented triggering system is synthesisable and was optimised to fit the V1495 user FPGA. Memory components made up of FIFOs were used as buffers in timing modules to avoid missing of interaction of interest in trigger systems. The FIFOs were limited to 4096 words, allowing timing modules to have maximum safe pulse stretch lengths and delays of up to 20.47μs, in 10ns precision. Alternative design involved using on-board programmable delay elements. But there were only four asynchronous timers available and could not be reused in more than four trigger modules. The triggers synthesized were 16$\pm$0.233 ns delayed, when programmed into the V1495 board, as compared to the analogue trigger setups. A maximum of ten read-only, and twenty five read and write registers can be used per trigger timing adjustments, real-time configurations and readout. The optimal trigger performance is solely dependent on user specified physics events. Timing adjustments can be made on-the-fly to reach the low jitter and latency trigger with optimal performance. This triggering platform therefore covers most of the triggering systems concepts such as clocked-readout, event-triggered BPR readout, signal conditioning and CTP decision making. Thus the project discussed in this thesis is a success.

6.2 RECOMMENDATIONS FOR FURTHER WORK

An improvement to the triggering system presented in this dissertation would be an increase in timing precisions, using FPGA devices with PLL synthesis upto 1000 MHz clock frequencies. Also, the solution to a full model-based triggering platform is reducing, as much as possible, the detectors output signals propagation path to the CTP. To reduce cost and provide a full cost effective triggering platform, both in terms of power and money, a reconfigurable hardware platform that functions with more than 120 analogue channels is proposed. The proposed platform must certainly implement several of reconfigurable sampling ADCs which can be tailored and reprogrammed as required.

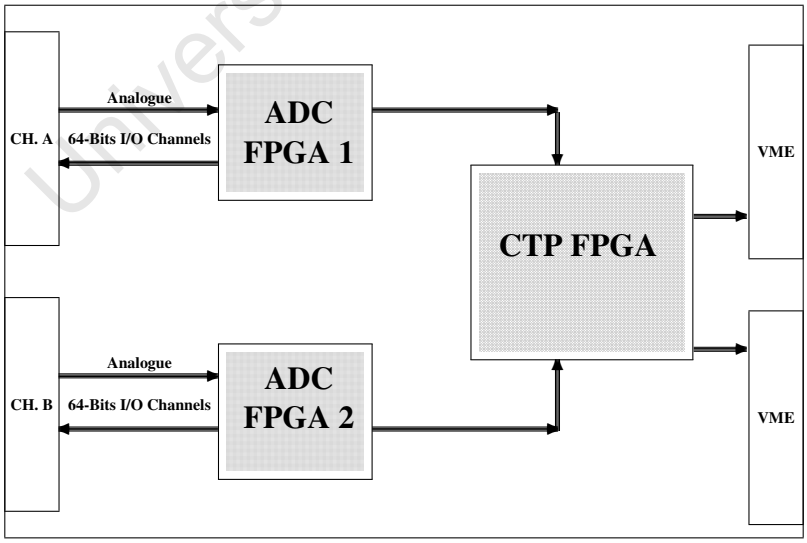


Figure 6.1: Recommended trigger board with upto 128 analogue input channels

As depicted in Figure 6.1, the recommended architectural overview will be similar to the ones presented by (Tekmicro, 2010) [49] and (T. Parnaby, 2009) [50]. This architecture will be com-

posed of two sampling ADC FPGAs, FPGA 1 and 2. Investigations on whether the pre-amplifier signals can directly be inputs to these FPGAs, and be converted into the digital format, shall be made. Also the maximum achievable I/O channel speed, specifying the I/O delays and type of connectors used, should be determined. The number of sampling ADCs implemented using buffers and registers, which when not optimised, as discovered in the project discussed in this thesis, can cause FPGA fitting problems, will need to be looked-at thoroughly. The proposed trigger applications board must also preserve similar features used in setting up triggering systems using the triggering platform presented in this dissertation. With that additional research done, and ToC control GUI that performs real-time trigger monitoring of signals, Physicists and Engineers design cycles will highly be reduced, and their productivity and focus on their research will increase. Therefore giving birth to a new triggering systems design methodology that is even beyond complement to the true-DSP[22] synthesis methodology.

BIBLIOGRAPHY

[1] CAEN Electronic Instrumentation, *Tools for Discovery, Digital Pulse Processing Overview*, 2010, pp 2-3.

[2] I.C. Brock and T. Schrner-Sadenius, *Physics at the Terascale*, WILEY-VCH, Boschstr. 12, 69469 Weinheim, Germany, 2011, pp 363 - 395.

[3] R. Grzywacz, *Applications of digital pulse processing in nuclear spectroscopy*, Nuclear Instruments and Methods in Physics Research B 204, Elsevier, 2003, pp. 649-659.

[4] J. M. Cardoso, J. B. Simes, C. M. B. A. Correia, A. Combo, R. Pereira, J. Sousa, N. Cruz, P. Carvalho, C. A. F. Varandas, *A High Performance Reconfigurable Hardware Platform for Digital Pulse Processing*, IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 51, NO. 3, JUNE 2004, pp. 921.

[5] C. Pohl, C. Paiz, M. Porrmann, *vMAGIC - Automatic Code Generation for VHDL*, International Journal of Reconfigurable Computing, VOL. 2009, Hindawi Publishing Corporation, 2009.

[6] S. Sharma, W. Chen, *Using Model-Based Design to Accelerate FPGA Development for Automotive Applications*, SAE World Congress, 2009.

[7] M. Faisal Nadeem, Mahmood Ahmadi, M. Nadeem and Stephan Wong. *Modeling and Simulation of Reconfigurable Processors in Grid Networks*. 2010 International Conference on Reconfigurable Computing. pp. 226.

[8] S.L. Campbell, J-P Chancelier and R. Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer Science and Business Media, Inc. pp. 194 - 197, 2006.

[9] J.S.J. Wong, P. Sedcole, and P.Y.K. Cheung, *Self-measurement of combinational circuit delays in FPGA*, ACM Transaction on Reconfigurable Technology and Systems, Vol. 2, no. 2, pp 1-22, 2009.

[10] Altera, *AN 311: Standard Cell ASIC to FPGA Design Methodology and Guidelines*, AN-311-3.1, April 2009.

[11] X. Dong, G. Demieux, *PGR: Period and glitch reduction via Clock skew scheduling, delay padding and glitches*, IEEE PPT, 2009.

[12] A. Fallu-Labruyere, H. Tan, W. Hennig and W.K Warburton, *Time Resolution studies using Digital Constant Fraction Discimination*, Elsier Science - XIA LLC 310557 Genstar Rd, 2001.

[13] AFRODITE Side, Nuclear Structure of AFRODITE, *http://www.tlabs.ac.za/nucafrodite.htm*, Last Accessed 22 July 2011.

[14] ORTEC's HTTP Site, *http://www.ortec-online.com*, Last Accessed 18 December 2011.

[15] Para Electronis - Nuclear Products HTTP Site, *http://www.paraelectronics.com/NIMModules.html*, Last Accessed 18 December 2010.

[16] CAEN V1495 Module HTTP site, *http://www.caen.it/jsp/Template2/CaenProd.jsp?idmod=484*, Last Accessed: 03 January 2011.

[17] Altera HTTP site, *http://quartushelp.altera.com/9.1/mergedProjects/hdl/mega/mega_file_scfifo.htm*, Last Accessed: 26 June 2012.

[18] R. Ruegg, *Methods for R&D programs: A Directory of Evaluation Methods Relevant to Technology Development Programs*, U.S. Department of energy - Energy efficiency and renewable energy, pp 24-31, 2007.

[19] B.W. Boehm, *Methods for R&D programs: A Directory of Evaluation Methods Relevant to Technology Development Programs*, T.R.W Defense Systems Group, pp 64, IEEE, 1988.

[20] SoftWeb Solutions HTTP Site, *http://www.softwebsolutions.com/our-processes.html*, Last Accessed: 22 July 2012.

[21] Altera Corporation, *Altera Quartus II Handbook Chapter 18: Anayzing designs with Quartus II Netlist Viewers*, Quartus II design flow with Netlist viewers, v. 12.0, vol. 1, 2012.

[22] Synplify HTTP site, true DSP synthesis, *www.synopsys.co.jp/literature/whitepapers/pdf/*, Last Accessed: 20 December 2010.

[23] S. Sharma, W. Chen, *Using Model-Based Design to Accelerate FPGA Development for Automotive Applications*, SAE World Congress, 2009.

[24] Scilab HTTP Site, *http://www.scilab.org/products/scilab*,Last Accessed 08 March 2011.

[25] Ptolemy II HTTP Site, *http://ptolemy.berkeley.edu/ptolemyII/*, Last Accessed 08 March 2011.

[26] Scicos-HDL HTTP Site, *http://scicoshdl.sourceforge.net/*, Last Accessed 19 December 2010.

[27] T. Filiba, M. Leung, V. Nagpal, *VHDL Code Generation in the Ptolemy II Environment*, University of California - Berkely (UCBL), pg 1, 2006.

[28] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Massachusetts: Addison-Wesley, 1992.

[29] A. Fallu-Labruyere, H. Tan, W. Hennig and W.K Warburton, *Time Resolution studies using Digital Constant Fraction Discimination*, XIA, LLC, 310557 Genstar Rd, Hayward CA 94544, USA. 2001 Elsier Science.

[30] M. Beuzekom, *Identifying fast hadrons with silicon detectors*, University of Groningen Faculty of Mathematics and Natural Sciences Dissertation, Appendix A, 2006.

[31] XIA, LLC, *Setup Guide for iThemba Labs Digital Data Acquisition System (DDAS)*, pg 12, 2009.

[32] J. Davis, C. Hylands, J. Janneck, E.A. Lee, J. Liu, X. Liu, S. Neuendorffer, S. Sachs, M. Stewart, K. Vissers, P. Whitaker, Y. Xiong, *Overview of the Ptolemy Project*, Department of Electrical Engineering and Computer Science, University of California - Berkely, California 94720, 2001.

[33] S.S Ntshangase, *Development of a Recoil detector and the study of exotic asymmetric shapes in nuclie*, Thesis presented in fulfilment of the requirement for the degree of Doctor of Philosophy at the University of cape Town, pg 48-52, 2011.

[34] CAEN, *Digital Pulse Processing in Nuclear Physics*, Electronic Instrumentation, WP2018, Rev. 3, pg 8-18, 2011.

[35] Wiener Manual, *NIMBox/NEMBox User's Manual*, pg 3-22, 2010.

[36] J. M. Cardoso, J.B. Simes, C.M.B.A. Correia, A. Combo, R. Pereira, J. Sousa, N. Cruz, P. Carvalho, and C. A. F. Varandas, *A High Performance Reconfigurable Hardware Platform for Digital Pulse Processing*, IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 51, NO. 3, IEEE, pg 921-925, 2004.

[37] XIA, LLC, *Digital Gamma Finder - PIXIE-16 Manual*, v1.4, pg 26, 2009.

[38] CAEN, V1495, *Technical Information Manual - MOD. V1495 General Purpose VME Board*, Revision 10, pg 26, 2010.

[39] V. Lindenstruth, I. Kisel, *Overview of trigger systems*, ELSEVIER Nuclear Instruments and Methods in Physics Research A 535 (2004) 4856, pg 48-56, 2004.

[40] R. Woods, J. McAllister, G. Lightbody, Y. Yi, *FPGA-based implementation of signal processing systems*, A John Wiley and Sons, Ltd. Publication, pg 20-24, 2008.

[41] W. Parsons, *Region 1 Detector Jefferson Lab Qweak Experiment*, Master of Science in the Department of Physics, Idaho State University, pg 11, 2010.

[42] S. Murray, *An investigation into the use of Photovoltaic Cells as fission fragment detectors for AFRODITE.*, Master of Science in the Department of Physics, University of Cape Town, pg 131-143, 2010.

[43] Canberra, *Preamplifier Introduction*, Canberra Industries, Inc., USA, pg 1, 2010.

[44] Many-Body Nuclear Dynamics Group Manuals for Electronics Modules Website, *http://nuchem.iucf.indiana.edu/Instrumentation/Elec_Manual/Elec_Manual.htm*, Center for the Exploration of Energy and Matter and Department of Chemistry, Indiana University, Bloomington, Last Accessed: 22 August 2012.

[45] CAEN Website, *http://www.caen.it/csite/Product.jsp?Type=Product&parent=12*, NIM Electronics Products, Last Accessed: 22 August 2012.

[46] M.S Moore, M. Brooks, G. Willden, *Keys to Success in Model-based design*, Workshop by DARPA IXO ARFL Rome, New York, pg 23, 2004.

[47] MIPS Technologies Inc., *Debugging Xilinx FPGA devices with Tektronix logic analysers and mixed signal scopes* , MIPS Technologies, Inc., USA, pg 1, 2008.

[48] Altera, *In-System Debugging Using External Logic Analysers*, Quartus II Handbook, version 12, pg 3, 2012.

[49] Tekmicro, *Titan-V5 VXS, Maximum FPGA Density, Combined with High Performance, Mixed Signal Technology, Without Compromise.*, TEK Microsystems, Titon-V5, pg 1, 2010.

[50] T. Barnaby, *FPGAs Help Measure Trajectory of Particles in CERNs Proton Synchrotron*, Xcell Journal, pg 23, 2009.

[51] T.D. Hendrix, M.P Schneider, *NASAs TReK Project: A Case Study in Using the Spiral Model of Software Development*, COMMUNICATIONS OF THE ACM, ACM, Vol. 45, No. 4ve, pg 154, 2002.

[52] C. Larman, V.R. Basili, *NASA's TReK Project: A Case Study in Using the Spiral Model of Software Development*, Valtech & University of Maryland, IEEE Computer Society, IEEE, pg 47-56, 2003.

[53] P.K. Ragunath, S. Velmourougan, P. Davachelvan,S. Kayalvizhi, R. Ravimohan, *Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC)*, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1, pg 115, 2010.

[54] N.M.A. Munassar and A. Govardhan, *A Comparison Between Five Models Of Software Engineering*, IJCSI International Journal of Computer Science Issues, VOL.7, Issue 5, pg 99, 2010.

[55] M. Lunardon, C. Bottosso, D. Fabris, S. Moretto, G. Nebbia, S. Pesente, G. Viesti, A. Bigongiari, A. Colonna, C. Tintori, V. Valkovic, D. Sudac, P. Peerani, V. Sequeira, M. Salvato, *Front-end electronics and DAQ for the EURITRACK tagged neutron inspection system*, ELSEVIER, Science Direct, Nuclear Instruments and Methods in Physics Research B 261, pg 391-395, 2007.

[56] M. Bhuyan, V.B. Chandratre, S. Dasgupta, V.M. Datar, S.D. Kalmani, S.M. Lahamge, N.K. Mondal, P. Nagaraj, S. Pal, S.K. Rao, A. Redij, D. Samuel, M.N. Saraf, B. Satyanarayana, R.R. Shinde, S.S. Upadhya, *VME-based data acquisition system for the India-based Neutrino Observatory prototype detector*, ELSEVIER, Science Direct, Nuclear Instruments and Methods in Physics Research A 661, pg S72-S76, 2010.

[57] E. Spiritia, M. De Napoli and F. Romano, *The FIRST experiment: interaction region and MAPS vertex detector*, ELSEVIER, Science Direct, Nuclear Physics (Proc. Suppl.) B 215, pg 157-161, 2011.

[58] R. Beyer, D. Bemmerer, E. Grosse, R. Hannaske, A.R. Junghans, M. Kempe, T. Kogler, R. Massarczyk, R. Nolte, R. Schwengnera, A. Wagnera, *Fast neutron inelastic scattering at the nELBE facility*, IOP PUBLISHING FOR SISSA, INTERNATIONAL WORKSHOP ON FAST NEUTRONS DETECTOR AND APPLICATIONS, pg 5, 2011.

[59] P.R. Wilson, *Design Recipes for FPGAs*, Newnes, ELSEVIER, USA, pg 225-282, 2007.

[60] N. Storey, *Eletrical and Electronic Systems - Chapter 27: Implementing digital systems*, Pearson Prentice Hall, England, pg 5, 2004.

[61] V.A. Pedroni, *Circuit Design With VHDL*, MIT Press, MIT, USA, pg 1-355, 2004.

[62] C. Maxfield, *The Design warrior's guide to FPGAs*, Newnes, ELSEVIER, USA, pg 101-300-408, 2004.

[63] Altera, *Quartus II Handbook Version 12.0 Volume 2: Design Implementation and Optimization - Command-Line Scripting*, Altera Corperation, pg 2-2, 2012.

[64] D.M. Binkerly, *Performance analysis of Non-Delay-Line Constant Fraction Discriminator Timing Circuits*, IEEE Transactions on Nuclear Science, VOL. 41, NO. 4, USA, pg 1160-1175, 1994.

[65] W-S. Choong, *The timing resolution of scintillation-detector systems: Monte Carlo analysis*, IOP Publishing, Physics in Medicine and Biology, VOL. 54, UK, pg 6495-6513, 2009.

[66] O. Barnaba, Y.B. Chen, G. Musitelli, R. Nardo, G.L. Raselli, M. Rossella, P. Torre, *A full-integrated pulse-shape discriminator for liquid scintillator counters*, ELSEVIER, Nuclear Instruments and Methods in Physics Research A 410, USA, pg 220-228, 1998.

[67] CANBERRA, *Model 2128 200 MHz Constant Fraction Discriminator*, CANBERRA - THE NUCLEAR MEASUREMENTS BUSINESS UNIT OF AREVA, USA, 2007.

APPENDIX A

TRIGGER MODELS WITH MULTIPLE NIM MODELS

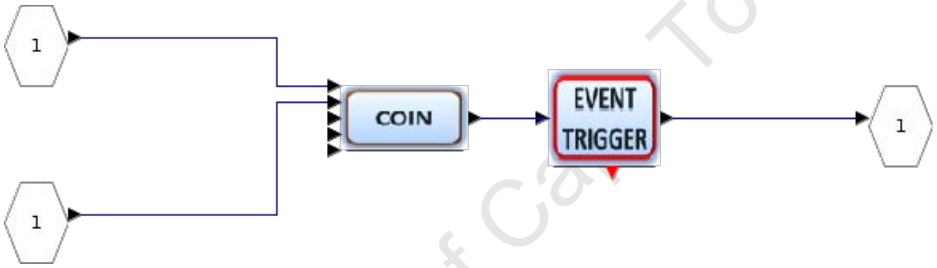


Figure A.1: 2 NIM Models in a Trigger Logic

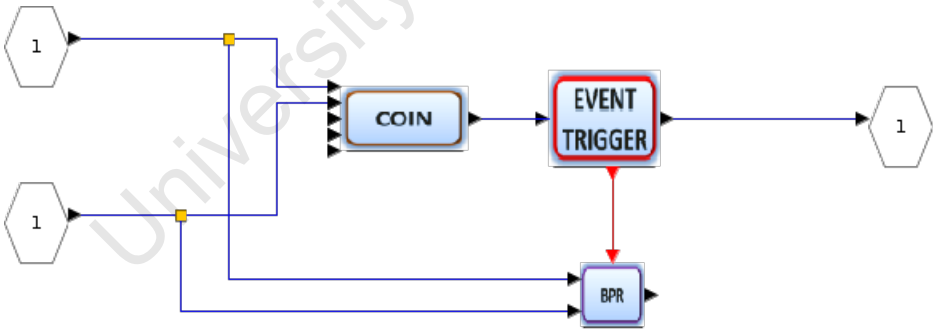


Figure A.2: 3 NIM Models in a Trigger Logic

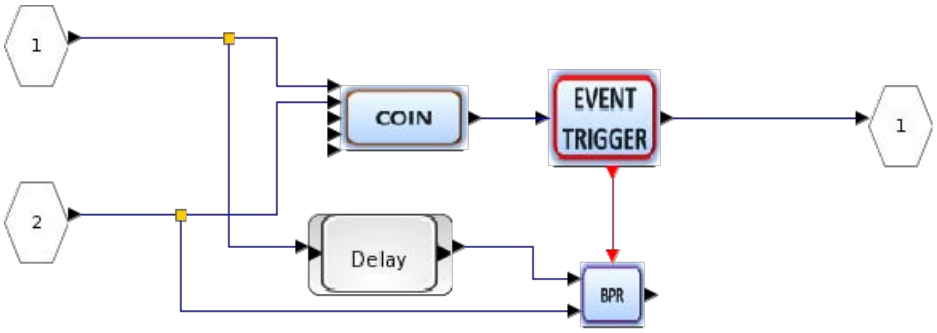


Figure A.3: 4 NIM Models in a Trigger Logic

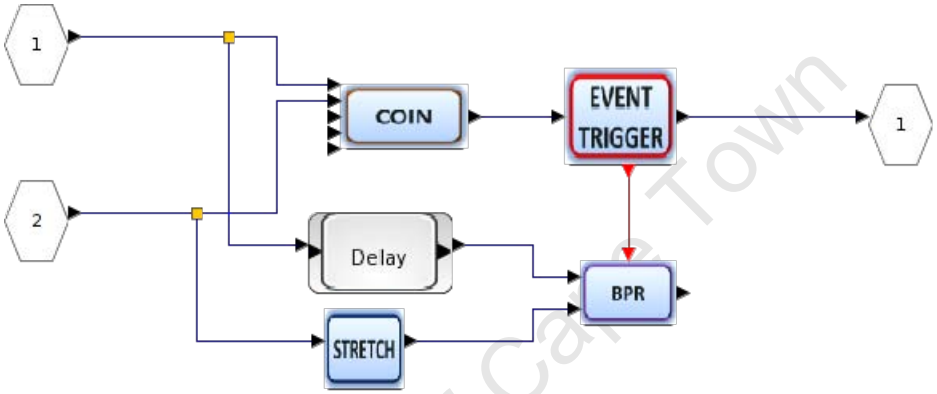


Figure A.4: 5 NIM Models in a Trigger Logic

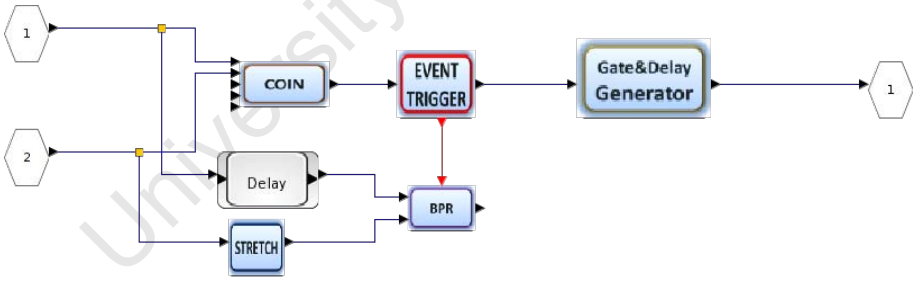


Figure A.5: 6 NIM Models in a Trigger Logic

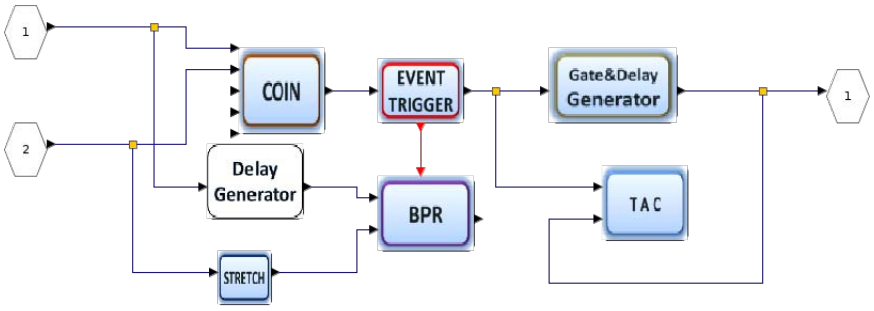
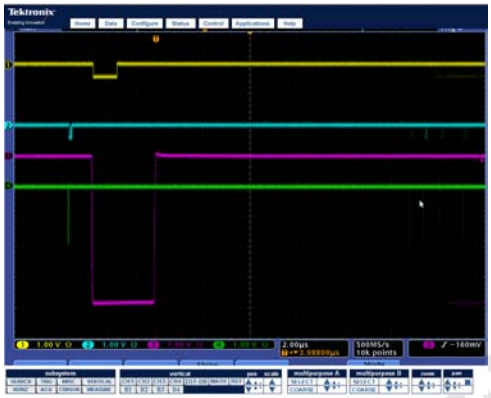


Figure A.6: 7 NIM Models in a Trigger Logic

APPENDIX B

NIM MODELS AND THEIR TIMING DIAGRAMS



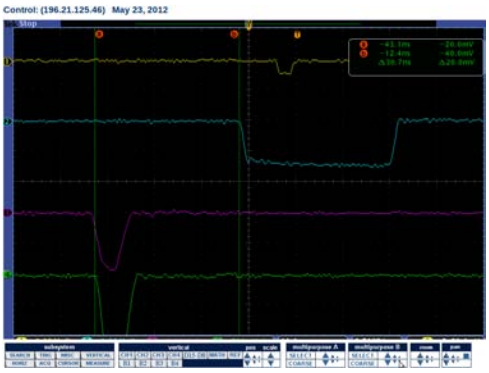
(a) Gate and delay generator timing diagram

(b) analog Gate and delay generator delay measurement

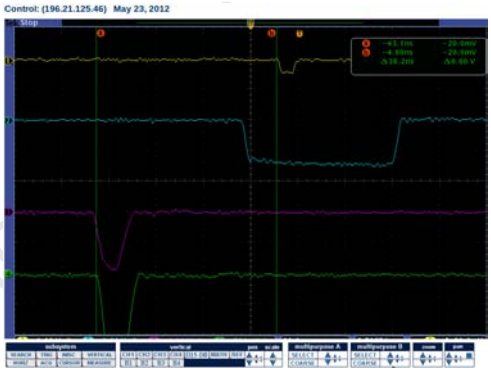
(c) digital Gate and delay generator delay measurement

(d) Gate and delay generator model

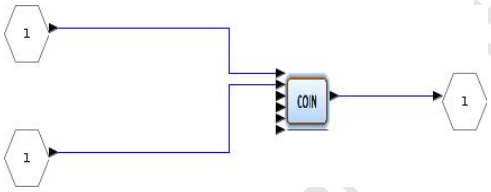
Figure B.1: Gate and delay generator model with its timing diagram



(a) analogue Coincidence model timing diagram



(b) Digital Coincidence model timing diagram



(c) Coincidence model

Figure B.2: Coincidence model with its timing diagrams

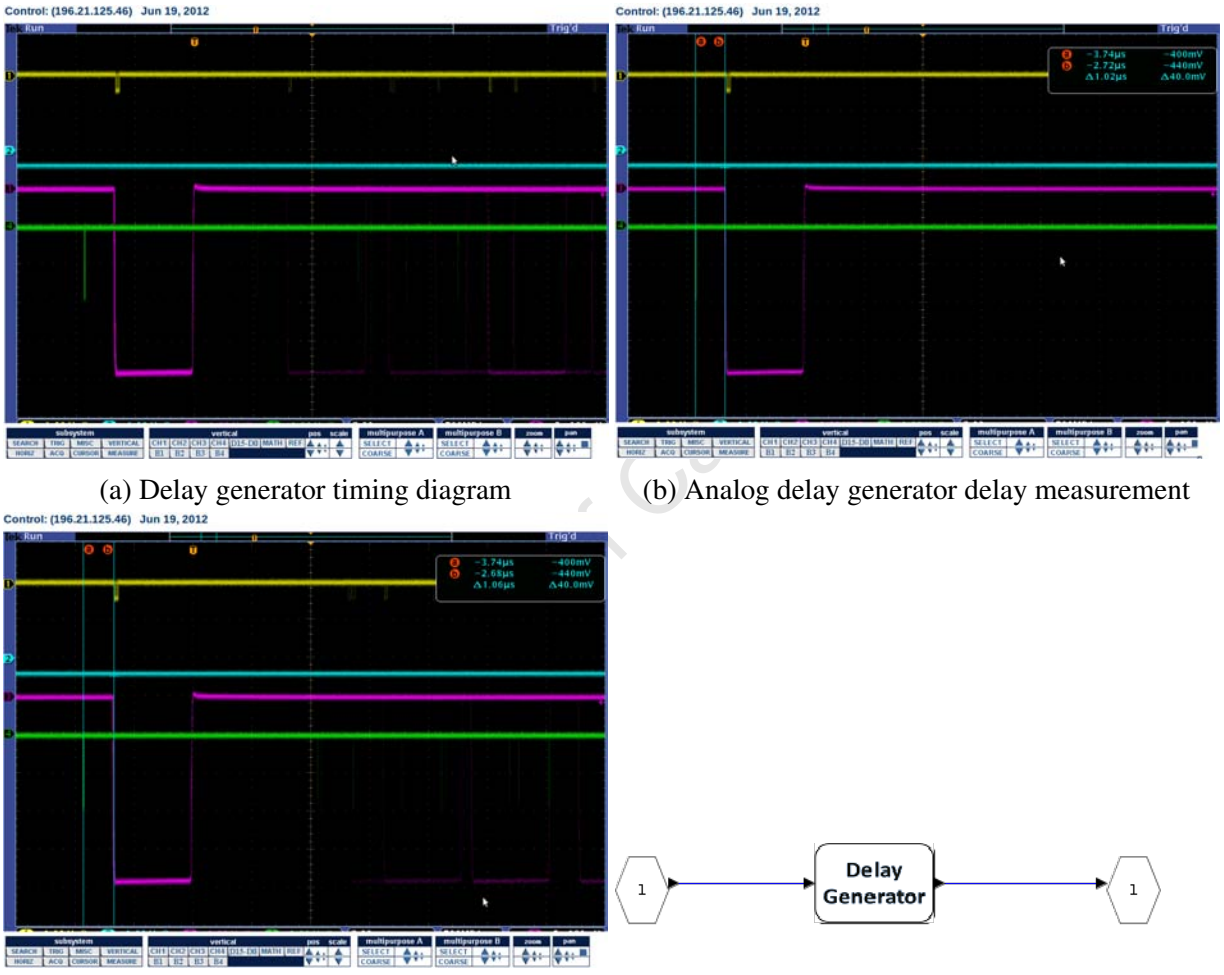
(a) Pulse stretcher timing diagram

(b) Analogue pulse stretcher delay measurement

(c) Digital pulse stretcher delay measurement

(d) Pulse stretcher model

Figure B.3: Pulse stretcher model with its timing diagram



(a) Delay generator timing diagram

(b) Analog delay generator delay measurement

(c) Digital delay generator delay measurement

(d) Delay generator model

Figure B.4: Gate and delay generator model with its timing diagram

APPENDIX C

NIM MODELS AND THEIR AUTOMATICALY GENERATED VHDL CODES

Figure C.1: Sample Signal Delayer Model

Listing C.1: Delay Model Generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library
library IEEE;
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use work.GenSATPack.all;

entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
        nOED : out std_logic;
        nOEE : out std_logic;
```

```
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;

architecture rtl of GenSATMain is
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal DSignal_I_7fd7 : std_logic_vector(0 downto 0);
    signal DSignal_O_7fd7 : std_logic_vector(0 downto 0);
    signal Dusedw_7fd7 : std_logic_vector(11 downto 0);
    signal DELAY_OUT_7fd7 : std_logic;
    signal DELAY_7fd7 : INTEGER;
    signal Drdreq_7fd7 : std_logic;
    signal Dwrreq_7fd7 : std_logic;
    signal Dgbpr_clr_7fd7 : std_logic;
    signal DisEmpty_7fd7 : std_logic;
    signal DisFull_7fd7 : std_logic;
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal RELOAD_REG_RW2 : std_logic := '0';
    component GenSATLocalRegisterAccess
        port (
            nLBRES : in std_logic;
            nBLAST : in std_logic;
```

```
            WnR : in std_logic;
            nADS : in std_logic;
            LCLK : in std_logic;
            nREADY : out std_logic;
            nINT : out std_logic;
            LAD : inout std_logic_vector(15 downto 0);
            REG_R1 : in std_logic_vector(31 downto 0);
            REG_R2 : in std_logic_vector(31 downto 0);
            REG_RW1 : buffer std_logic_vector(31 downto 0);
            REG_RW2 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW2 : out std_logic
        );
    end component GenSATLocalRegisterAccess;
begin
    SELG <= '0';
    SELE <= '0';
    nOEG <= '1';
    nOEE <= '0';
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    E(0) <= DSignal_O_7fd7(0);
    DSignal_I_7fd7(0) <= NOT(GIN(0));
    DELAY_7fd7 <= conv_integer(unsigned(REG_RW2(8 downto 0)));
    Dwrreq_7fd7 <= '1';
    Dgbpr_clr_7fd7 <= RELOAD_REG_RW2;
    GenSATDelayer_7fd7 : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_7fd7,
CNTRELOAD   => RELOAD_REG_RW2,
DELAY_OUT   => Drdreq_7fd7
        );
    GenSATDelayerFIFO_7fd7 : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_7fd7,
clock   => PCLK,
data    => DSignal_I_7fd7,
rdreq   => Drdreq_7fd7,
wrreq   => Dwrreq_7fd7,
empty   => DisEmpty_7fd7,
full    => DisFull_7fd7,
q   => DSignal_O_7fd7,
usedw   => Dusedw_7fd7
        );
    HOLD <= REG_RW1(0);
    GenSATMainLRA : GenSATLocalRegisterAccess
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
```

```
LCLK         => LCLK,
nREADY       => nREADY,
nINT         => nINT,
LAD          => LAD,
REG_R1    => REG_R1,
REG_R2    => REG_R2,
REG_RW1    => REG_RW1,
REG_RW2    => REG_RW2,
RELOAD_REG_RW2    => RELOAD_REG_RW2
        );
end;
```

Figure C.2: Gate and Delay Generator Model

Listing C.2: Gate and Delay Generator Model Generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
        nOED : out std_logic;
        nOEE : out std_logic;
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
```

```
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal DSignal_I_7fde : std_logic_vector(0 downto 0);
    signal DSignal_O_7fde : std_logic_vector(0 downto 0);
    signal Dusedw_7fde : std_logic_vector(11 downto 0);
    signal DELAY_OUT_7fde : std_logic;
    signal DELAY_7fde : INTEGER;
    signal Drdreq_7fde : std_logic;
    signal Dwrreq_7fde : std_logic;
    signal Dgbpr_clr_7fde : std_logic;
    signal DisEmpty_7fde : std_logic;
    signal DisFull_7fde : std_logic;
    signal HOLD_7fde : std_logic := '0';
    signal Signal_O_7fde : std_logic := '0';
    signal STRETCH_LEN_7fde : INTEGER := 0;
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal RELOAD_REG_RW2 : std_logic := '0';
    component GenSATLocalRegisterAccess
        port (
            nLBRES : in std_logic;
            nBLAST : in std_logic;
            WnR : in std_logic;
            nADS : in std_logic;
            LCLK : in std_logic;
            nREADY : out std_logic;
            nINT : out std_logic;
```

```
            LAD : inout std_logic_vector(15 downto 0);
            REG_R1 : in std_logic_vector(31 downto 0);
            REG_R2 : in std_logic_vector(31 downto 0);
            REG_RW1 : buffer std_logic_vector(31 downto 0);
            REG_RW2 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW2 : out std_logic
        );
    end component GenSATLocalRegisterAccess;
begin
    SELD <= '0';
    SELE <= '0';
    SELF <= '0';
    SELG <= '0';
    nOED <= '1';
    nOEE <= '0';
    nOEF <= '0';
    nOEG <= '0';
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    DSignal_I_7fde(0) <= NOT(D(2));
    DELAY_7fde <= conv_integer(unsigned(REG_RW2(8 downto 0)));
    Dwrreq_7fde <= '1';
    Dgbpr_clr_7fde <= RELOAD_REG_RW2;
    STRETCH_LEN_7fde <= conv_integer(unsigned(REG_RW2(8 downto 0)));
    E(0) <= Signal_O_7fde;
    GenSATDelayer_7fde : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_7fde,
CNTRELOAD   => RELOAD_REG_RW2,
DELAY_OUT   => Drdreq_7fde
        );
    GenSATDelayerFIFO_7fde : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_7fde,
clock   => PCLK,
data    => DSignal_I_7fde,
rdreq   => Drdreq_7fde,
wrreq   => Dwrreq_7fde,
empty   => DisEmpty_7fde,
full    => DisFull_7fde,
q   => DSignal_O_7fde,
usedw   => Dusedw_7fde
        );
    Stretcher_7fde : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_7fde,
PULSE_IN  => DSignal_O_7fde(0),
PULSE_OUT => Signal_O_7fde
```

```
        );
    HOLD <= REG_RW1(0);
    GenSATMainLRA : GenSATLocalRegisterAccess
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
REG_R1   => REG_R1,
REG_R2   => REG_R2,
REG_RW1   => REG_RW1,
REG_RW2   => REG_RW2,
RELOAD_REG_RW2   => RELOAD_REG_RW2
        );
end;
```

Figure C.3: Pulse Stretcher Model

Listing C.3: Pulse Stretcher Model Generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
```

```
        nOED : out std_logic;
        nOEE : out std_logic;
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal HOLD_7f7a : std_logic := '0';
    signal Signal_I_7f7a : std_logic := '0';
    signal Signal_O_7f7a : std_logic := '0';
    signal STRETCH_LEN_7f7a : INTEGER := 0;
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal RELOAD_REG_RW2 : std_logic := '0';
    component GenSATLocalRegisterAccess
        port (
            nLBRES : in std_logic;
            nBLAST : in std_logic;
            WnR : in std_logic;
            nADS : in std_logic;
            LCLK : in std_logic;
            nREADY : out std_logic;
            nINT : out std_logic;
```

```
            LAD : inout std_logic_vector(15 downto 0);
            REG_R1 : in std_logic_vector(31 downto 0);
            REG_R2 : in std_logic_vector(31 downto 0);
            REG_RW1 : buffer std_logic_vector(31 downto 0);
            REG_RW2 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW2 : out std_logic
        );
    end component GenSATLocalRegisterAccess;
begin
    SELG <= '0';
    SELE <= '0';
    nOEG <= '1';
    nOEE <= '0';
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    E(0) <= Signal_O_7f7a;
    Signal_I_7f7a <= NOT(GIN(0));
    STRETCH_LEN_7f7a <= conv_integer(unsigned(REG_RW2(8 downto 0)));
    Stretcher_7f7a : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_7f7a,
PULSE_IN  => Signal_I_7f7a,
PULSE_OUT => Signal_O_7f7a
        );
    HOLD <= REG_RW1(0);
    GenSATMainLRA : GenSATLocalRegisterAccess
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
REG_R1   => REG_R1,
REG_R2   => REG_R2,
REG_RW1   => REG_RW1,
REG_RW2   => REG_RW2,
RELOAD_REG_RW2   => RELOAD_REG_RW2
        );
end;
```


Listing C.4: Coincidence Model Generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library
```

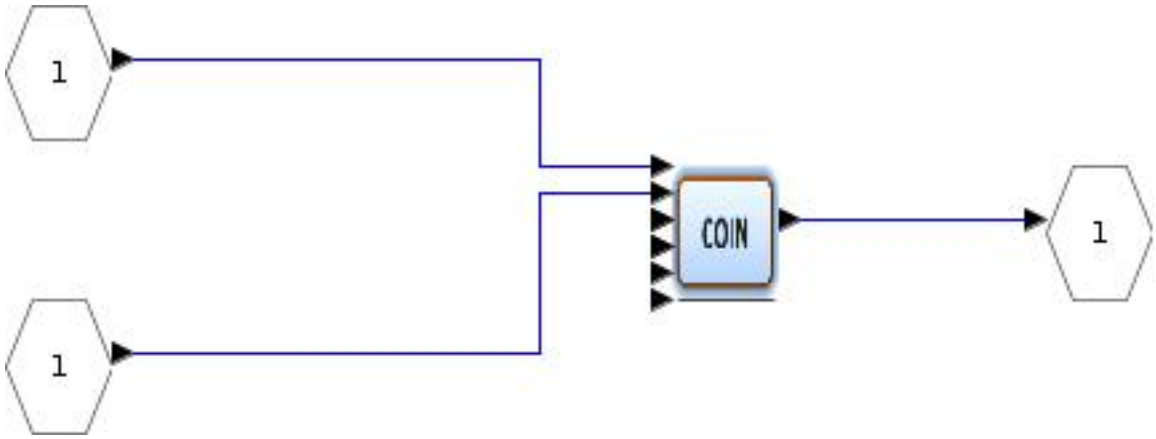


Figure C.4: Coincidence Model


```
library IEEE;
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use IEEE.numeric_std.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
        nOED : out std_logic;
        nOEE : out std_logic;
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
```

```
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal BIT_1_7fc7 : std_logic;
    signal BIT_2_7fc7 : std_logic;
    signal BIT_3_7fc7 : std_logic;
    signal BIT_4_7fc7 : std_logic;
    signal BIT4_OUT_7fc7 : std_logic := '0';
    signal REG_RW_7fc7 : std_logic_vector(3 downto 0);
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    component GenSATLocalRegisterAccess
        port (
            nLBRES : in std_logic;
            nBLAST : in std_logic;
            WnR : in std_logic;
            nADS : in std_logic;
            LCLK : in std_logic;
            nREADY : out std_logic;
            nINT : out std_logic;
            LAD : inout std_logic_vector(15 downto 0);
            REG_R1 : in std_logic_vector(31 downto 0);
            REG_R2 : in std_logic_vector(31 downto 0);
            REG_RW1 : buffer std_logic_vector(31 downto 0)
        );
    end component GenSATLocalRegisterAccess;
begin
    SELD <= '0';
    SELE <= '0';
    nOED <= '1';
    nOEE <= '0';
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    BIT_1_7fc7 <= NOT(D(2));
    REG_RW_7fc7(0) <= '1';
    BIT_2_7fc7 <= NOT(D(18));
    REG_RW_7fc7(1) <= '1';
    BIT_3_7fc7 <= '0';
    BIT_4_7fc7 <= '0';
```

```
    E(0) <= BIT4_OUT_7fc7;
    GenSAT4BitCoin_7fc7 : GenSAT4BitCoin
        port map (
            BIT_1   => BIT_1_7fc7,
BIT_2   => BIT_2_7fc7,
BIT_3   => BIT_3_7fc7,
BIT_4   => BIT_4_7fc7,
SW_1    => REG_RW_7fc7(0),
SW_2    => REG_RW_7fc7(1),
SW_3    => REG_RW_7fc7(2),
SW_4    => REG_RW_7fc7(3),
BIT_OUT => BIT4_OUT_7fc7
        );
    HOLD <= REG_RW1(0);
    GenSATMainLRA : GenSATLocalRegisterAccess
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
REG_R1   => REG_R1,
REG_R2   => REG_R2,
REG_RW1   => REG_RW1
        );
end;
```

Listing C.5: The Full Coincidence module VHDL code showing how a switch and inputs are combined to make a coincidence checker

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY GenSATMultBitCoin IS PORT
(
    BITS    : IN STD_LOGIC_VECTOR(63 downto 0);
    SW      : IN STD_LOGIC_VECTOR(63 downto 0);
    BIT_OUT :OUT  STD_LOGIC
  );
END GenSATMultBitCoin;

ARCHITECTURE rtl OF GenSATMultBitCoin IS
BEGIN
 -- The output BIT_OUT will give a pulse for each coincidence of defined
    simultaneous inputs
 -- Variable SW represent 64-bits vector of switches set dynamically
 -- by model-to-VHDL interpreter. If one of these switches is set to
    active
 -- high then the signal from the corresponding input port will not be
    ignored.
 -- then a coincidence is expected at that point with another port with
```

```
    its
 -- switch set to active high.

  BIT_OUT <= (BITS(0) or NOT(SW(0))) and (BITS(1) or NOT(SW(1))) and (
     BITS(2) or NOT(SW(2))) and (BITS(3) or NOT(SW(3))) and (BITS(4) or
     NOT(SW(4))) and (BITS(5) or NOT(SW(5))) and (BITS(6) or NOT(SW(6)))
      and (BITS(7) or NOT(SW(7))) and
  (BITS(8) or NOT(SW(8))) and (BITS(9) or NOT(SW(9))) and (BITS(10) or
     NOT(SW(10))) and (BITS(11) or NOT(SW(11))) and (BITS(12) or NOT(SW
     (12))) and (BITS(13) or NOT(SW(13))) and (BITS(14) or NOT(SW(14)))
     and (BITS(15) or NOT(SW(15))) and
  (BITS(16) or NOT(SW(16))) and (BITS(17) or NOT(SW(17))) and (BITS(18)
     or NOT(SW(18))) and (BITS(19) or NOT(SW(19))) and (BITS(20) or NOT(
     SW(20))) and (BITS(21) or NOT(SW(21))) and (BITS(22) or NOT(SW(22))
     ) and (BITS(23) or NOT(SW(23))) and
  (BITS(24) or NOT(SW(24))) and (BITS(25) or NOT(SW(25))) and (BITS(26)
     or NOT(SW(26))) and (BITS(27) or NOT(SW(27))) and (BITS(28) or NOT(
     SW(28))) and (BITS(29) or NOT(SW(29))) and (BITS(30) or NOT(SW(30))
     ) and (BITS(31) or NOT(SW(31))) and
  (BITS(32) or NOT(SW(32))) and (BITS(33) or NOT(SW(33))) and (BITS(34)
     or NOT(SW(34))) and (BITS(35) or NOT(SW(35))) and (BITS(36) or NOT(
     SW(36))) and (BITS(37) or NOT(SW(37))) and (BITS(38) or NOT(SW(38))
     ) and (BITS(39) or NOT(SW(39))) and
  (BITS(40) or NOT(SW(40))) and (BITS(41) or NOT(SW(41))) and (BITS(42)
     or NOT(SW(42))) and (BITS(43) or NOT(SW(43))) and (BITS(44) or NOT(
     SW(44))) and (BITS(45) or NOT(SW(45))) and (BITS(46) or NOT(SW(46))
     ) and (BITS(47) or NOT(SW(47))) and
  (BITS(48) or NOT(SW(48))) and (BITS(49) or NOT(SW(49))) and (BITS(50)
     or NOT(SW(50))) and (BITS(51) or NOT(SW(51))) and (BITS(52) or NOT(
     SW(52))) and (BITS(53) or NOT(SW(53))) and (BITS(54) or NOT(SW(54))
     ) and (BITS(55) or NOT(SW(55))) and
  (BITS(56) or NOT(SW(56))) and (BITS(57) or NOT(SW(57))) and (BITS(58)
     or NOT(SW(58))) and (BITS(59) or NOT(SW(59))) and (BITS(60) or NOT(
     SW(60))) and (BITS(61) or NOT(SW(61))) and (BITS(62) or NOT(SW(62))
     ) and (BITS(63) or NOT(SW(63)));
END rtl;
```

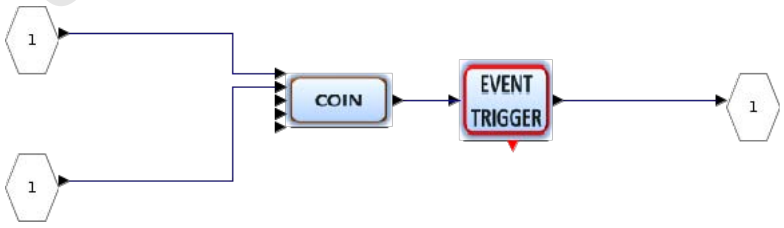


Figure C.5: Event trigger Model

Listing C.6: Event trigger Model Generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
```

```
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use IEEE.numeric_std.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
        nOED : out std_logic;
        nOEE : out std_logic;
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R3 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R4 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R5 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R6 : std_logic_vector(31 downto 0) := (others=>'Z');
```

```
    signal REG_R7 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R8 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R9 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R10 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R11 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R12 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW3 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW4 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW5 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW6 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW7 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW8 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW9 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW10 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW11 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW12 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW13 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW14 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal SPLITBLK_7f4a : std_logic;
    signal Data_7ecb : std_logic;
    signal EventTrig_7ecb : std_logic;
    signal Data_7f51 : std_logic_vector(31 downto 0);
    signal Strobe_7f51 : std_logic;
    signal REG_OUT_7f51 : std_logic_vector(31 downto 0);
begin
    SELD <= '0';
    SELE <= '0';
    nOED <= '1';
    nOEE <= '0';
    HOLD <= REG_RW1(0);
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    E(0) <= EventTrig_7ecb;
    SPLITBLK_7f4a <= NOT(D(2));
    Data_7ecb <= SPLITBLK_7f4a;
    Data_7f51(0) <= SPLITBLK_7f4a;
    Data_7f51(1) <= NOT(D(18));
    Data_7f51(2) <= '0';
    Data_7f51(3) <= '0';
    Data_7f51(4) <= '0';
    Data_7f51(5) <= '0';
    Data_7f51(6) <= '0';
    Data_7f51(7) <= '0';
    Data_7f51(8) <= '0';
    Data_7f51(9) <= '0';
    Data_7f51(10) <= '0';
    Data_7f51(11) <= '0';
```

```
    Data_7f51(12) <= '0';
    Data_7f51(13) <= '0';
    Data_7f51(14) <= '0';
    Data_7f51(15) <= '0';
    Data_7f51(16) <= '0';
    Data_7f51(17) <= '0';
    Data_7f51(18) <= '0';
    Data_7f51(19) <= '0';
    Data_7f51(20) <= '0';
    Data_7f51(21) <= '0';
    Data_7f51(22) <= '0';
    Data_7f51(23) <= '0';
    Data_7f51(24) <= '0';
    Data_7f51(25) <= '0';
    Data_7f51(26) <= '0';
    Data_7f51(27) <= '0';
    Data_7f51(28) <= '0';
    Data_7f51(29) <= '0';
    Data_7f51(30) <= '0';
    Data_7f51(31) <= '0';
    Strobe_7f51 <= EventTrig_7ecb;
    REG_RW2 <= REG_OUT_7f51;
    GenSATEventTrigger_7ecb : GenSATEventTrigger
        port map (
            HOLD    => HOLD,
Data  => Data_7ecb,
EventTrig => EventTrig_7ecb
        );
    GenSATBPR_7f51 : GenSATBitPatternRegister
        port map (
            Data         => Data_7f51,
Strobe     => Strobe_7f51,
REG_OUT    => REG_OUT_7f51
        );
    GenSATLocalBus : LB_INT
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
WR_DLY_CMD  => WR_DLY_CMD,
REG_R1     => REG_R1,
REG_R2     => REG_R2,
REG_R3     => REG_R3,
REG_R4     => REG_R4,
REG_R5     => REG_R5,
REG_R6     => REG_R6,
REG_R7     => REG_R7,
REG_R8     => REG_R8,
REG_R9     => REG_R9,
REG_R10     => REG_R10,
```

```
REG_RW1     => REG_RW1,
REG_RW2     => REG_RW2,
REG_RW3     => REG_RW3,
REG_RW4     => REG_RW4,
REG_RW5     => REG_RW5,
REG_RW6     => REG_RW6,
REG_RW7     => REG_RW7,
REG_RW8     => REG_RW8,
REG_RW9     => REG_RW9,
REG_RW10     => REG_RW10,
REG_RW11     => REG_RW11,
REG_RW12     => REG_RW12,
REG_RW13      => REG_RW13,
REG_RW14      => REG_RW14
        );
end;
```

APPENDIX D

TOC DEBUGGED MODELS TOP-LEVEL ENTITY GENERATED VHDL CODE

D.1 FOR TIMING SAFE MODEL IN CHAPTER 5.2

Listing D.1: Timing Safe Model generated VHDL Code

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
--Used Libraries
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);-- 32 ECL/LVDS Inputs
        B : in std_logic_vector(31 downto 0);--32 ECL/LVDS Inputs
        C : out std_logic_vector(31 downto 0);--32 ECL/LVDS Outputs
        D : inout std_logic_vector(31 downto 0);-- Mezzanine slot D
        E : inout std_logic_vector(31 downto 0);-- Mezzanine slot E
        F : inout std_logic_vector(31 downto 0);-- Mezzanine slot G
        GIN : in std_logic_vector(1 downto 0);-- 2 TTL/NIM Inputs
        GOUT : out std_logic_vector(1 downto 0);-- 2 TTL/NIM Outputs
        nOED : out std_logic;--D Slot Ports I/O Enable 1=Input,0=Output
        nOEE : out std_logic;--E Slot Ports I/O Enable 1=Input,0=Output
        nOEF : out std_logic;--F Slot Ports I/O Enable 1=Input,0=Output
        nOEG : out std_logic;--G Slot Ports I/O Enable 1=Input,0=Output
        SELD : out std_logic;--D Slot Ports Level Select 1=TTL,0=NIM
        SELE : out std_logic;--E Slot Ports Level Select 1=TTL,0=NIM
        SELF : out std_logic;--F Slot Ports Level Select 1=TTL,0=NIM
        SELG : out std_logic;--G Slot Ports Level Select 1=TTL,0=NIM
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
```

```
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;

architecture rtl of GenSATMain is
  --***************************************************
  --********* Signal Declarations **********************
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
    signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
    signal Data_7e07 : std_logic_vector(31 downto 0);
    signal Strobe_7e07 : std_logic;
    signal REG_OUT_7e07 : std_logic_vector(31 downto 0);
    signal BITS_7ddf : std_logic_vector(63 downto 0);
    signal COIN_SW_7ddf : std_logic_vector(63 downto 0);
    signal BIT_OUT_7ddf : std_logic := '0';
    signal SPLITBLK_7f62 : std_logic;
    signal SPLITBLK_7dd1 : std_logic;
    signal CTRLREG_7f72 : std_logic_vector(31 downto 0) := (others=>'0');
    signal Data_7f72 : std_logic_vector(7 downto 0);
    signal ScopeOut_7f72 : std_logic_vector(3 downto 0);
    signal SPLITBLK_7f5a : std_logic;
    signal Data_7daf : std_logic;
    signal EventTrig_7daf : std_logic;
    signal SPLITBLK_7f53 : std_logic;
    signal SPLITBLK_7f56 : std_logic;
    signal SPLITBLK_7f4c : std_logic;
    signal HOLD_7ed3 : std_logic := '0';
    signal Signal_I_7ed3 : std_logic := '0';
    signal Signal_O_7ed3 : std_logic := '0';
    signal STRETCH_LEN_7ed3 : INTEGER := 0;
    signal SPLITBLK_7f45 : std_logic;
    signal HOLD_7ecd : std_logic := '0';
    signal Signal_I_7ecd : std_logic := '0';
    signal Signal_O_7ecd : std_logic := '0';
```

```
    signal STRETCH_LEN_7ecd : INTEGER := 0;
    signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_R3 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW3 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal REG_RW4 : std_logic_vector(31 downto 0) := (others=>'Z');
    signal RELOAD_REG_RW2 : std_logic := '0';
    signal RELOAD_REG_RW3 : std_logic := '0';
    signal RELOAD_REG_RW4 : std_logic := '0';

    --*****************************************************
  --********* LRA Component Dynamically Generated ******
    component GenSATLocalRegisterAccess
        port (
            nLBRES : in std_logic;
            nBLAST : in std_logic;
            WnR : in std_logic;
            nADS : in std_logic;
            LCLK : in std_logic;
            nREADY : out std_logic;
            nINT : out std_logic;
            LAD : inout std_logic_vector(15 downto 0);
            --Read Only Registers
            REG_R1 : in std_logic_vector(31 downto 0);
            REG_R2 : in std_logic_vector(31 downto 0);
            REG_R3 : in std_logic_vector(31 downto 0);
            -- Read/Write Registers
            REG_RW1 : buffer std_logic_vector(31 downto 0);
            REG_RW2 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW2 : out std_logic;
            REG_RW3 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW3 : out std_logic;
            REG_RW4 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW4 : out std_logic
        );
    end component GenSATLocalRegisterAccess;
begin
    SELD <= '0';-- Setting Port Level in Slot D to be NIM
    SELE <= '0';-- Setting Port Level in Slot E to be NIM
    SELF <= '0';-- Setting Port Level in Slot F to be NIM
    SELG <= '0';-- Setting Port Level in Slot G to be NIM
    nOED <= '1';-- Input Enabling Ports in Slot D
    nOEE <= '0';-- Output Enabling Ports in Slot E
    nOEF <= '0';-- Output Enabling Ports in Slot F
    nOEG <= '0';-- Output Enabling Ports in Slot G

    --PLL instantiation - Output is 100MHz PCLK
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    --  Dynamic Signal Assignments: Unique ID to determine which signal
```

```
    is for which NIM component/block
Data_7e07(0) <= SPLITBLK_7dd1;
Data_7e07(1) <= SPLITBLK_7f56;
Data_7e07(2) <= '0';
Data_7e07(3) <= '0';
Data_7e07(4) <= '0';
Data_7e07(5) <= '0';
Data_7e07(6) <= '0';
Data_7e07(7) <= '0';
Data_7e07(8) <= '0';
Data_7e07(9) <= '0';
Data_7e07(10) <= '0';
Data_7e07(11) <= '0';
Data_7e07(12) <= '0';
Data_7e07(13) <= '0';
Data_7e07(14) <= '0';
Data_7e07(15) <= '0';
Data_7e07(16) <= '0';
Data_7e07(17) <= '0';
Data_7e07(18) <= '0';
Data_7e07(19) <= '0';
Data_7e07(20) <= '0';
Data_7e07(21) <= '0';
Data_7e07(22) <= '0';
Data_7e07(23) <= '0';
Data_7e07(24) <= '0';
Data_7e07(25) <= '0';
Data_7e07(26) <= '0';
Data_7e07(27) <= '0';
Data_7e07(28) <= '0';
Data_7e07(29) <= '0';
Data_7e07(30) <= '0';
Data_7e07(31) <= '0';
Strobe_7e07 <= EventTrig_7daf;
REG_R1 <= REG_OUT_7e07;
BITS_7ddf(0) <= SPLITBLK_7dd1;
COIN_SW_7ddf(0) <= '1';
BITS_7ddf(1) <= SPLITBLK_7f56;
COIN_SW_7ddf(1) <= '1';
COIN_SW_7ddf(63 downto 2) <= (others=>'0');
SPLITBLK_7f62 <= Signal_O_7ed3;
SPLITBLK_7dd1 <= SPLITBLK_7f62;
CTRLREG_7f72 <= REG_RW2;
Data_7f72(0) <= SPLITBLK_7f62;
Data_7f72(1) <= EventTrig_7daf;
Data_7f72(2) <= SPLITBLK_7f5a;
Data_7f72(3) <= SPLITBLK_7f53;
Data_7f72(4) <= SPLITBLK_7f4c;
Data_7f72(5) <= SPLITBLK_7f45;
Data_7f72(6) <= '0';
Data_7f72(7) <= '0';
SPLITBLK_7f5a <= BIT_OUT_7ddf;
Data_7daf <= SPLITBLK_7f5a;
SPLITBLK_7f53 <= Signal_O_7ecd;
SPLITBLK_7f56 <= SPLITBLK_7f53;
```

```
    SPLITBLK_7f4c <= NOT(D(2));
    Signal_I_7ed3 <= SPLITBLK_7f4c;
    STRETCH_LEN_7ed3 <= conv_integer(unsigned(REG_RW3(8 downto 0)));
    SPLITBLK_7f45 <= NOT(D(18));
    Signal_I_7ecd <= SPLITBLK_7f45;
    STRETCH_LEN_7ecd <= conv_integer(unsigned(REG_RW4(8 downto 0)));
    E(0) <= ScopeOut_7f72(0);
    E(16) <= ScopeOut_7f72(1);
    E(1) <= ScopeOut_7f72(2);
    E(17) <= ScopeOut_7f72(3);
    GenSATBPR_7e07 : GenSATBitPatternRegister
        port map (
            Data         => Data_7e07,
Strobe     => Strobe_7e07,
REG_OUT    => REG_OUT_7e07
        );
    GenSATMultBitCoin_7ddf : GenSATMultBitCoin
        port map (
            BITS    => BITS_7ddf,
SW    => COIN_SW_7ddf,
BIT_OUT => BIT_OUT_7ddf
        );
    GenSATScope_7f72 : GenSATScope
        port map (
            CTRLREG       => CTRLREG_7f72,
Data       => Data_7f72,
ScopeOut   => ScopeOut_7f72
        );
    GenSATEventTrigger_7daf : GenSATEventTrigger
        port map (
            LCLK    => PCLK,HOLD    => HOLD,
Data  => Data_7daf,
EventTrig => EventTrig_7daf
        );
    Stretcher_7ed3 : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_7ed3,
PULSE_IN  => Signal_I_7ed3,
PULSE_OUT => Signal_O_7ed3
        );
    Stretcher_7ecd : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_7ecd,
PULSE_IN  => Signal_I_7ecd,
PULSE_OUT => Signal_O_7ecd
        );
    HOLD <= REG_RW1(0); -- Synchronous Clear register REG_RW(0)=>'1'
       means clear everything. Default is '0'

    --GenSAT LRA Instantiation
    GenSATMainLRA : GenSATLocalRegisterAccess
```

```
        port map (
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
REG_R1   => REG_R1,
REG_R2   => REG_R2,
REG_R3   => REG_R3,
REG_RW1   => REG_RW1,
REG_RW2   => REG_RW2,
RELOAD_REG_RW2   => RELOAD_REG_RW2,
REG_RW3   => REG_RW3,
RELOAD_REG_RW3   => RELOAD_REG_RW3,
REG_RW4   => REG_RW4,
RELOAD_REG_RW4   => RELOAD_REG_RW4
        );
end;
```

APPENDIX E

SCILAB SCRIPT TO CREATE NIM ELECTRONICS PALETTES

Listing E.1: Scilab Script

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@ Create a New palette Named NIM Electronics
   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
loadScicosLibs();
pal = xcosPal("NIM Electronics");

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Declare Current Directory @@@@@
newDir=SCI+"/modules/xcos/src/NIMBlocks"

//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Define Modules Here @@@
//===========================================================
exec(newDir+"/Coincidence.sci");
scs_m =Coincidence("define");
export_to_hdf5(newDir +"/Coincidence.h5","scs_m");
//===========================================================
exec(newDir+"/Stretcher.sci");
scs_m =Stretcher("define");
export_to_hdf5(newDir +"/Stretcher.h5","scs_m");
//===========================================================
exec(newDir+"/MultBitStretcher.sci");
scs_m =MultBitStretcher("define");
export_to_hdf5(newDir +"/MultBitStretcher.h5","scs_m");
//===========================================================
exec(newDir+"/MultBitDelayer.sci");
scs_m =MultBitDelayer("define");
export_to_hdf5(newDir +"/MultBitDelayer.h5","scs_m");
//===========================================================
exec(newDir+"/MultBitORG.sci");
scs_m =MultBitORG("define");
export_to_hdf5(newDir +"/MultBitORG.h5","scs_m");
//===========================================================
exec(newDir+"/MultBitANDG.sci");
scs_m =MultBitANDG("define");
```

```
export_to_hdf5(newDir +"/MultBitANDG.h5","scs_m");
//=========================================================
exec(newDir+"/MultBitSignalMerger.sci");
scs_m =MultBitSignalMerger("define");
export_to_hdf5(newDir +"/MultBitSignalMerger.h5","scs_m");
//=========================================================
exec(newDir+"/BitPatternRegister.sci");
scs_m =BitPatternRegister("define");
export_to_hdf5(newDir +"/BitPatternRegister.h5","scs_m");
//=========================================================
exec(newDir+"/DNIMInputs.sci");
scs_m =DNIMInputs("define");
export_to_hdf5(newDir +"/DNIMInputs.h5","scs_m");
//=========================================================
exec(newDir+"/ENIMInputs.sci");
scs_m =ENIMInputs("define");
export_to_hdf5(newDir +"/ENIMInputs.h5","scs_m");
//=========================================================
exec(newDir+"/FNIMInputs.sci");
scs_m =FNIMInputs("define");
export_to_hdf5(newDir +"/FNIMInputs.h5","scs_m");
//=========================================================
exec(newDir+"/DECLInputs.sci");
scs_m =DECLInputs("define");
export_to_hdf5(newDir +"/DECLInputs.h5","scs_m");
//=========================================================
exec(newDir+"/EECLInputs.sci");
scs_m =EECLInputs("define");
export_to_hdf5(newDir +"/EECLInputs.h5","scs_m");
//=========================================================
exec(newDir+"/FECLInputs.sci");
scs_m =FECLInputs("define");
export_to_hdf5(newDir +"/FECLInputs.h5","scs_m");
//=========================================================
exec(newDir+"/GNIMInputs.sci");
scs_m =GNIMInputs("define");
export_to_hdf5(newDir +"/GNIMInputs.h5","scs_m");
//=========================================================
exec(newDir+"/AInputs.sci");
scs_m =AInputs("define");
export_to_hdf5(newDir +"/AInputs.h5","scs_m");
//=========================================================
exec(newDir+"/BInputs.sci");
scs_m =BInputs("define");
export_to_hdf5(newDir +"/BInputs.h5","scs_m");
//=========================================================
exec(newDir+"/COutputs.sci");
scs_m =COutputs("define");
export_to_hdf5(newDir +"/COutputs.h5","scs_m");
//=========================================================
exec(newDir+"/DNIMOutputs.sci");
scs_m =DNIMOutputs("define");
export_to_hdf5(newDir +"/DNIMOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/ENIMOutputs.sci");
```

```
scs_m =ENIMOutputs("define");
export_to_hdf5(newDir +"/ENIMOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/FNIMOutputs.sci");
scs_m =FNIMOutputs("define");
export_to_hdf5(newDir +"/FNIMOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/DECLOutputs.sci");
scs_m =DECLOutputs("define");
export_to_hdf5(newDir +"/DECLOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/EECLOutputs.sci");
scs_m =EECLOutputs("define");
export_to_hdf5(newDir +"/EECLOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/FECLOutputs.sci");
scs_m =FECLOutputs("define");
export_to_hdf5(newDir +"/FECLOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/GNIMOutputs.sci");
scs_m =GNIMOutputs("define");
export_to_hdf5(newDir +"/GNIMOutputs.h5","scs_m");
//=========================================================
exec(newDir+"/CFD.sci");
scs_m =CFD("define");
export_to_hdf5(newDir +"/CFD.h5","scs_m");
//=========================================================
exec(newDir+"/EventTrigger.sci");
scs_m =EventTrigger("define");
export_to_hdf5(newDir +"/EventTrigger.h5","scs_m");
//=========================================================
exec(newDir+"/GatenDelayGen.sci");
scs_m =GatenDelayGen("define");
export_to_hdf5(newDir +"/GatenDelayGen.h5","scs_m");
//=========================================================
//exec(newDir+"/GenSATFDLGnDG.sci");
//scs_m =GenSATFDLGnDG("define");
//export_to_hdf5(newDir +"/GenSATFDLGnDG.h5","scs_m");
//=========================================================
exec(newDir+"/LogicFan.sci");
scs_m =LogicFan("define");
export_to_hdf5(newDir +"/LogicFan.h5","scs_m");
//=========================================================
exec(newDir+"/Timer.sci");
scs_m =Timer("define");
export_to_hdf5(newDir +"/Timer.h5","scs_m");
//=========================================================
exec(newDir+"/TimetoAmpConverter.sci");
scs_m =TimetoAmpConverter("define");
export_to_hdf5(newDir +"/TimetoAmpConverter.h5","scs_m");
//=========================================================
exec(newDir+"/FANIO.sci");
scs_m =FANIO("define");
export_to_hdf5(newDir +"/FANIO.h5","scs_m");
//=========================================================
```

```
exec(newDir+"/FanOut.sci");
scs_m =FanOut("define");
export_to_hdf5(newDir +"/FanOut.h5","scs_m");
//=========================================================
exec(newDir+"/ORG.sci");
scs_m =ORG("define");
export_to_hdf5(newDir +"/ORG.h5","scs_m");
//=========================================================
exec(newDir+"/NIM_ECL.sci");
scs_m =NIM_ECL("define");
export_to_hdf5(newDir +"/NIM_ECL.h5","scs_m");
//=========================================================
exec(newDir+"/MULTIPLEXER.sci");
scs_m =MULTIPLEXER("define");
export_to_hdf5(newDir +"/MULTIPLEXER.h5","scs_m");
//=========================================================
exec(newDir+"/Scope.sci");
scs_m =Scope("define");
export_to_hdf5(newDir +"/Scope.h5","scs_m");
//=========================================================
exec(newDir+"/Delayer.sci");
scs_m =Delayer("define");
export_to_hdf5(newDir +"/Delayer.h5","scs_m");
//=========================================================
exec(newDir+"/DelayGenerator.sci");
scs_m =DelayGenerator("define");
export_to_hdf5(newDir +"/DelayGenerator.h5","scs_m");
//=========================================================
exec(newDir+"/Comparator.sci");
scs_m =Comparator("define");
export_to_hdf5(newDir +"/Comparator.h5","scs_m");
//=========================================================
exec(newDir+"/NOTG.sci");
scs_m =NOTG("define");
export_to_hdf5(newDir +"/NOTG.h5","scs_m");
//=========================================================
exec(newDir+"/ANDG.sci");
scs_m =ANDG("define");
export_to_hdf5(newDir +"/ANDG.h5","scs_m");
//=========================================================
exec(newDir+"/DISC.sci");
scs_m =DISC("define");
export_to_hdf5(newDir +"/DISC.h5","scs_m");
//=========================================================
exec(newDir+"/QFFLU.sci");
scs_m =QFFLU("define");
export_to_hdf5(newDir +"/QFFLU.h5","scs_m");
//=========================================================
exec(newDir+"/CODEGEN.sci");
scs_m =CODEGEN("define");
export_to_hdf5(newDir +"/CODEGEN.h5","scs_m");
//=========================================================
//exec(newDir+"/NIM3_15.sci");
//scs_m =NIM3_15("define");
//export_to_hdf5(newDir +"/NIM3_15.h5","scs_m");
```

```
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Add Blocks to the NIM palette @@@
pal = xcosPalAddBlock(pal,newDir+"/Coincidence.h5",newDir+"/Images/
    NIMBlocks/COINC.png",newDir+"/Images/NIMBlocks/COINC.png");
pal = xcosPalAddBlock(pal,newDir+"/Stretcher.h5",newDir+"/Images/
    NIMBlocks/STRETCH.png",newDir+"/Images/NIMBlocks/STRETCH.png");
pal = xcosPalAddBlock(pal,newDir+"/MultBitStretcher.h5",newDir+"/Images/
    NIMBlocks/STRETCH.png",newDir+"/Images/NIMBlocks/STRETCH.png");
pal = xcosPalAddBlock(pal,newDir+"/BitPatternRegister.h5",newDir+"/Images
    /NIMBlocks/BPR.png",newDir+"/Images/NIMBlocks/BPR.png");
//pal = xcosPalAddBlock(pal,newDir+"/GenSAT32InBPR.h5",newDir+"/Images/
    NIMBlocks/BPR.png",newDir+"/Images/NIMBlocks/BPR.png");
pal = xcosPalAddBlock(pal,newDir+"/CFD.h5",newDir+"/Images/NIMBlocks/CFD0
    .png",newDir+"/Images/NIMBlocks/CFD0.png");
pal = xcosPalAddBlock(pal,newDir+"/EventTrigger.h5",newDir+"/Images/
    NIMBlocks/ETRIG.png",newDir+"/Images/NIMBlocks/ETRIG.png");
pal = xcosPalAddBlock(pal,newDir+"/GatenDelayGen.h5",newDir+"/Images/
    NIMBlocks/GNDG.png",newDir+"/Images/NIMBlocks/GNDG.png");
//pal = xcosPalAddBlock(pal,newDir+"/GenSATFDLGnDG.h5",newDir+"/Images/
    NIMBlocks/GNDG.png",newDir+"/Images/NIMBlocks/GNDG.png");
pal = xcosPalAddBlock(pal,newDir+"/ORG.h5",newDir+"/Images/NIMBlocks/org.
    png",newDir+"/Images/NIMBlocks/org.png");
pal = xcosPalAddBlock(pal,newDir+"/MultBitORG.h5",newDir+"/Images/
    NIMBlocks/org.png",newDir+"/Images/NIMBlocks/org.png");
pal = xcosPalAddBlock(pal,newDir+"/Timer.h5",newDir+"/Images/NIMBlocks/
    TIME.png",newDir+"/Images/NIMBlocks/TIME.png");
pal = xcosPalAddBlock(pal,newDir+"/TimetoAmpConverter.h5",newDir+"/Images
    /NIMBlocks/TAC.png",newDir+"/Images/NIMBlocks/TAC.png");
pal = xcosPalAddBlock(pal,newDir+"/FANIO.h5",newDir+"/Images/NIMBlocks/
    LFAN.png",newDir+"/Images/NIMBlocks/LFAN.png");
pal = xcosPalAddBlock(pal,newDir+"/FanOut.h5",newDir+"/Images/NIMBlocks/
    LFAN.png",newDir+"/Images/NIMBlocks/LFAN.png");
pal = xcosPalAddBlock(pal,newDir+"/NIM_ECL.h5",newDir+"/Images/NIMBlocks/
    nimecl.png",newDir+"/Images/NIMBlocks/nimecl.png");
//pal = xcosPalAddBlock(pal,newDir+"/MULTIPLEXER.h5",newDir+"/Images/
    NIMBlocks/mult.png",newDir+"/Images/NIMBlocks/mult.png");
pal = xcosPalAddBlock(pal,newDir+"/Scope.h5",newDir+"/Images/NIMBlocks/
    mult.png",newDir+"/Images/NIMBlocks/mult.png");
pal = xcosPalAddBlock(pal,newDir+"/Delayer.h5",newDir+"/Images/NIMBlocks/
    del.png",newDir+"/Images/NIMBlocks/del.png");
pal = xcosPalAddBlock(pal,newDir+"/MultBitDelayer.h5",newDir+"/Images/
    NIMBlocks/del.png",newDir+"/Images/NIMBlocks/del.png");
pal = xcosPalAddBlock(pal,newDir+"/MultBitSignalMerger.h5",newDir+"/
    Images/NIMBlocks/multbitmerger.png",newDir+"/Images/NIMBlocks/
    multbitmerger.png");
//pal = xcosPalAddBlock(pal,newDir+D"/DelayGenerator.h5",newDir+"/Images/
    NIMBlocks/del.png",newDir+"/Images/NIMBlocks/del.png");
pal = xcosPalAddBlock(pal,newDir+"/Comparator.h5",newDir+"/Images/
    NIMBlocks/comp.png",newDir+"/Images/NIMBlocks/comp.png");
pal = xcosPalAddBlock(pal,newDir+"/ANDG.h5",newDir+"/Images/NIMBlocks/
    andg.png",newDir+"/Images/NIMBlocks/andg.png");
pal = xcosPalAddBlock(pal,newDir+"/MultBitANDG.h5",newDir+"/Images/
    NIMBlocks/andg.png",newDir+"/Images/NIMBlocks/andg.png");
pal = xcosPalAddBlock(pal,newDir+"/NOTG.h5",newDir+"/Images/NIMBlocks/
    notg.png",newDir+"/Images/NIMBlocks/notg.png");
```

```
pal = xcosPalAddBlock(pal,newDir+"/DISC.h5",newDir+"/Images/NIMBlocks/
    disc.png",newDir+"/Images/NIMBlocks/disc.png");
pal = xcosPalAddBlock(pal,newDir+"/QFFLU.h5",newDir+"/Images/NIMBlocks/
    QFFLU.png",newDir+"/Images/NIMBlocks/QFFLU.png");
pal = xcosPalAddBlock(pal,newDir+"/CODEGEN.h5",newDir+"/Images/NIMBlocks/
    codegen.png",newDir+"/Images/NIMBlocks/codegen.png");
//pal = xcosPalAddBlock(pal,newDir+"/NIM3_15.h5");
pal = xcosPalAddBlock(pal,newDir+"/DNIMInputs.h5",newDir+"/Images/
    NIMBlocks/dniminputs.png",newDir+"/Images/NIMBlocks/dniminputs.png");
pal = xcosPalAddBlock(pal,newDir+"/ENIMInputs.h5",newDir+"/Images/
    NIMBlocks/eniminputs.png",newDir+"/Images/NIMBlocks/eniminputs.png");
pal = xcosPalAddBlock(pal,newDir+"/FNIMInputs.h5",newDir+"/Images/
    NIMBlocks/fniminputs.png",newDir+"/Images/NIMBlocks/fniminputs.png");
pal = xcosPalAddBlock(pal,newDir+"/DECLInputs.h5",newDir+"/Images/
    NIMBlocks/declinputs.png",newDir+"/Images/NIMBlocks/declinputs.png");
pal = xcosPalAddBlock(pal,newDir+"/EECLInputs.h5",newDir+"/Images/
    NIMBlocks/eeclinputs.png",newDir+"/Images/NIMBlocks/eeclinputs.png");
pal = xcosPalAddBlock(pal,newDir+"/FECLInputs.h5",newDir+"/Images/
    NIMBlocks/feclinputs.png",newDir+"/Images/NIMBlocks/feclinputs.png");
pal = xcosPalAddBlock(pal,newDir+"/GNIMInputs.h5",newDir+"/Images/
    NIMBlocks/gniminputs.png",newDir+"/Images/NIMBlocks/gniminputs.png");
pal = xcosPalAddBlock(pal,newDir+"/AInputs.h5",newDir+"/Images/NIMBlocks/
    ainputs.png",newDir+"/Images/NIMBlocks/ainputs.png");
pal = xcosPalAddBlock(pal,newDir+"/BInputs.h5",newDir+"/Images/NIMBlocks/
    binputs.png",newDir+"/Images/NIMBlocks/binputs.png");
pal = xcosPalAddBlock(pal,newDir+"/COutputs.h5",newDir+"/Images/NIMBlocks
    /coutputs.png",newDir+"/Images/NIMBlocks/coutputs.png");
pal = xcosPalAddBlock(pal,newDir+"/DNIMOutputs.h5",newDir+"/Images/
    NIMBlocks/dnimoutputs.png",newDir+"/Images/NIMBlocks/dnimoutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/ENIMOutputs.h5",newDir+"/Images/
    NIMBlocks/enimoutputs.png",newDir+"/Images/NIMBlocks/enimoutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/FNIMOutputs.h5",newDir+"/Images/
    NIMBlocks/fnimoutputs.png",newDir+"/Images/NIMBlocks/fnimoutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/DECLOutputs.h5",newDir+"/Images/
    NIMBlocks/decloutputs.png",newDir+"/Images/NIMBlocks/decloutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/EECLOutputs.h5",newDir+"/Images/
    NIMBlocks/eecloutputs.png",newDir+"/Images/NIMBlocks/eecloutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/FECLOutputs.h5",newDir+"/Images/
    NIMBlocks/fecloutputs.png",newDir+"/Images/NIMBlocks/fecloutputs.png"
    );
pal = xcosPalAddBlock(pal,newDir+"/GNIMOutputs.h5",newDir+"/Images/
    NIMBlocks/gnimoutputs.png",newDir+"/Images/NIMBlocks/gnimoutputs.png"
    );
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ Deploy NIM Palette @@@
//
xcosPalAdd(pal,"Palettes");
```

APPENDIX F

TOP-LEVEL ENTITY TEMPLATE

Listing F.1: The template for top-level entity for triggering system modelled using the model interpreter presented

```
  --*************************************************
  --********* Library inclusions ***********************

entity GenSATMain is port
(
        A       : in std_logic_vector(31 downto 0);--32 ECL/LVDS Inputs
        B       : in std_logic_vector(31 downto 0);--32 ECL/LVDS Inputs
        C       : out std_logic_vector(31 downto 0);--32 ECL/LVDS Outputs
        D       : inout std_logic_vector(31 downto 0);-- Mezzanine slot D
        E       : inout std_logic_vector(31 downto 0);-- Mezzanine slot E
        F       : inout std_logic_vector(31 downto 0);-- Mezzanine slot F
        GIN     : in std_logic_vector(1 downto 0);-- Mezzanine slot G
        GOUT    : out std_logic_vector(1 downto 0);-- Mezzanine slot G
        nOED    : out std_logic;--D Slot Ports I/O Enable 1=Input,0=
           Output
        nOEE    : out std_logic;--E Slot Ports I/O Enable 1=Input,0=
           Output
        nOEF    : out std_logic;--F Slot Ports I/O Enable 1=Input,0=
           Output
        nOEG    : out std_logic;--G Slot Ports I/O Enable 1=Input,0=
           Output
        SELD    : out std_logic;--D Slot Ports Level Select 1=TTL,0=NIM
        SELE    : out std_logic;--E Slot Ports Level Select 1=TTL,0=NIM
        SELF    : out std_logic;--F Slot Ports Level Select 1=TTL,0=NIM
        SELG    : out std_logic;--G Slot Ports Level Select 1=TTL,0=NIM
        IDD     : out std_logic_vector(2 downto 0);
        IDE     : out std_logic_vector(2 downto 0);
        IDF     : out std_logic_vector(2 downto 0);
        PULSE     : in std_logic_vector(3 downto 0);
        nSTART    : out std_logic_vector(1 downto 0);
        START     : out std_logic_vector(1 downto 0);
        DDLY    : inout std_logic_vector(7 downto 0);
        WR_DLY0   : out std_logic;
        WR_DLY1   : out std_logic;
        DIRDDLY   : out std_logic;
        nOEDDLY0  : out std_logic;
```

```
        nOEDDLY1  : out std_logic;
        nLEDG     : out std_logic;
        nLEDR     : out std_logic;
        SPARE     : inout std_logic_vector(11 downto 0);
        nlBRES    : in std_logic;
        nBLAST    : in std_logic;
        WnR     : in std_logic;
        nADS    : in std_logic;
        LCLK    : in std_logic;
        nREADY    : out std_logic;
        nINT    : out std_logic;
        LAD     : inout std_logic_vector(15 downto 0)
);
end entity GenSATMain;

architecture rtl of GenSATMain is
  --*******************************************************
  --********* Signal Declarations *********************
begin
  --*******************************************************
  --********* V1495 Port I/O Level Selects **********
  --********* 0=NIM, 1=NIM, etc ************************

  --*******************************************************
  --********* Port Assignments ************************

  --*******************************************************
  --********* Instantiations ************************
end;
```

APPENDIX G

QUARTUS II PROJECT FILE TEMPLATE

Listing G.1: Quartus II Project File template used to create trigger Project files

```
#============================================================
#========== Altera Quartus II Project File Template ======

set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C20F400C6
set_global_assignment -name TOP_LEVEL_ENTITY GenSATMain
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 8.0
set_global_assignment -name PROJECT_CREATION_TIME_DATE "12:44:20  MARCH
    10, 2010"
set_global_assignment -name LAST_QUARTUS_VERSION "10.1 SP1"
set_global_assignment -name USE_GENERATED_PHYSICAL_CONSTRAINTS OFF -
    section_id eda_palace
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_COLOR 14622752 -section_id Top
set_global_assignment -name LL_ROOT_REGION ON -section_id "Root Region"
set_global_assignment -name LL_MEMBER_STATE LOCKED -section_id "Root
    Region"


# Pin & Location Assignments
# ==========================================================
set_location_assignment PIN_V3 -to B[17]
set_location_assignment PIN_U2 -to C[5]
set_location_assignment PIN_V2 -to C[11]
set_location_assignment PIN_R4 -to C[17]
set_location_assignment PIN_U3 -to C[22]
set_location_assignment PIN_T3 -to E[21]
set_location_assignment PIN_T2 -to F[15]
set_location_assignment PIN_K5 -to LCLK
set_location_assignment PIN_F3 -to PULSE[0]
set_location_assignment PIN_T4 -to PULSE[2]
set_location_assignment PIN_G3 -to A[0]
set_location_assignment PIN_F2 -to A[1]
set_location_assignment PIN_F6 -to A[3]
set_location_assignment PIN_J5 -to A[8]
```

```
set_location_assignment PIN_H1 -to A[11]
set_location_assignment PIN_H2 -to A[12]
set_location_assignment PIN_H3 -to A[13]
set_location_assignment PIN_G5 -to A[14]
set_location_assignment PIN_H6 -to A[18]
set_location_assignment PIN_F5 -to A[26]
set_location_assignment PIN_J3 -to A[29]
set_location_assignment PIN_P5 -to B[0]
set_location_assignment PIN_P2 -to B[1]
set_location_assignment PIN_M4 -to B[6]
set_location_assignment PIN_M2 -to B[7]
set_location_assignment PIN_H5 -to B[9]
set_location_assignment PIN_G6 -to B[10]
set_location_assignment PIN_P3 -to B[16]
set_location_assignment PIN_P6 -to B[20]
set_location_assignment PIN_M7 -to B[21]
set_location_assignment PIN_N3 -to B[22]
set_location_assignment PIN_M1 -to B[23]
set_location_assignment PIN_H4 -to B[24]
set_location_assignment PIN_G4 -to B[25]
set_location_assignment PIN_F4 -to B[26]
set_location_assignment PIN_G2 -to B[27]
set_location_assignment PIN_M3 -to C[4]
set_location_assignment PIN_R6 -to C[6]
set_location_assignment PIN_R2 -to C[7]
set_location_assignment PIN_R1 -to C[10]
set_location_assignment PIN_R5 -to C[14]
set_location_assignment PIN_N4 -to C[18]
set_location_assignment PIN_R3 -to C[23]
set_location_assignment PIN_P4 -to C[26]
set_location_assignment PIN_M5 -to C[31]
set_location_assignment PIN_P1 -to E[7]
set_location_assignment PIN_J4 -to E[14]
set_location_assignment PIN_P7 -to E[17]
set_location_assignment PIN_N2 -to E[22]
set_location_assignment PIN_N1 -to E[23]
set_location_assignment PIN_F1 -to E[24]
set_location_assignment PIN_G1 -to E[25]
set_location_assignment PIN_E2 -to E[26]
set_location_assignment PIN_M6 -to F[2]
set_location_assignment PIN_N7 -to F[10]
set_location_assignment PIN_N6 -to F[11]
set_location_assignment PIN_E3 -to FPGA[1]
set_location_assignment PIN_J6 -to IDD[1]
set_location_assignment PIN_J1 -to LAD[3]
set_location_assignment PIN_G7 -to LAD[9]
set_location_assignment PIN_J2 -to nOED
set_location_assignment PIN_C2 -to A[2]
set_location_assignment PIN_C4 -to A[5]
set_location_assignment PIN_D1 -to A[9]
set_location_assignment PIN_C5 -to A[15]
set_location_assignment PIN_D5 -to A[16]
set_location_assignment PIN_F7 -to A[17]
set_location_assignment PIN_D6 -to A[20]
set_location_assignment PIN_E5 -to A[22]
```

```
set_location_assignment PIN_B6 -to A[28]
set_location_assignment PIN_B4 -to B[11]
set_location_assignment PIN_E7 -to B[29]
set_location_assignment PIN_E4 -to FPGA[0]
set_location_assignment PIN_B3 -to FPGA[2]
set_location_assignment PIN_B5 -to LAD[4]
set_location_assignment PIN_B7 -to LAD[10]
set_location_assignment PIN_A7 -to nINT
set_location_assignment PIN_A6 -to nREADY
set_location_assignment PIN_A4 -to WnR
set_location_assignment PIN_D8 -to A[4]
set_location_assignment PIN_A8 -to A[6]
set_location_assignment PIN_C8 -to A[10]
set_location_assignment PIN_A9 -to A[24]
set_location_assignment PIN_F8 -to B[28]
set_location_assignment PIN_D7 -to B[30]
set_location_assignment PIN_C7 -to B[31]
set_location_assignment PIN_B8 -to FPGA[3]
set_location_assignment PIN_E8 -to LAD[11]
set_location_assignment PIN_E11 -to LAD[12]
set_location_assignment PIN_C6 -to PULSE[1]
set_location_assignment PIN_C15 -to PULSE[3]
set_location_assignment PIN_B11 -to A[7]
set_location_assignment PIN_C9 -to A[19]
set_location_assignment PIN_D9 -to A[21]
set_location_assignment PIN_A12 -to A[23]
set_location_assignment PIN_C14 -to A[25]
set_location_assignment PIN_B12 -to A[27]
set_location_assignment PIN_C12 -to A[30]
set_location_assignment PIN_A13 -to A[31]
set_location_assignment PIN_C11 -to B[12]
set_location_assignment PIN_D11 -to B[13]
set_location_assignment PIN_C10 -to B[14]
set_location_assignment PIN_D10 -to B[15]
set_location_assignment PIN_E14 -to D[0]
set_location_assignment PIN_E13 -to D[1]
set_location_assignment PIN_D12 -to D[2]
set_location_assignment PIN_F12 -to D[3]
set_location_assignment PIN_E9 -to D[4]
set_location_assignment PIN_C13 -to D[5]
set_location_assignment PIN_F14 -to D[6]
set_location_assignment PIN_D16 -to D[10]
set_location_assignment PIN_E10 -to D[11]
set_location_assignment PIN_E15 -to D[14]
set_location_assignment PIN_D14 -to D[15]
set_location_assignment PIN_A17 -to E[27]
set_location_assignment PIN_D13 -to E[29]
set_location_assignment PIN_C16 -to E[30]
set_location_assignment PIN_E12 -to E[31]
set_location_assignment PIN_E16 -to IDD[0]
set_location_assignment PIN_C17 -to IDD[2]
set_location_assignment PIN_B17 -to LAD[0]
set_location_assignment PIN_B15 -to LAD[1]
set_location_assignment PIN_B18 -to LAD[2]
set_location_assignment PIN_A11 -to LAD[5]
```

```
set_location_assignment PIN_B14 -to LAD[6]
set_location_assignment PIN_B10 -to LAD[7]
set_location_assignment PIN_B13 -to LAD[8]
set_location_assignment PIN_B9 -to LAD[14]
set_location_assignment PIN_B16 -to LAD[15]
set_location_assignment PIN_A10 -to nADS
set_location_assignment PIN_A15 -to nBLAST
set_location_assignment PIN_A14 -to nLBRES
set_location_assignment PIN_F16 -to D[7]
set_location_assignment PIN_F19 -to D[8]
set_location_assignment PIN_H15 -to D[9]
set_location_assignment PIN_C19 -to D[12]
set_location_assignment PIN_C18 -to D[13]
set_location_assignment PIN_G14 -to D[16]
set_location_assignment PIN_G17 -to D[17]
set_location_assignment PIN_G16 -to D[18]
set_location_assignment PIN_F17 -to D[19]
set_location_assignment PIN_D19 -to D[20]
set_location_assignment PIN_E17 -to D[21]
set_location_assignment PIN_D20 -to D[22]
set_location_assignment PIN_F20 -to D[23]
set_location_assignment PIN_E18 -to D[24]
set_location_assignment PIN_E19 -to D[25]
set_location_assignment PIN_D18 -to DDLY[4]
set_location_assignment PIN_G15 -to E[8]
set_location_assignment PIN_G19 -to E[11]
set_location_assignment PIN_G20 -to E[13]
set_location_assignment PIN_L14 -to GIN[0]
set_location_assignment PIN_F18 -to GIN[1]
set_location_assignment PIN_P20 -to GOUT[0]
set_location_assignment PIN_H16 -to IDE[0]
set_location_assignment PIN_J13 -to IDE[1]
set_location_assignment PIN_M14 -to IDE[2]
set_location_assignment PIN_K19 -to nLEDG
set_location_assignment PIN_D17 -to nLEDR
set_location_assignment PIN_N14 -to nOEE
set_location_assignment PIN_F15 -to SELD
set_location_assignment PIN_T18 -to SELE
set_location_assignment PIN_J16 -to D[28]
set_location_assignment PIN_H19 -to E[9]
set_location_assignment PIN_H18 -to E[10]
set_location_assignment PIN_H17 -to E[12]
set_location_assignment PIN_G18 -to E[15]
set_location_assignment PIN_J15 -to E[28]
set_location_assignment PIN_J14 -to B[8]
set_location_assignment PIN_T16 -to C[9]
set_location_assignment PIN_J18 -to D[26]
set_location_assignment PIN_K15 -to D[27]
set_location_assignment PIN_M15 -to D[29]
set_location_assignment PIN_K16 -to D[30]
set_location_assignment PIN_J17 -to D[31]
set_location_assignment PIN_U15 -to DDLY[0]
set_location_assignment PIN_M16 -to DDLY[1]
set_location_assignment PIN_N17 -to DDLY[2]
set_location_assignment PIN_W16 -to DDLY[3]
```

```
set_location_assignment PIN_N16 -to DDLY[5]
set_location_assignment PIN_U16 -to DDLY[6]
set_location_assignment PIN_V15 -to DDLY[7]
set_location_assignment PIN_W14 -to DIRDDLY
set_location_assignment PIN_T17 -to F[4]
set_location_assignment PIN_R16 -to F[5]
set_location_assignment PIN_P14 -to F[7]
set_location_assignment PIN_P18 -to F[8]
set_location_assignment PIN_R17 -to F[9]
set_location_assignment PIN_V18 -to F[12]
set_location_assignment PIN_U18 -to F[13]
set_location_assignment PIN_N15 -to F[14]
set_location_assignment PIN_U14 -to F[16]
set_location_assignment PIN_T15 -to F[17]
set_location_assignment PIN_V14 -to F[18]
set_location_assignment PIN_R14 -to F[19]
set_location_assignment PIN_T14 -to F[24]
set_location_assignment PIN_Y14 -to F[25]
set_location_assignment PIN_L13 -to GOUT[1]
set_location_assignment PIN_R18 -to IDF[1]
set_location_assignment PIN_H14 -to LAD[13]
set_location_assignment PIN_P17 -to nOEDDLY0
set_location_assignment PIN_R15 -to nOEDDLY1
set_location_assignment PIN_P15 -to nOEG
set_location_assignment PIN_R20 -to nSTART[2]
set_location_assignment PIN_R19 -to nSTART[3]
set_location_assignment PIN_P16 -to SELG
set_location_assignment PIN_V17 -to START[0]
set_location_assignment PIN_V16 -to START[1]
set_location_assignment PIN_Y17 -to WR_DLY0
set_location_assignment PIN_W17 -to WR_DLY1
set_location_assignment PIN_W9 -to B[19]
set_location_assignment PIN_U13 -to C[0]
set_location_assignment PIN_Y9 -to C[21]
set_location_assignment PIN_W10 -to C[25]
set_location_assignment PIN_Y10 -to C[30]
set_location_assignment PIN_V10 -to E[0]
set_location_assignment PIN_Y15 -to E[1]
set_location_assignment PIN_T6 -to E[2]
set_location_assignment PIN_W15 -to E[3]
set_location_assignment PIN_R11 -to E[4]
set_location_assignment PIN_U12 -to E[5]
set_location_assignment PIN_R13 -to E[6]
set_location_assignment PIN_U10 -to E[18]
set_location_assignment PIN_V13 -to E[20]
set_location_assignment PIN_T13 -to F[0]
set_location_assignment PIN_Y12 -to F[1]
set_location_assignment PIN_W13 -to F[3]
set_location_assignment PIN_T10 -to F[6]
set_location_assignment PIN_T11 -to F[20]
set_location_assignment PIN_U11 -to F[21]
set_location_assignment PIN_V11 -to F[22]
set_location_assignment PIN_R9 -to F[23]
set_location_assignment PIN_W11 -to F[26]
set_location_assignment PIN_Y11 -to F[27]
```

```
set_location_assignment PIN_T9 -to F[28]
set_location_assignment PIN_T12 -to F[29]
set_location_assignment PIN_W12 -to IDF[0]
set_location_assignment PIN_V12 -to IDF[2]
set_location_assignment PIN_Y13 -to nOEF
set_location_assignment PIN_W6 -to B[2]
set_location_assignment PIN_Y8 -to B[3]
set_location_assignment PIN_U8 -to B[4]
set_location_assignment PIN_U6 -to B[5]
set_location_assignment PIN_W8 -to B[18]
set_location_assignment PIN_Y4 -to C[1]
set_location_assignment PIN_V8 -to C[2]
set_location_assignment PIN_W7 -to C[3]
set_location_assignment PIN_R7 -to C[8]
set_location_assignment PIN_T5 -to C[12]
set_location_assignment PIN_U5 -to C[13]
set_location_assignment PIN_Y7 -to C[15]
set_location_assignment PIN_V5 -to C[16]
set_location_assignment PIN_Y6 -to C[19]
set_location_assignment PIN_V6 -to C[20]
set_location_assignment PIN_U9 -to C[24]
set_location_assignment PIN_W3 -to C[27]
set_location_assignment PIN_U7 -to C[28]
set_location_assignment PIN_V7 -to C[29]
set_location_assignment PIN_V4 -to E[16]
set_location_assignment PIN_W4 -to E[19]
set_location_assignment PIN_V9 -to F[30]
set_location_assignment PIN_T8 -to F[31]
set_location_assignment PIN_W5 -to SELF
set_location_assignment PIN_H7 -to SPARE[11]
set_location_assignment PIN_J7 -to SPARE[10]
set_location_assignment PIN_N5 -to SPARE[9]
set_location_assignment PIN_U4 -to SPARE[8]
set_location_assignment PIN_D3 -to SPARE[6]
set_location_assignment PIN_D2 -to SPARE[7]
set_location_assignment PIN_U19 -to SPARE[0]
set_location_assignment PIN_V19 -to SPARE[3]
set_location_assignment PIN_U20 -to SPARE[2]
set_location_assignment PIN_W18 -to SPARE[1]
set_location_assignment PIN_N20 -to SPARE[5]
set_location_assignment PIN_P19 -to SPARE[4]

# Fitter Assignments
# ==================
set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
    nLEDG
set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
    nLEDR
set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
    D
set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
    E
set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
    F
```

```
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to IDD
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to IDE
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to IDF

set_instance_assignment -name SLOW_SLEW_RATE ON -to D
set_instance_assignment -name SLOW_SLEW_RATE ON -to E
set_instance_assignment -name SLOW_SLEW_RATE ON -to F

# ================================================
set_global_assignment -name FMAX_REQUIREMENT "40 MHz" -section_id LCLK
set_instance_assignment -name CLOCK_SETTINGS LCLK -to LCLK
set_global_assignment -name ENABLE_INIT_DONE_OUTPUT ON
set_global_assignment -name GENERATE_RBF_FILE ON
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATED"
set_global_assignment -name RESERVE_ALL_UNUSED_PINS_NO_OUTPUT_GND "AS
    OUTPUT DRIVING AN UNSPECIFIED SIGNAL"

# ================================================
# ==================== Triggering Platform Packages ===
#set_global_assignment -name VHDL_FILE ../GenSATIPCores/GenSATPack/lb_int
    .vhd
set_global_assignment -name VHDL_FILE ../GenSATIPCores/GenSATPack/
    gbpr_fifo.vhd
set_global_assignment -name VHDL_FILE ../GenSATIPCores/GenSATPack/
    GenSATPack.vhd
set_global_assignment -name VHDL_FILE ../GenSATIPCores/GenSATPack/
    GenSATArrays.vhd
set_global_assignment -name VHDL_FILE ../GenSATIPCores/GenSATPack/PLL.vhd

# ========================================================
# ========= Triggering Platform Project NIM VHDL files ==
```

APPENDIX H

LOCAL REGISTER ACCESS COMPONENTS

H.1 SAMPLE LRA VHDL CODE GENERATED BY MODEL INTERPRETER

Listing H.1: Sample LRA VHDL Code generated by Model Interpreter

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity GenSATLocalRegisterAccess is
    port (
        nLBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0);
        REG_R1 : in std_logic_vector(31 downto 0);
        REG_R2 : in std_logic_vector(31 downto 0);
        REG_R3 : in std_logic_vector(31 downto 0);
        REG_RW1 : buffer std_logic_vector(31 downto 0);
        REG_RW2 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW2 : out std_logic;
        REG_RW3 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW3 : out std_logic;
        REG_RW4 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW4 : out std_logic
    );
end entity GenSATLocalRegisterAccess;
architecture rtl of GenSATLocalRegisterAccess is
    type LBSTATE_type is (LBIDLE, LBWRITEL, LBWRITEH, LBREADL, LBREADH);
    signal LBSTATE : LBSTATE_type;
```

```
    signal LADoe : std_logic;
    signal LADout : std_logic_vector(15 downto 0);
    signal DTL : std_logic_vector(15 downto 0);
    signal ADDR : std_logic_vector(15 downto 0);
    constant A_REG_R1 : std_logic_vector(15 downto 0) := x"1000";
    constant A_REG_R2 : std_logic_vector(15 downto 0) := x"1004";
    constant A_REG_R3 : std_logic_vector(15 downto 0) := x"1008";
    constant A_REG_RW1 : std_logic_vector(15 downto 0) := x"1028";
    constant A_REG_RW2 : std_logic_vector(15 downto 0) := x"102C";
    constant A_REG_RW3 : std_logic_vector(15 downto 0) := x"1030";
    constant A_REG_RW4 : std_logic_vector(15 downto 0) := x"1034";
begin
    LAD <= LADout when LADoe = '1' else (others => 'Z');
    process(LCLK, nLBRES)
        variable rreg : std_logic_vector(31 downto 0);
        variable wreg : std_logic_vector(31 downto 0);
    begin
        if (nLBRES = '0') then
            REG_RW1 <= x"00000000";
            REG_RW2 <= x"32160801";
            REG_RW3 <= x"00000064";
            REG_RW4 <= x"00000064";
            nREADY <= '1';
            LADoe <= '0';
            ADDR <= (others => '0');
            DTL <= (others => '0');
            LADout <= (others => '0');
            rreg := (others => '0');
            wreg := (others => '0');
            LBSTATE <= LBIDLE;
        elsif rising_edge(LCLK) then
            case LBSTATE is
                when LBIDLE =>
                    LADoe <= '0';
                    nREADY <= '1';
                    RELOAD_REG_RW2 <= '0';
                    RELOAD_REG_RW3 <= '0';
                    RELOAD_REG_RW4 <= '0';
                    if (nADS = '0') then
                        ADDR <= LAD;
                        if (WnR = '1') then
                            nREADY <= '0';
                            LBSTATE <= LBWRITEL;
                        else
                            nREADY <= '1';
                            LBSTATE <= LBREADL;
                        end if;
                    end if;

                when LBWRITEL =>
                    DTL <= LAD;
                    if (nBLAST = '0') then
                        LBSTATE <= LBIDLE;
                        nREADY <= '1';
                    else
```

```
            LBSTATE <= LBWRITEH;
        end if;

    when LBWRITEH =>
        wreg := LAD & DTL;
        case ADDR is
            when A_REG_RW1 =>
                REG_RW1 <= wreg;

            when A_REG_RW2 =>
                REG_RW2 <= wreg;
                RELOAD_REG_RW2 <= '1';

            when A_REG_RW3 =>
                REG_RW3 <= wreg;
                RELOAD_REG_RW3 <= '1';

            when A_REG_RW4 =>
                REG_RW4 <= wreg;
                RELOAD_REG_RW4 <= '1';

            when others =>
                null;

        end case;
        nREADY <= '1';
        LBSTATE <= LBIDLE;

    when LBREADL =>
        nREADY <= '0';
        case ADDR is
            when A_REG_R1 =>
                rreg := REG_R1;

            when A_REG_R2 =>
                rreg := REG_R2;

            when A_REG_R3 =>
                rreg := REG_R3;

            when A_REG_RW1 =>
                rreg := REG_RW1;

            when A_REG_RW2 =>
                rreg := REG_RW2;

            when A_REG_RW3 =>
                rreg := REG_RW3;

            when A_REG_RW4 =>
                rreg := REG_RW4;

            when others =>
                null;
```

```
                    end case;
                    LBSTATE <= LBREADH;
                    LADout <= rreg(15 downto 0);
                    LADoe <= '1';

                when LBREADH =>
                    LADout <= rreg(31 downto 16);
                    LBSTATE <= LBIDLE;

            end case;
        end if;
    end process;
end;
```

H.2 LIST OF REGISTERS

Table H.1: Table of 32-Bits Registers and their VME addresses

Register Name	VME Address	Operation
REG_R1	1000	Read Only
REG_R2	1004	Read Only
REG_R3	1008	Read Only
REG_R4	100C	Read Only
REG_R5	1010	Read Only
REG_R6	1014	Read Only
REG_R7	1018	Read Only
REG_R8	101C	Read Only
REG_R9	1020	Read Only
REG_R10	1024	Read Only
REG_RW1	1028	Read/Write
REG_RW2	102C	Read/Write
REG_RW3	1030	Read/Write
REG_RW4	1034	Read/Write
REG_RW5	1038	Read/Write
REG_RW6	103C	Read/Write
REG_RW7	1040	Read/Write
REG_RW8	1044	Read/Write
REG_RW9	1048	Read/Write
REG_RW10	1100	Read/Write
REG_RW11	1104	Read/Write
REG_RW12	1108	Read/Write
REG_RW13	110C	Read/Write
REG_RW14	1010	Read/Write
REG_RW15	1014	Read/Write
REG_RW16	1018	Read/Write
REG_RW17	101C	Read/Write
REG_RW18	1120	Read/Write
REG_RW19	1124	Read/Write
REG_RW20	1128	Read/Write
REG_RW21	112C	Read/Write
REG_RW22	1130	Read/Write
REG_RW23	1134	Read/Write
REG_RW24	1138	Read/Write
REG_RW25	113C	Read/Write

APPENDIX I

AFRODITE ELECTRONICS DIAGRAM AND GENERATED VHDL CODE

I.1 AFRODITE ELECTRONICS DIAGRAM

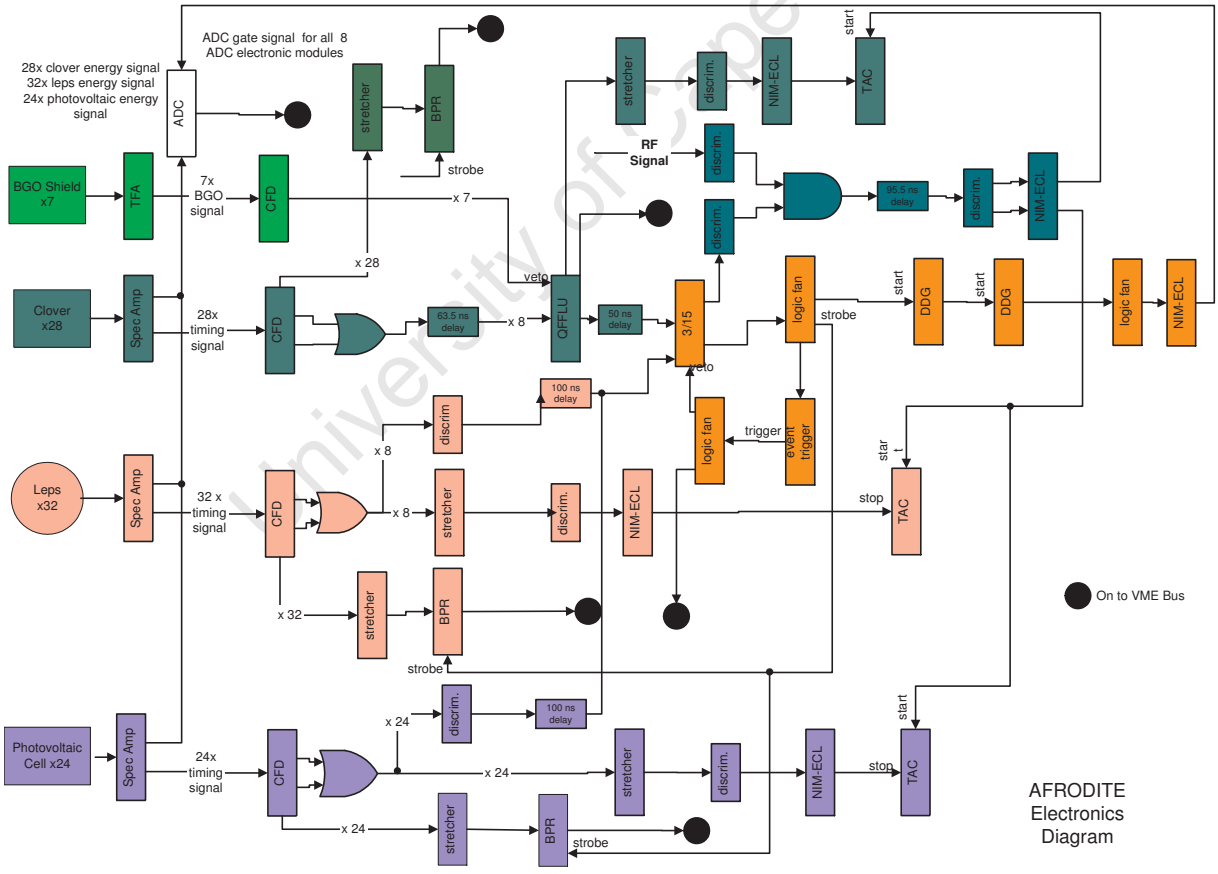


Figure I.1: AFRODITE Electronics Diagram (Image based on [42])

I.2 AFRODITE GENERATED VHDL CODE

Listing I.1: AFRODITE Comparative Trigger VHDL Code generated by Model Interpreter

```
-- This VHDL Code is automatically generated
-- by the model-to-VHDL interpreter that uses
-- vMAGIC Library

library IEEE;
use ieee.std_logic_1164.all;
use IEEE.std_Logic_arith.all;
use IEEE.std_Logic_unsigned.all;
use work.GenSATPack.all;
entity GenSATMain is
    port (
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        D : inout std_logic_vector(31 downto 0);
        E : inout std_logic_vector(31 downto 0);
        F : inout std_logic_vector(31 downto 0);
        GIN : in std_logic_vector(1 downto 0);
        GOUT : out std_logic_vector(1 downto 0);
        nOED : out std_logic;
        nOEE : out std_logic;
        nOEF : out std_logic;
        nOEG : out std_logic;
        SELD : out std_logic;
        SELE : out std_logic;
        SELF : out std_logic;
        SELG : out std_logic;
        IDD : out std_logic_vector(2 downto 0);
        IDE : out std_logic_vector(2 downto 0);
        IDF : out std_logic_vector(2 downto 0);
        PULSE : in std_logic_vector(3 downto 0);
        nSTART : out std_logic_vector(1 downto 0);
        START : out std_logic_vector(1 downto 0);
        DDLY : inout std_logic_vector(7 downto 0);
        WR_DLY0 : out std_logic;
        WR_DLY1 : out std_logic;
        DIRDDLY : out std_logic;
        nOEDDLY0 : out std_logic;
        nOEDDLY1 : out std_logic;
        nLEDG : out std_logic;
        nLEDR : out std_logic;
        SPARE : inout std_logic_vector(11 downto 0);
        nlBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0)
    );
end entity GenSATMain;
architecture rtl of GenSATMain is
    signal PCLK : std_logic := '1';
    signal HOLD : std_logic := '0';
```

```
signal WR_DLY_CMD : std_logic_vector(1 downto 0) := (others=>'0');
signal Data_792f : std_logic_vector(31 downto 0);
signal Strobe_792f : std_logic;
signal REG_OUT_792f : std_logic_vector(31 downto 0);
signal SPLITBLK_6863 : std_logic;
signal Signal_I_7a5d : std_logic_vector(31 downto 0);
signal Signal_O_7a5d : std_logic_vector(31 downto 0);
signal STRETCH_LEN_7a5d : INTEGER;
signal X_75d1 : std_logic_vector(31 downto 0);
signal ORG_Out_75d1 : std_logic;
signal SPLITBLK_685c : std_logic;
signal SPLITBLK_6855 : std_logic;
signal SPLITBLK_684e : std_logic;
signal SPLITBLK_6847 : std_logic;
signal SPLITBLK_6840 : std_logic;
signal SPLITBLK_6839 : std_logic;
signal SPLITBLK_6832 : std_logic;
signal SPLITBLK_682b : std_logic;
signal SPLITBLK_6824 : std_logic;
signal SPLITBLK_681d : std_logic;
signal SPLITBLK_6816 : std_logic;
signal SPLITBLK_680f : std_logic;
signal SPLITBLK_6808 : std_logic;
signal SPLITBLK_6800 : std_logic;
signal SPLITBLK_67f9 : std_logic;
signal SPLITBLK_67f2 : std_logic;
signal SPLITBLK_67eb : std_logic;
signal SPLITBLK_67e3 : std_logic;
signal SPLITBLK_67dc : std_logic;
signal SPLITBLK_67d4 : std_logic;
signal SPLITBLK_67cd : std_logic;
signal SPLITBLK_67c6 : std_logic;
signal SPLITBLK_67bf : std_logic;
signal SPLITBLK_6616 : std_logic;
signal Signal_I_6feb : std_logic_vector(31 downto 0);
signal Signal_O_6feb : std_logic_vector(31 downto 0);
signal STRETCH_LEN_6feb : INTEGER;
signal Signal_I_72c4 : std_logic_vector(31 downto 0);
signal Signal_O_72c4 : std_logic_vector(31 downto 0);
signal STRETCH_LEN_72c4 : INTEGER;
signal SPLITBLK_660f : std_logic;
signal SPLITBLK_6608 : std_logic;
signal SPLITBLK_6601 : std_logic;
signal SPLITBLK_65fa : std_logic;
signal SPLITBLK_65f3 : std_logic;
signal SPLITBLK_65ec : std_logic;
signal SPLITBLK_65e5 : std_logic;
signal SPLITBLK_65de : std_logic;
signal SPLITBLK_65d7 : std_logic;
signal SPLITBLK_65d0 : std_logic;
signal SPLITBLK_65c9 : std_logic;
signal SPLITBLK_65c2 : std_logic;
signal SPLITBLK_65bb : std_logic;
signal SPLITBLK_65b4 : std_logic;
signal SPLITBLK_65ad : std_logic;
```

```
signal SPLITBLK_65a6 : std_logic;
signal SPLITBLK_659e : std_logic;
signal SPLITBLK_6597 : std_logic;
signal SPLITBLK_6590 : std_logic;
signal SPLITBLK_6589 : std_logic;
signal SPLITBLK_6582 : std_logic;
signal SPLITBLK_657b : std_logic;
signal SPLITBLK_6574 : std_logic;
signal DSignal_I_64eb : std_logic_vector(31 downto 0);
signal DSignal_O_64eb : std_logic_vector(31 downto 0);
signal Dusedw_64eb : std_logic_vector(11 downto 0);
signal DELAY_OUT_64eb : std_logic;
signal DELAY_64eb : INTEGER;
signal Drdreq_64eb : std_logic;
signal Dwrreq_64eb : std_logic;
signal Dgbpr_clr_64eb : std_logic;
signal DisEmpty_64eb : std_logic;
signal DisFull_64eb : std_logic;
signal Signal_I_6147 : std_logic_vector(31 downto 0);
signal Signal_O_6147 : std_logic_vector(31 downto 0);
signal STRETCH_LEN_6147 : INTEGER;
signal DSignal_I_2fed : std_logic_vector(31 downto 0);
signal DSignal_O_2fed : std_logic_vector(31 downto 0);
signal Dusedw_2fed : std_logic_vector(11 downto 0);
signal DELAY_OUT_2fed : std_logic;
signal DELAY_2fed : INTEGER;
signal Drdreq_2fed : std_logic;
signal Dwrreq_2fed : std_logic;
signal Dgbpr_clr_2fed : std_logic;
signal DisEmpty_2fed : std_logic;
signal DisFull_2fed : std_logic;
signal SPLITBLK_28a6 : std_logic;
signal X_37c2 : std_logic_vector(31 downto 0);
signal ORG_Out_37c2 : std_logic;
signal Signal_I_3bb4 : std_logic_vector(31 downto 0);
signal Signal_O_3bb4 : std_logic_vector(31 downto 0);
signal STRETCH_LEN_3bb4 : INTEGER;
signal SPLITBLK_289f : std_logic;
signal SPLITBLK_2897 : std_logic;
signal SPLITBLK_2890 : std_logic;
signal SPLITBLK_2889 : std_logic;
signal SPLITBLK_2882 : std_logic;
signal SPLITBLK_287b : std_logic;
signal SPLITBLK_2874 : std_logic;
signal SPLITBLK_286d : std_logic;
signal SPLITBLK_2866 : std_logic;
signal SPLITBLK_285f : std_logic;
signal SPLITBLK_2858 : std_logic;
signal SPLITBLK_2851 : std_logic;
signal SPLITBLK_284a : std_logic;
signal SPLITBLK_2843 : std_logic;
signal SPLITBLK_283c : std_logic;
signal SPLITBLK_2835 : std_logic;
signal SPLITBLK_282e : std_logic;
signal SPLITBLK_2827 : std_logic;
```

```
signal SPLITBLK_2820 : std_logic;
signal SPLITBLK_2819 : std_logic;
signal SPLITBLK_2812 : std_logic;
signal SPLITBLK_280b : std_logic;
signal SPLITBLK_2804 : std_logic;
signal SPLITBLK_27fd : std_logic;
signal SPLITBLK_27f6 : std_logic;
signal SPLITBLK_27ef : std_logic;
signal SPLITBLK_27e8 : std_logic;
signal SPLITBLK_27e1 : std_logic;
signal SPLITBLK_27da : std_logic;
signal SPLITBLK_27d3 : std_logic;
signal SPLITBLK_27cc : std_logic;
signal Data_369c : std_logic_vector(31 downto 0);
signal Strobe_369c : std_logic;
signal REG_OUT_369c : std_logic_vector(31 downto 0);
signal SPLITBLK_27a5 : std_logic;
signal Signal_I_33bc : std_logic_vector(31 downto 0);
signal Signal_O_33bc : std_logic_vector(31 downto 0);
signal STRETCH_LEN_33bc : INTEGER;
signal Signal_I_3236 : std_logic_vector(31 downto 0);
signal Signal_O_3236 : std_logic_vector(31 downto 0);
signal STRETCH_LEN_3236 : INTEGER;
signal SPLITBLK_279e : std_logic;
signal SPLITBLK_2797 : std_logic;
signal SPLITBLK_2790 : std_logic;
signal SPLITBLK_2789 : std_logic;
signal SPLITBLK_2782 : std_logic;
signal SPLITBLK_277b : std_logic;
signal SPLITBLK_2774 : std_logic;
signal Signal_I_299a : std_logic_vector(31 downto 0);
signal Signal_O_299a : std_logic_vector(31 downto 0);
signal STRETCH_LEN_299a : INTEGER;
signal DSignal_I_6131 : std_logic_vector(31 downto 0);
signal DSignal_O_6131 : std_logic_vector(31 downto 0);
signal Dusedw_6131 : std_logic_vector(11 downto 0);
signal DELAY_OUT_6131 : std_logic;
signal DELAY_6131 : INTEGER;
signal Drdreq_6131 : std_logic;
signal Dwrreq_6131 : std_logic;
signal Dgbpr_clr_6131 : std_logic;
signal DisEmpty_6131 : std_logic;
signal DisFull_6131 : std_logic;
signal BITS_1e61 : std_logic_vector(63 downto 0);
signal COIN_SW_1e61 : std_logic_vector(63 downto 0);
signal BIT_OUT_1e61 : std_logic := '0';
signal DSignal_I_1cb3 : std_logic_vector(0 downto 0);
signal DSignal_O_1cb3 : std_logic_vector(0 downto 0);
signal Dusedw_1cb3 : std_logic_vector(11 downto 0);
signal DELAY_OUT_1cb3 : std_logic;
signal DELAY_1cb3 : INTEGER;
signal Drdreq_1cb3 : std_logic;
signal Dwrreq_1cb3 : std_logic;
signal Dgbpr_clr_1cb3 : std_logic;
signal DisEmpty_1cb3 : std_logic;
```

```
signal DisFull_1cb3 : std_logic;
signal HOLD_1cb3 : std_logic := '0';
signal Signal_O_1cb3 : std_logic := '0';
signal STRETCH_LEN_1cb3 : INTEGER := 0;
signal BITS_59e0 : std_logic_vector(63 downto 0);
signal COIN_SW_59e0 : std_logic_vector(63 downto 0);
signal BIT_OUT_59e0 : std_logic := '0';
signal SPLITBLK_5a76 : std_logic;
signal X_6213 : std_logic_vector(31 downto 0);
signal ORG_Out_6213 : std_logic;
signal Signal_I_6092 : std_logic_vector(31 downto 0);
signal Signal_O_6092 : std_logic_vector(31 downto 0);
signal STRETCH_LEN_6092 : INTEGER;
signal SPLITBLK_5a6f : std_logic;
signal SPLITBLK_5a68 : std_logic;
signal SPLITBLK_5a61 : std_logic;
signal SPLITBLK_5a5a : std_logic;
signal SPLITBLK_5a53 : std_logic;
signal SPLITBLK_5a4c : std_logic;
signal SPLITBLK_5a45 : std_logic;
signal SPLITBLK_5a3e : std_logic;
signal SPLITBLK_5a37 : std_logic;
signal SPLITBLK_5a30 : std_logic;
signal SPLITBLK_5a29 : std_logic;
signal SPLITBLK_5a22 : std_logic;
signal SPLITBLK_5a1b : std_logic;
signal SPLITBLK_5a14 : std_logic;
signal SPLITBLK_5a0d : std_logic;
signal SPLITBLK_5a06 : std_logic;
signal SPLITBLK_59ff : std_logic;
signal SPLITBLK_59f8 : std_logic;
signal SPLITBLK_59f1 : std_logic;
signal SPLITBLK_59ea : std_logic;
signal SPLITBLK_59df : std_logic;
signal SPLITBLK_59d8 : std_logic;
signal SPLITBLK_59d1 : std_logic;
signal SPLITBLK_59ca : std_logic;
signal SPLITBLK_5900 : std_logic;
signal SPLITBLK_58f9 : std_logic;
signal SPLITBLK_58f2 : std_logic;
signal HOLD_5819 : std_logic := '0';
signal Signal_I_5819 : std_logic := '0';
signal Signal_O_5819 : std_logic := '0';
signal STRETCH_LEN_5819 : INTEGER := 0;
signal SPLITBLK_570c : std_logic;
signal DSignal_I_56a6 : std_logic_vector(0 downto 0);
signal DSignal_O_56a6 : std_logic_vector(0 downto 0);
signal Dusedw_56a6 : std_logic_vector(11 downto 0);
signal DELAY_OUT_56a6 : std_logic;
signal DELAY_56a6 : INTEGER;
signal Drdreq_56a6 : std_logic;
signal Dwrreq_56a6 : std_logic;
signal Dgbpr_clr_56a6 : std_logic;
signal DisEmpty_56a6 : std_logic;
signal DisFull_56a6 : std_logic;
```

```
signal HOLD_571d : std_logic := '0';
signal Signal_I_571d : std_logic := '0';
signal Signal_O_571d : std_logic := '0';
signal STRETCH_LEN_571d : INTEGER := 0;
signal HOLD_56f0 : std_logic := '0';
signal Signal_I_56f0 : std_logic := '0';
signal Signal_O_56f0 : std_logic := '0';
signal STRETCH_LEN_56f0 : INTEGER := 0;
signal X_5832 : std_logic;
signal Y_5832 : std_logic;
signal ANDG_Out_5832 : std_logic;
signal HOLD_56d1 : std_logic := '0';
signal Signal_I_56d1 : std_logic := '0';
signal Signal_O_56d1 : std_logic := '0';
signal STRETCH_LEN_56d1 : INTEGER := 0;
signal STARTTAC_5713 : std_logic;
signal STOPTAC_5713 : std_logic;
signal SECFLAG_5713 : std_logic;
signal INDEX_5713 : INTEGER;
signal COUNT_5713 : INTEGER;
signal NUMRESETS_5713 : std_logic_vector(31 downto 0);
signal PLLREG_5713 : std_logic_vector(31 downto 0);
signal STARTTAC_2911 : std_logic;
signal STOPTAC_2911 : std_logic;
signal SECFLAG_2911 : std_logic;
signal INDEX_2911 : INTEGER;
signal COUNT_2911 : INTEGER;
signal NUMRESETS_2911 : std_logic_vector(31 downto 0);
signal PLLREG_2911 : std_logic_vector(31 downto 0);
signal SPLITBLK_56ba : std_logic;
signal STARTTAC_7ce1 : std_logic;
signal STOPTAC_7ce1 : std_logic;
signal SECFLAG_7ce1 : std_logic;
signal INDEX_7ce1 : INTEGER;
signal COUNT_7ce1 : INTEGER;
signal NUMRESETS_7ce1 : std_logic_vector(31 downto 0);
signal PLLREG_7ce1 : std_logic_vector(31 downto 0);
signal SPLITBLK_56aa : std_logic;
signal SPLITBLK_74aa : std_logic;
signal HOLD_56ce : std_logic := '0';
signal Signal_I_56ce : std_logic := '0';
signal Signal_O_56ce : std_logic := '0';
signal STRETCH_LEN_56ce : INTEGER := 0;
signal FANOUTInput_75dd : std_logic;
signal FANOUT_75dd : std_logic_vector(7 downto 0);
signal DSignal_I_1cdf : std_logic_vector(0 downto 0);
signal DSignal_O_1cdf : std_logic_vector(0 downto 0);
signal Dusedw_1cdf : std_logic_vector(11 downto 0);
signal DELAY_OUT_1cdf : std_logic;
signal DELAY_1cdf : INTEGER;
signal Drdreq_1cdf : std_logic;
signal Dwrreq_1cdf : std_logic;
signal Dgbpr_clr_1cdf : std_logic;
signal DisEmpty_1cdf : std_logic;
signal DisFull_1cdf : std_logic;
```

```
signal HOLD_1cdf : std_logic := '0';
signal Signal_O_1cdf : std_logic := '0';
signal STRETCH_LEN_1cdf : INTEGER := 0;
signal Data_7c29 : std_logic;
signal EventTrig_7c29 : std_logic;
signal Data_760d : std_logic_vector(31 downto 0);
signal Strobe_760d : std_logic;
signal REG_OUT_760d : std_logic_vector(31 downto 0);
signal SPLITBLK_7624 : std_logic;
signal DSignal_I_5809 : std_logic_vector(0 downto 0);
signal DSignal_O_5809 : std_logic_vector(0 downto 0);
signal Dusedw_5809 : std_logic_vector(11 downto 0);
signal DELAY_OUT_5809 : std_logic;
signal DELAY_5809 : INTEGER;
signal Drdreq_5809 : std_logic;
signal Dwrreq_5809 : std_logic;
signal Dgbpr_clr_5809 : std_logic;
signal DisEmpty_5809 : std_logic;
signal DisFull_5809 : std_logic;
signal REG_R1 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R2 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R3 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R4 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R5 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R6 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R7 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R8 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_R9 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW1 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW2 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW3 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW4 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW5 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW6 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW7 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW8 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW9 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW10 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW11 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW12 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW13 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW14 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW15 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW16 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW17 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW18 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW19 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW20 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW21 : std_logic_vector(31 downto 0) := (others=>'Z');
signal REG_RW22 : std_logic_vector(31 downto 0) := (others=>'Z');
signal RELOAD_REG_RW2 : std_logic := '0';
signal RELOAD_REG_RW3 : std_logic := '0';
signal RELOAD_REG_RW4 : std_logic := '0';
signal RELOAD_REG_RW5 : std_logic := '0';
signal RELOAD_REG_RW6 : std_logic := '0';
```

```
signal RELOAD_REG_RW7 : std_logic := '0';
signal RELOAD_REG_RW8 : std_logic := '0';
signal RELOAD_REG_RW9 : std_logic := '0';
signal RELOAD_REG_RW10 : std_logic := '0';
signal RELOAD_REG_RW11 : std_logic := '0';
signal RELOAD_REG_RW12 : std_logic := '0';
signal RELOAD_REG_RW13 : std_logic := '0';
signal RELOAD_REG_RW14 : std_logic := '0';
signal RELOAD_REG_RW15 : std_logic := '0';
signal RELOAD_REG_RW16 : std_logic := '0';
signal RELOAD_REG_RW17 : std_logic := '0';
signal RELOAD_REG_RW18 : std_logic := '0';
signal RELOAD_REG_RW19 : std_logic := '0';
signal RELOAD_REG_RW20 : std_logic := '0';
signal RELOAD_REG_RW21 : std_logic := '0';
signal RELOAD_REG_RW22 : std_logic := '0';
component GenSATLocalRegisterAccess
    port (
        nLBRES : in std_logic;
        nBLAST : in std_logic;
        WnR : in std_logic;
        nADS : in std_logic;
        LCLK : in std_logic;
        nREADY : out std_logic;
        nINT : out std_logic;
        LAD : inout std_logic_vector(15 downto 0);
        REG_R1 : in std_logic_vector(31 downto 0);
        REG_R2 : in std_logic_vector(31 downto 0);
        REG_R3 : in std_logic_vector(31 downto 0);
        REG_R4 : in std_logic_vector(31 downto 0);
        REG_R5 : in std_logic_vector(31 downto 0);
        REG_R6 : in std_logic_vector(31 downto 0);
        REG_R7 : in std_logic_vector(31 downto 0);
        REG_R8 : in std_logic_vector(31 downto 0);
        REG_R9 : in std_logic_vector(31 downto 0);
        REG_RW1 : buffer std_logic_vector(31 downto 0);
        REG_RW2 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW2 : out std_logic;
        REG_RW3 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW3 : out std_logic;
        REG_RW4 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW4 : out std_logic;
        REG_RW5 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW5 : out std_logic;
        REG_RW6 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW6 : out std_logic;
        REG_RW7 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW7 : out std_logic;
        REG_RW8 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW8 : out std_logic;
        REG_RW9 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW9 : out std_logic;
        REG_RW10 : buffer std_logic_vector(31 downto 0);
        RELOAD_REG_RW10 : out std_logic;
        REG_RW11 : buffer std_logic_vector(31 downto 0);
```

```
            RELOAD_REG_RW11 : out std_logic;
            REG_RW12 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW12 : out std_logic;
            REG_RW13 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW13 : out std_logic;
            REG_RW14 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW14 : out std_logic;
            REG_RW15 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW15 : out std_logic;
            REG_RW16 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW16 : out std_logic;
            REG_RW17 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW17 : out std_logic;
            REG_RW18 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW18 : out std_logic;
            REG_RW19 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW19 : out std_logic;
            REG_RW20 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW20 : out std_logic;
            REG_RW21 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW21 : out std_logic;
            REG_RW22 : buffer std_logic_vector(31 downto 0);
            RELOAD_REG_RW22 : out std_logic
        );
    end component GenSATLocalRegisterAccess;
begin
    SELE <= '0';
    IDD <= "000";
    SELF <= '0';
    SELG <= '0';
    nOEE <= '1';
    nOED <= '1';
    nOEF <= '0';
    nOEG <= '0';
    instance_pll : PLL
        port map (
            inclk0   => LCLK,
c0 => PCLK
        );
    Data_792f(0) <= Signal_O_7a5d(0);
    Data_792f(1) <= Signal_O_7a5d(1);
    Data_792f(2) <= Signal_O_7a5d(2);
    Data_792f(3) <= Signal_O_7a5d(3);
    Data_792f(4) <= Signal_O_7a5d(4);
    Data_792f(5) <= Signal_O_7a5d(5);
    Data_792f(6) <= Signal_O_7a5d(6);
    Data_792f(7) <= Signal_O_7a5d(7);
    Data_792f(8) <= Signal_O_7a5d(8);
    Data_792f(9) <= Signal_O_7a5d(9);
    Data_792f(10) <= Signal_O_7a5d(10);
    Data_792f(11) <= Signal_O_7a5d(11);
    Data_792f(12) <= Signal_O_7a5d(12);
    Data_792f(13) <= Signal_O_7a5d(13);
    Data_792f(14) <= Signal_O_7a5d(14);
    Data_792f(15) <= Signal_O_7a5d(15);
```

```
Data_792f(16) <= Signal_O_7a5d(16);
Data_792f(17) <= Signal_O_7a5d(17);
Data_792f(18) <= Signal_O_7a5d(18);
Data_792f(19) <= Signal_O_7a5d(19);
Data_792f(20) <= Signal_O_7a5d(20);
Data_792f(21) <= Signal_O_7a5d(21);
Data_792f(22) <= Signal_O_7a5d(22);
Data_792f(23) <= Signal_O_7a5d(23);
Data_792f(24) <= '0';
Data_792f(25) <= '0';
Data_792f(26) <= '0';
Data_792f(27) <= '0';
Data_792f(28) <= '0';
Data_792f(29) <= '0';
Data_792f(30) <= '0';
Data_792f(31) <= '0';
Strobe_792f <= FANOUT_75dd(0);
REG_R1 <= REG_OUT_792f;
SPLITBLK_6863 <= A(0);
Signal_I_7a5d(0) <= SPLITBLK_6863;
Signal_I_7a5d(1) <= SPLITBLK_685c;
Signal_I_7a5d(2) <= SPLITBLK_6855;
Signal_I_7a5d(3) <= SPLITBLK_684e;
Signal_I_7a5d(4) <= SPLITBLK_6847;
Signal_I_7a5d(5) <= SPLITBLK_6840;
Signal_I_7a5d(6) <= SPLITBLK_6839;
Signal_I_7a5d(7) <= SPLITBLK_6832;
Signal_I_7a5d(8) <= SPLITBLK_682b;
Signal_I_7a5d(9) <= SPLITBLK_6824;
Signal_I_7a5d(10) <= SPLITBLK_681d;
Signal_I_7a5d(11) <= SPLITBLK_6816;
Signal_I_7a5d(12) <= SPLITBLK_680f;
Signal_I_7a5d(13) <= SPLITBLK_6808;
Signal_I_7a5d(14) <= SPLITBLK_6800;
Signal_I_7a5d(15) <= SPLITBLK_67f9;
Signal_I_7a5d(16) <= SPLITBLK_67f2;
Signal_I_7a5d(17) <= SPLITBLK_67eb;
Signal_I_7a5d(18) <= SPLITBLK_67e3;
Signal_I_7a5d(19) <= SPLITBLK_67dc;
Signal_I_7a5d(20) <= SPLITBLK_67d4;
Signal_I_7a5d(21) <= SPLITBLK_67cd;
Signal_I_7a5d(22) <= SPLITBLK_67c6;
Signal_I_7a5d(23) <= SPLITBLK_67bf;
Signal_I_7a5d(31 downto 24) <= (others=>'0');
STRETCH_LEN_7a5d <= conv_integer(unsigned(REG_RW2(8 downto 0)));
X_75d1(0) <= SPLITBLK_6863;
X_75d1(1) <= SPLITBLK_685c;
X_75d1(2) <= SPLITBLK_6855;
X_75d1(3) <= SPLITBLK_684e;
X_75d1(4) <= SPLITBLK_6847;
X_75d1(5) <= SPLITBLK_6840;
X_75d1(6) <= SPLITBLK_6839;
X_75d1(7) <= SPLITBLK_6832;
X_75d1(8) <= SPLITBLK_682b;
X_75d1(9) <= SPLITBLK_6824;
```

```
X_75d1(10) <= SPLITBLK_681d;
X_75d1(11) <= SPLITBLK_6816;
X_75d1(12) <= SPLITBLK_680f;
X_75d1(13) <= SPLITBLK_6808;
X_75d1(14) <= SPLITBLK_6800;
X_75d1(15) <= SPLITBLK_67f9;
X_75d1(16) <= SPLITBLK_67f2;
X_75d1(17) <= SPLITBLK_67eb;
X_75d1(18) <= SPLITBLK_67e3;
X_75d1(19) <= SPLITBLK_67dc;
X_75d1(20) <= SPLITBLK_67d4;
X_75d1(21) <= SPLITBLK_67cd;
X_75d1(22) <= SPLITBLK_67c6;
X_75d1(23) <= SPLITBLK_67bf;
SPLITBLK_685c <= A(1);
SPLITBLK_6855 <= A(2);
SPLITBLK_684e <= A(3);
SPLITBLK_6847 <= A(4);
SPLITBLK_6840 <= A(5);
SPLITBLK_6839 <= A(6);
SPLITBLK_6832 <= A(7);
SPLITBLK_682b <= A(8);
SPLITBLK_6824 <= A(9);
SPLITBLK_681d <= A(10);
SPLITBLK_6816 <= A(11);
SPLITBLK_680f <= A(12);
SPLITBLK_6808 <= A(13);
SPLITBLK_6800 <= A(14);
SPLITBLK_67f9 <= A(15);
SPLITBLK_67f2 <= A(16);
SPLITBLK_67eb <= A(17);
SPLITBLK_67e3 <= A(18);
SPLITBLK_67dc <= A(19);
SPLITBLK_67d4 <= A(20);
SPLITBLK_67cd <= A(21);
SPLITBLK_67c6 <= A(22);
SPLITBLK_67bf <= A(23);
SPLITBLK_6616 <= ORG_Out_75d1;
Signal_I_6feb(0) <= SPLITBLK_6616;
Signal_I_6feb(1) <= SPLITBLK_660f;
Signal_I_6feb(2) <= SPLITBLK_6608;
Signal_I_6feb(3) <= SPLITBLK_6601;
Signal_I_6feb(4) <= SPLITBLK_65fa;
Signal_I_6feb(5) <= SPLITBLK_65f3;
Signal_I_6feb(6) <= SPLITBLK_65ec;
Signal_I_6feb(7) <= SPLITBLK_65e5;
Signal_I_6feb(8) <= SPLITBLK_65de;
Signal_I_6feb(9) <= SPLITBLK_65d7;
Signal_I_6feb(10) <= SPLITBLK_65d0;
Signal_I_6feb(11) <= SPLITBLK_65c9;
Signal_I_6feb(12) <= SPLITBLK_65c2;
Signal_I_6feb(13) <= SPLITBLK_65bb;
Signal_I_6feb(14) <= SPLITBLK_65b4;
Signal_I_6feb(15) <= SPLITBLK_65ad;
Signal_I_6feb(16) <= SPLITBLK_65a6;
```

```
Signal_I_6feb(17) <= SPLITBLK_659e;
Signal_I_6feb(18) <= SPLITBLK_6597;
Signal_I_6feb(19) <= SPLITBLK_6590;
Signal_I_6feb(20) <= SPLITBLK_6589;
Signal_I_6feb(21) <= SPLITBLK_6582;
Signal_I_6feb(22) <= SPLITBLK_657b;
Signal_I_6feb(23) <= SPLITBLK_6574;
Signal_I_6feb(31 downto 24) <= (others=>'0');
STRETCH_LEN_6feb <= conv_integer(unsigned(REG_RW3(8 downto 0)));
Signal_I_72c4(0) <= SPLITBLK_6616;
Signal_I_72c4(1) <= SPLITBLK_660f;
Signal_I_72c4(2) <= SPLITBLK_6608;
Signal_I_72c4(3) <= SPLITBLK_6601;
Signal_I_72c4(4) <= SPLITBLK_65fa;
Signal_I_72c4(5) <= SPLITBLK_65f3;
Signal_I_72c4(6) <= SPLITBLK_65ec;
Signal_I_72c4(7) <= SPLITBLK_65e5;
Signal_I_72c4(8) <= SPLITBLK_65de;
Signal_I_72c4(9) <= SPLITBLK_65d7;
Signal_I_72c4(10) <= SPLITBLK_65d0;
Signal_I_72c4(11) <= SPLITBLK_65c9;
Signal_I_72c4(12) <= SPLITBLK_65c2;
Signal_I_72c4(13) <= SPLITBLK_65bb;
Signal_I_72c4(14) <= SPLITBLK_65b4;
Signal_I_72c4(15) <= SPLITBLK_65ad;
Signal_I_72c4(16) <= SPLITBLK_65a6;
Signal_I_72c4(17) <= SPLITBLK_659e;
Signal_I_72c4(18) <= SPLITBLK_6597;
Signal_I_72c4(19) <= SPLITBLK_6590;
Signal_I_72c4(20) <= SPLITBLK_6589;
Signal_I_72c4(21) <= SPLITBLK_6582;
Signal_I_72c4(22) <= SPLITBLK_657b;
Signal_I_72c4(23) <= SPLITBLK_6574;
Signal_I_72c4(31 downto 24) <= (others=>'0');
STRETCH_LEN_72c4 <= conv_integer(unsigned(REG_RW4(8 downto 0)));
SPLITBLK_660f <= ORG_Out_75d1;
SPLITBLK_6608 <= ORG_Out_75d1;
SPLITBLK_6601 <= ORG_Out_75d1;
SPLITBLK_65fa <= ORG_Out_75d1;
SPLITBLK_65f3 <= ORG_Out_75d1;
SPLITBLK_65ec <= ORG_Out_75d1;
SPLITBLK_65e5 <= ORG_Out_75d1;
SPLITBLK_65de <= ORG_Out_75d1;
SPLITBLK_65d7 <= ORG_Out_75d1;
SPLITBLK_65d0 <= ORG_Out_75d1;
SPLITBLK_65c9 <= ORG_Out_75d1;
SPLITBLK_65c2 <= ORG_Out_75d1;
SPLITBLK_65bb <= ORG_Out_75d1;
SPLITBLK_65b4 <= ORG_Out_75d1;
SPLITBLK_65ad <= ORG_Out_75d1;
SPLITBLK_65a6 <= ORG_Out_75d1;
SPLITBLK_659e <= ORG_Out_75d1;
SPLITBLK_6597 <= ORG_Out_75d1;
SPLITBLK_6590 <= ORG_Out_75d1;
SPLITBLK_6589 <= ORG_Out_75d1;
```

```
SPLITBLK_6582 <= ORG_Out_75d1;
SPLITBLK_657b <= ORG_Out_75d1;
SPLITBLK_6574 <= ORG_Out_75d1;
DSignal_I_64eb(0) <= Signal_O_72c4(0);
DSignal_I_64eb(1) <= Signal_O_72c4(1);
DSignal_I_64eb(2) <= Signal_O_72c4(2);
DSignal_I_64eb(3) <= Signal_O_72c4(3);
DSignal_I_64eb(4) <= Signal_O_72c4(4);
DSignal_I_64eb(5) <= Signal_O_72c4(5);
DSignal_I_64eb(6) <= Signal_O_72c4(6);
DSignal_I_64eb(7) <= Signal_O_72c4(7);
DSignal_I_64eb(8) <= Signal_O_72c4(8);
DSignal_I_64eb(9) <= Signal_O_72c4(9);
DSignal_I_64eb(10) <= Signal_O_72c4(10);
DSignal_I_64eb(11) <= Signal_O_72c4(11);
DSignal_I_64eb(12) <= Signal_O_72c4(12);
DSignal_I_64eb(13) <= Signal_O_72c4(13);
DSignal_I_64eb(14) <= Signal_O_72c4(14);
DSignal_I_64eb(15) <= Signal_O_72c4(15);
DSignal_I_64eb(16) <= Signal_O_72c4(16);
DSignal_I_64eb(17) <= Signal_O_72c4(17);
DSignal_I_64eb(18) <= Signal_O_72c4(18);
DSignal_I_64eb(19) <= Signal_O_72c4(19);
DSignal_I_64eb(20) <= Signal_O_72c4(20);
DSignal_I_64eb(21) <= Signal_O_72c4(21);
DSignal_I_64eb(22) <= Signal_O_72c4(22);
DSignal_I_64eb(23) <= Signal_O_72c4(23);
DELAY_64eb <= conv_integer(unsigned(REG_RW5(8 downto 0)));
Dwrreq_64eb <= '1';
Dgbpr_clr_64eb <= RELOAD_REG_RW5;
Signal_I_6147(0) <= Signal_O_6feb(0);
Signal_I_6147(1) <= Signal_O_6feb(1);
Signal_I_6147(2) <= Signal_O_6feb(2);
Signal_I_6147(3) <= Signal_O_6feb(3);
Signal_I_6147(4) <= Signal_O_6feb(4);
Signal_I_6147(5) <= Signal_O_6feb(5);
Signal_I_6147(6) <= Signal_O_6feb(6);
Signal_I_6147(7) <= Signal_O_6feb(7);
Signal_I_6147(8) <= Signal_O_6feb(8);
Signal_I_6147(9) <= Signal_O_6feb(9);
Signal_I_6147(10) <= Signal_O_6feb(10);
Signal_I_6147(11) <= Signal_O_6feb(11);
Signal_I_6147(12) <= Signal_O_6feb(12);
Signal_I_6147(13) <= Signal_O_6feb(13);
Signal_I_6147(14) <= Signal_O_6feb(14);
Signal_I_6147(15) <= Signal_O_6feb(15);
Signal_I_6147(16) <= Signal_O_6feb(16);
Signal_I_6147(17) <= Signal_O_6feb(17);
Signal_I_6147(18) <= Signal_O_6feb(18);
Signal_I_6147(19) <= Signal_O_6feb(19);
Signal_I_6147(20) <= Signal_O_6feb(20);
Signal_I_6147(21) <= Signal_O_6feb(21);
Signal_I_6147(22) <= Signal_O_6feb(22);
Signal_I_6147(23) <= Signal_O_6feb(23);
Signal_I_6147(31 downto 24) <= (others=>'0');
```

```
STRETCH_LEN_6147 <= conv_integer(unsigned(REG_RW6(8 downto 0)));
DSignal_I_2fed(0) <= Signal_O_33bc(0);
DSignal_I_2fed(1) <= Signal_O_33bc(1);
DSignal_I_2fed(2) <= Signal_O_33bc(2);
DSignal_I_2fed(3) <= Signal_O_33bc(3);
DSignal_I_2fed(4) <= Signal_O_33bc(4);
DSignal_I_2fed(5) <= Signal_O_33bc(5);
DSignal_I_2fed(6) <= Signal_O_33bc(6);
DSignal_I_2fed(7) <= Signal_O_33bc(7);
DELAY_2fed <= conv_integer(unsigned(REG_RW7(8 downto 0)));
Dwrreq_2fed <= '1';
Dgbpr_clr_2fed <= RELOAD_REG_RW7;
SPLITBLK_28a6 <= B(0);
X_37c2(0) <= SPLITBLK_28a6;
X_37c2(1) <= SPLITBLK_289f;
X_37c2(2) <= SPLITBLK_2897;
X_37c2(3) <= SPLITBLK_2890;
X_37c2(4) <= SPLITBLK_2889;
X_37c2(5) <= SPLITBLK_2882;
X_37c2(6) <= SPLITBLK_287b;
X_37c2(7) <= SPLITBLK_2874;
X_37c2(8) <= SPLITBLK_286d;
X_37c2(9) <= SPLITBLK_2866;
X_37c2(10) <= SPLITBLK_285f;
X_37c2(11) <= SPLITBLK_2858;
X_37c2(12) <= SPLITBLK_2851;
X_37c2(13) <= SPLITBLK_284a;
X_37c2(14) <= SPLITBLK_2843;
X_37c2(15) <= SPLITBLK_283c;
X_37c2(16) <= SPLITBLK_2835;
X_37c2(17) <= SPLITBLK_282e;
X_37c2(18) <= SPLITBLK_2827;
X_37c2(19) <= SPLITBLK_2820;
X_37c2(20) <= SPLITBLK_2819;
X_37c2(21) <= SPLITBLK_2812;
X_37c2(22) <= SPLITBLK_280b;
X_37c2(23) <= SPLITBLK_2804;
X_37c2(24) <= SPLITBLK_27fd;
X_37c2(25) <= SPLITBLK_27f6;
X_37c2(26) <= SPLITBLK_27ef;
X_37c2(27) <= SPLITBLK_27e8;
X_37c2(28) <= SPLITBLK_27e1;
X_37c2(29) <= SPLITBLK_27da;
X_37c2(30) <= SPLITBLK_27d3;
X_37c2(31) <= SPLITBLK_27cc;
Signal_I_3bb4(0) <= SPLITBLK_28a6;
Signal_I_3bb4(1) <= SPLITBLK_289f;
Signal_I_3bb4(2) <= SPLITBLK_2897;
Signal_I_3bb4(3) <= SPLITBLK_2890;
Signal_I_3bb4(4) <= SPLITBLK_2889;
Signal_I_3bb4(5) <= SPLITBLK_2882;
Signal_I_3bb4(6) <= SPLITBLK_287b;
Signal_I_3bb4(7) <= SPLITBLK_2874;
Signal_I_3bb4(8) <= SPLITBLK_286d;
Signal_I_3bb4(9) <= SPLITBLK_2866;
```

```
Signal_I_3bb4(10) <= SPLITBLK_285f;
Signal_I_3bb4(11) <= SPLITBLK_2858;
Signal_I_3bb4(12) <= SPLITBLK_2851;
Signal_I_3bb4(13) <= SPLITBLK_284a;
Signal_I_3bb4(14) <= SPLITBLK_2843;
Signal_I_3bb4(15) <= SPLITBLK_283c;
Signal_I_3bb4(16) <= SPLITBLK_2835;
Signal_I_3bb4(17) <= SPLITBLK_282e;
Signal_I_3bb4(18) <= SPLITBLK_2827;
Signal_I_3bb4(19) <= SPLITBLK_2820;
Signal_I_3bb4(20) <= SPLITBLK_2819;
Signal_I_3bb4(21) <= SPLITBLK_2812;
Signal_I_3bb4(22) <= SPLITBLK_280b;
Signal_I_3bb4(23) <= SPLITBLK_2804;
Signal_I_3bb4(24) <= SPLITBLK_27fd;
Signal_I_3bb4(25) <= SPLITBLK_27f6;
Signal_I_3bb4(26) <= SPLITBLK_27ef;
Signal_I_3bb4(27) <= SPLITBLK_27e8;
Signal_I_3bb4(28) <= SPLITBLK_27e1;
Signal_I_3bb4(29) <= SPLITBLK_27da;
Signal_I_3bb4(30) <= SPLITBLK_27d3;
Signal_I_3bb4(31) <= SPLITBLK_27cc;
STRETCH_LEN_3bb4 <= conv_integer(unsigned(REG_RW8(8 downto 0)));
SPLITBLK_289f <= B(1);
SPLITBLK_2897 <= B(2);
SPLITBLK_2890 <= B(3);
SPLITBLK_2889 <= B(4);
SPLITBLK_2882 <= B(5);
SPLITBLK_287b <= B(6);
SPLITBLK_2874 <= B(7);
SPLITBLK_286d <= B(8);
SPLITBLK_2866 <= B(9);
SPLITBLK_285f <= B(10);
SPLITBLK_2858 <= B(11);
SPLITBLK_2851 <= B(12);
SPLITBLK_284a <= B(13);
SPLITBLK_2843 <= B(14);
SPLITBLK_283c <= B(15);
SPLITBLK_2835 <= B(16);
SPLITBLK_282e <= B(17);
SPLITBLK_2827 <= B(18);
SPLITBLK_2820 <= B(19);
SPLITBLK_2819 <= B(20);
SPLITBLK_2812 <= B(21);
SPLITBLK_280b <= B(22);
SPLITBLK_2804 <= B(23);
SPLITBLK_27fd <= B(24);
SPLITBLK_27f6 <= B(25);
SPLITBLK_27ef <= B(26);
SPLITBLK_27e8 <= B(27);
SPLITBLK_27e1 <= B(28);
SPLITBLK_27da <= B(29);
SPLITBLK_27d3 <= B(30);
SPLITBLK_27cc <= B(31);
Data_369c(0) <= Signal_O_3bb4(0);
```

```
Data_369c(1) <= Signal_O_3bb4(1);
Data_369c(2) <= Signal_O_3bb4(2);
Data_369c(3) <= Signal_O_3bb4(3);
Data_369c(4) <= Signal_O_3bb4(4);
Data_369c(5) <= Signal_O_3bb4(5);
Data_369c(6) <= Signal_O_3bb4(6);
Data_369c(7) <= Signal_O_3bb4(7);
Data_369c(8) <= Signal_O_3bb4(8);
Data_369c(9) <= Signal_O_3bb4(9);
Data_369c(10) <= Signal_O_3bb4(10);
Data_369c(11) <= Signal_O_3bb4(11);
Data_369c(12) <= Signal_O_3bb4(12);
Data_369c(13) <= Signal_O_3bb4(13);
Data_369c(14) <= Signal_O_3bb4(14);
Data_369c(15) <= Signal_O_3bb4(15);
Data_369c(16) <= Signal_O_3bb4(16);
Data_369c(17) <= Signal_O_3bb4(17);
Data_369c(18) <= Signal_O_3bb4(18);
Data_369c(19) <= Signal_O_3bb4(19);
Data_369c(20) <= Signal_O_3bb4(20);
Data_369c(21) <= Signal_O_3bb4(21);
Data_369c(22) <= Signal_O_3bb4(22);
Data_369c(23) <= Signal_O_3bb4(23);
Data_369c(24) <= Signal_O_3bb4(24);
Data_369c(25) <= Signal_O_3bb4(25);
Data_369c(26) <= Signal_O_3bb4(26);
Data_369c(27) <= Signal_O_3bb4(27);
Data_369c(28) <= Signal_O_3bb4(28);
Data_369c(29) <= Signal_O_3bb4(29);
Data_369c(30) <= Signal_O_3bb4(30);
Data_369c(31) <= Signal_O_3bb4(31);
Strobe_369c <= FANOUT_75dd(1);
REG_R2 <= REG_OUT_369c;
SPLITBLK_27a5 <= ORG_Out_37c2;
Signal_I_33bc(0) <= SPLITBLK_27a5;
Signal_I_33bc(1) <= SPLITBLK_279e;
Signal_I_33bc(2) <= SPLITBLK_2797;
Signal_I_33bc(3) <= SPLITBLK_2790;
Signal_I_33bc(4) <= SPLITBLK_2789;
Signal_I_33bc(5) <= SPLITBLK_2782;
Signal_I_33bc(6) <= SPLITBLK_277b;
Signal_I_33bc(7) <= SPLITBLK_2774;
Signal_I_33bc(31 downto 8) <= (others=>'0');
STRETCH_LEN_33bc <= conv_integer(unsigned(REG_RW9(8 downto 0)));
Signal_I_3236(0) <= SPLITBLK_27a5;
Signal_I_3236(1) <= SPLITBLK_279e;
Signal_I_3236(2) <= SPLITBLK_2797;
Signal_I_3236(3) <= SPLITBLK_2790;
Signal_I_3236(4) <= SPLITBLK_2789;
Signal_I_3236(5) <= SPLITBLK_2782;
Signal_I_3236(6) <= SPLITBLK_277b;
Signal_I_3236(7) <= SPLITBLK_2774;
Signal_I_3236(31 downto 8) <= (others=>'0');
STRETCH_LEN_3236 <= conv_integer(unsigned(REG_RW10(8 downto 0)));
SPLITBLK_279e <= ORG_Out_37c2;
```

```
SPLITBLK_2797 <= ORG_Out_37c2;
SPLITBLK_2790 <= ORG_Out_37c2;
SPLITBLK_2789 <= ORG_Out_37c2;
SPLITBLK_2782 <= ORG_Out_37c2;
SPLITBLK_277b <= ORG_Out_37c2;
SPLITBLK_2774 <= ORG_Out_37c2;
Signal_I_299a(0) <= Signal_O_3236(0);
Signal_I_299a(1) <= Signal_O_3236(1);
Signal_I_299a(2) <= Signal_O_3236(2);
Signal_I_299a(3) <= Signal_O_3236(3);
Signal_I_299a(4) <= Signal_O_3236(4);
Signal_I_299a(5) <= Signal_O_3236(5);
Signal_I_299a(6) <= Signal_O_3236(6);
Signal_I_299a(7) <= Signal_O_3236(7);
Signal_I_299a(31 downto 8) <= (others=>'0');
STRETCH_LEN_299a <= conv_integer(unsigned(REG_RW11(8 downto 0)));
F(0) <= Signal_O_1cb3;
DSignal_I_6131(0) <= ORG_Out_6213;
DSignal_I_6131(1) <= ORG_Out_6213;
DSignal_I_6131(2) <= ORG_Out_6213;
DSignal_I_6131(3) <= ORG_Out_6213;
DSignal_I_6131(4) <= ORG_Out_6213;
DSignal_I_6131(5) <= ORG_Out_6213;
DSignal_I_6131(6) <= ORG_Out_6213;
DSignal_I_6131(7) <= ORG_Out_6213;
DELAY_6131 <= conv_integer(unsigned(REG_RW12(8 downto 0)));
Dwrreq_6131 <= '1';
Dgbpr_clr_6131 <= RELOAD_REG_RW12;
BITS_1e61(0) <= DSignal_O_2fed(0);
COIN_SW_1e61(0) <= '1';
BITS_1e61(1) <= DSignal_O_2fed(1);
COIN_SW_1e61(1) <= '1';
BITS_1e61(2) <= DSignal_O_2fed(2);
COIN_SW_1e61(2) <= '1';
BITS_1e61(3) <= DSignal_O_2fed(3);
COIN_SW_1e61(3) <= '1';
BITS_1e61(4) <= DSignal_O_2fed(4);
COIN_SW_1e61(4) <= '1';
BITS_1e61(5) <= DSignal_O_2fed(5);
COIN_SW_1e61(5) <= '1';
BITS_1e61(6) <= DSignal_O_2fed(6);
COIN_SW_1e61(6) <= '1';
BITS_1e61(7) <= DSignal_O_2fed(7);
COIN_SW_1e61(7) <= '1';
BITS_1e61(8) <= DSignal_O_64eb(0);
COIN_SW_1e61(8) <= '1';
BITS_1e61(9) <= DSignal_O_64eb(1);
COIN_SW_1e61(9) <= '1';
BITS_1e61(10) <= DSignal_O_64eb(2);
COIN_SW_1e61(10) <= '1';
BITS_1e61(11) <= DSignal_O_64eb(3);
COIN_SW_1e61(11) <= '1';
BITS_1e61(12) <= DSignal_O_64eb(4);
COIN_SW_1e61(12) <= '1';
BITS_1e61(13) <= DSignal_O_64eb(5);
```

```
COIN_SW_1e61(13) <= '1';
BITS_1e61(14) <= DSignal_O_64eb(6);
COIN_SW_1e61(14) <= '1';
BITS_1e61(15) <= DSignal_O_64eb(7);
COIN_SW_1e61(15) <= '1';
BITS_1e61(16) <= DSignal_O_64eb(8);
COIN_SW_1e61(16) <= '1';
BITS_1e61(17) <= DSignal_O_64eb(9);
COIN_SW_1e61(17) <= '1';
BITS_1e61(18) <= DSignal_O_64eb(10);
COIN_SW_1e61(18) <= '1';
BITS_1e61(19) <= DSignal_O_64eb(11);
COIN_SW_1e61(19) <= '1';
BITS_1e61(20) <= DSignal_O_64eb(12);
COIN_SW_1e61(20) <= '1';
BITS_1e61(21) <= DSignal_O_64eb(13);
COIN_SW_1e61(21) <= '1';
BITS_1e61(22) <= DSignal_O_64eb(14);
COIN_SW_1e61(22) <= '1';
BITS_1e61(23) <= DSignal_O_64eb(15);
COIN_SW_1e61(23) <= '1';
BITS_1e61(24) <= DSignal_O_64eb(16);
COIN_SW_1e61(24) <= '1';
BITS_1e61(25) <= DSignal_O_64eb(17);
COIN_SW_1e61(25) <= '1';
BITS_1e61(26) <= DSignal_O_64eb(18);
COIN_SW_1e61(26) <= '1';
BITS_1e61(27) <= DSignal_O_64eb(19);
COIN_SW_1e61(27) <= '1';
BITS_1e61(28) <= DSignal_O_64eb(20);
COIN_SW_1e61(28) <= '1';
BITS_1e61(29) <= DSignal_O_64eb(21);
COIN_SW_1e61(29) <= '1';
BITS_1e61(30) <= DSignal_O_64eb(22);
COIN_SW_1e61(30) <= '1';
BITS_1e61(31) <= DSignal_O_64eb(23);
COIN_SW_1e61(31) <= '1';
BITS_1e61(32) <= DSignal_O_64eb(24);
COIN_SW_1e61(32) <= '1';
BITS_1e61(33) <= DSignal_O_64eb(25);
COIN_SW_1e61(33) <= '1';
BITS_1e61(34) <= DSignal_O_64eb(26);
COIN_SW_1e61(34) <= '1';
BITS_1e61(35) <= DSignal_O_64eb(27);
COIN_SW_1e61(35) <= '1';
BITS_1e61(36) <= DSignal_O_64eb(28);
COIN_SW_1e61(36) <= '1';
BITS_1e61(37) <= DSignal_O_64eb(29);
COIN_SW_1e61(37) <= '1';
BITS_1e61(38) <= DSignal_O_56a6(0);
COIN_SW_1e61(38) <= '1';
COIN_SW_1e61(63 downto 39) <= (others=>'0');
DSignal_I_1cb3(0) <= Signal_O_1cdf;
DELAY_1cb3 <= conv_integer(unsigned(REG_RW13(8 downto 0)));
Dwrreq_1cb3 <= '1';
```

```
Dgbpr_clr_1cb3 <= RELOAD_REG_RW13;
STRETCH_LEN_1cb3 <= conv_integer(unsigned(REG_RW13(8 downto 0)));
BITS_59e0(0) <= NOT(E(2));
COIN_SW_59e0(0) <= '1';
BITS_59e0(1) <= NOT(E(18));
COIN_SW_59e0(1) <= '1';
BITS_59e0(2) <= NOT(E(3));
COIN_SW_59e0(2) <= '1';
BITS_59e0(3) <= NOT(E(19));
COIN_SW_59e0(3) <= '1';
BITS_59e0(4) <= NOT(E(14));
COIN_SW_59e0(4) <= '1';
BITS_59e0(5) <= NOT(E(30));
COIN_SW_59e0(5) <= '1';
BITS_59e0(6) <= NOT(E(15));
COIN_SW_59e0(6) <= '1';
BITS_59e0(7) <= NOT(E(31));
COIN_SW_59e0(7) <= '1';
BITS_59e0(8) <= DSignal_O_6131(0);
COIN_SW_59e0(8) <= '1';
BITS_59e0(9) <= DSignal_O_6131(1);
COIN_SW_59e0(9) <= '1';
BITS_59e0(10) <= DSignal_O_6131(2);
COIN_SW_59e0(10) <= '1';
BITS_59e0(11) <= DSignal_O_6131(3);
COIN_SW_59e0(11) <= '1';
BITS_59e0(12) <= DSignal_O_6131(4);
COIN_SW_59e0(12) <= '1';
BITS_59e0(13) <= DSignal_O_6131(5);
COIN_SW_59e0(13) <= '1';
BITS_59e0(14) <= DSignal_O_6131(6);
COIN_SW_59e0(14) <= '1';
BITS_59e0(15) <= DSignal_O_6131(7);
COIN_SW_59e0(15) <= '1';
COIN_SW_59e0(63 downto 16) <= (others=>'0');
SPLITBLK_5a76 <= D(0);
X_6213(0) <= SPLITBLK_5a76;
X_6213(1) <= SPLITBLK_5a6f;
X_6213(2) <= SPLITBLK_5a68;
X_6213(3) <= SPLITBLK_5a61;
X_6213(4) <= SPLITBLK_5a5a;
X_6213(5) <= SPLITBLK_5a53;
X_6213(6) <= SPLITBLK_5a4c;
X_6213(7) <= SPLITBLK_5a45;
X_6213(8) <= SPLITBLK_5a3e;
X_6213(9) <= SPLITBLK_5a37;
X_6213(10) <= SPLITBLK_5a30;
X_6213(11) <= SPLITBLK_5a29;
X_6213(12) <= SPLITBLK_5a22;
X_6213(13) <= SPLITBLK_5a1b;
X_6213(14) <= SPLITBLK_5a14;
X_6213(15) <= SPLITBLK_5a0d;
X_6213(16) <= SPLITBLK_5a06;
X_6213(17) <= SPLITBLK_59ff;
X_6213(18) <= SPLITBLK_59f8;
```

```
X_6213(19) <= SPLITBLK_59f1;
X_6213(20) <= SPLITBLK_59ea;
X_6213(21) <= SPLITBLK_59df;
X_6213(22) <= SPLITBLK_59d8;
X_6213(23) <= SPLITBLK_59d1;
X_6213(24) <= SPLITBLK_59ca;
X_6213(25) <= SPLITBLK_5900;
X_6213(26) <= SPLITBLK_58f9;
X_6213(27) <= SPLITBLK_58f2;
Signal_I_6092(0) <= SPLITBLK_5a76;
Signal_I_6092(1) <= SPLITBLK_5a6f;
Signal_I_6092(2) <= SPLITBLK_5a68;
Signal_I_6092(3) <= SPLITBLK_5a61;
Signal_I_6092(4) <= SPLITBLK_5a5a;
Signal_I_6092(5) <= SPLITBLK_5a53;
Signal_I_6092(6) <= SPLITBLK_5a4c;
Signal_I_6092(7) <= SPLITBLK_5a45;
Signal_I_6092(8) <= SPLITBLK_5a3e;
Signal_I_6092(9) <= SPLITBLK_5a37;
Signal_I_6092(10) <= SPLITBLK_5a30;
Signal_I_6092(11) <= SPLITBLK_5a29;
Signal_I_6092(12) <= SPLITBLK_5a22;
Signal_I_6092(13) <= SPLITBLK_5a1b;
Signal_I_6092(14) <= SPLITBLK_5a14;
Signal_I_6092(15) <= SPLITBLK_5a0d;
Signal_I_6092(16) <= SPLITBLK_5a06;
Signal_I_6092(17) <= SPLITBLK_59ff;
Signal_I_6092(18) <= SPLITBLK_59f8;
Signal_I_6092(19) <= SPLITBLK_59f1;
Signal_I_6092(20) <= SPLITBLK_59ea;
Signal_I_6092(21) <= SPLITBLK_59df;
Signal_I_6092(22) <= SPLITBLK_59d8;
Signal_I_6092(23) <= SPLITBLK_59d1;
Signal_I_6092(24) <= SPLITBLK_59ca;
Signal_I_6092(25) <= SPLITBLK_5900;
Signal_I_6092(26) <= SPLITBLK_58f9;
Signal_I_6092(27) <= SPLITBLK_58f2;
Signal_I_6092(31 downto 28) <= (others=>'0');
STRETCH_LEN_6092 <= conv_integer(unsigned(REG_RW14(8 downto 0)));
SPLITBLK_5a6f <= D(1);
SPLITBLK_5a68 <= D(2);
SPLITBLK_5a61 <= D(3);
SPLITBLK_5a5a <= D(4);
SPLITBLK_5a53 <= D(5);
SPLITBLK_5a4c <= D(6);
SPLITBLK_5a45 <= D(7);
SPLITBLK_5a3e <= D(8);
SPLITBLK_5a37 <= D(9);
SPLITBLK_5a30 <= D(10);
SPLITBLK_5a29 <= D(11);
SPLITBLK_5a22 <= D(12);
SPLITBLK_5a1b <= D(13);
SPLITBLK_5a14 <= D(14);
SPLITBLK_5a0d <= D(15);
SPLITBLK_5a06 <= D(16);
```

```
SPLITBLK_59ff <= D(17);
SPLITBLK_59f8 <= D(18);
SPLITBLK_59f1 <= D(19);
SPLITBLK_59ea <= D(20);
SPLITBLK_59df <= D(21);
SPLITBLK_59d8 <= D(22);
SPLITBLK_59d1 <= D(23);
SPLITBLK_59ca <= D(24);
SPLITBLK_5900 <= D(25);
SPLITBLK_58f9 <= D(26);
SPLITBLK_58f2 <= D(27);
Signal_I_5819 <= DSignal_O_5809(0);
STRETCH_LEN_5819 <= conv_integer(unsigned(REG_RW15(8 downto 0)));
SPLITBLK_570c <= BIT_OUT_59e0;
DSignal_I_56a6(0) <= SPLITBLK_570c;
DELAY_56a6 <= conv_integer(unsigned(REG_RW16(8 downto 0)));
Dwrreq_56a6 <= '1';
Dgbpr_clr_56a6 <= RELOAD_REG_RW16;
Signal_I_571d <= SPLITBLK_570c;
STRETCH_LEN_571d <= conv_integer(unsigned(REG_RW17(8 downto 0)));
Signal_I_56f0 <= Signal_O_571d;
STRETCH_LEN_56f0 <= conv_integer(unsigned(REG_RW18(8 downto 0)));
X_5832 <= Signal_O_56ce;
Y_5832 <= Signal_O_56d1;
Signal_I_56d1 <= null;
STRETCH_LEN_56d1 <= conv_integer(unsigned(REG_RW19(8 downto 0)));
STARTTAC_5713 <= Signal_O_56f0;
STOPTAC_5713 <= SPLITBLK_56ba;
REG_R3 <= PLLREG_5713;
STARTTAC_2911 <= Signal_O_299a(0);
STOPTAC_2911 <= SPLITBLK_56aa;
REG_R4 <= PLLREG_2911;
SPLITBLK_56ba <= Signal_O_5819;
STARTTAC_7ce1 <= Signal_O_6147(0);
STOPTAC_7ce1 <= SPLITBLK_56aa;
REG_R5 <= PLLREG_7ce1;
SPLITBLK_56aa <= SPLITBLK_56ba;
SPLITBLK_74aa <= BIT_OUT_1e61;
Signal_I_56ce <= SPLITBLK_74aa;
STRETCH_LEN_56ce <= conv_integer(unsigned(REG_RW20(8 downto 0)));
FANOUTInput_75dd <= SPLITBLK_74aa;
DSignal_I_1cdf(0) <= FANOUT_75dd(2);
DELAY_1cdf <= conv_integer(unsigned(REG_RW21(8 downto 0)));
Dwrreq_1cdf <= '1';
Dgbpr_clr_1cdf <= RELOAD_REG_RW21;
STRETCH_LEN_1cdf <= conv_integer(unsigned(REG_RW21(8 downto 0)));
Data_7c29 <= FANOUT_75dd(3);
Data_760d(0) <= Signal_O_6092(0);
Data_760d(1) <= Signal_O_6092(1);
Data_760d(2) <= Signal_O_6092(2);
Data_760d(3) <= Signal_O_6092(3);
Data_760d(4) <= Signal_O_6092(4);
Data_760d(5) <= Signal_O_6092(5);
Data_760d(6) <= Signal_O_6092(6);
Data_760d(7) <= Signal_O_6092(7);
```

```
    Data_760d(8) <= Signal_O_6092(8);
    Data_760d(9) <= Signal_O_6092(9);
    Data_760d(10) <= Signal_O_6092(10);
    Data_760d(11) <= Signal_O_6092(11);
    Data_760d(12) <= Signal_O_6092(12);
    Data_760d(13) <= Signal_O_6092(13);
    Data_760d(14) <= Signal_O_6092(14);
    Data_760d(15) <= Signal_O_6092(15);
    Data_760d(16) <= Signal_O_6092(16);
    Data_760d(17) <= Signal_O_6092(17);
    Data_760d(18) <= Signal_O_6092(18);
    Data_760d(19) <= Signal_O_6092(19);
    Data_760d(20) <= Signal_O_6092(20);
    Data_760d(21) <= Signal_O_6092(21);
    Data_760d(22) <= Signal_O_6092(22);
    Data_760d(23) <= Signal_O_6092(23);
    Data_760d(24) <= Signal_O_6092(24);
    Data_760d(25) <= Signal_O_6092(25);
    Data_760d(26) <= Signal_O_6092(26);
    Data_760d(27) <= Signal_O_6092(27);
    Data_760d(28) <= '0';
    Data_760d(29) <= '0';
    Data_760d(30) <= '0';
    Data_760d(31) <= '0';
    Strobe_760d <= FANOUT_75dd(4);
    REG_R7 <= REG_OUT_760d;
    SPLITBLK_7624 <= ANDG_Out_5832;
    DSignal_I_5809(0) <= SPLITBLK_7624;
    DELAY_5809 <= conv_integer(unsigned(REG_RW22(8 downto 0)));
    Dwrreq_5809 <= '1';
    Dgbpr_clr_5809 <= RELOAD_REG_RW22;
    F(16) <= SPLITBLK_7624;
    GenSATBPR_792f : GenSATBitPatternRegister
        port map (
            Data        => Data_792f,
Strobe     => Strobe_792f,
REG_OUT    => REG_OUT_792f
        );
    Stretcher_7a5d : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_7a5d,
PULSE_32IN  => Signal_I_7a5d,
PULSE_32OUT => Signal_O_7a5d
        );
    ORG_75d1 : GenSAT32BitORG
        port map (
            X => X_75d1,
ORG_Out => ORG_Out_75d1
        );
    Stretcher_6feb : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
```

```
STRETCH_LEN     => STRETCH_LEN_6feb,
PULSE_32IN  => Signal_I_6feb,
PULSE_32OUT => Signal_O_6feb
        );
    Stretcher_72c4 : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_72c4,
PULSE_32IN  => Signal_I_72c4,
PULSE_32OUT => Signal_O_72c4
        );
    GenSATDelayer_64eb : GenSAT32BitDelayer
        port map (
            PCLK          => PCLK,
HOLD          => HOLD,
Signal_I        => DSignal_I_64eb,
Signal_O        => DSignal_O_64eb,
DELAY         => DELAY_64eb,
wrreq         => Dwrreq_64eb,
RELOAD_REG_RW2                          => RELOAD_REG_RW5,
isFull        => DisFull_64eb,
rdreq         => Drdreq_64eb,
usedw         => Dusedw_64eb,
gbpr_clr        => Dgbpr_clr_64eb,
isEmpty       => DisEmpty_64eb
        );
    Stretcher_6147 : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_6147,
PULSE_32IN  => Signal_I_6147,
PULSE_32OUT => Signal_O_6147
        );
    GenSATDelayer_2fed : GenSAT32BitDelayer
        port map (
            PCLK          => PCLK,
HOLD          => HOLD,
Signal_I        => DSignal_I_2fed,
Signal_O        => DSignal_O_2fed,
DELAY         => DELAY_2fed,
wrreq         => Dwrreq_2fed,
RELOAD_REG_RW2                          => RELOAD_REG_RW7,
isFull        => DisFull_2fed,
rdreq         => Drdreq_2fed,
usedw         => Dusedw_2fed,
gbpr_clr        => Dgbpr_clr_2fed,
isEmpty       => DisEmpty_2fed
        );
    ORG_37c2 : GenSAT32BitORG
        port map (
            X => X_37c2,
ORG_Out => ORG_Out_37c2
        );
```

```
    Stretcher_3bb4 : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_3bb4,
PULSE_32IN  => Signal_I_3bb4,
PULSE_32OUT => Signal_O_3bb4
        );
    GenSATBPR_369c : GenSATBitPatternRegister
        port map (
            Data         => Data_369c,
Strobe     => Strobe_369c,
REG_OUT    => REG_OUT_369c
        );
    Stretcher_33bc : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_33bc,
PULSE_32IN  => Signal_I_33bc,
PULSE_32OUT => Signal_O_33bc
        );
    Stretcher_3236 : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_3236,
PULSE_32IN  => Signal_I_3236,
PULSE_32OUT => Signal_O_3236
        );
    Stretcher_299a : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_299a,
PULSE_32IN  => Signal_I_299a,
PULSE_32OUT => Signal_O_299a
        );
    GenSATDelayer_6131 : GenSAT32BitDelayer
        port map (
            PCLK          => PCLK,
HOLD          => HOLD,
Signal_I        => DSignal_I_6131,
Signal_O        => DSignal_O_6131,
DELAY         => DELAY_6131,
wrreq         => Dwrreq_6131,
RELOAD_REG_RW2                        => RELOAD_REG_RW12,
isFull        => DisFull_6131,
rdreq         => Drdreq_6131,
usedw         => Dusedw_6131,
gbpr_clr        => Dgbpr_clr_6131,
isEmpty       => DisEmpty_6131
        );
    GenSATMultBitCoin_1e61 : GenSATMultBitCoin
        port map (
```

```
            BITS    => BITS_1e61,
SW    => COIN_SW_1e61,
BIT_OUT => BIT_OUT_1e61
        );
    GenSATDelayer_1cb3 : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_1cb3,
CNTRELOAD   => RELOAD_REG_RW13,
DELAY_OUT   => Drdreq_1cb3
        );
    GenSATDelayerFIFO_1cb3 : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_1cb3,
clock   => PCLK,
data    => DSignal_I_1cb3,
rdreq   => Drdreq_1cb3,
wrreq   => Dwrreq_1cb3,
empty   => DisEmpty_1cb3,
full    => DisFull_1cb3,
q   => DSignal_O_1cb3,
usedw   => Dusedw_1cb3
        );
    Stretcher_1cb3 : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_1cb3,
PULSE_IN  => DSignal_O_1cb3(0),
PULSE_OUT => Signal_O_1cb3
        );
    GenSATMultBitCoin_59e0 : GenSATMultBitCoin
        port map (
            BITS    => BITS_59e0,
SW    => COIN_SW_59e0,
BIT_OUT => BIT_OUT_59e0
        );
    ORG_6213 : GenSAT32BitORG
        port map (
            X => X_6213,
ORG_Out => ORG_Out_6213
        );
    Stretcher_6092 : GenSAT32BitStretcher
        port map (
            HOLD    => HOLD,
PCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_6092,
PULSE_32IN  => Signal_I_6092,
PULSE_32OUT => Signal_O_6092
        );
    Stretcher_5819 : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
```

```
STRETCH_LEN     => STRETCH_LEN_5819,
PULSE_IN  => Signal_I_5819,
PULSE_OUT => Signal_O_5819
        );
    GenSATDelayer_56a6 : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_56a6,
CNTRELOAD   => RELOAD_REG_RW16,
DELAY_OUT   => Drdreq_56a6
        );
    GenSATDelayerFIFO_56a6 : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_56a6,
clock   => PCLK,
data    => DSignal_I_56a6,
rdreq   => Drdreq_56a6,
wrreq   => Dwrreq_56a6,
empty   => DisEmpty_56a6,
full    => DisFull_56a6,
q   => DSignal_O_56a6,
usedw   => Dusedw_56a6
        );
    Stretcher_571d : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_571d,
PULSE_IN  => Signal_I_571d,
PULSE_OUT => Signal_O_571d
        );
    Stretcher_56f0 : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_56f0,
PULSE_IN  => Signal_I_56f0,
PULSE_OUT => Signal_O_56f0
        );
    ANDG_5832 : GenSATANDG
        port map (
            X => X_5832,
Y => Y_5832,
ANDG_Out => ANDG_Out_5832
        );
    Stretcher_56d1 : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_56d1,
PULSE_IN  => Signal_I_56d1,
PULSE_OUT => Signal_O_56d1
        );
    GenSATTAC_5713 : GenSATTimeAccum
```

```
        port map (
            TCLK    =>PCLK,
STARTTAC      =>STARTTAC_5713,
STOPTAC =>STOPTAC_5713,
SECFLAG =>SECFLAG_5713,
INDEX   =>INDEX_5713,
COUNT   =>COUNT_5713,
NUMRESETS =>NUMRESETS_5713,
PLLREG  =>PLLREG_5713
        );
    GenSATTAC_2911 : GenSATTimeAccum
        port map (
            TCLK    =>PCLK,
STARTTAC      =>STARTTAC_2911,
STOPTAC =>STOPTAC_2911,
SECFLAG =>SECFLAG_2911,
INDEX   =>INDEX_2911,
COUNT   =>COUNT_2911,
NUMRESETS =>NUMRESETS_2911,
PLLREG  =>PLLREG_2911
        );
    GenSATTAC_7ce1 : GenSATTimeAccum
        port map (
            TCLK    =>PCLK,
STARTTAC      =>STARTTAC_7ce1,
STOPTAC =>STOPTAC_7ce1,
SECFLAG =>SECFLAG_7ce1,
INDEX   =>INDEX_7ce1,
COUNT   =>COUNT_7ce1,
NUMRESETS =>NUMRESETS_7ce1,
PLLREG  =>PLLREG_7ce1
        );
    Stretcher_56ce : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN    => STRETCH_LEN_56ce,
PULSE_IN  => Signal_I_56ce,
PULSE_OUT => Signal_O_56ce
        );
    GenSATFANOUT_75dd : GenSATFanOut
        port map (

Input   =>FANOUTInput_75dd,
SW    =>REG_R6(7 downto 0),
FANOUT  =>FANOUT_75dd
        );
    GenSATDelayer_1cdf : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_1cdf,
CNTRELOAD   => RELOAD_REG_RW21,
DELAY_OUT   => Drdreq_1cdf
        );
```

```
    GenSATDelayerFIFO_1cdf : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_1cdf,
clock   => PCLK,
data    => DSignal_I_1cdf,
rdreq   => Drdreq_1cdf,
wrreq   => Dwrreq_1cdf,
empty   => DisEmpty_1cdf,
full    => DisFull_1cdf,
q   => DSignal_O_1cdf,
usedw   => Dusedw_1cdf
        );
    Stretcher_1cdf : GenSATV1495stretcher
        port map (
            HOLD    => HOLD,
LCLK    => PCLK,
STRETCH_LEN     => STRETCH_LEN_1cdf,
PULSE_IN  => DSignal_O_1cdf(0),
PULSE_OUT => Signal_O_1cdf
        );
    GenSATEventTrigger_7c29 : GenSATEventTrigger
        port map (
            LCLK    => PCLK,HOLD    => HOLD,
Data  => Data_7c29,
EventTrig => EventTrig_7c29
        );
    GenSATBPR_760d : GenSATBitPatternRegister
        port map (
            Data         => Data_760d,
Strobe     => Strobe_760d,
REG_OUT    => REG_OUT_760d
        );
    GenSATDelayer_5809 : GenSATV1495delayer
        port map (
            HOLD    => HOLD,
LCLK  => PCLK,
DELAY => DELAY_5809,
CNTRELOAD   => RELOAD_REG_RW22,
DELAY_OUT   => Drdreq_5809
        );
    GenSATDelayerFIFO_5809 : gbpr_fifo
        port map (
            aclr    => Dgbpr_clr_5809,
clock   => PCLK,
data    => DSignal_I_5809,
rdreq   => Drdreq_5809,
wrreq   => Dwrreq_5809,
empty   => DisEmpty_5809,
full    => DisFull_5809,
q   => DSignal_O_5809,
usedw   => Dusedw_5809
        );
    HOLD <= REG_RW1(0);
    GenSATMainLRA : GenSATLocalRegisterAccess
        port map (
```

```
            nLBRES      => nLBRES,
nBLAST      => nBLAST,
WnR         => WnR,
nADS        => nADS,
LCLK        => LCLK,
nREADY      => nREADY,
nINT        => nINT,
LAD         => LAD,
REG_R1   => REG_R1,
REG_R2   => REG_R2,
REG_R3   => REG_R3,
REG_R4   => REG_R4,
REG_R5   => REG_R5,
REG_R6   => REG_R6,
REG_R7   => REG_R7,
REG_R8   => REG_R8,
REG_R9   => REG_R9,
REG_RW1   => REG_RW1,
REG_RW2   => REG_RW2,
RELOAD_REG_RW2   => RELOAD_REG_RW2,
REG_RW3   => REG_RW3,
RELOAD_REG_RW3   => RELOAD_REG_RW3,
REG_RW4   => REG_RW4,
RELOAD_REG_RW4   => RELOAD_REG_RW4,
REG_RW5   => REG_RW5,
RELOAD_REG_RW5   => RELOAD_REG_RW5,
REG_RW6   => REG_RW6,
RELOAD_REG_RW6   => RELOAD_REG_RW6,
REG_RW7   => REG_RW7,
RELOAD_REG_RW7   => RELOAD_REG_RW7,
REG_RW8   => REG_RW8,
RELOAD_REG_RW8   => RELOAD_REG_RW8,
REG_RW9   => REG_RW9,
RELOAD_REG_RW9   => RELOAD_REG_RW9,
REG_RW10   => REG_RW10,
RELOAD_REG_RW10   => RELOAD_REG_RW10,
REG_RW11   => REG_RW11,
RELOAD_REG_RW11   => RELOAD_REG_RW11,
REG_RW12   => REG_RW12,
RELOAD_REG_RW12   => RELOAD_REG_RW12,
REG_RW13   => REG_RW13,
RELOAD_REG_RW13   => RELOAD_REG_RW13,
REG_RW14   => REG_RW14,
RELOAD_REG_RW14   => RELOAD_REG_RW14,
REG_RW15   => REG_RW15,
RELOAD_REG_RW15   => RELOAD_REG_RW15,
REG_RW16   => REG_RW16,
RELOAD_REG_RW16   => RELOAD_REG_RW16,
REG_RW17   => REG_RW17,
RELOAD_REG_RW17   => RELOAD_REG_RW17,
REG_RW18   => REG_RW18,
RELOAD_REG_RW18   => RELOAD_REG_RW18,
REG_RW19   => REG_RW19,
RELOAD_REG_RW19   => RELOAD_REG_RW19,
REG_RW20   => REG_RW20,
```

```
RELOAD_REG_RW20   => RELOAD_REG_RW20,
REG_RW21   => REG_RW21,
RELOAD_REG_RW21   => RELOAD_REG_RW21,
REG_RW22   => REG_RW22,
RELOAD_REG_RW22   => RELOAD_REG_RW22
        );
end;
```