# A FORMAL VERIFICATION METHODOLOGY FOR REAL-TIME FPGA

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Shaista Jabeen

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Electrical and Computer Engineering

July 2017

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

**Title**

A FORMAL VERIFICATION METHODOLOGY FOR REAL-TIME FPGA

**By**

Shaista Jabeen

The supervisory committee certifies that this dissertation complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Sudarshan Srinivasan

Chair

Dr. Na Gong

Dr. Dharmakeerthi Nawarathna

Dr. Yechun Wang

Approved:

| | |
|---|---|
| August 01 2017 | Dr. Alan Kallmeyer |
| Date | Department Chair |

# ABSTRACT

Real-time systems in safety-critical and mission-critical domains have stringent or hard timing constraints. The correctness of such systems is of prime importance to avoid any unacceptable consequences like a big financial loss or a human life loss. With dynamic performance demands and the system complexity, there is an increased difficulty to prove and verify the correctness of a real-time system design. There is a shifting trend of implementing the complex real-time systems using hardware based solutions instead of software based solutions. The interaction between multiple system components and functional behavior of individual modules needs to be checked for correctness not only functionally, but also with respect to the time. Field programmable gate arrays (FPGAs) are getting popular in a wide variety of safety-critical real- time system applications for last two decades. FPGAs have predictable timing behavior, low cost and they outperform over general purpose CPUs.

In this dissertation, we present a new formal verification approach that addresses the functional and timing correctness attributes of FPGA-based designs in safety-critical real-time applications. Our technique is a refinement-based deductive verification technique, which tells what it means for a system at lower abstraction level to be equivalent to a system specification at a higher level . We used the notion of Well-Founded Simulation, which explains the reasoning for a single step transition of RTL design in FPGA implementation. Initially, the system specification is obtained as a timed transition model. The implementation circuit in FPGA is also modeled as a timed transition system. Stuttering phenomenon and rank are used to prove the safety and liveness properties of the system. We devised a set of proof obligation templates for functional and timing verification, respectively. The proof obligation templates were successfully applied to some case studies. The developed technique can be extended to the applications which employ network on chip in their design.

# ACKNOWLEDGEMENTS

First and foremost, I would like to express my immense and deep gratitude to ALMIGHTY ALLAH for providing me an opportunity to accomplish this achievement in my life. All the strength, courage, support, motivation, power, wisdom and favors are bestowed by Him during the entire course of my studies.

I would like to thank my adviser Dr. Sudarshan Srinivasan for his help, guidance and innovative ideas. He has been very supportive throughout the program and I have learnt a lot from him during my PhD. I truly respect him for being a nice, kind, and genuine person. I respect him even more because of his persistent guidance and help that he is always ready to offer. I attained the true confidence of the research work that I have done with him.

I would like to extend my appreciation to my committee members Dr. Na Gong, Dr. Dharmakeerthi Nawarathna, and Dr. Yechun Wang. I am grateful for their encouragement and support. I am thankful to the Electrical and Computer Engineering Department for facilitating me during my studies. I am thankful to COMSATS Institute of Information Technology Pakistan for providing me a wonderful opportunity for my PhD in USA.

I am grateful to my friend Sana Shuja for her constant support and encouragement throughout my PhD. I would like to thank my friends and siblings who facilitated me in the hour of need during this journey.

Last but not the least, I am so grateful to my parents for sending me to this part of world and trusting on my capabilities. Their unconditional love, faith and support made me the person I am today. Without their support, I would not have been able to achieve all this.

# DEDICATION

To my Father, Mother and Siblings

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 1. INTRODUCTION

## 1.1. Background: Motivation

Our everyday lives have greatly become dependent on a large number of systems which employ one form or another form of electronic control(s) such as home appliances, transportation, medical equipment, industrial control and automation. Such systems are embedded systems consisting of tiny computers to control the system operations electronically. An embedded system is a computer system that is capable of performing a limited set of specific functions. It often interacts with its environment. Most embedded systems are real-time in nature.

A real-time (RT) system is a one that is required to process the information and generate the response within a fixed timeline, called a deadline. If the system generates the correct response after a certain deadline it could be regarded as having failed or the response after the deadline can be equivalent to a wrong response. Due to the addition of new, advanced features to the real-time systems, there is an ever increasing complexity involved in the design of such systems. Many real-time systems are safety-critical.

If any failure in a system leads to the consequences that are determined to be unacceptable, then that system is the safety-critical system. Examples of safety-critical real-time systems include a variety of systems like an automobile, an airplane, an implantable medical device etc. Such a system is typically a combination of complex heterogeneous components or modules working together. A nuclear power plant contains a number of control and automation functions performed by multiple embedded controllers and computer systems. All modules work together by communicating to each other in a timely manner. If there is any error in one of the system components, then that error can propagate further to cause a system failure. Eventually, that failure can cause an undesired system behavior, which will lead to death or injury to persons or damage to property.

In order to avoid undesirable consequences, it has become imperative for the developers to ensure that a system performs accurately under all scenarios. The correct behavior of a system is guaranteed by verifying that the system is functionally correct and the timing requirements are met. Hence, the two aspects related to the verification of real-time system design are: 1) logical correctness, and 2) response at the right time.

There are several different verification techniques employed to guarantee the correctness of a system. Traditionally, there are two major classifications used for system verification: 1)testing based techniques, and 2) formal methods based techniques. In testing based techniques, different test vectors are applied to cover all the possible combinations of inputs. The response of a system is observed and the performance is evaluated for all the inputs. Testing is applied once a system has been designed.

Formal methods employ mathematical techniques in which real world systems are modeled as mathematical objects. Required safety properties are encoded and checked on the system models, which help to find forbidden situation(s) in the system operation. Formal methods further include two approaches, model checking and theorem proving.

Model Checking is a completely automatic technique. It performs reachability analysis by going through the entire state space, evaluating the system correctness, and pointing out the erroneous behavior. The required safety properties are expressed as formulae in temporal logic [4]. If a safety property fails for a system, the model checker tool provides a counterexample which describes about the reachability of such particular unsafe state(s) in the system. Examples of model-checking tools for timed systems are Spin [**?**], UPPAAL [5] and Kronos [6].

Theorem proving is a deductive procedure in which theorems are formalized to capture the behavior and the correctness requirements of a system. If the corresponding theorem can be proved in theory, it shows that the system holds or meets the requirements in practice. Theorem proving methods are capable of dealing with parameterized systems and infinite states. Such methods do not suffer with the state space explosion problems as compared to the model checking. Theorem proving is a manual technique which needs a key step of generating proof by the user. The theorem proving tool can then fill in the smaller holes or gaps. Formal methods are applied right from the start of system design phase.

The problem associated with the testing is that it cannot ensure the absence of bugs from a system. Another problem is that the testing of real-time systems requires real-time simulation, which need to consider not only the value domain but also the temporal domain. . There are several important requirements of testing in the temporal domain. The input to the test object may need to be delivered at a particular moment. At the beginning of test execution, the temporal state of the test object may need to be controlled. The timing of the result may need to be observed.

There is a probability for non-determinism etc. A significant benefit of using formal varication in the design process is getting proofs for critical parts of a system that are not easy to verify with simulation, or for the modules used at different places.

Formal methods can give the assurance that a system is bug free. Therefore, it is a better and a reliable approach to be used in the early design phase of real-time safety- critical systems.

## 1.2. Hard Real-Time Versus Soft Real-Time Systems

Systems that are required to meet hard or firm deadlines to generate a response are called hard real-time systems. A late response (even if correct) may be a fatal flaw, can be of no use or cause disastrous consequences. On the other hand, soft real-time systems need to meet a soft deadline. A soft deadline means that the timely completion is desirable but a delayed response is also useful to some extent. Examples of hard real-time systems include automobiles, airplanes, traffic control systems, implantable medical devices, nuclear reactor controllers etc. Examples of soft-real-time systems include email server, streaming audio, video, online reservation, MS word processor etc. All safety critical systems are hard real-time systems. Temporal accuracy is a prime concern in these systems. Bugs and errors that occur in hard real-time systems are very often irreversible. If an error occurs in a soft real-time system, the computation is rolled back to a previously established checkpoint to initiate a recovery action. A hard real-time system must remain synchronous with the state of the environment in all cases. On the other hand, soft real-time systems will slow down their response time if the load is very high. Formal verification is important because a real-time safety critical system must not behave anything, except for what it is designed to do. Our work aim to devise a formal verification methodology for real-time systems implemented on Field Programmable Gate Arrays (FPGAs).

## 1.3. Field Programmable Gate Arrays (FPGAs)

An FPGA is an integrated circuit that contains thousands of connected logic cells. It is programmed by the user or designer after it is manufactured, so its name is field programmable. Each logic cell in FPGA has the same basic structure but they can be configured to perform different operations. These cells are connected together by a series of interconnects to perform higher level processing tasks. Any design or application in FPGA is implemented as a hardware module. This is in contrast to the microcontroller based solution to perform a particular function. In microcontroller, there is a fixed architecture and program instructions are mapped sequentially to that architecture. There are underlying software complexities involved in software based design like interrupt handling etc. FPGAs are parallel devices with multiple logic blocks that get converted into a circuit. An example is shown in Figure 1.1 to show a comparison between software based and hardware based design. FPGAs are a good choice for real-time processing because of predictable timing behavior. There is an increased trend of research and development in FPGA based real-time systems in industry and academia for a last couple of decades [7] [8] [9] [10] [11] [12]. FPGA verification has two categories: design verification and implementation verification. In design verification, attempts are made to find the design errors or coding errors. It also includes finding errors in specification by applying simulation or formal verification. In implementation verification, the focus is to locate the errors introduced in translation or synthesis stage. Our prime focus is FPGA design verification for real-time (RT) safety critical applications. We propose a verification framework for RTL design verification for real-time applications, excluding the implementation verification of FPGA fabric or IC.

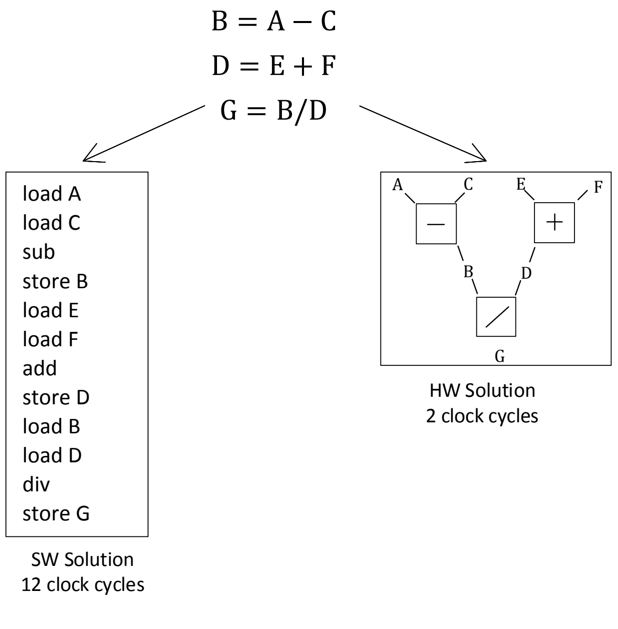

Figure 1.1. Software vs. Hardware

### 1.3.1. RTL Design Verification of Real-Time Systems

At first stage, a functional level design is modeled for a system. The second stage is to model the register transfer level (RTL) design by refining the architectural description. This phase involves designing the functional components and memory elements using hardware description language(HDL). This stage also includes the development of the clocking system of the design and architectural trade-offs such as speed/power. Once RTL design is finalized, then a fundamental stage of design verification starts. In the verification process, practices and techniques are employed to make sure that the design is free of errors and defects. The purpose of verification is to remove all the bugs before the expensive phase of physical implementation of a system. Each time a functional error is found, the functional design model is checked and is modified to provide the correct behavior and the RTL design is updated consequently. Sometimes certain attributes may be unnoticed or there is a possibility of unaddressed aspects in the original set of specification. In such scenario, the specification needs to be updated.

The flaws and errors gone unnoticed in initial design and found in final implementation of a system are expensive to be taken care of. RTL verification is one of the challenging activities in development of real-time and digital systems. FPGA based real-time safety critical systems have multiple components like controllers, inputs, environmental interfaces, logic and temporal dependencies. During the development phase of a design, a system may often go through multiple transformations from the original set of specification to the final system product. The goal is to make the final product safe to operate and free of any kinds of bugs and errors.

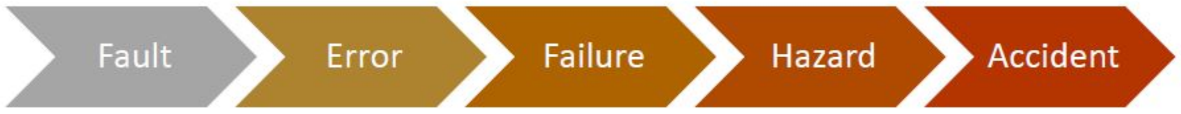

Figure 1.2. Error Flow

## 1.4. Why Formal Verification?

Formal methods have become common and essential tools to be used in design of safety-critical and mission-critical systems. The presence of any fault can lead towards a mishap, an unfortunate accident or unaccepted consequences. The possible factors that can lead to a hazardous consequence are depicted in figure 1.2. Examples of safety-critical and mission-critical systems are automotives, implantable medical devices, aerospace, railways, nuclear systems, industrial equipment controllers, digital motor control, power line monitoring and process control etc. In order to avoid the occurrence of unacceptable consequences, it is required to make sure that the system design is free of any faulty behavior and it operates safely in all modes.

Verification makes sure that systems goes through all the paths and transitions how it was intended to perform and does not exhibit any unwanted system behavior. Formal verification builds universal correctness properties about the particular design, independent of any specific set of inputs. It checks all possible cases including corner case situations which otherwise go untested in simulation based verification approaches.

## 1.5. Why is Testing Not Sufficient?

Testing strategies revolve around applying the test vectors for inputs and observing the system response against those inputs. If a system is large and complex with N number of inputs, then testing requires to generate $2^N$ number of input combinations to check all possible behaviors. In reality, it is very time consuming to check a system against each and every combination. Testing is an exhaustive technique and it also does not assure the absence of bugs. It is diifcult to uncover corner case bugs with the help of testing.

For last two decades, there has been an exciting development of modern electronic technologies and products. Real-time systems are becoming complex and computation intensive. We have developed a formal verification methodology for FPGA based system at RTL design level. Our proposed scheme is applicable to several applications mapped onto FPGAs. We carried out some case studies and explained the proposed scheme. A refinement based verification technique is applied, that provides reasoning for each step taken by the system at a circuit level while assuring certain safety properties.

### 1.5.1. Problem Statement: Our Contribution

Real-time system models are verfied by using several available model cheking tools. UPPAAL [5] and KRONOS [6] are two popular tools used to check the safety properties of real-time systems. In these tools, system is modeled as a timed automata and the desired safety properties are modeled using computation tree logic (CTL) [13]. Reachability analysis is performed by expanding the state space of system. The problem associated with model checking is the state space explosion. Another problem is that the model checking tools verify the system properties at a higher level of system. Errors and defects can be introduced when a system is implemented at a lower level like the RTL level. Our goal is to verify the RTL design of a system. RTL is a low level of a system implementation which contains the minor circuit details along with interfaces. We proposed a methodology that tends to account for a single step transition of the implementation system to be equivalent or matching to the specification. In this regard, we used the notion of Well-Founded Simulation (WFS) which was originally developed by Pete Manolios [14]. WFS based refinement is used to show that the RTL circuit implementation in FPGA is equivalent to a high level specification model. This attempt is more than verifying a set of properties for the safety of a system.

Real-time systems are built from various components working together as a big entity. The properties and behaviors of a system are distinct from those of its components. Engineers and designers predict and calculate the properties of system from its components working together. There are two main aspects related to an overall system performance in complex real-time systems, the computation and the communication between several applications. The correctness of a system is associated to the correct behavior of the computation part and the communication part happening in a correct timely manner. We propose a verification methodology which can verify the FPGA controllers for computation. The proposed technique can also be extended to the applications that use network-on-chip for communication purposes to accoplish the tasks in FPGA based real-time systems. Our approach is applied on a few case studies which include the applications mapped on FPGAs. Our particular contributions are:

- To develop a set of functional proof obligation templates for verifying the functional correctness of a system, consisting of seperate proof obligations for stuttering and non-stuttering phenomenon, respectively
- Introducing the timing proof obligation templates using Timed Well-founded Simulation (TWFS)
- To use the rank functions and FPGA properties to provide reasoning for latecny bounds
- To devise the three high-level steps for timing verification using the defined proof obligation template-properties for timing verification without expanding the implementation system

The proposed approach is based on theorem proving technique which is scalable and parametric so state space explosion is not a problem here. We believe that a certain class of applications mapped on FPGAs, which are used in real-time safety-critical domain, can be verified using our developed verification methodology.

The organization of this dissertation is as follow; chapter 2 decribes some preliminary concepts that are required to understand this dissertation. In chapter 3, we describe the related work for the real-time systems verification. The formal verification of FPGA based systems is also mentioned. Chapter 4 explains the developed methodology with some case studies. Chapter 5 is about the formal verification scheme applied on the stepper motor with different modes of operation. In chapter 6, we cover the case for formal verification of those applications mapped on FPGAs, which employ network-on-chip (NOC) in their design. Chapter 7 concludes the dissertation.

# 2. PRELIMINARIES

The development of proposed verification methodology is built upon existing mathematical constructs. In this chapter, we provide an introduction to the concepts required to understand this dissertation. We describe the mathematical structures and the particular approach adopted to solve the problem of verifying the correctness of hardware based safety critical systems. Typically, the correctness verification of a system requires showing that the system is equivalent to another system that serves as a specification. Specification is given as the abstract (mathematical) model. To model a real-time system in mathematical domain, it is required to capture the particular aspects of a system in a correct way. Such formalisms to construct a mathematical model may capture structural and architectural or focus strictly on behavioral aspects. For verification purposes, formal methods are aimed for simplicity to conduct efficient analysis of design implementation. A possible abstract view on the behavior of a system is to regard it as an object having some internal state and, depending on that state, is able to take transition leading to other states. Such a transition might be autonomous or stimulated by the environment. In our case, a mathematical structure that tries to capture this abstract view on a system is a transition system. We capture the system specification requirements for a real-time system as a timed-transition system.

## 2.1. Timed Transition System (TTS)

The mathematical model used to specify the system requirements is a timed transition system (TTS). The formal definition for a TTS is given below.

**Definition 1** *A TTS $M$ is a 3-tuple $\langle S, R, L\rangle$, where $S$ is the set of states, $R$ is the transition relation that defines the state transitions, and $L$ is a labeling function that defines what is visible at each state. $R$ is of the form $\langle w, v, lb, ub\rangle$, where $w$, $v \in S$ and $lb, ub \in \Re$. $lb$ and $ub$ indicate the lower bound and the upper bound on the time delay of the transition, respectively.*

The purpose of a mathematical model is to capture the system behavior and its evolution under all possible situations. A system can be only in one of the possible states at any time as defined by set S. Current state can take a step and transition to next state, which is captured by the transition relation set R. The upper and lower bounds indicatie the window of time in which that transition is required to take place. The labeling function tells us about the information visible at a particular state. The TTS is, therefore, a labeled transition system.

## 2.2. Refinement Based Verification: Refinement Map

In simple words, refinement means to add details to a system design. Refinement is a mechanism for structuring complex systems for increased readability. To develop a system there are several levels of abstraction, from requirements specification to concrete implementation. These levels or phases are typically classified as: 1) Developing the formal specification from informal description of system requirements, 2) Developing the architecture of a system and refining it up to the required level of granularity, known as the system design phase (in our case it is RTL design), 3) Finally, the transformation of design to the detailed implementation (in FPGAs this phase refers to physical design including synthesis, place and route) We present an approach that focuses mainly on the first two phases (i.e. requirement specification and RTL design).

Translation of a high level specification to an architecture involves adding details by defining structures and interfaces to the design of a real-time system. In this study, we use the term implementation which refers to the system design obtained in second phase of development as described above. The implementation is usually at low-level, which includes underlying circuit details like timers, flags, counters, registers, etc. Refinement-based verification is a method in which the functionality of an abstract system model is proved to be correctly implemented by a low-level implementation. An example of refinement-based verification can be one that verifies if a clock-accurate model of a pipelined processor correctly implements a given instruction-set architecture. (i.e, a programmer's model of a machine). In our case, the abstract model/specification of the system is a TTS. Implementation is the RTL design (circuit) for FPGA. We use the refinement maps to translate the behavior of implementation in low level to match to the behavior of high level specification. Refinement map is a function that projects the implementation states to specification states. Use of refinement map makes it possible to verify the small parts of the low-level design in the context of the abstract model. An advantage of refinement methodology is that it makes possible to verify designs that are much too large to be handled directly by model checking.

## 2.3. Stuttering and Non-Stuttering Phenomenon: Rank Function

Stuttering and non-stuttering are the two notions used to represent the behaviors of the implementation system. In the context of verification, these two notions classify the progress of implementation system into two categories. A system is either stuttering which means that it is making progress and not caught in a deadlock, or a system is taking a non-stuttering transition which means it is making progress with respect to the specification.

Implementation is at the lower level and takes mutiple transition steps to correspond or match to a single step transition on the specification side. Stuttering is the phenomenon that accounts for the multiple transitions on the implementation side to be equivalent to a single-step transition on the specification side.

Stuttering phenomenon on the implementation side should manifest that a system is making progress and it is not caught in a deadlock situation. Some witness function, known as *rank*, exists to account for the stuttering phenomenon. The domain of a *rank* function is the set of states of a system and it's range is the natural numbers. Whenever implementation makes progress, the value of rank decreases. Therefore, it is important to define an appropriate *rank* function for a particular system which is verified. We have extended the role of this *rank* function to account for the timing behavior of a system implemented in FPGA. The concept of stuttering, non stuttering and *rank* function can be explained by the help of the following example.

### 2.3.1. Example

Suppose we want to design a counter that counts in the sequence 0,2,0,2,..... The specification and implementation are both shown as transition system in figure 2.1. We have not shown the time bounds for space constraints here. The inner part shows two states 0 and 2 which are the specification states of specification TS. The outer part has four states which represent the implementation states. Typically, an implementation of a system on the hardware or software takes several multiple transitions to make progress with respect to the specification. This example depicts a smaller case in which two implementation steps are taken to match to a single transition on specification side. For a single specification transition $(0, 2)$, the implementation goes through two transitions for example r1, $(00, 01)$, and r2 $(01, 10)$. In the first transition r1, the successor state corresponds to the same specification state 0. Such multiple implementation states that correspond to a single specification state are classified as stuttering states and the phenomenon is called stuttering. When the implementation transition is similar to the specification transition, then the successor state in implementation corresponds to the successor state of specification transition. This transition is classified as non-stuttering transition. In order to ensure that the implementation

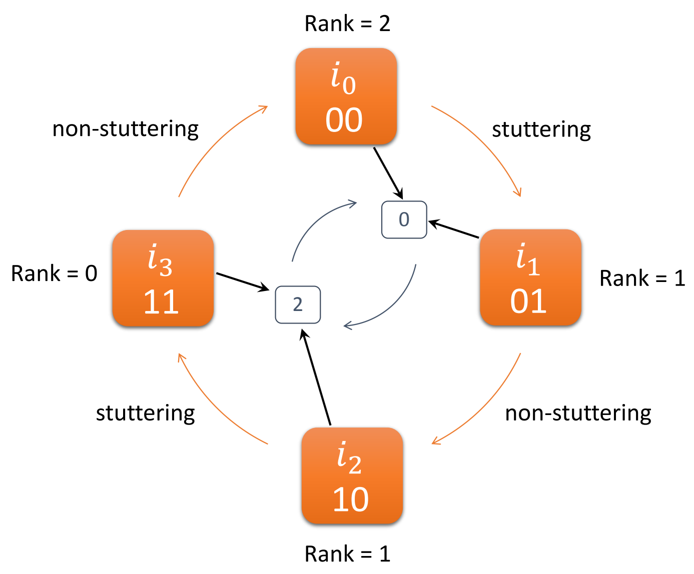

Figure 2.1. Example: Counter as a Transition System

is making progress and it is not caught in a deadlock, it is important to define a rank function. A rank always exists whenever there is stuttering phenomenon and its range lies in natural numbers. The value of rank function decreases whenever an implementation makes progress by each step. In the above example, rank function is the number of zeros. We can see in the figure that whenever there is a stuttering phenomenon, the number of zeros decrease in the successor state.

## 2.4. Simulation Relation

The formal definition of simulation relation can be given as below [15].

[14] $R$ is a *simulation relation* on transition system $TS = \langle S, R, L \rangle$ if $R \subseteq S \times S$ and for $s$,$w$ such that $sRw$ we have the following.

1. $L.s = L.w$

2. $\langle \forall u : sRu : \langle \exists v :: wRv \quad \wedge \quad uRv \rangle \rangle$

When there are different levels of hierarchy in a system, then all such hierarchical levels should be consistent with respect to their functional, performance or certain attributes. Simulation relations are used to model the consistency or uniformity between such hierarchical system models. For example, given the two transition systems S1 and S2, S2 simulates the transition system S1 if every transition made by S1 can correspond to a similar transition taken by S2. According to above mentioned definition, we can say that the state $s$ of S1 is similar to the state $w$ of S2, if there exists a simulation relation R such that $sRw$. Here, S1 can be at a higher level (specification) and S2 can be at lower or more complex level (implementation). S1 and S2 model the two levels of a hierarchical system, one is a simpler model and the other one is a refined or optimized model. These two models are related together through a simulation relation R.

P. Manolios [14] developed the theory of refinement based on the stuttering simulation. Their work provides reasoning for a single-step rather than reasoning for infinite computations. We use the notion of well-founded simulation refinement to verify the functional correctness of FPGA based real-time systems. We extend it to the Timed Well-founded Simluation (TWFS) to provide reasoning for the timing behavior of the real-time FPGA systems.

## 2.5. Theorem Proving and Hardware Verification

In theorem proving technique, the verification of system correctness is equivalent to proving the corresponding correctness condition theorem. For example, the system specification is formalized as a set of formulae $\Phi$ and the implementation is formalized in another form $\Psi$. In order to prove that the system is correct, we need to prove $\Phi \models \Psi$. The method or procedure of theorem proving is supported by different theorem proving tools which are based on first-order or higher-order logic. The are many automated theorem proving tools which are equipped with the sets of axioms and the sets of inference rules. These tools are also combined with some background theories. A person must interact and guide the tool to obtain the correctness proof of a system. The modeling languages used in the theorem proving tools are very expressive and can be easily used to represent the properties. One of the main difficulty of using theorem proving tools is that if a user does not successfully complete the proof of property, the tool is not capable to tell about where is the shortcoming or whether the property is indeed unprovable [16]. On the other hand, theorem proving method offers better flexibility and control in doing proofs compared to model checking. We propose a verification technique that is based on theorem proving. We use Z3 tool which is an automated theorem prover developed by Microsoft [17].

### 2.5.1. Hardware Formalization

An initial criterion to formally verify the hardware design is to formalize the implementation by translating the hardware model into a mathematical model. The translation procedure depends on the abstraction level of the implementation model. Typically, an FPGA design process has several layers from the top specification to the bottom netlist. Depending on the level of abstraction, formalizing a hardware implementation is generally classified into two models, register transfer level hardware description language (RTL HDL) and netlist schematic circuit diagram.

Our focus is on the mapping of safety-critical applications onto FPGA and to verify the RTL HDL model. In FPGA, all inputs are synchronized by a global clock. Since the RTL model is a more abstract level, the problem of clock jitter and clock skew is not taken into consideration. Under this assumption, the clock can be abstracted as a natural number representing the number of completed clock cycles. In our case, hardware implementation is also modeled as a timed transition system (TTS). The difference from the specification TTS is that in implementation we have millions of transition relations depending on the clock of FPGA. We have formalized the specification and the hardware implementation as TTS in first-order logic using semantics of Satisfiability Modulo Theories (SMT) language [18] [19].

## 2.6. An Overall View of Proposed Verification Methodology

Figure 2.2 describes the flow of our proposed methodology. At first stage, specification of a particular system is modeled as a timed transition system (TTS). At second stage, it is refined and implemented on RTL level. At the final stage, the RTL is converted into SMT language. SMT is a dedicated domain that is based on automated deduction. It is used to check the satisfiability of first-order logic formulae with respect to some background logical theories. Characterization benchmarks are the logical theories incorporated in SMT. SMT contains several quantifier-free formulae which confine the language to express or formalize the particular system. We use the bitvectors theory because on RTL level everything has a fixed bit-width. We construct the two kinds of proof obligations, stuttering and non-stuttering, using the theory of well-founded simulation (WFS) refinement in SMT [18]. The proof obligations are then discharged into a solver Z3 [17]. If Z3 evaluates that all the conditions are satisfied, then our verification process ends. If requirements are not satisfied, Z3 generates counterexamples. By looking onto the counterexamples, the specification and the design modeling in SMT are modified and adjusted. The invariants are added to capture only the reachable states of a system.

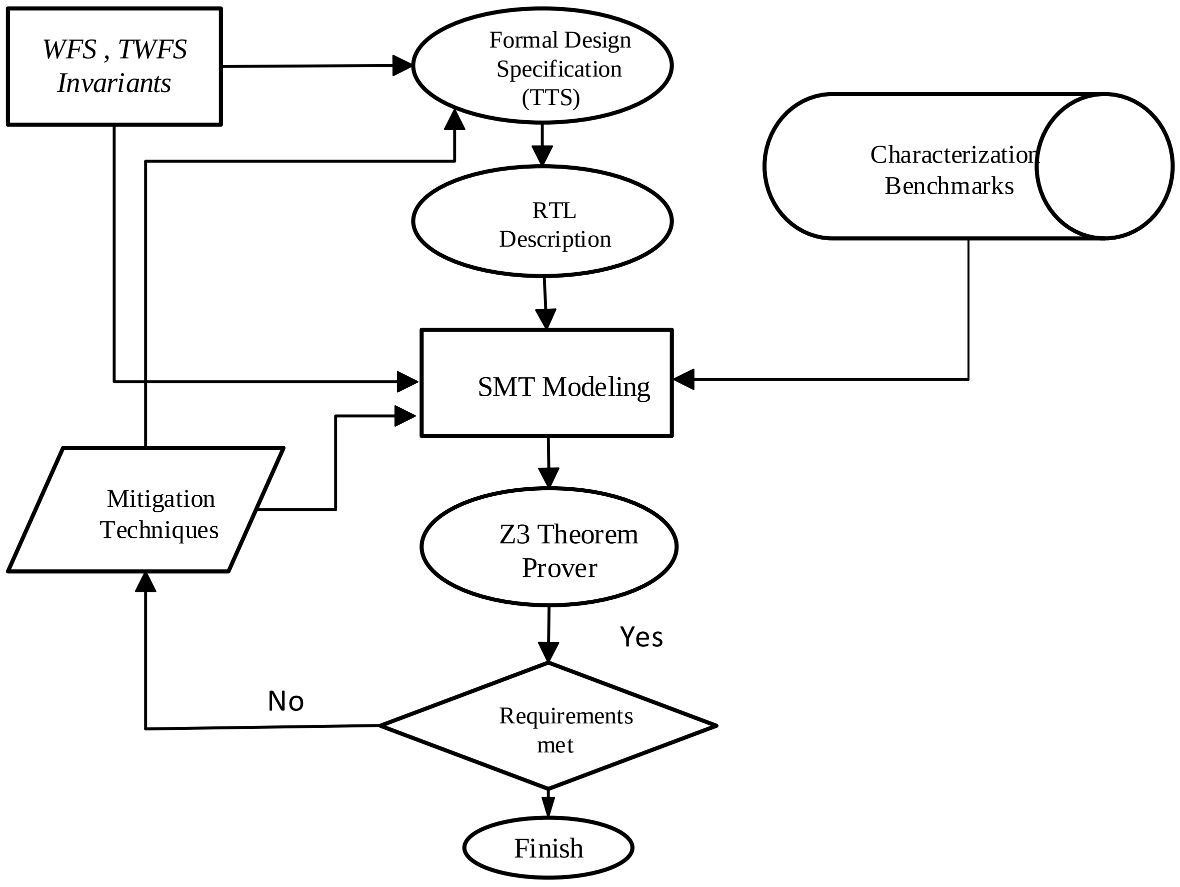

Figure 2.2. Verification Methodology Flow Diagram

# 3. RELATED WORK

The constantly rising demands for greater flexibility and efficiency in many complex real-time applications are leading towards increased deployment of FPGAs. At the same time, the safety analysis of such systems and applications is essential to ensure the safe operation. Modern FPGAs are tremendously complex. They are powerful enough devices that they can be even configured to host a complete microprocessor, or even a System-on-Chip, i.e., a complete system, composed of processor, memory and peripherals, all placed on the same chip. [20] provides a more detailed discussion about FPGA architectures.

Mission-critical systems include systems such as aerospace control systems, nuclear reactor control system, automotive safety and control systems. Safety-critical systems such as railroad/subway control systems, medical devices and mission critical systems must avoid the occurrence of faults even at high costs to prevent the undesirable consequences. There are several general frameworks available that are manifested as safety standards for hardware and software system design. Some of such standards are given in table 3.1 [21].

These standards put rigorous requirements on the system development to ensure high-levels of safety, both functionally and with respect to timing requirements. Most of these available standards do not directly address the specific issues of the FPGA technology, or provide only limited guidance about them. The FPGA design flow is more automated than the ASIC, and thus it leads designers to rely much more on the CAD tools provided by the FPGA vendors. In FPGA based real-time design, the predominant approach has been to apply sufficiently effective sets of tests to simulate the design. There is less attention paid to formally verifying the correctness of the intermediate products of the FPGA design phases and the trend has been to rely too much on the CAD tools.

## 3.1. Approaches for Verification of Real-Time Systems

The predominant approach has been to use the testing and simulation based verification methods for FPGA based systems. Several formal verification techniques have also been proposed by research communities for hardware and software based real-time systems. Such techniques have been deployed on various case studies. We will discuss some of the significant work that has been

Table 3.1. Safety Standards

| Standard | Application Type | Industry | Created by |
|---|---|---|---|
| DO-178B | Software | Aerospace and Defense | Radio Technical Commission for Aeronautics (RTCA) |
| DO-254<br>EN 50128 | Hardware<br>Software | Aerospace & Defense<br>Railway Transportation | RTCA<br>European Committee for Electrotechnical Standardization (CENELEC) |
| FSDA | Hardware | Cryptographic Equipment | National Security Agency (NSA) |
| IEC 60601 | Hardware | Medical Equipment | International Electro technical Commission (IEC) |
| IEC 60880 | Software | Nuclear Power | IEC |
| IEC 61508 | Hardware & Software | Heavy Equipment and Energy | IEC |
| ISO 26262 | Hardware & Software | Automotive Electronics | International Organization for Standardization (ISO) |

developed for formally verifying the real-time system designs and FPGA based real-time systems.

### 3.1.1. Model Driven Verification Approach

In model-driven approach, the functional verification of a system is carried out by defining a platform independent model. A popular technique for model-driven verification is known as model checking. Model checking is used to check the temporal properties which are expressed using a temporal logic [4]. There is an extensive work done in verification of temporal properties over finite state systems. Large complex real-time systems are often designed using component-based development approaches. Various components are used as building blocks to make a larger system as a functional body. Erroneous behavior or failure in component-based systems are generally due to unintended or improper interactions among the components. There is an obvious likelihood that a fault in one function could propagate to the other in component based systems. Therefore, a system should not only compute accurately but also communicate safely and correctly in a modular design. For safety-critical systems, it is essential to remove unintended interactions and to verify the correctness of those that are intended. Research work shows that model checking of very abstract designs can support human-guided assumption synthesis for verification purposes.

There are several model checking tools available to formally verify the systems behavior. Two popular tools, specifically targeting real-time systems are UPPAAL [5], Epsilon [22] and Kronos [23]. These tools are based on the theory of timed automata [24]. In timed-automata, the finite state machines are extended with the continuous-valued variables (clocks) used to measure the time delays. A system is modeled as a network of timed automata in both UPPAAL and Kronos. Several case studies have been carried out after the development these tools to perform verification. Some of the case studies among many others include verification of industrial communication protocols like audio-transmission protocol by Philips [25] [26], Collision Avoidance Protocol (CSMA-CD) protocol [27] [28] and Fischer's real-time mutual exclusion protocol [29]. There have been many extended works based on timed-automata for verifying the timed-systems [30]. David et al., [31] have proposed an UPPAAL based tool to check the specifications of real-timed systems. They provided the important concepts for refinement, consistency checking, logical and structural composition, and quotient of specifications, all of which are indispensable in compositional design practice. They implemented the theory on the engine of UPPAAL-Tiga [32] and demonstrated a small case study.

The concept of robust partitioning has been developed since the modern aircrafts started employing Integrated Modular Avionics (IMA) [33]. The idea of partitioning is to protect the applications from each other which use the shared resources, as if they were not sharing and each had their own private resources. The key resources involved in this partitioning are communication and computation: i.e., networks and processors. The goal of partitioning is to eliminate the unintended pathways for communication in networks and processors and high-level resources built on these. Partitioning provides surety that when new components are added to an existing system, the prior properties are preserved. To calculate the properties of interacting components, the conventional and well-known way is to use assume/guarantee reasoning. [34]. Cimatti et al. [35] have proposed a tool, OCRA, based on assume-guarantee reasoning. This tool checks the contract-based designs in complex embedded and real-time systems. In contract-based design, properties are added to the component model in the form of contracts. Each contract contains the properties to be satisfied by the component environment (assumptions) and properties guaranteed by the component in response (guarantees). The tool allows for checking the refinement of contracts specified in linear-time temporal logic. Several benchmarks have been verified using this tool. Our verification approach is different since we deal with the refinement of actual hardware design, while OCRA deals with contract-based refinements for a design.

E. Endres et al. [36] proposed a strategy to build, formally and pervasively verify a complex distributed automotive system. The idea of pervasive verification entails the complete system verification including the interaction among all the components thus reducing the number of system assumptions. The authors aim was to get a top-level single theorem that describes the correctness of whole system. Their work targets a gate-level model which comprises of several interconnected electronic control units (ECUs) with independent clocks. The automotive system is implemented on several FPGA boards. The whole system is a distributed asynchronous communication system with a set of ECUs connected by a bus. Pervasive verification was performed by combining the three models into a single model. The three models at different levels of abstraction for system are : 1. a formal model of an asynchronous real-time triggered system on the bus side, 2. a formal gate-level design of digital hardware for local properties on the controller side, and 3. a formal model as seen by an assembler programmer. The unified model was formally specified in Isabelle/HOL theorem prover [37]. Then a gate-level prototype was synthesized from formal models. The verification was done using combination of interactive theorem proving (Isabelle/HOL) and model checking (LTL).

P. Conmy et al. [38] have formally done the failure analysis of modular design embedded on FPGAs. They presented a bottom-up technique to trace the low-level faults to high level system hazards. The failure and safety properties are derived by performing the exhaustive analysis on the design embedded on FPGA, to identify the safety case. A comprehensive failure investigation of FPGA's circuit is carried out to fix the potentially hazardous outputs that can occur at the input output pins of FPGA device. The scale of analysis is then upgraded in a hierarchical component-based strategy. Their approach is comparable to the Model Driven Architecture (MDA) approach to a certain extent. MDA was described by Object group management [39] that uses different levels of hardware abstraction. The authors said that the proposed approach is compatible with current widely used international safety standards such as DO254 [40] and IEC61508 [41]. A case study of mine control system embedded on an FPGA was carried out to demonstrate the developed techniques. J. Hammarberg and S. Nadjm-Tehrani [42] presented the development and verification of electronic components for hydraulic monitoring system (HMS). HMS are used in aircrafts for actuating the control surfaces and landing gears which are highly critical components.

They wrote the formal specifications of system in Esterel language and verified the safety properties. They used model checking combined with failure modes and effects analysis (FMEA) and fault tree analysis (FTA). Once the safety properties were satisfied, the Esterel model was then automatically translated into VHDL and implemented on FPGA.

Typically, model checking verifies a system model at a higher-level, which employs to check for certain properties ensuring safety and correctness. After the model verification, next step is to translate the model into some specific platform model that can run independently or on a computer. This process of model translation onto a lower level platform is the implementation phase and it is vulnerable to errors or defects. Therefore, verification at the lower level is equally important. We primarily focus to verify the lower level design for FPGA systems by modeling the RTL system design and using theorem proving technique.

### 3.1.2. Proof Based Methods: Deductive Verification

Deductive verification is comprised of theorem proving techniques. The deductive proof obligation technique involves creating the proof obligations from the system specifications. The truth value evaluation of these proof obligations infers that the system conforms to its specification. The proof obligations are discharged into an interactive theorem prover (for example HOL, Coq [43] PVS, ACL2, Isabelle), an automatic theorem prover or satisfiability modulo theories (SMT) solvers. Deductive verification requires a lot of input effort from the user to understand and encode the system information in a correct manner and convey this information to generate a verification framework. It involves formulating the theorems, functions or lemmas to be proved.

There is a considerable amout of work done in deductive verification domain. We mention some of these works here. H. Deng [44] has (1) proposed a method for formalizing FPGA implementations at different levels of abstraction, and (2) proved the functional correctness. The FPGA implementation of some frequently used safety subsystem was formalized through PVS theorem prover. The components were modeled both at the netlist level and at the Verilog Register Transfer (HDL) level using some correctness conditions. This work described the general strategies that can be used to prove the properties for synchronous circuit design. This work did not verify the FPGA for real-time systems. B. Barras et. al provided significant rules for the modular construction and verification of cyber-physical systems. They evaluated the proposed rules by implementing the controllers for safety properties of quadcopter using the Coq proof assistant [43].

Nikolaj S.Bjørner et. al. [45] presented a modular framework to prove the temporal properties of real-time systems. They used the deductive verification rules and automatic invariant generation to create the properties of real-time systems. Using the proposed methodology, the authors presented mechanical verification of the generalized railroad crossing case study using Stanford Temporal Prover, STeP.

We presented a formal verification technique based on the theorem proving method. We developed a unified approach for functional and timing verification of real-time systems implemented on FPGA. It is a scalable technique.

## 3.2. On Formal Verification of RTL Design

RTL verification comprises of attaining a genuine confidence that a circuit will function correctly, under the assumption that no manufacturing fault is present. There is a significant work done on the RTL verification for digital ICs development. Verification is, actually, a bottleneck in the time-to-market for integrated circuit development. Different commercial CAD tools such as Logic Simulator by Xilinx [46] are available for conducting the timing simulation of designs targeted for FPGA. An approach for circuit-level timing simulation for FPGA was presented by Robertson et. al, [47]. Their proposed method performs the static timing analysis and it is applicable at the design phase of FPGA IC. Our method, on the contrary targets the verification of timing properties of applications that are mapped to FPGA IC.

H. Mangassarian et al. [48] proposed a technique that reduces the verification effort in automated RTL debuggers. Automated debuggers use formal tools such as Boolean satisfiability (SAT) [49] and reduce the debugging problem into a propositional formula. Authors have presented an iterative algorithm to compute the dominance relationships between RTL blocks. By leveraging this dominance relationship, the number of calls to the formal engine are reduced and the run-time of each call to formal engine was accelerated. Their work is different from ours since they are targeting the RTL verification of VLSI designs which contain typically millions of gates. Our work does not target the VLSI design, rather we deal with the RTL verification of safety-critical applications mapped to FPGA IC. Joakim Urdahl et al., [50] have presented a formal verification methodology for system-on-ship (SoC) designs. Their idea was to create a formal link between system models and concrete RTL implementation only by employing the standard techniques of property checking. They used the path predicate abstraction of an RTL design based on standard property language. Their proposed abstraction mechanism is related to the notion of a stuttering bisimulation which is employed in the field of theorem proving, but they are not relying on theorem proving. The authors claim that the expressive power of modern property languages such as SVA and influence of state-of-art property checking technology such as SPIN helped to reduce the semantic gap between the system level and RTL description. By conducting two comprehensive industrial case studies, they proved the practical feasibility of proposed approach.

Our contribution is the verification technique for RTL design of real-time applications that are implemented using FPGA platform. We devised seperate proof obligation templates for functional and timing verification of RTL circuit design. Our timing verification approach is more direct and effecient as it does not require to open up the implementation transition system.

# 4. A FORMAL VERIFICATION METHODOLOGY FOR REAL-TIME FPGA

## 4.1. Introduction

A real-time system is one that must process information and respond within a deadline. Field Programmable Gate Array (FPGA) based control offers several benefits over software control including hardware acceleration and more predictable timing behavior. Use of FPGA-based control in conjunction with software control is therefore a growing trend in real-time system design. Example applications of FPGA control in safety-critical real-time applications include high consequence industrial systems [10] [51], avionics systems [52], and nuclear power plants [53].

FPGA based real-time system design and control is a very active area of development and research in both industry and academia [54] [55] [56]. The predominant approach used in industry to check for system (hardware or software) errors is testing [57] [58]. Testing is very effective in finding bugs, but cannot show the absence of bugs. Formal verification (FV) approaches are effective in finding deep, corner case bugs, and can provide safety guarantees. Therefore, FV approaches have become indispensable in the design and validation of safety-critical systems. This paper presents an FV approach for real-time FPGA controllers that find application in safety-critical systems.

There are two aspects to real-time verification, functional verification and timing verification. There are many approaches for formal functional verification of FPGA-based designs. There are also many approaches for FV of both functional and timing properties of software. However, FV approaches targeted at functional and timing verification of FPGA are currently not available. This paper presents a unified formal approach for functional and timing verification of FPGA. The approach is applicable to register transfer level (RTL) designs.

## 4.2. Related Work

UPPAAL [59] and Kronos [6] are formal model checkers that have been very successful in the verification of real-time systems. They are primarily targeted at system-level models. Another FV tool is Epsilon [22], which has been used to reason about high-level models of communication protocols. The above mentioned tools have not been demonstrated to check the correctness of real-time Register Transfer Level (RTL) hardware designs, which is typically the level at which digital controllers for FPGA are designed. There is however, a big complexity gap between system-level models and low-level RTL.

Robertson et al., [23] present an approach to perform circuit-level timing simulation for FPGA, that plays the role of static timing analysis (STA), and is applicable at the phase of designing the FPGA IC. In contrast, our approach targets the verification of timing properties of applications that are mapped on to the FPGA IC. Many commercial CAD tools such as the Logic Simulator by Xilinx [46] are available and can be used to perform timing simulation for designs targeted for FPGA. Such approaches are similar to functional testing and rely on the use of input test vectors and can be classified as testing-based approaches to timing verification. In contrast, our approach is based on formal verification, and is a proof-based approach to timing verification.

Dubasi et al. [60] have proposed Timed Well-Founded Equivalence Bisimulation (TWEB) refinement, a notion of correctness for real-time systems. They have shown how to apply TWEB refinement for verification of low-level object code. Their approach is to perform symbolic simulation on the object code and construct its timed transition system (TTS), which is then analyzed using algorithms that check TWEB refinement. The resulting TTS can be quite large with millions of transitions. The approach proposed in the paper also uses a variation of TWEB refinement. However, the approach exploits the property of FPGA that every step/transition has the same delay equal to the clock period delay, to provide a more efficient technique that does not simulate the design and does not open up the TTS. Instead, the proposed approach reduces verification to a series of proof obligations that can be checked using a Satisfiability Modulo Theories (SMT) solver [18] such as Z3 [17]. Note that the delay property is true for a wide-class of systems. There are many systems that may not satisfy the aforementioned property such as those based on multi-clock designs, which will be studied as part of future work.

Shuja et al. [61] have applied refinement-based verification for pacemaker control. They only verify object code. Their approach is not targeted at FPGA. They also only perform functional verification and have not addressed timing verification. There are numerous formal approaches targeted at verification of RTL designs [62] [63]. However, these approaches do not address timing verification. In contrast, this paper provides a unified approach for verification of both functional and timing requirements for RTL/FPGA designs.

An UPPAAL-based tool to check refinement between specifications of real-time systems has been developed [31]. A bisimulation relation for real-time systems with priorities has been developed and provides a method for encoding and verifying the problem using UPPAAL [64]. The above refinement approaches for real-time systems are targeted at high-level models and do not consider stuttering and refinement maps. The proposed approach incorporates stuttering and refinement maps and is, therefore, unique in this regard and applicable to the verification of low-level implementations such as RTL for FPGA.

## 4.3. Background: Timed Well-Founded Simulation Refinement (TWFS)

In FV, the implementation is the design artifact to be verified. The specification is a mathematical model that defines the correct behaviors of the design. In this approach, Timed Transition Systems (TTSs) are used as the modeling framework for both implementations and specifications. The definition for TTS is provided below.

**Definition 2** *A TTS $M$ is a 3-tuple $\langle S, R, L\rangle$, where $S$ is the set of states, $R$ is the transition relation that defines the state transitions, and $L$ is a labeling function that defines what is visible at each state. $R$ is of the form $\langle w, v, lb, ub\rangle$, where $w$, $v \in S$ and $lb, ub \in \Re$. $lb$ and $ub$ indicate the lower bound and the upper bound on the time delay of the transition, respectively.*

In the definition above, if the delay bounds are excluded, then it is a transition system (TS). There are two benefits to using TTS over temporal logic to define specifications. First, specifications can be easily encoded as a simple TTS, while they may be more convoluted if expressed as temporal properties. This may not always be true, but is true for many applications including the case studies in the paper. The second and more important benefit is that if the specification is a TTS, then refinement-based approaches can be used for verification. Refinement-based approaches typically scale very well for low-level implementations such as FPGA. If the specification is expressed as temporal properties, then model checking tools need to be used for verification. Model checking suffers from state explosion limitations and therefore does not scale well for verification of low-level artifacts such as FPGA.

For verification, we use the SMT-LIB language [18] to describe the circuit TTS and the specification TTS. This allows us to use SMT solvers such as Z3 [17] to discharge the verification proof obligations. The digital circuit corresponding to an FPGA implementation can be modeled as a TTS (called the implementation TTS) by modeling the RTL circuit as an SMT-LIB function that computes the next state and outputs of the circuit, given the current state and inputs of the circuit. A call of the function is a step of the circuit. Therefore, this function defines the transition relation. The time delay associated with a step is always the time period of one clock cycle. Therefore, the lower bound and upper bound delays corresponding to a transition are both always the time period of one clock cycle. The set of states of the circuit are characterized by circuit invariants that are specified as predicates in the SMT-LIB language.

The functional verification methodology for FPGA is developed based on Well-Founded Simulation (WFS) refinement [65] as the notion of correctness. If it can be proved that an implementation transition system (TS) is a WFS refinement of a specification TS, then every behavior of the implementation is guaranteed to match a behavior of the specification. An important feature of WFS is that it accounts for stuttering, which is the phenomenon where multiple but finite transitions of the implementation will match the same specification transition. WFS definition is given below.

**Definition 3** [65] (*Well-Founded Simulation (WFS)*) $B \subseteq S \times S$ is a *well-founded simulation* on TS $\mathcal{M} = \langle S, R, L \rangle$ iff:

(Wfs1) $\langle \forall s, w \in S : sBw \ : \ L(s) = L(w) \rangle$

(Wfs2) There exists functions, $\mathit{rankt} : S \times S \to W$,

$\mathit{rankl} : S \times S \times S \to \mathbb{N}$,

such that $\langle W, \lessdot \rangle$ is well-founded, and

$\langle \forall s, u, w \in S :: sBw \ \wedge \ sRu \ :$

(a) $\langle \exists v \ : \ wRv \ \wedge \ uBv \rangle \ \vee$

(b) $(uBw \ \wedge \ \mathit{rankt}(u, w) \lessdot \mathit{rankt}(s, w)) \ \vee$

(c) $\langle \exists v : wRv : sBv \wedge \mathit{rankl}(v, s, u) < \mathit{rankl}(w, s, u) \rangle \rangle$

**Definition 4** [65] (*WFS Refinement*) Let $M = \langle S,R,L \rangle$, $M' = \langle S', R', L' \rangle$, and $r : S \to S'$. $M$ is a simulation refinement of $M'$ with respect to refinement map $r$, written $M \sqsubseteq_r M'$, if there exists a relation, $B$, such that $\langle \forall s \in S :: sB(r.s)\rangle$ and $B$ is a WFS on the TS $\langle S \uplus S', R \uplus R', \mathcal{L}\rangle$, where $\mathcal{L}.s = L'(s)$ for $s$ an $S'$ state and $\mathcal{L}.s = L'(r.s)$ otherwise.

In the above definitions, $M$ and $M^{'}$ are the implementation TS and specification TS, respectively. $r$ is the refinement map, which is a function used to map implementation states to specification states. Informally, the definitions state that every implementation transition should either be a stuttering transition (Definition 2, case c) or a non-stuttering transition (Definition 2, case a). Definition 2, case b corresponds to stuttering on the specification side. For the problems considered, the specifications are very simple and high-level, and stuttering does not happen on the specification side. Therefore, case b is ignored. A stuttering implementation transition ($w$, $v$) is one in which both $w$ and $v$ map to the same specification state, say $s$, i.e., $r(w) = r(v) = s$.

A non-stuttering implementation transition ($w$, $v$) is one that matches with a specification transition. More specifically, if $r(w) = s$ and $r(v) = u$, then ($s$, $u$) should be a specification transition. If $r(w)$ or $r(v)$ do not correspond to specification states or if ($s$, $u$) is not a transition of the specification, then these situations correspond to errors in the circuit. Two rank functions are used to distinguish stutter from deadlock (essentially infinite stutter). When stuttering occurs, the rank function is designed in such a way that its value should decrease. Since the rank functions are based on well-founded structures, they cannot decrease forever. Deadlock bugs will, therefore, be detected when there is stuttering behavior but the rank does not decrease. *rankt* and *rankl* correspond to stuttering on the specification side and the implementation side, respectively. Since we are not concerned with stuttering on the specification side, we ignore *rankt*. Also, for the following discussions, we use just *rank(w)* to denote *rankl*$(w, s, u)$. We can ignore the specification states $s$ and $u$ because an implementation state gets mapped only to one specification state (as stuttering does not occur on the specification side).

The notion of WFS refinement does not account for timing requirements. The notion of Timed WFS (TWFS) refinement accounts for timing requirements and is given below. Dubasi et al. [60] proposed Timed Well-Founded Equivalence Bisimulation (TWEB) refinement as an extension of Well-Founded Equivalence Bisimulation (WEB) refinement to handle timing requirements. A similar extension can be applied to WFS refinement to obtain TWFS refinement, which is what is given below.

**Definition 5** [60] $M_I$ is a TWFS refinement of $M_S$ if:

1. $M_I$ is a WFS refinement of $M_S$ w.r.t. refinement map $r$.

2. Let $MM_I$ be the marked TTS of $M_I$ w.r.t. $M_S$. Then, for every non-stuttering transition of $MM_I$ $\langle w_a, w_b\rangle$, and for every stuttering segment $\pi$ of $\langle w_a, w_b\rangle$, the following should be satisfied:

$$lb_s^{\langle r(w_a),r(w_b)\rangle} \leq \sum_{p\in\pi} lb_i^p \leq \sum_{p\in\pi} ub_i^p \leq ub_s^{\langle r(w_a),r(w_b)\rangle}$$

In the above, $M_I$ and $M_S$ are the implementation TTS and specification TTS, respectively. $MM_I$ is a TTS obtained by marking the transitions in $M_I$ as either stuttering transitions or non-stuttering transitions. An $MM_I$ corresponding to the SM specification (Figure 2) is shown in Figure 4.5(a). Figure 4.5(b) shows the specification states that the implementation states get mapped to under the refinement map. In Figure 4.5(a), transitions (13,1) and (1,2) are examples of non-stuttering and stuttering transitions, respectively. A stuttering segment $\pi$ in $MM_I$ is a finite sequence of transitions of $MM_I$ such that the last transition in the sequence is a non-stuttering transition and all others are stuttering transitions. Additionally, this sequence should be preceded by another non-stuttering transition in $MM_I$. (1,2), (2,3) and (3, 6) is a stuttering segment that corresponds to the specification transition ($s_1$, $s_2$). $lb_s^{\langle x,y\rangle}$ and $ub_s^{\langle x,y\rangle}$ refer to the delay lower bound and delay upper bound of the specification transition $\langle x, y\rangle$. Similarly, $lb_i^{\langle x,y\rangle}$ and $ub_i^{\langle x,y\rangle}$ refer to the delay lower bound and delay upper bound of the implementation transition $\langle x, y\rangle$. TWFS refinement essentially requires that the combined delay of all the transitions in every stuttering segment of the implementation should lie within the delay bounds of the corresponding specification transition.

## 4.4. Case Studies and Specifications

This section describes three case studies and provides the TTS specification for each of the case studies.

### 4.4.1. Pulse Width Modulation (PWM) Control for DC Motor

PWM is a technique by which the average amount of voltage/power supply to any device can be varied. PWM has a very broad range of applications including speed control of DC motors, power electronics and DC/DC converters, which are further used in many safety-critical applications

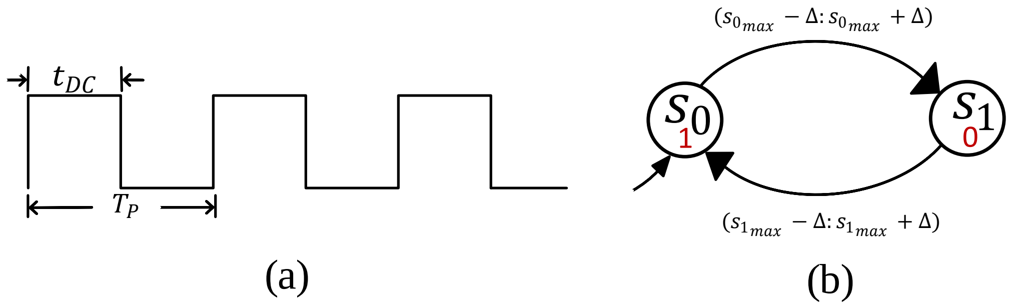
$t_{DC}$
$T_P$
(a)
$(s_{0_{max}} - \Delta : s_{0_{max}} + \Delta)$
$s_0$
1
$s_1$
0
$(s_{1_{max}} - \Delta : s_{1_{max}} + \Delta)$
(b)

Figure 4.1. (a) Pulse Width Modulated Signal (PWM Wave), (b) TTS Specification for PWM

such as aerospace, vehicular electronic systems, and process control in industry.

The PWM waveform is shown in Figure 4.1 and must be generated by a controller (software or FPGA) to control a DC motor. PWM is a binary signal with two discrete values 1 and 0, which is applied to the DC motor to vary its speed. The width of the signal can be varied between 0 and the time period of the pulse indicated by $T_P$ in Figure 4.1. The time during which the signal is high is called the duty cycle ($t_{DC}$). The duty cycle determines the amount of average power delivered and hence controls the speed of the DC motor. The greater the duty cycle, the more average power is delivered.

The specification for PWM is given as a timed transition system (TTS) in Figure 4.1. It has two discrete states $s_0$ and $s_1$. Initially, the signal is high in $s_0$ state. The delay lower bound on transition ($s_0$, $s_1$) is $s_{0_{max}}= t_{DC}$ indicating that the signal must be high for a time period equal to $t_{DC}$. Then the signal becomes low in $s_1$ state. The delay lower bound on transition ($s_1$, $s_0$) is $s_{1_{max}}=$ $T_P$ - $t_{DC}$ indicating that the signal must be low for a time period equal to $T_P$ - $t_{DC}$. This sequence repeats. Some tolerance is allowed in the delay bounds, which is incorporated as $\Delta$ in the lower and upper delay bounds on both transitions. The PWM specification TTS follows. The set of states $S$=($s_0$, $s_1$). The transition relation $R$=($\langle s_0, s_1, s_{0_{max}} - \Delta, s_{0_{max}} + \Delta\rangle$, $\langle s_1, s_0, s_{1_{max}} - \Delta, s_{1_{max}} + \Delta\rangle$). The labeling function is defined as $L(s_0)$=1 and $L(s_1)$=0.

### 4.4.2. Stepper Motor (SM) Control

A SM is a brushless DC motor that completes one full rotation by taking multiple discrete steps. Stepper motors are used in a wide-class of safety-critical applications including but not limited to industrial process control, medical applications, military applications, robotics, space applications, throttle valve control for vehicles, etc. FPGA based control of SMs is a growing trend

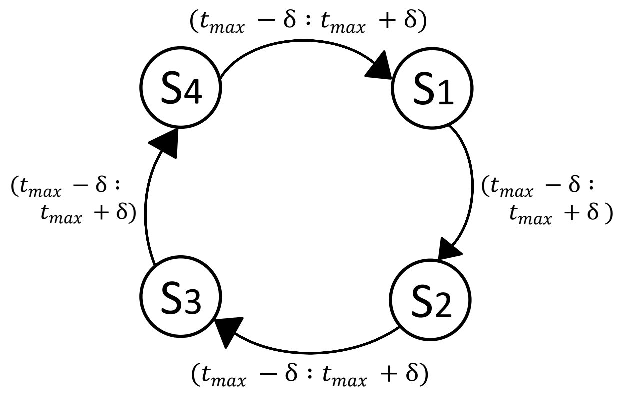

Figure 4.2. TTS Specification for SM Control

in research and development.

A four-lead SM can be made to rotate by applying a repeating sequence of predefined bit patterns such as 0001, 0010, 0100, 1000, 0001 ... . The delay between bit patterns determines the speed of the motor. This bit pattern is generated by FPGA to control the rotation of the motor. In Figure 4.2, the TTS specification is given for full stepping control. The speed of rotation of the SM is determined by this transition delay $t_{max}$. The timed transition system is given as below;

$$S = \{\ s_1,\ s_2,\ s_3,\ s_4\ \}$$

$$\begin{aligned} R = \{\ &\langle s_1, s_2, t_{max} - \delta, t_{max} + \delta\rangle, \\ &\langle s_2, s_3, t_{max} - \delta, t_{max} + \delta\rangle, \\ &\langle s_3, s_4, t_{max} - \delta, t_{max} + \delta\rangle, \\ &\langle s_4, s_1, t_{max} - \delta, t_{max} + \delta\rangle\ \} \end{aligned}$$

The labeling function defines the values for each state as the following bit patterns: $L(s_1)$=0001, $L(s_2)$=0010, $L(s_3)$=0100, $L(s_4)$=1000.

Refinement-based functional verification of SM control has been presented [66]. The approach is specific to SM control and does not address timing verification. The approach in this paper is also refinement-based, but is more general and applicable not only to SM control, but any form of control that can be specified using a TTS. In addition, timing verification is also addressed.

### 4.4.3. Pacemaker (PM) Control

A pacemaker is an implantable medical device that provides electrical impulses to the heart muscles to maintain an adequate heart rate. Pacemakers are used to treat arrhythmias. The clinical

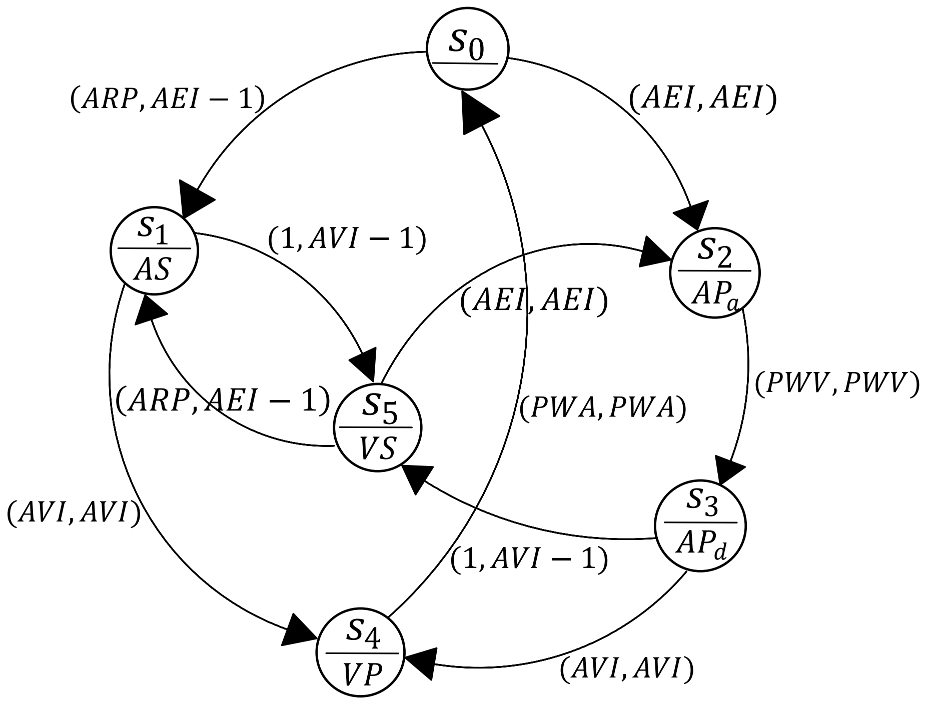

Figure 4.3. TTS Specification for Pacemaker (PM)

requirements of a DDD mode pacemaker are given by Boston Scientific [24], and can be specified as a TTS. Shuja et al. [61] have provided a TTS specification for these requirements, which is shown in Figure 4.3.

The states of the TTS are marked with atomic propositions AS, AP$_a$, AP$_d$, VS and VP, which are predicates critical to the operation of the pacemaker. Atrial Sense (AS) and Ventricle Sense (VS) represent the sense event of the electrical pulse in the atrial and ventricle chambers of the heart, respectively. Atrial Pace (AP) and Ventricle Pace (VP) represent the pace event of the pacemaker, i.e., the application of an electrical pulse by the pacemaker to the atrial and ventricle chambers, respectively. AP$_a$ and AP$_d$ represent the assertion and desertion of the AP signal, respectively. The very high-level idea of the operation of a pacemaker is that, if AS or VS is not detected, i.e., if the heart has not generated the electrical pulses that it should have in a given time period, then the pacemaker will compensate by generating these signals. As such the timing cycles of the pacemaker operation are quite complex. The TTS specification has a number of timing requirements captured as upper and lower limits on the state transitions. These limits are specified using parameters such as ARP, AEI, etc. These parameters are essentially constant values and their significance is described in detail in [61] [67]. The formal TTS specification is given below. The set of states $S$=($s_0$, $s_1$, $s_2$, $s_3$, $s_4$, $s_5$). The transition relation is;

$$\begin{aligned} R = (&\langle s_0, s_1, ARP, AEI-1\rangle, \langle s_0, s_2, AEI, AEI\rangle,\\ &\langle s_2, s_3, PWV, PWV\rangle, \langle s_3, s_4, AVI, AVI\rangle,\\ &\langle s_1, s_4, AVI, AVI\rangle, \langle s_1, s_5, 1, AVI-1\rangle,\\ &\langle s_5, s_1, ARP, AEI-1\rangle, \langle s_5, s_2, AEI, AEI\rangle,\\ &\langle s_3, s_5, 1, AVI-1\rangle, \langle s_4, s_0, PWA, PWA\rangle). \end{aligned}$$

The labeling function defines the values of the five atomic propositions (AS, AP$_a$, AP$_d$, VS, VP) for each state: $L(s_0)$=00000, $L(s_1)$=10000, $L(s_2)$=01000, $L(s_3)$=00100, $L(s_4)$=00001, $L(s_5)$=00010. Note that the pacemaker specification TTS is non-deterministic. The pacemaker case study is included to demonstrate that the approach is also applicable to a wide class of systems, including ones that require complex timing cycles such as this one.

## 4.5. Functional Verification

The contribution of this work is a set of decidable proof obligation templates that when applied to a specification TS and a corresponding implementation circuit, amounts to WFS refinement verification. The templates exploit the fact that the specifications are simple and have few states and transitions.

There are some challenges in verifying the lower level implementation against high level specification. The proof obligation templates also entail the solutions addressing these challenges. The first challenge in checking the implementation TS against the specification TS is that the states of the two TSs can look very different. The specification TS is very simple and high-level (as with the case studies), and this is how they should be. Whereas, the controller circuit implementation is typically defined at the RTL and is very low-level. The circuit will typically have data registers, counters, flags and control registers. The values of all these components will correspond to the state of a circuit. The question is therefore, how to compare implementation and specification states. WFS refinement employs refinement maps to address this problem. Refinement maps are functions that map implementation states onto specification states. The first proof obligation template is for stuttering transitions:

$$\forall w :: [C_k(w, \vec{i}) \wedge r(w){=}s \wedge v{=}\mathit{impl\text{-}ckt}(w, \vec{i})]$$
$$\longrightarrow [r(v){=}s \wedge \mathit{rank}(v) < \mathit{rank}(w)]$$

In the above, $s$ is a specification state and $w$ is an implementation state. $v$ is the successor state of $w$. *rank(w)* and *rank(v)* are the rank values of implementation states $w$ and $v$, respectively. $r$ is the refinement map. For the case studies considered (which represent a large class of FPGA controllers), refinement maps can be defined as simple projection functions. For example, for the SM case study, the refinement map is defined as a function that projects the four motor control bits from the SM controller circuit state. The four control bits give the specification state. Vector $\vec{i}$ is the circuit inputs. $\mathit{impl\text{-}ckt}(w,\ \vec{i})$ is the function corresponding to the implementation circuit that takes as input the current state and values to circuit inputs and gives the next circuit state $v$. $C_k$ corresponds to the conditions on the implementation state and circuit inputs under which the circuit stutters in the specification state $s$. *One stuttering proof obligation should be generated for every state of the specification TS.*

The second challenge in the functional verification methodology is that typically the implementation controller circuit TS will have millions of states and transitions, whereas the specification TSs have a much smaller number of transitions. The above is true with all the case studies. The consequence is that multiple but finite transitions of the implementation will match to the same specification transition. This phenomenon is known as stuttering and is incorporated by WFS refinement. The second proof obligation template is for non-stuttering transitions:

$$\forall w :: [C_k(w, \vec{i}) \wedge r(w){=}s \wedge v{=}\mathit{impl\text{-}ckt}(w, \vec{i})]$$
$$\longrightarrow [r(v){=}u]$$

For the non-stuttering proof obligation, $C_k$ corresponds to the conditions on the implementation state and circuit inputs under which the circuit transition should match with the specification transition ($s$, $u$). *One non-stuttering proof obligation should be generated for every transition of the specification TS.* Based on these proof obligation templates, we derived the proof obligations for each of the case study as given below in the following subsections.

### 4.5.1. Proof Obligations for Stepper Motor

**Stuttering POs**

$$(r(w){=}s_1) \quad \wedge \quad (0 \leq w.counter \leq t_{max}) \;\longrightarrow\; \{\ (r(v){=}s_1) \quad \wedge \quad rank(v) < rank(w)\ \}$$

$$(r(w){=}s_2) \quad \wedge \quad (0 \leq w.counter \leq t_{max}) \;\longrightarrow\; \{\ (r(v){=}s_2) \quad \wedge \quad rank(v) < rank(w)\ \}$$

$$(r(w){=}s_3) \quad \wedge \quad (0 \leq w.counter \leq t_{max}) \;\longrightarrow\; \{\ (r(v){=}s_3) \quad \wedge \quad rank(v) < rank(w)\ \}$$

$$(r(w){=}s_4) \quad \wedge \quad (0 \leq w.counter \leq t_{max}) \;\longrightarrow\; \{\ (r(v){=}s_4) \quad \wedge \quad rank(v) < rank(w)\ \}$$

**Non-Stuttering POs**

$$(r(w){=}s_1) \quad \wedge \quad (w.counter = t_{max}) \;\longrightarrow\; (r(v){=}s_2)$$

$$(r(w){=}s_2) \quad \wedge \quad (w.counter = t_{max}) \;\longrightarrow\; (r(v){=}s_3)$$

$$(r(w){=}s_3) \quad \wedge \quad (w.counter = t_{max}) \;\longrightarrow\; (r(v){=}s_4)$$

$$(r(w){=}s_4) \quad \wedge \quad (w.counter = t_{max}) \;\longrightarrow\; (r(v){=}s_1)$$

### 4.5.2. Proof Obligations for PWM Control

**Stuttering POs**

$$(r(w){=}s_0) \quad \wedge \quad (0 \leq w.PWM_c \leq s_0\text{-}max) \;\longrightarrow\; \{\ (r(v){=}s_0) \quad \wedge \quad rank(v) < rank(w)\ \}$$

$$(r(w){=}s_1) \quad \wedge \quad (s_0\text{-}max \leq w.PWM_c \leq s_1\text{-}max) \;\longrightarrow\; \{\ (r(v){=}s_1) \quad \wedge \quad rank(v) < rank(w)\ \}$$

**Non-Stuttering POs**

$$(r(w){=}s_0) \quad \wedge \quad (w.PWM_c = s_0\text{-}max) \;\longrightarrow\; (r(v){=}s_1)$$

$$(r(w){=}s_1) \quad \wedge \quad (w.PWM_c = s_1\text{-}max) \;\longrightarrow\; (r(v){=}s_0)$$

### 4.5.3. Proof Obligations for Pacemaker

**Stuttering POs**

$$(r(w){=}s_0) \;\wedge\; (0 \leq w.t_v \leq \text{ARP}) \;\longrightarrow \{\; (r(v){=}s_0) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_0) \;\wedge\; (\text{ARP} \leq w.t_v \leq \text{AEI-1}) \;\wedge\; (A_{in}{=}0) \;\longrightarrow \{\; (r(v){=}s_0) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_1) \;\wedge\; (1 \leq w.t_a \leq \text{AVI} - 1) \;\wedge\; (V_{in} = 0) \;\longrightarrow \{\; (r(v){=}s_1) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_3) \;\wedge\; (1 \leq w.t_a \leq \text{AVI} - 1) \;\wedge\; (V_{in} = 0) \;\longrightarrow \{\; (r(v){=}s_3) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_4) \;\wedge\; (AVI \leq w.t_a \leq \text{P}WA) \;\longrightarrow \{\; (r(v){=}s_4) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_5) \;\wedge\; (0 \leq w.t_a < \text{A}RP) \;\longrightarrow \{\; (r(v){=}s_5) \;\wedge\; rank(v) < rank(w) \;\}$$

$$(r(w){=}s_5) \;\wedge\; (\text{ARP} \leq w.t_v \leq \text{AEI-1}) \;\wedge\; (A_{in}{=}0) \;\longrightarrow\; (r(v){=}s_1) \;\wedge\; rank(v) < rank(w) \;\}$$

**Non-Stuttering POs**

$$(r(w){=}s_0) \;\wedge\; (\text{ARP} \leq w.t_v \leq \text{AEI-1}) \;\wedge\; (A_{in}{=}1) \;\longrightarrow\; (r(v){=}s_1)$$

$$(r(w){=}s_0) \;\wedge\; (w.t_v = \text{AEI}) \;\longrightarrow\; (r(v){=}s_2)$$

$$(r(w){=}s_1) \;\wedge\; (1 \leq w.t_a \leq \text{AVI-1}) \;\wedge\; (V_{in}{=}1) \;\longrightarrow\; (r(v){=}s_5)$$

$$(r(w){=}s_1) \;\wedge\; (w.t_a = \text{AVI}) \;\longrightarrow\; (r(v){=}s_4)$$

$$(r(w){=}s_2) \;\wedge\; (w.t_v = \text{PWV}) \;\longrightarrow\; (r(v){=}s_3)$$

$$(r(w){=}s_3) \;\wedge\; (1 \leq w.t_a \leq \text{AVI-1}) \;\wedge\; (V_{in}{=}1) \;\longrightarrow\; (r(v){=}s_5)$$

$$(r(w){=}s_3) \;\wedge\; (w.t_a = \text{AVI}) \;\longrightarrow\; (r(v){=}s_4)$$

$$(r(w){=}s_4) \;\wedge\; (w.t_a = \text{PWA}) \;\longrightarrow\; (r(v){=}s_0)$$

$$(r(w){=}s_5) \;\wedge\; (\text{ARP} \leq w.t_v \leq \text{AEI-1}) \;\wedge\; (A_{in}{=}1) \;\longrightarrow\; (r(v){=}s_1)$$

$$(r(w){=}s_5) \;\wedge\; (w.t_v = \text{AEI}) \;\longrightarrow\; (r(v){=}s_2)$$

The proof obligations can be directly expressed in the SMT-LIB language and can be checked using an SMT solver. The templates essentially correspond to cases (a) and (c) of Definition 2 and are obtained by symbolic manipulation of the definition and case analysis. The correctness of the set of proof obligations depends on the $C_k$ conditions covering all the implementation states. The set of states of an implementation controller circuit that are reachable from the reset states (states the circuit is initialized to after system reset) are typically characterized by one or more invariants. The invariants for the three case studies are given in Figure 4.4.

$$
\begin{aligned}
\mathrm{INV}_{\mathrm{PWM}}: \;& [(r(w) = s_0) \wedge (0 \leq w.\mathrm{PWM_C} \leq s_0\text{-max})] \\
& \vee\; [(r(w) = s_1) \wedge (s_0\text{-max} \leq w.\mathrm{PWM_C} \leq s_1\text{-max})] \\
\mathrm{INV}^1_{\mathrm{SM}}: \;& (0 \leq w.counter \leq t_{max}) \\
\mathrm{INV}^2_{\mathrm{SM}}: \;& (r(w) \in S_s) \qquad \text{where } S_s = \{s_1, s_2, s_3, s_4\} \\
\mathrm{INV}_{\mathrm{PM}}: \;& [\;[r(w) = s_0) \wedge (0 \leq w.t_v \leq \mathrm{AEI})] \\
& \vee [(r(w) = s_1) \wedge (0 \leq w.t_a \leq \mathrm{AVI})] \\
& \vee [(r(w) = s_2) \wedge (\mathrm{AEI} \leq w.t_v \leq \mathrm{AEI} + \mathrm{PWV})] \\
& \vee [(r(w) = s_3) \wedge (0 \leq w.t_a \leq \mathrm{AVI})] \\
& \vee [(r(w) = s_4) \wedge (\mathrm{AVI} \leq w.t_a \leq \mathrm{AVI{+}PWA})] \\
& \vee [(r(w) = s_5) \wedge (0 \leq w.t_v \leq \mathrm{AEI})]\;]
\end{aligned}
$$

Figure 4.4. Invariants of PWM, SM and Pacemaker, Respectively

If $K$ proof obligations were generated using the templates, the following additional proof obligation should be checked, which ensures that the conditions cover all the implementation states.

$$\bigvee_{k=1}^{K} [C_k(w, \vec{i}) \wedge r(w){=}s_k] \rightarrow Inv_{ckt}$$

The above proof obligation for the PWM case study is given below.

$$
\begin{aligned}
&\big\{[r(w){=}s_0 \wedge (0 \leq w.\mathrm{PWM}_c < s_0\text{-}max)] \vee \\
&\;[r(w){=}s_1 \wedge (s_0\text{-}max \leq w.\mathrm{PWM}_c < s_1\text{-}max)] \vee \\
&\;[r(w){=}s_0 \wedge (w.\mathrm{PWM}_c = s_0\text{-}max)] \vee \\
&\;[r(w){=}s_1 \wedge (w.\mathrm{PWM}_c = s_1\text{-}max)]\big\} \\
&\Longrightarrow \quad \big\{[r(w){=}s_0 \wedge (0 \leq w.\mathrm{PWM}_c \leq s_0\text{-}max)] \vee \\
&\qquad\quad [r(w){=}s_1 \wedge s_0\text{-}max \leq w.\mathrm{PWM}_c \leq s_1\text{-}max)]\big\}
\end{aligned}
$$

To summarize, if the functional behavior of an FPGA controller circuit can be described as a simple TS, then the controller circuit can be verified against the specification TS by generating

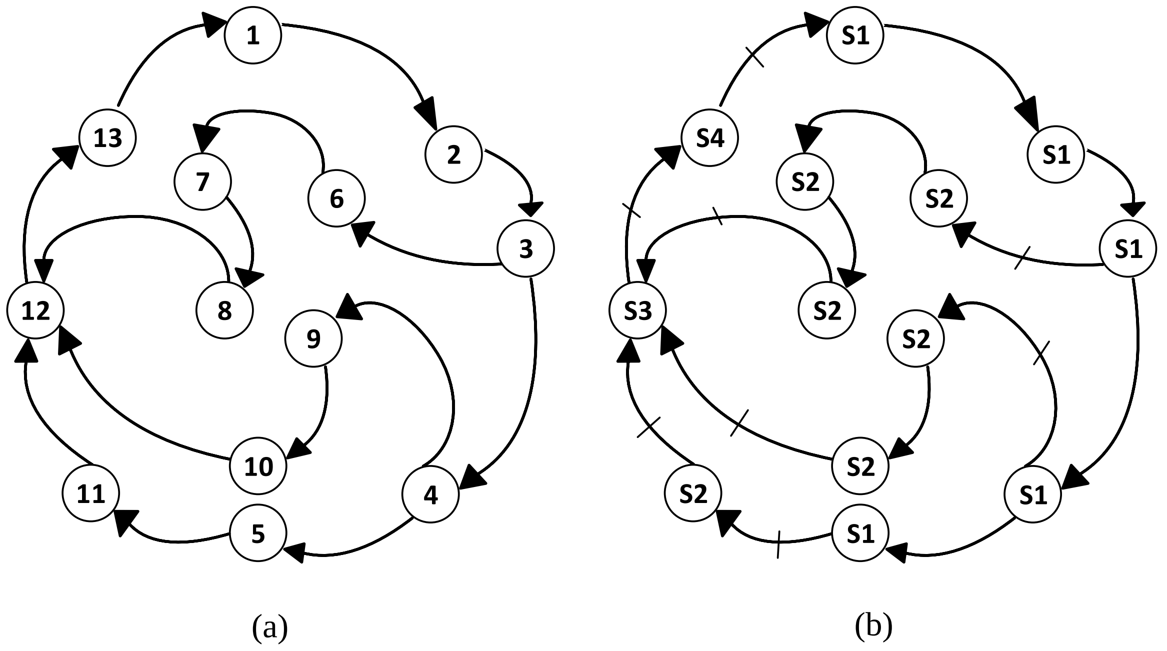

Figure 4.5. (a) Implementation TTS (b) Refinement Map Applied to Get Specification States

and checking the proof obligation templates developed in this Section. Verifying the stuttering and non-stuttering proof obligations amounts to the functional verification of the implementation system.

## 4.6. Timing Verification

Dubasi et al. [60] applied TWEB refinement for verification of real-time object code programs. The idea was to use symbolic simulation to extract the TTS corresponding to object code (the implementation TTS). They developed algorithms that would then check the implementation TTS against the specification using TWEB refinement as the notion of correctness. One of the challenges was that the implementation TTS of object code had millions or more transitions. This was overcome using abstraction techniques. The extraction of the implementation TTS is quite complex and time consuming.

Our proposed approach exploits two properties of FPGA to perform verification without computing the object code TTS and is, therefore, very efficient. The first property is that the delay associated with any transition in the FPGA implementation TTS is a constant and equal to the FPGA clock period. In contrast, for object code, the time required for the execution of an instruction varies and depends on the hardware platform. The second property is that all the transitions of the FPGA TTS can be modeled using a function that corresponds to the hardware controller design. In contrast, for object code, numerous such functions are required, one for each instruction. The first FPGA property is employed by the methodology based on the observation that since the delay of all implementation transitions are the same and equal to the clock period, the delay of a stuttering segment can be calculated if the length of the stuttering segment is known. The second FPGA property is employed by the methodology by reducing TWFS verification to a set of proof obligations that can be checked on the FPGA function using an SMT solver.

The proposed approach overloads the use of *rank* for computing the length of stuttering segments. The idea is to devise a *rank* function whose value decreases by a constant value in every stuttering implementation transition. Typically, *rank* functions are computed using the values of one or more timers/counters, whose values are increasing or decreasing by a constant positive value. Therefore, *rank* functions can be devised to satisfy this property. There are three high-level steps for timing verification. The first step is to check that for all stuttering transitions, the *rank* decreases by a constant value. This can be verified by modifying the stuttering proof obligation template (from Section 4.5) as follows:

$$\begin{aligned}\forall w :: [C_k(w, \vec{i}) \wedge r(w){=}s \wedge v{=}\mathit{impl\text{-}ckt}(w, \vec{i})]\\ \rightarrow [r(v){=}s \wedge (\mathit{rank}(w)\text{-}\mathit{rank}(v){=}C_{rank})]\end{aligned}$$

In the above, $w$ is the implementation state and $v$ is $w$'s successor. The original stuttering proof obligation template required that *rank(v)* decrease w.r.t *rank(w)*. In the modified template shown above, the difference *rank(w)-rank(v)* is required to be a constant positive value $C_{rank}$.

The second step is to verify the *entry-state* property. *entry-state* of a stuttering segment is the first state of a stuttering segment. In Figure 4.5(a), implementation state 1 is the *entry-state* for stuttering segments corresponding to specification transition ($s_1$, $s_2$). All *entry-state*s corresponding to a specification state will have the same rank value. This property is used for stuttering segment length estimation. The *entry-state* rank values are to be verified using the proof obligation template given on the next page. In the template below, ($w$, $v$) is an implementation transition that maps to specification transition ($u$, $s$). Therefore, ($w$, $v$) is a non-stuttering transition and $v$ corresponds to entry states of stuttering segments that stutter in specification state $s$. $\mathit{entry\text{-}state\text{-}rank}_s$ is the value that the *rank* function should have for entry states corresponding to specification state $s$. Note that every specification state will have a single corresponding *entry-state-rank* value.

$$\forall w, v :: [r(w){=}u \wedge v{=}\mathit{impl\text{-}ckt}(w, \vec{i}) \wedge r(v){=}s]$$
$$\rightarrow \mathit{rank}(v){=}\mathit{entry\text{-}state\text{-}rank}_s]$$

For example, for the SM case study, there will be four *entry-state-rank* values, one corresponding to each of the specification states. Thus, the entry state rank values for the SM case study will be $\mathit{entry\text{-}state\text{-}rank}_{s1}$, $\mathit{entry\text{-}state\text{-}rank}_{s2}$, $\mathit{entry\text{-}state\text{-}rank}_{s3}$, and $\mathit{entry\text{-}state\text{-}rank}_{s4}$. There can be many specification transitions that come into specification state $s$. Therefore, using the above proof obligation template, one proof obligation must be generated for every specification transition ($u$, $s$).

Next, the notion of an exit state of a stuttering segment is used, which is the last stuttering state of the stuttering segment. For example, in Figure 6.(a) for stuttering segment (1,2), (2,3) and (3, 6), 3 is the exit state. For stuttering segment (1,2), (2,3), (3,4) and (4, 9), 4 is the exit state. The difference between the *rank* of the entry state and the exit state of a stuttering segment, divided by $C_{rank}$ will give the length of the stuttering segment. While the *rank* of the entry state of all stuttering segments corresponding to a specification transition are the same, the *rank* of the exit states of these stuttering segments will vary. For example, in the pacemaker case study, when the input $\mathrm{A}_{in}$ becomes 1, the controller will transition from $s_0$ to $s_1$. There is a window of time where $\mathrm{A}_{in}$ can become 1, during which time the implementation controller will stutter in $s_0$. Since, there are a range of implementation states where $\mathrm{A}_{in}$ can become 1, these will amount to stuttering segments of $s_0$ of varying lengths. What is to be verified is that the length of all these stuttering segments lie between the lower bound and upper bound of the delay of the corresponding specification transition.

$$\forall w, v, \vec{i} :: [r(w){=}s_1 \wedge v{=}\mathit{impl\text{-}ckt}(w, \vec{i}) \wedge r(v){=}s_2] \rightarrow$$
$$[\mathit{fpga\text{-}cc\text{-}lb}_{s1-s2} \leq (\mathit{entry\text{-}state\text{-}rank}_{s1} - \mathit{rank}(w)) + 1 \leq \mathit{fpga\text{-}cc\text{-}ub}_{s1-s2}]$$

The proof obligation template is given above. This will be the third and final step of the timing verification process. In the above, ($w$, $v$) is an implementation transition that maps to specification transition ($s_1$, $s_2$). $w$ therefore represents exit states corresponding to the stuttering segments of ($s_1$, $s_2$). $fpga\text{-}cc\text{-}lb_{s1-s2}$ and $fpga\text{-}cc\text{-}ub_{s1-s2}$ are the delay lower bound and delay upper bound of the specification transition ($s_1$, $s_2$) represented in terms of the clock cycles of the FPGA platform and can be obtained by dividing the delay bounds in time units by the FPGA clock period. The above proof obligation requires that the length of all the stuttering segments of ($s_1$, $s_2$), which corresponds to the delay of the stuttering segments interms of FPGA clock cycles, always lie between the delay lower bound and the delay upper bound of ($s_1$, $s_2$) transition. The "+ 1" corresponds to the ($w$, $v$) transition, which should be included in the length of the stuttering segment. Using the above proof obligation template, one proof obligation must be generated for every specification transition ($s_1$, $s_2$).

## 4.7. Overview of Proposed Methodology

We described the concepts of functional and timing verification in the previous two sections, respectively. Now we provide an overview of the high-level steps required to apply the proposed methodology. These steps are given for anyone to implement or replicate our devised approach for the verification of a system that can be expressed as a timed-transition system (TTS).

1. Capture the system requirements as a TTS and encode TTS as an SMT-LIB function.

2. Translate implementation circuit RTL to SMT-LIB.

3. Devise a refinement map and a rank function(s) for the implementation circuit. The rank function should decrease by a constant value when the circuit stutters in a particular specification state. Encode both functions in SMT-LIB.

4. Devise circuit invariants and $C_k$ conditions that distinguish when the circuit stutters and when it does not.

5. Compute *entry-state* rank values, one value for each specification state.

6. Synthesize the functional proof obligations using the devised PO templates. By using the functions of rank, refinement map and $C_k$ conditions as described above, encode the proof obligations in SMT-LIB.

7. Once done with functional proofs, synthesize the timing proof obligations using the devised timing PO templates, encode the proof obligations in SMT-LIB.

8. The encoded file containing the specification TTS, the implementation TTS and the proof obligation(s) is discharged into an automatic theorem prover (SMT solver) like Z3.

If the solver generates any counterexample(s), it points to the presence of any bugs in a system. The counterexapmles are analyzed to identify the causes of errors. Depending on the cause(s) of error, we have to move one step back to the implementation system and further investigate the error(s). Each proof obligation is checked seperately to check the respective correctness property.

## 4.8. Experimental Results

We implemented the controllers for the three case studies in RTL VHDL and synthesized onto an Altera DE2 FPGA board. The verification experiments were performed on a Windows based Intel(R) Core(TM) i7 2.40 GHz machine, with 8GB RAM. The verification statistics are given in Table 4.1. In the table, PO stands for proof obligations. For all the case studies, each of the proof obligations took less than a second to check. The verification efficiency is due to the very nice property of the methodology, which is that the proof obligations reduce correctness verification to only reasoning about single steps of the implementation and specification, and also due to advances in SMT solver technology. The verification also caught several bugs in the implementation circuits. One error is described from the pacemaker case study. For the state transition $(s_0, s_2)$, the pacemaker circuit did not assert AP signal at AEI when no $A_{in}$ was sensed from the heart. Therefore, the circuit was stalled in $s_0$ state for longer than allowed and is only released from this state when the next $A_{in}$ is detected from the heart. This behavior is incorrect and would cause missed rhythms.

Table 4.1. Verification Statistics for Three Case Studies

| Case Study | # Specification Transitions | # Implementation Transitions | # of Functional PO | Timing PO | # of Invariants |
|---|---|---|---|---|---|
| PWM | 2 | 0.1 million | 4 | 6 | 2 |
| SM | 4 | 4 million | 8 | 12 | 2 |
| PM | 10 | 200 million | 17 | 26 | 2 |

### 4.9. Conclusions

There are two properties that make the overall approach very attractive. First, the use of timed transition systems to model requirements allows for simple and concise specifications. Therefore, it is easier to ensure that the specifications themselves are correct. Second, both functional and timing verification are performed without simulating the circuit. Instead, verification is performed directly by checking if the implementation circuits satisfy the proof obligations using an SMT solver.

The proposed methodology is based on the application of rank function and equal delay property of FPGA platform. This results in very efficient verification times as compared to the simulation based techniques or model checking based techniques. Application Specific Integrated Circuits (ASICs) is another implementation platform that also enjoys the property of equal delay as in FPGA. Each transition of the application implemented on FPGA or as an ASIC circuit will have the same delay, equal to the delay of their clock cycle period, respectively. Therefore, in future work we plan to explore the applicability of our approach to ASICs.

# 5. A FORMAL VERIFICATION METHODOLOGY FOR FPGA BASED STEPPER MOTOR CONTROL

## 5.1. Introduction

A stepper motor is a brushless, synchronous motor in which a full rotation is divided into a number of smaller steps. It is different from a simple DC motor which rotates continuously when DC voltage is applied to it. Stepper motor contains a multiple-lead wire, with either 4 or 6 or 8 leads. A repeating seqence of electric pulses is applied with the help of a digital controller which rotates the motor in discrete steps. Motion control is achieved by discrete rotation of the mechanical shaft. Hence, stepper motor is controlled digitally through to provide the precise motion control. Because stepper motors can move in accurate, discrete angular increments (steps) in response to electrical input pulses, they are ideal for those applications which require controlled and precise movements. A significant advantage of stepper motor is that it does not require any feedback information to determine its position, rather it is capable to operate in an open loop fashion. The position of motor rotation is simply known by keeping track of input step pulses. A stepper motor can only take one step at a time and each step taken is of the same size. The number of electric pulses directly control the amount of rotation produced, while the motor's speed is proportional to the frequency of the pulses.

Stepper motors are also employed in many safety-critical applications where precise motion control is required. Such applications include surgical robots (robots used in surgery) [9] [68] and carburetor adjustment for air fuel mixture in automotive electronic throttle control [69]. Typically, microcontrollers are used for Stepper Motor (SM) control. A more recent trend is to use FPGA for SM control [8]. FPGA-based motor control drives are becoming popular because FPGAs allow a platform for direct implementation of control functions in hardware, resulting in much higher performance, steep reductions in power consumption, easily achievable concurrency and parallel processing, and more deterministic timing behavior (that is well-suited for real-time applications such as SM control).

## 5.2. Motor Driver and Modes of Operation

A motor driver is a controller that provides the required electric pulses to the motor for its operation. Each pulse moves the shaft in fixed increments. Stepper drives control the operation of a stepper motor. Generally, stepper motors are manufactured with steps per revolution of 12, 24, 72, 144, 180, and 200, resulting in the shaft increments of 30, 15, 5, 2.5, 2, and 1.8 degrees per step. Stepper motors are either bipolar or unipolar. There are multiple coils organized in groups called phases. By energizing each phase in sequence, the motor will rotate, one step at a time. The torque and motor running properties are affected by the excitation mode applied. The speed and direction of stepper motor are controlled by a control system that applies pulses through a stepper motor driver.

There are three commonly used excitation modes for stepper motors: full step, half step and microstepping. Stepper motors come in many different sizes and styles and electrical characteristics. For our case study, we used a four-lead stepper motor. **Full Step:** In full stepping, the digital sequence applied to motor has four combinations. A stepper motor has two pairs of stator windings. The two phases are energized in alternate fashion and in reverse polarity also. Just as the rotor aligns with one of the stator poles, the second phase is energized. There are four steps, so, four discrete state transitions are required for one rotation of stepper motor. **Half Step:** In half-step mode, there are eight steps instead of four. The main difference is that the second phase is turned on before the first phase is turned off. Hence, sometimes the two phases are energized at the same time. During the half-steps, the rotor hold on between the two full-step positions. A half-step mode requires eight discrete state transitions to complete one rotation of stepper motor. **Microstepping:** Microstepping is a technique of moving the stator flux of a stepper more smoothly as compared to in full-step or half-step drive modes. Thus, it produces less vibration and makes noiseless stepping possible with no detectable “stepping”. In microstepping mode, the step angle is further divided into multiple subdivisions to improve the control over the stepper motor. The applications where a more refined motor work with greater resolution is required, microstepper controller is used. The physical limitation of any particular application should also be considered while employing microstepping control mode. There is a resonance hazard in some applications which can be eliminated by using microstepping controller. In many applications microstepping can increase system performance, and decrease the system complexity and cost, compared to full- and half-step driving techniques. In our case study, we applied formal verification technique to the half-stepping and full-stepping modes. The purpose was to verify the correct operation, speed and direction controls for stepper motor in the desired mode. If the drive circuit missed the time limits then the speed is affected.

We present a formal verification methodology for validation of FPGA-based SM control. Our verification methodology is based on the theory of Well Founded Equivalence Bisimulation (WEB) refinement [14]. In the context of refinement, both specifications and implementations are modeled as transition systems (TSs). A TS $\mathcal{M} = \langle S, R, L \rangle$ is a three tuple that includes a set of states (S), a transition relation ($R$) that provides the transitions between the states, and a labeling function ($L$) that describes what is visible at each state. WEB refinement is a notion of equivalence that defines what it means for two TSs to be functionally equivalent and allows for the implementation TS to be significantly more complex when compared to its specification. In our case studies, automatic and efficient verification of FPGA controllers with millions of transitions against specifications with less than 50 transitions was achieved. The specific contributions of our work are the following:

(a) generic TS specifications for 6 types of SM control;

(b) a set of decidable proof obligations (based on the specification TS) that can be used to automatically check and validate FPGA-based SM control designs and implementations;

(c) a set of invariant properties to be used in conjunction with proof obligations that eliminate unreachable controller states and should be satisfied by FPGA SM control implementations;

(d) rank functions used for liveness verification (detecting bugs that can cause deadlock) of FPGA SM controllers.

## 5.3. Related Work

There has been a lot of recent work on the use of FPGA for SM control. Below we outline three salient works. Yu et al. [70] have developed an FPGA-based controller for stepper motors that incorporates bidirectional control. Dahm et al. demonstrate the advantages of using FPGAs for SM control in comparision to software-based control [71]. Zaferullaht et al. [72] have developed an FPGA controller for Collimator Jaws Positioning that employs SM control. There are many other works that propose FPGA-based controllers for stepper motors. The recent level of research activity in this area suggests that the research trend of using FPGA for SM control is increasing. The above three works are focused on FPGA-based controller design, However, these works do not address the problem of functional verification of the hardware control design. We are not aware of any prior work on formal verification methodologies for FPGA-based SM control. We believe this is the first attempt at such a solution. TS specifications for two types of SM control (clockwise half stepping and clockwise full stepping) have been developed previously by Dubasi et al. [73]. We have developed TS specifications for six types of SM control including the above two mentioned. Also, their work is targeted at object code verification, whereas we target FPGA verification.

### 5.4. Specification of Stepper Motor Control

In this section, we provide formal specifications for SM control as a transition system model. We have considered six case studies for the functional verification of a stepper motor which has four leads. Three case studies are for full stepping mode i.e., full step clockwise, full-step anticlockwise and full-step bidirectional control. The other three case studies are for half stepping mode comprisng of half step clockwise, half step anti-clockwise and halfstep bidirectional control. The specification transition system (TS) for each case is given below.

#### 5.4.1. Full Stepping Clockwise

For a motor with 4 leads (a,b,c,d), application of the following repeating sequence (called full stepping) of values causes the motor to rotate in the clockwise direction:

$$abcd = 0001, 0010, 0100, 1000, 0001...$$

Each value in the sequence causes the motor to rotate by a discrete angle. A digital circuit implemented on an FPGA generates this sequence, and when interfaced with a motor, can control the motor. The time delay between each value in the sequence determines the speed of the motor. To define TS specifications, we use the convention that state $s_i$ corresponds to the state where the value to the leads is $i$. For example, state 0010 is represented by $s_2$. The TS ($M_{FC}=\langle S_{FC}, R_{FC}\rangle$) specification shown for SM control with four leads is (also shown in Figure 5.1):

$$\begin{aligned} S_{FC} =& \{s_1, s_2, s_4, s_8\} \\ R_{FC} =& \{\langle s_1, s_2\rangle, \langle s_2, s_4\rangle, \langle s_4, s_8\rangle, \langle s_8, s_1\rangle\} \end{aligned}$$

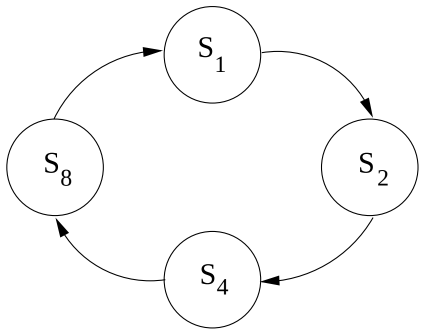

Figure 5.1. Specifications for Full Stepping SM Control

The correct specification for bidirectional full stepping control is given below and has 24 transitions. Therefore, a part of this specification TS $M_{FB}$ is shown in Figure 5.2.

The labeling function $L$ for all specification TSs is the value of the control bits in each state, i.e., $L(s)$=$s$. Therefore, in defining the specification TSs, we do not explicitly define $L$.

### 5.4.2. Half Stepping Clockwise

A four-lead stepper motor can also be operated using a half stepping sequence. In half stepping sequence, only one bit can be flipped going from one state to next (subsequent) state. The values for half-stepping repeating sequence are 0001, 0011, 0010, 0110, 0100, 1100, 1000, 1001, 0001. TS ($M_{HC}$=$\langle S_{HC}$,$R_{HC}\rangle$) specification for half stepping is given below:

$$S_{HC} = \{s_1, s_2, s_3, s_4, s_6, s_8, s_9, s_{12}\}$$

$$R_{HC} = \{\langle s_1, s_3\rangle, \langle s_3, s_2\rangle, \langle s_2, s_6\rangle, \langle s_6, s_4\rangle, \langle s_4, s_{12}\rangle, \langle s_{12}, s_8\rangle, \langle s_8, s_9\rangle, \langle s_9, s_1\rangle\}$$

### 5.4.3. Full Stepping Anti-clockwise

Stepper motors can also be rotated in the anti-clockwise direction by reversing the sequence of values. The TS specification $M_{FA}$=$\langle S_{FA}$,$R_{FA}\rangle$ for anti-clockwise full stepping is defined below.

$$S_{FA} = S_{FC};$$

$$R_{FA} = \{\langle s_j, s_i\rangle : \forall\langle s_i, s_j\rangle \in R_{FC}\};$$

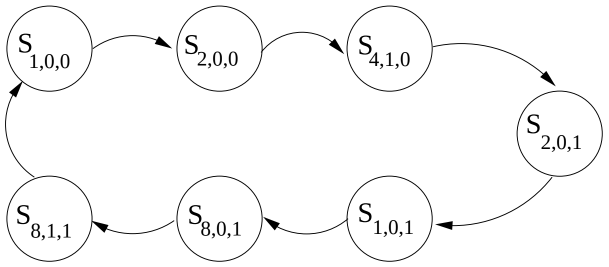

Figure 5.2. Specifications for Full Stepping Bidirectional SM Control

### 5.4.4. Half Stepping Anti-clockwise

$M_{HA}$=$\langle S_{HA}$,$R_{HA}\rangle$, is given below:

$$S_{HA} = S_{HC}; \text{and}$$
$$R_{HA} = \{\langle s_j, s_i\rangle : \forall\langle s_i, s_j\rangle \in R_{HC}\}.$$

### 5.4.5. Full Stepping Bidirectional Control

Stepper motors can also be controlled bidirectionally, i.e., controlled to rotate both in clockwise and anti-clockwise direction. The corresponding TS specifications are not straightforward for these cases. For example, the TS obtained by taking the union of clockwise and anti-clockwise transitions is not a correct specification for bidirectional control. Its implementation ignores user input about changes in direction and simply transitions from $s_1$ to $s_2$, $s_2$ to $s_4$, $s_4$ to $s_8$, $s_8$ to $s_1$ and then transitions from $s_1$ to $s_8$, $s_8$ to $s_4$, $s_4$ to $s_2$, and $s_2$ to $s_1$. The specification TS $M_{FB}$=$\langle S_{FB}, R_{FB}\rangle$ is given below:

$$\begin{aligned}
S_{FB} &= \{s_{i,b_1,b_2} : \forall s_i \in S_{FC} \wedge \forall b_1, b_2 \in \{0,1\}\}\\
R_{FB} &= \{\langle s_{i,0,0}, s_{j,b,0}\rangle : \forall\langle s_i, s_j\rangle \in R_{FC} \wedge \forall b \in \{0,1\}\}\cup\\
&\quad \{\langle s_{i,0,0}, s_{i,1,0}\rangle : \forall s_i \in S_{FC}\}\cup\\
&\quad \{\langle s_{i,0,1}, s_{i,1,1}\rangle : \forall s_i \in S_{FA}\}\cup\\
&\quad \{\langle s_{i,0,1}, s_{j,b,1}\rangle : \forall\langle s_i, s_j\rangle \in R_{FA} \wedge \forall b \in \{0,1\}\}\cup\\
&\quad \{\langle s_{i,1,1}, s_{j,0,0}\rangle : \forall\langle s_i, s_j\rangle \in R_{FC}\}\cup\\
&\quad \{\langle s_{i,1,0}, s_{j,0,1}\rangle : \forall\langle s_i, s_j\rangle \in R_{FA}\}
\end{aligned}$$

In the above specification state $s_{i,b_1,b_2}$, $i$ indicates the value to the four leads. $b_1$ is a predicate that indicates if a request to change direction has been initiated by the user, and $b_2$ is a predicate that determines the current direction of the motor. $b_2 = 0/1$ incidates clockwise/anti-clockwise. Note that $M_{FB}$ does not allow the implementation to ignore user input regarding direction.

### 5.4.6. Half Stepping Bidirectional Control

For bidirectional control of half stepping, the transition specification $M_{HB}$=$\langle S_{HB}, R_{HB}\rangle$ is given below:

$$
\begin{aligned}
S_{HB} &= \{s_{i,b_1,b_2} : \forall s_i \in S_{HC} \wedge \forall b_1, b_2 \in \{0,1\}\}\\
R_{HB} &= \{\langle s_{i,0,0}, s_{j,b,0}\rangle : \forall \langle s_i, s_j\rangle \in R_{HC} \wedge \forall b \in \{0,1\}\} \cup\\
&\qquad \{\langle s_{i,0,0}, s_{i,1,0}\rangle : \forall s_i \in S_{HC}\} \cup\\
&\qquad \{\langle s_{i,0,1}, s_{i,1,1}\rangle : \forall s_i \in S_{HA}\} \cup\\
&\qquad \{\langle s_{i,0,1}, s_{j,b,1}\rangle : \forall \langle s_i, s_j\rangle \in R_{HA} \wedge \forall b \in \{0,1\}\} \cup\\
&\qquad \{\langle s_{i,1,1}, s_{j,0,0}\rangle : \forall \langle s_i, s_j\rangle \in R_{HC}\} \cup\\
&\qquad \{\langle s_{i,1,0}, s_{j,0,1}\rangle : \forall \langle s_i, s_j\rangle \in R_{HA}\}
\end{aligned}
$$

## 5.5. Verification Methodology

In this section, we develop a verification methodology for all the different types of SM control outlined in Section 5.4. Our methodology (as stated earlier) is based on the theory of WEB refinement, which is a notion of equivalence between two transition systems. We have provided transition system specifications for the different types of SM control in Section 5.4. Our goal is to develop a verification methodology for FPGA-based digital SM control. The digital circuit corresponding to SM control can be modeled as a TS by modeling the circuit as a function that describes the next state of the circuit, given the current state and inputs. This function along with the initial state of the circuit gives the TS model of the circuit.

If we compare the TS models of the specification and FPGA implementation for SM control, there are two primary differences. First, the states of the specification have usually 4 or 6 bits. Whereas, the FPGA circuit state looks different. For example, the circuit has counters that are not present in the specification state. WEB refinement accounts for this by providing a refinement map ($r()$), which is a function that maps circuit states to specification states. A second difference is that the FPGA circuit has many more transitions than the specification. The digital circuit may take several transitions to match a transition of the specification. This phenomenon is known as stuttering and is accounted for by WEB refinement. The general theory of WEB refinement is quite complex. Below we provide the formal definition. We refer the reader to [14], for a more detailed account of this theory.

**Definition 1:** [14] $B \subseteq S \times S$ is a WEB on TS $\mathcal{M}$=$\langle S, R, L\rangle$iff:

(1) $B$ is an equivalence relation on $S$; and

(2) $\langle \forall s, w \in S :: sBw \quad \rightarrow \; L(s) = L(w)\rangle$; and

(3) There exist functions $erankl : S \times S \rightarrow \mathbb{N}$, $erankt : S\rangle W$,

such that $\langle W, \lessdot\rangle$ is well-founded, and

$\langle \forall s, u, w \in S :: sBw \wedge sRu \quad \rightarrow$

(a)$\langle \exists v :: wRv \wedge uBv\rangle \vee$

(b)$(uBw \wedge erankt(u) \lessdot erankt(s)) \vee$

(c)$\langle \exists v :: wRv \wedge sBv \wedge erankl(v, u) < erankl(w, u)\rangle\rangle$

**Definition 2:** [14] Let $M$=$\langle S$,$R$,$L\rangle$, $M'$=$\langle S', R', L'\rangle$, and $r : S \rightarrow S'$. We say that $M$ is a WEB refinement of $M'$ with respect to refinement map $r$, written $M \approx_r M'$, if there exists a relation, $B$, such that $\langle \forall s \in S :: sBr(s)\rangle$ and $B$ is a WEB on the TS $\langle S \uplus S', R \uplus R', \mathcal{L}\rangle$, where $\mathcal{L}(s) = L'(s)$ for $s$ an $S'$ state and $\mathcal{L}(s)$ =$L'(r(s))$ otherwise.

In the above definitions, $M$ is the implementation TS and $M^{'}$ is the specification TS. *rank* is a function from implementation states to natural numbers and its value decreases when there is stuttering. *rank* is used to distinguish stuttering from deadlock. Note that the labeling function for implementation states is the refinement map ($r()$), i.e., $L(s)$=$r(s)$. The above WEB refinement formula is hard to check. We have derived proof obligations for each of the specification TSs ($M_{HC}$, $M_{HA}$, $M_{FC}$, $M_{FA}$, $M_{HB}$ and $M_{FB}$). The proof obligations are expressible in a decidable fragment of first-order logic and can be efficiently checked with an SMT solver, unlike the original WEB refinement formula. In developing the proof obligations, we exploit the fact that we know the specification TS. We use two invariants in our proof obligations. These invariants should be satisfied by SM control implementations. We have defined two inavriants. The SM controller will use a counter to determine delay and hence the speed of the motor. The first invariant defines the range for this counter. $m$ is the maximum value of the counter. If the counter reaches $m$, it is reset. The invariants are given below;

$$\text{INV1: } 0 \leq w.counter \leq m$$

$$\text{INV2: } r(w) \in S_s$$

In the above, $S_s = S_{HC}/S_{FC}/S_{HA}/S_{FA}/S_{HB}/S_{FB}$. The refinement map projects the control bits from an implementation state, which gives the corresponding specification state. The second invariant indicates that these control bits of the implementation states can have only the values allowed by the specification. The concrete form of INV2 is given below for full stepping and half stepping modes, respectively.

$$\text{INV2}_{\text{FS}}: \ a'bcd + ab'cd + abc'd + abcd'$$

$$\text{INV2}_{\text{HS}}: \ a'b'c + a'cd' + bc'd' + ab'c'$$

$\text{INV2}_{\text{FS}}$ is true for all three case studies of full stepping control. Similarly, $\text{INV2}_{\text{FS}}$ is true in all case studies of half stepping control. Next we describe the proof obligations. Specifications $M_{HC}$, $M_{FC}$, $M_{HA}$ and $M_{FA}$, are deterministic. For these four cases we only need two proof obligations.

$$\text{PF1: } [(r(w) = s_i) \wedge (w.counter = m)] \longrightarrow [r(v) = s_j]$$

$$\text{PF2: } [(r(w) = s_i) \wedge (w.counter \neq m)]$$
$$\longrightarrow [(r(v) = s_i) \wedge (rank(v) < rank(w))]$$

$$\text{RANK: } rank(w) = m - w.counter$$

In the proof obligations above, $w$ is an implementation state and $v$ is its only successor, (implementations are also deterministic). $(s_i,s_j) \in R_s$ where $R_s{=}R_{HC}/R_{FC}/R_{HA}/R_{FA}$. $w.counter$ is the value of the counter in state $w$. When $counter{=}m$, the implementation makes progress with respect to the specification, otherwise the implementation stutters. $r(w)$ and $r(v)$ are the specification states obtained by applying the refinement map to implementation states $w$ and $v$, respectively. PF1 gives the proof obligation corresponding to the non-stuttering case, and PF2 gives the proof obligation corresponding to the stuttering case. When stutter occurs we have to show that a witness *rank* function decreases. We define the *rank* of an implementation state $w$ as the difference between the maximum value the counter can take and the current value of counter. Note that INV1 guarantees that PF1 and PF2 cover all reachable states of the implementation.

**Theorem 1:** Let $M^{'}$=$M_{HA}/M_{HC}/M_{FA}/M_{FC}$. Let $M$ be an implementation of $M^{'}$. $\bigwedge\limits_{n=1}^{2} PF_n \rightarrow M \approx_r M^{'}$.

The above theorem can be proved using case analysis. We derived a separate set of proof obligations for bidirectional SM control based on specification TSs $M_{HB}$ and $M_{FB}$. We require only invariants INV1 and INV2 for bidirectional control as well. A separate set of proof obligations were required as unlike $M_{FC}/M_{HC}/M_{FA}/M_{HA}$, $M_{HB}/M_{FB}$ are non-deterministic. The proof obligations for $M_{HB}/M_{FB}$ are given below:

$$\begin{aligned}
\text{PF3: } & [(r(w) = s_{i,0,0}) \wedge (w.dir) \wedge (w.counter = m)] \\
& \longrightarrow [r(v) = s_{j,1,0}] \\
\text{PF4: } & [(r(w) = s_{i,0,0}) \wedge (w.dir) \wedge (w.counter \neq m)] \\
& \longrightarrow [r(v) = s_{i,1,0}] \\
\text{PF5: } & [(r(w) = s_{i,0,0}) \wedge \neg w.dir \wedge (w.counter = m)] \\
& \longrightarrow [r(v) = s_{j,0,0}] \\
\text{PF6: } & [(r(w) = s_{i,0,0}) \wedge \neg w.dir \wedge (w.counter \neq m)] \\
& \longrightarrow [(r(v) = r(w)) \wedge (rank(v) < rank(w))] \\
\text{PF7: } & [(r(w) = s_{i,1,0}) \wedge (w.counter = m)] \\
& \longrightarrow [r(v) = s_{j,0,1}] \\
\text{PF8: } & [(r(w) = s_{i,1,0}) \wedge (w.counter \neq m)] \\
& \longrightarrow [(r(v) = r(w)) \wedge (rank(v) < rank(w))] \\
\text{PF9: } & [(r(w) = s_{i,0,1}) \wedge (w.dir) \wedge (w.counter = m)] \\
& \longrightarrow [r(v) = s_{j,1,1}]
\end{aligned}$$

PF10: $[(r(w) = s_{i,0,1}) \wedge (w.dir) \wedge (w.counter \neq m)]$

$\longrightarrow [r(v) = s_{i,1,1}]$

PF11: $[(r(w) = s_{i,0,1}) \wedge \neg w.dir \wedge (w.counter = m)]$

$\longrightarrow [r(v) = s_{j,0,1}]$

PF12: $[(r(w) = s_{i,0,1}) \wedge \neg w.dir \wedge (w.counter \neq m)]$

$\longrightarrow [(r(v) = r(w)) \wedge (rank(v) < rank(w))]$

PF13: $[(r(w) = s_{i,1,1}) \wedge (w.counter = m)]$

$\longrightarrow [r(v) = s_{j,0,0}]$

PF14: $[(r(w) = s_{i,1,1}) \wedge (w.counter \neq m)]$

$\longrightarrow [(r(v) = r(w)) \wedge (rank(v) < rank(w))]$

In the above, $w$ is an implementation state and $v$ is a successor of $w$. Here $(s_{i,b_1,b_2}, s_{j,b_1,b_2}) \in R_{SB}$, where $R_{SB} = R_{HB}/R_{FB}$ and $b_1, b_2 = \{0,1\}$. *w.dir* is a flag in the implementation that registers a request to change the direction of rotation. PF3-PF8 and PF9-PF14 are applicable when the motor's current direction is clockwise and anti-clockwise, respectively. PF9-PF14 are symmetric to PF3-PF8, therefore we only describe PF3-PF8. PF3 and PF4 are applicable when a request to change direction has been registered (*w.dir*). PF3 corresponds to the case where the counter value has reached its maximum limit and PF4 corresponds to the case where the counter value has not reached its maximum value. After a direction change request is initiated ($s_{i,1,0}$), if *counter*$\neq m$, then the implementation will continue to stutter (PF8). Once *counter*$= m$, the implementation should make progress, which is captured by PF7 where the motor changes direction. PF5 and PF6 correspond to the case where no direction change has been requested and hence are similar to PF1 and PF2. PF5 is the non-stuttering case and PF6 is the stuttering case.

**Theorem 2:** Let $M' = M_{HB}/M_{FB}$. Let $M$ be an implementation of $M'$. $\bigwedge_{n=3}^{14} PF_n \rightarrow M \approx_r M'$.

The above theorem can be proved using case analysis on the implementation states and symbolic manipulation. The implementation states are characterized by invariants INV1 and INV2. Each proof obligation addresses a subset of the implementation states. PF3 is applicable to implementation states where *w.counter*$=m$ and $r(w) = s_{i,0,0}$; PF4 is applicable to implementation states where *w.counter*$\neq m$ and $r(w) = s_{i,0,0}$, and so on. It can be seen that each proof obligation matches

Table 5.1. Verification Statistics

| Case Study | Conflicts | Decisions | Memory (MB) | Verif. Time(ms) | # of bugs found |
|---|---|---|---|---|---|
| FC | 6349 | 13874 | 2.05 | 170 | 7 |
| FA | 7061 | 14581 | 2.01 | 180 | 8 |
| HC | 7651 | 15038 | 2.09 | 200 | 8 |
| HA | 7185 | 13708 | 2.09 | 180 | 11 |
| FB | 8078 | 16537 | 2.27 | 630 | 12 |
| HB | 8292 | 18468 | 2.42 | 680 | 16 |

Table 5.2. Number of States in Each Case Study

| Case Study | # of transitions in Specification | # of transitions in Impl (millions) |
|---|---|---|
| FC | 4 | 6 million |
| FA | 4 | 6 million |
| HC | 8 | 12 million |
| HA | 8 | 12 million |
| FB | 24 | 24 million |
| HB | 48 | 48 million |

with one of the three cases (a), (b), or (c) of the WEB definition (Definition 1). Also, PF3-PF14 together handle all the implementation states.

The proof obligations are to be used for verification. Checking that the implementation satisfies the proof obligations guarantees that the implementation is functionally equivalent to the specification. Theorems 1 and 2 establish the correctness of the proof obligations, but are not directly used for verification.

## 5.6. Experimental Results

We have applied our verification methodology to check the correctness of six FPGA-based SM controllers. The six FPGA controllers correspond to the six different types of SM control outlined in Section 5.4. The controllers were implemented in RTL VHDL and then synthesized onto FPGA. For verification of the controller, we modeled the specification TS and the FPGA controllers in the SMT-LIB language [19].

We then checked the proof obligations using the Z3 SMT solver [17]. The verification experiments were performed on an Intel(R) Core(TM) i3 2.53 GHz processor with 2GB RAM. The results are shown in Table 1. The second and third columns show the conflicts and decisions generated by

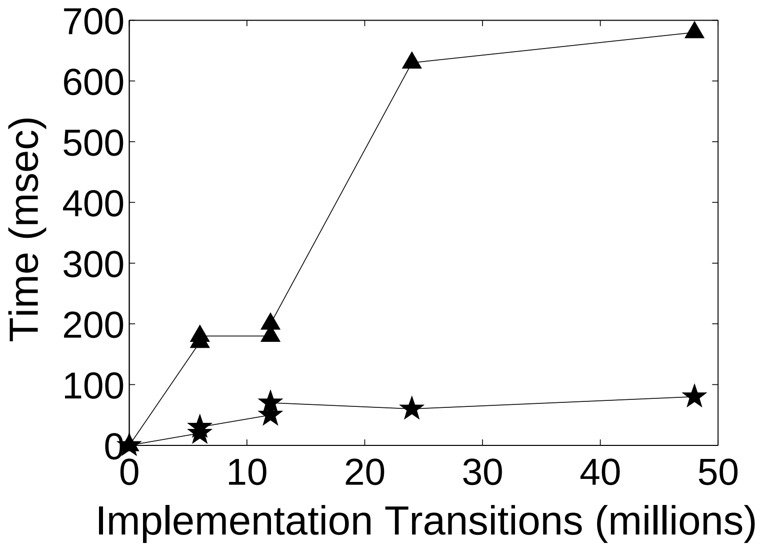

Figure 5.3. The Figure Plots Verification Time Against Number of Transitions of the Implementation TS. The Triangular Line and the Asterisk Line Plot Verification Times for Correct and Buggy Implementations, Respectively.

Z3. The fourth and fifth columns show the total amount of memory and the total time, respectively, required to check all the WEB refinement proof obligations for each case study. The table also gives the number of transitions of the SM FPGA controller TS and the corresponding specification TS. As shown in the table, the FPGA controllers have a very large number of transitions (in the order of millions). This is because the FPGAs have high speed clock signals and circuit operations are synchronized with the clock. Each tick of a clock corresponds to an implementation state of a system. On the other hand, the number of transitions of the specification TS is less than 50. Even still, the verification times and memory usage are very efficient. During verification, we also found many bugs in the controller designs as depicted in the last column.

Figure 5.3 shows that the time required to find bugs is much less than the time required to prove correctness. This is a very positive result because much of the verification process involves finding and fixing bugs. Also, the time required to find bugs is less sensitive to the size of the implementation TS than the time for proving correctness (which increases linearly with the size of the implementation TS).

## 5.7. Conclusion

We converted the VHDL implementation models into the input language of SMT solvers. The proof obligations were encoded in the format as explained earlier in this chapter. The encoded proof obligations were discharged into an automated theorem proving tool Z3 and were checked for validity. Our approach is highly automated and also very effecient. Experimental result show the scalability of our approach. Our methods can, hence, be easily incorporated for commercial use and can provide a high degree of savings in time and resources for validation of FPGA-based SM controllers.

# 6. FORMAL VERIFICATION OF REAL-TIME NETWORK ON CHIP APPLICATIONS

## 6.1. Introduction

Network on chip (NOC) is an on-chip communication architecture that provides robust, flexible and scalable communication services in multiprocessor system-on-chip designs (SOC). SOC is a complex interconnection of several functional blocks. Real-time systems and applications are becoming computation-intensive and complex, at the same time demanding high performance and low-power. The number of computing resources are growing in implementing such high end systems like a nuclear power-plant controller, an airplane or a satellite system. Several computing resources like CPU, specific IPs, DSP etc are added together to make up a SOC. The interconnection between each computing element in SOC is a challenging issue. Typically, a bus based interconnection is used between different functional blocks. But there are certain limitations to the scalability of shared-bus based interconnect. NOC is an emerging solution to obtain the scalable bandwidth requirements in SOC. It is basically the on-chip packet based communication system between computing blocks which are connected together through routers. The basic idea of NOC originated from the traditional large-scale multi-processors and distributed computing networks. In terms of hardware resources, NOC interconnect is easier to scale beacuse it requires shorter, unidirectional and point-to-point wires.

## 6.2. Basics of Network on Chip

Typically, there are two main components in a NOC architecture, computation nodes and routers. To make up a communication network, several routers are connected to each other via small length wires. Each router is further connected to the processor, memory or the IP block through a local port. A simple 2D NOC is depicted in figure 6.1. It represents a mesh topology which is a popular NOC topology because of network scalability and the use of a simple routing algorithm. The processing elements (PEs) are the nodes which perform computation part and share the data among them. According to Roson and Sangiovanni-Vincentelli [74], each node is made up of an application and an interface. The interface is connected to the communication architecture.

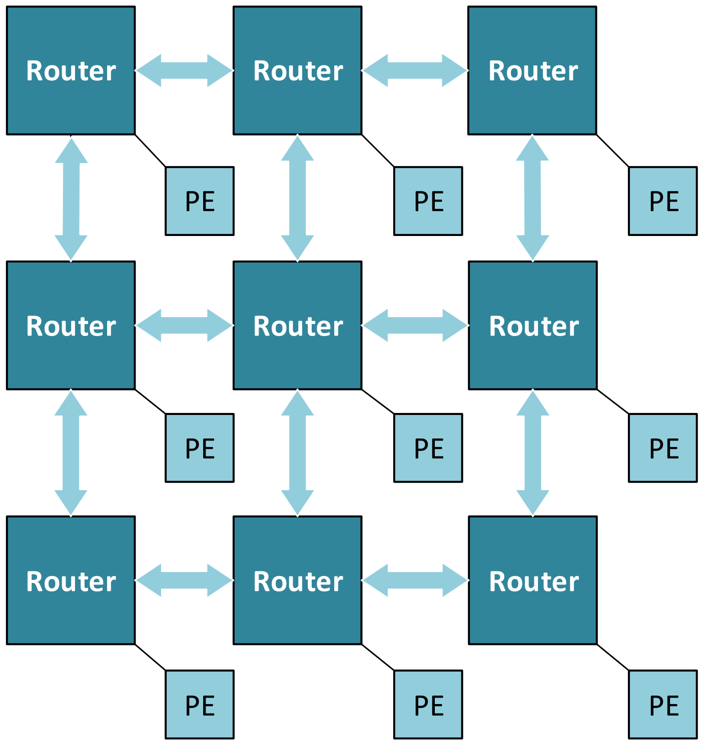

Figure 6.1. An Example of a Simple 2D Mesh NOC

The communication architecture contains links and routers. Applications communicate through the interface by means of some protocol. Applications can be either active or passive. The active applications are processors and passive applications are memories. Typically, the design of a NOC should be pertinent to the applications. Depending on the specific application, NOC based architectures can be constructed according to functional, structural and performance specifications. There is much research literature available which explains the performance evaluation and design tradeoffs for NOC interconnect architecture [75].

The router is a core building block of NOC. A router is a switch that takes a data packet at its input port and sends it to the desired output port based on some underlying routing mechanism. A router of an NOC can be implemented in a centralized or a distributed way. In the centralized design, a router contains: (i) a crossbar; (ii) centralized controllers for routing, switching and arbitration; and (ii) FIFO buffers and link controllers for buffering and flow control. In the distributed approach, the controllers and crossbar are broken down into smaller modules and implemented among the input and output channels of the ports at the router interface.

**Routing and Arbitration:** Two primary algorithms are routing and arbitration that tell us about where and when a packet will move. A routing algorithm decides which direction each input packet should travel. The arbitration resolves which input should be granted when there are more than one input packet requests for the same output port. The logic block or the controller implements the routing and arbitration schemes.

It is evident that the NOC designer has different possible strategies to implement a router (and, therefore, the network communication protocol) leading to a very large design space. This is the main advantage of NOC platform that the system communication requirements can be met by selecting a particular design approach. These advantages make NOC different from bus structure allowing to achieve the guaranteed performance through specific requirements of the application.

**Objective:** Our main objective here is to present a methodology for formal verification of a NOC architecture. The proposed verification technique can be applied to any NOC design by defining the formal specification of a particular design. We have considered the basic router design as proposed by [2] and [1]. The following specifications have been considered for the router design in our case:

**Buffering:** There is a FIFO buffer at both the input and output channels, respectively.

**Flow Control:** The store and forward flow control mechanism is used. In store and forward approach, the packet is first fully stored in the input channel and then it is forwarded to the respective output channel.

**Arbiter:** A fixed priority arbiter is designed. It assigns priorities to each input in a fixed priority order.

**Routing Algorithm:** XY-routing logic is employed in a distributed fashion which means at each input channel. It is a deterministic logic that selects the output port based on the routing information available in the packet header.

We present a case study in which one node is having an active application and it should communicate to another node, having a passive application. In between the two nodes is the FPGA communication architecture as NOC. We address the functional and timing requirements and the correctness verification for a such scenario.

## 6.3. The Router Architecture

There are a number of router implementations already been done [76] [77] [78] [79]. To keep things simple and avoid complexity, we have considered a simple router design based on the distributed approach, that avoids congestion and communication bottleneck. Our proposed router design is similar to the designs presented in [1] [2]. A simple router is shown in the figure 6.2. This router can support five parallel connections at the same time. It uses store and forward flow control mechanism in which network resources are allocated in a packet by packet basis. It has five ports named as east(E),west(W), north(N), south(S) and local(L). Each port contains two unidirectional channels, an input channel and an output channel. Each channel has multiple state components for receiving and tranferring the data between input and output channels. There are multiple signals in each channel like data, flow control and wires etc. The router design architecture with input and output channels is shown in figure 6.3. There are five input channels given on left side and the five output channels on the right side. In the figure, each input and output channel aligned in a row correspond to the East, West, North, South and Local ports, respectively. It is a distributed router which does not have a central control logic. The routing logic and arbitration logic are implemented by each port independently, in input and output channels, respectively. We provide specification for a simple router design as a transition system (TS). The specification TS of a router is comprised of large number of states. We describe the state components for router input and output channel seperately.

### 6.3.1. Specification of Router Input Channel

The input channel contains three main components; a FIFO buffer, an input controller and a routing logic. FIFO is the first in first out buffer, containing registers to store the incoming packets. We have taken the parallel FIFO buffer implementation with demultiplexer and multiplexers. This FIFO structure is same both in the input and outpur channels. The difference is only of the control signals that allow the read and write operation in input and output channels seperately. The FIFO circuit is shown in figure 6.4. The read and write pointers are managed by the FIFO controller taking into account the register status flags. There is a status flag for each FIFO register to indicate the status of that particular register. When all flags are 1, it indicates that FIFO is full and all registers are containing data. When all flags are zero, it indicates that FIFO buffer is empty. The routing logic that we have considered is a simple and deterministic one, known as XY-routing logic. The input channel state variables are given below. All these state variables are updated by the input controller.

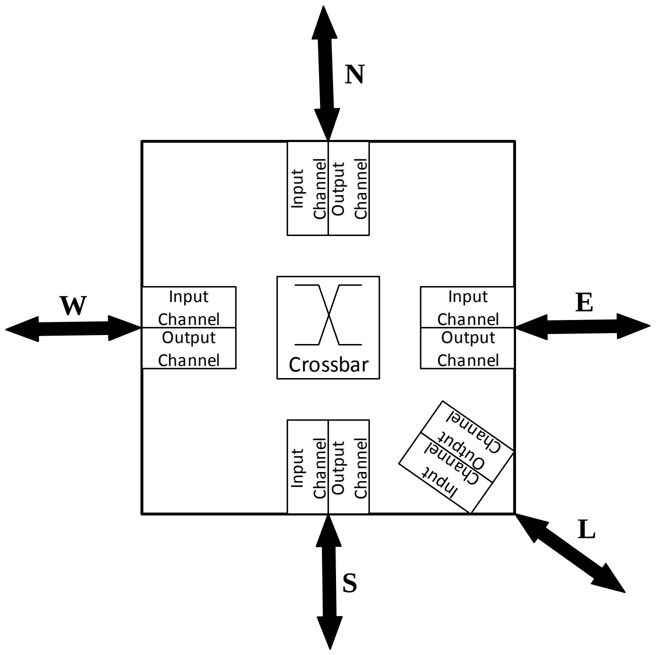

Figure 6.2. A Simple Router Interface [1]

$$S_{in} = \{req_{in}, wp_{in}, rp_{in}, f0_{in}, f1_{in}, f2_{in}, f3_{in}, R0_{in}, R1_{in}, R2_{in}, R3_{in}\}$$

Apart from state variables, there are also many signals and wires in the router circuit. We have grouped those wires and signals into a set $\mathcal{W}$.

$$\mathcal{W}_{in} = \{ack_{in}, full_{in}, empty_{in}, XY_{decode}, r_1, r_2, r_3, r_4, r_5, \}$$

In our proposed NOC, the communication from the source node begins with a hand-shake process. The processing node i.e the controller sends a request to the first router with which it is connected. This request is a state variable, $req_{in}$, that remains asserted until the acknowledgement by the router is received. $ack_{in}$ is the signal in response to request variable. $req_{in}$ is updated by the value generated by a controller. It is generated after a fixed time interval periodically and it is covered as a part of the overall NOC architecture in section 6.4.1.

$$\begin{aligned} {ack_{in}}' \leftarrow \quad if \quad (full = 1) \qquad & ack = 0 \\ else \qquad \qquad & ack = 1 \end{aligned}$$

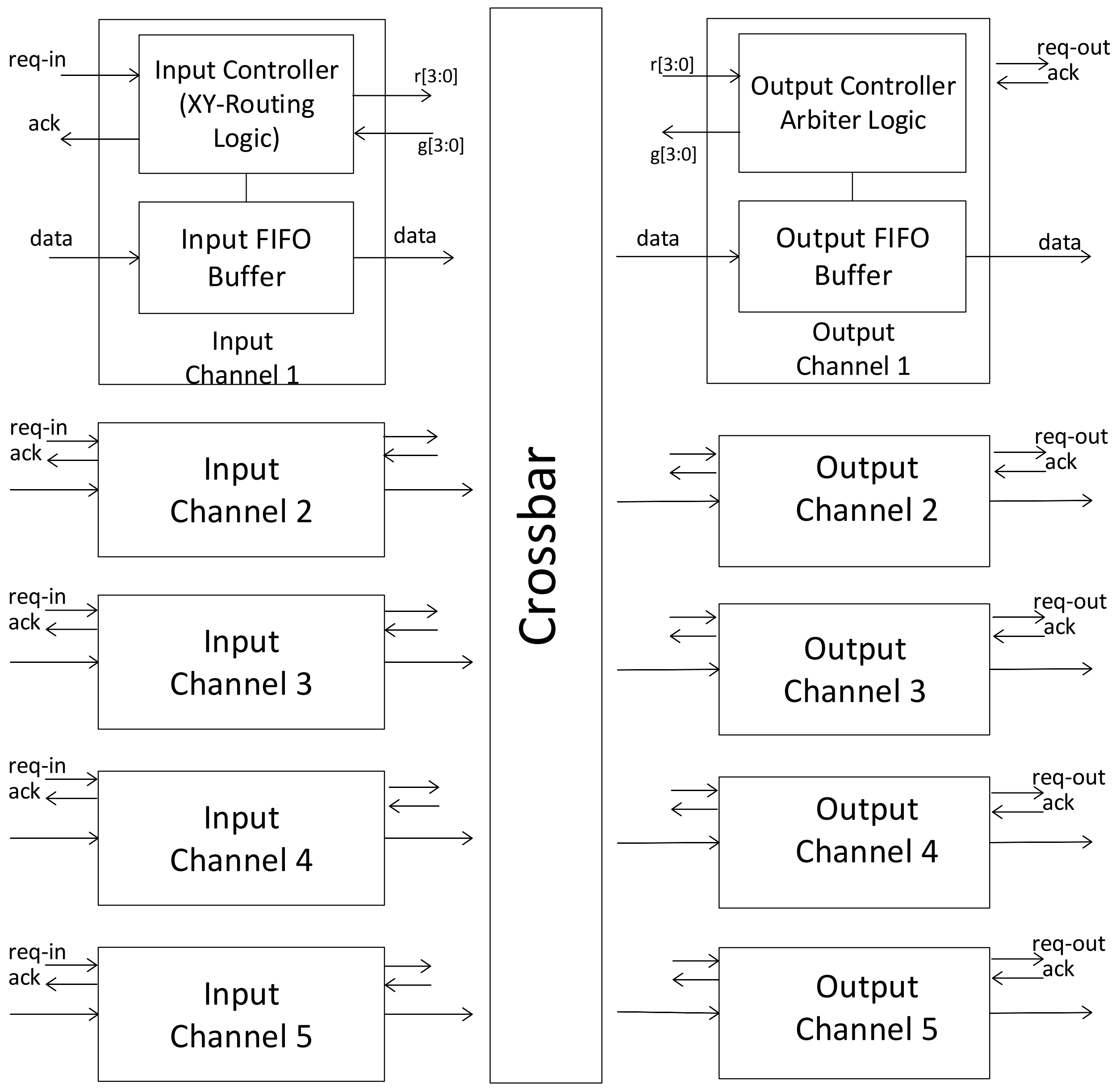

Figure 6.3. A General Architecture for a Router [2] [1]

The above function decribes the $ack_{in}$ signal, which is the acknowledgement given by input channel to the output channel as a part of handshake mechanism for communication. $full$ is the buffer status flag which tell whether buffer is having space to collect the incoming data packet. We explain the description of states and signals in following context.

**6.3.1.1. FIFO Flags**

$full_{in}$ and $empty_{in}$ are the two flag signals representing the status of input buffers either as full or empty. These flags confirm about the availibility of buffer space for incoming packets into a specific channel.

$$\begin{aligned} full_{in}' &\leftarrow if \ \{(f0_{in}=1)\wedge(f1_{in}=1)\wedge(f2_{in}=1)\wedge(f3_{in}=1)\wedge(wp_{in}=rp_{in})\} \\ & \quad full_{in}=1 \quad else \quad full_{in}=0 \\ empty_{in}' &\leftarrow if \ \{(f0_{in}=0)\wedge(f1_{in}=0)\wedge(f2_{in}=0)\wedge(f3_{in}=0)\wedge(wp_{in}=rp_{in})\} \\ & \quad empty_{in}=1 \quad else \quad empty_{in}=0 \end{aligned}$$

$f0_{in}$ , $f1_{in}$, $f2_{in}$ and $f3_{in}$ are the four flags indicating the individual status of input FIFO buffer registers R0, R1, R2 and R3, respectively. The respective flag is 1 only if the register is holding the input data, otherwise if it is empty the flag is 0.

$$\begin{aligned} f1_{in}' \leftarrow \ & if \ \{(f1_{in}=0) \ \wedge \ (wp_{in}=0) \ \wedge \ (req_{in}=1) \ \wedge \ (ack=1) \ \} \qquad (f1_{in}=1) \\ & elseif \ \left\{ \ (f1_{in}=1) \ \wedge \ (rp_{in}=0) \ \wedge \ (g1=1\vee g2=1\vee g3=1\vee g4=1) \ \} \ \right\} \\ & \quad (f1_{in}=0) \\ & else \quad f1_{in} \end{aligned}$$

$$\begin{aligned} f2_{in}' \leftarrow \ & if \ \{(f2_{in}=0) \ \wedge \ (wp_{in}=1) \ \wedge \ (req_{in}=1) \ \wedge \ (ack=1) \ \} \qquad (f2_{in}=1) \\ & elseif \ \left\{ \ (f2_{in}=1) \ \wedge \ (rp_{in}=1) \ \wedge \ (g1=1\vee g2=1\vee g3=1\vee g4=1) \ \} \ \right\} \\ & \quad (f2_{in}=0) \\ & else \quad f2_{in} \end{aligned}$$

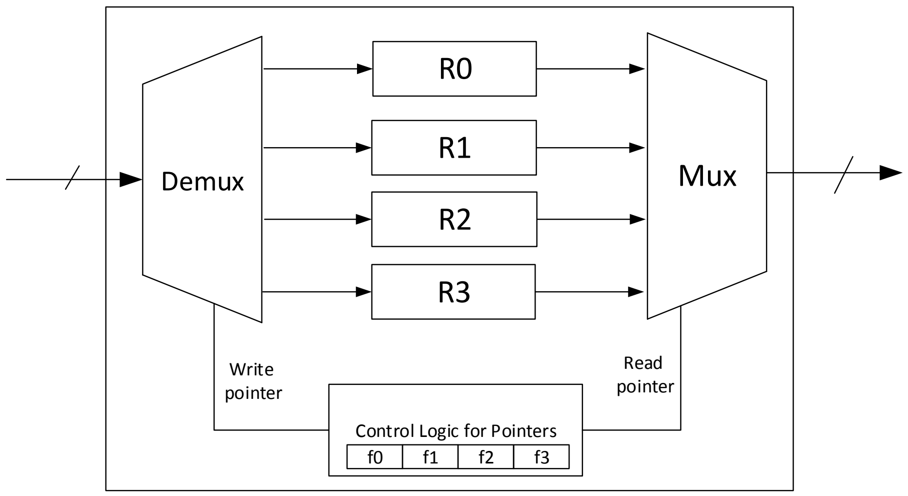

Figure 6.4. FIFO Buffer Structure [3]

$$
\begin{aligned}
f3_{in}' \leftarrow \quad & if \;\; \{(f3_{in}=0) \;\wedge\; (wp_{in}=2) \;\wedge\; (req_{in}=1) \;\wedge\; (ack=1)\;\} \qquad (f3_{in}=1) \\
& elseif \; \Big\{ \; (f3_{in}=1) \;\wedge\; (rp_{in}=2) \;\wedge\; (g1=1 \vee g2=1 \vee g3=1 \vee g4=1) \;\; \} \; \Big\} \\
& \quad (f3_{in}=0) \\
& else \qquad f3_{in}
\end{aligned}
$$

$$
\begin{aligned}
f4_{in}' \leftarrow \quad & if \;\; \{(f4_{in}=0) \;\wedge\; (wp_{in}=3) \;\wedge\; (req_{in}=1) \;\wedge\; (ack=1)\;\} \qquad (f4_{in}=1) \\
& elseif \; \Big\{ \; (f4_{in}=1) \;\wedge\; (rp_{in}=3) \;\wedge\; (g1=1 \vee g2=1 \vee g3=1 \vee g4=1) \;\; \} \; \Big\} \\
& \quad (f4_{in}=0) \\
& else \qquad f4_{in}
\end{aligned}
$$

#### 6.3.1.2. FIFO$_{in}$ Write Pointer ($wp_{in}$)

The buffer used in the router design is a parallel buffer implementation [3]. The circuit for such a FIFO is shown in figure 6.4. It consists of a demultiplexer and a multiplexer. $wp_{in}$ is the write pointer for FIFO buffer. In RTL circuit $wp_{in}$ is actually the select lines of demultiplexer. The pointer gets updated based on the status of its current value and the flags. We have considered four registers in our FIFO design so the wrtie pointer gets its value range from 0 to 3.

```
$wp_{in}' \leftarrow if \quad (full_{in} = 1) \qquad wp_{in}$
        $elseif \ \{(wp_{in} = 0) \ \wedge \ (req_{in} = 1) \wedge (ack = 1) \wedge (f0_{in} = 0)\}$
            $wp_{in} = 1$
        $elseif \ \{(wp_{in} = 1) \ \wedge \ (req_{in} = 1) \wedge (ack = 1) \wedge (f1_{in} = 0)\}$
            $wp_{in} = 2$
        $elseif \ \{(wp_{in} = 2) \ \wedge \ (req_{in} = 1) \wedge (ack = 1) \wedge (f2_{in} = 0)\}$
            $wp_{in} = 3$
        $elseif \ \{(wp_{in} = 3) \ \wedge \ (req_{in} = 1) \wedge (ack = 1) \wedge (f3_{in} = 0)\}$
            $wp_{in} = 0$
        $else \quad wp_{in}$
```

### 6.3.1.3. FIFO$_{in}$ Read Pointer ($\boldsymbol{rp_{in}}$)

$rp_{in}$ is the read pointer of FIFO buffer. The circuit part correcponding to read pointer is actually the select lines of a multiplexer. The read pointer $rp_{in}$ gets updated only if the register is holding data, i.e. the respective flag status is 1,and the grant signal has been asserted by requested output channel. The read pointer gets updated in an order as the registers are updated by $wp_{in}$.

```
$rp_{in}' \leftarrow if \quad (empty_{in} = 1) \qquad rp_{in}$
        $elseif \ \{ \ (rp_{in} = 0) \ \wedge \ (f1_{in} = 1) \ \wedge \ (g1 = 1 \vee g2 = 1 \vee g3 = 1 \vee g4 = 1) \ \}$
            $(rp_{in} = 1)$
        $elseif \ \{ \ (rp_{in} = 1) \ \wedge \ (f2_{in} = 1) \ \wedge \ (g1 = 1 \vee g2 = 1 \vee g3 = 1 \vee g4 = 1) \ \}$
            $(rp_{in} = 2)$
        $elseif \ \{ \ (rp_{in} = 2) \ \wedge \ (f3_{in} = 1) \ \wedge \ (g1 = 1 \vee g2 = 1 \vee g3 = 1 \vee g4 = 1) \ \}$
            $(rp_{in} = 3)$
        $elseif \ \{ \ (rp_{in} = 3) \ \wedge \ (f4_{in} = 1) \ \wedge \ (g1 = 1 \vee g2 = 1 \vee g3 = 1 \vee g4 = 1) \ \}$
            $(rp_{in} = 0)$
        $else \quad rp_{in}$
```

### 6.3.1.4. FIFO Registers

The functions for updating the register variables are given next.

$$
\begin{aligned}
R0_{in}{}' \leftarrow \quad & if \ \left\{ \ (r_{in} = 1) \wedge (ack_{in} = 1) \wedge (f0_{in} = 0) \wedge (wp_{in} = 0) \ \right\} \\
& R0_{in} = data \\
& else \qquad R0_{in} \\
R1_{in}{}' \leftarrow \quad & if \ \left\{ \ (r_{in} = 1) \wedge (ack_{in} = 1) \wedge (f1_{in} = 0) \wedge (wp_{in} = 1) \ \right\} \\
& R1_{in} = data \\
& else \qquad R1_{in} \\
R2_{in}{}' \leftarrow \quad & if \ \left\{ \ (r_{in} = 1) \wedge (ack_{in} = 1) \wedge (f2_{in} = 0) \wedge (wp_{in} = 2) \ \right\} \\
& R2_{in} = data \\
& else \qquad R2_{in} \\
R3_{in}{}' \leftarrow \quad & if \ \left\{ \ (r_{in} = 1) \wedge (ack_{in} = 1) \wedge (f3_{in} = 0) \wedge (wp_{in} = 3) \ \right\} \\
& R3_{in} = data \\
& else \qquad R3_{in}
\end{aligned}
$$

Inside the router, the central connection hub which connects each input channel to other neighbouring output channels is called the crossbar. The datalines from each input channel is connected to the four outputs with the help of wires and multiplexers. The routing logic decodes the destination encoded in the packet and sends request to the respective output channel only. The routing logic we use is a deterministic logic i.e. XY-logic. We describe the routing and control logic in the section below.

### 6.3.1.5. Routing Scheme and Control Logic

The control logic determines the underlying scheme based on which the router sends data packets from input channel to the output channel. In our proposed router design, XY-routing algorithm is implemented in the input channel. It compares the actual router address (cx, cy), i.e. the coordinates in 2D mesh topology, to the target/destination address (dx, dy) stored in the packet header. First, the x-coordinates of current router (cx) and destination router (dx) are compared. If $cx < dx$ the packet will be routed to the East, if $cx > cy$ it will be routed to the West and if $cx = dx$

then the packet is already horizontally aligned. After horizontal alignment, the y-coordinates are compared. If $cy < dy$, the incoming packet will be routed to the South, if $cy > dy$ it will be routed to the North. The packet must go to the Local port when the local address (cx,cy) of router is equal to the destination address (dx,dy). Once the destination port is determined, a request is sent from the input channel to the respective destination/output channel. The input channel will wait for the grant signal from output channel. If the chosen port does not allow grant, the packet will be blocked in the input buffers unless the grant signal is asserted. In order to address the issue of mutiple requests to the same output channel, there is an arbitration logic implemented in the output channel. The arbitration logic determines which input port data should go first into the output channel. We have used fixed priority arbiter design in output channel.

The input channel implements the decoding logic for an incoming packet while the packet is stored in the input buffer. Based on the value of $XY_{decode}$ signals, the corresponding request is generated to the output channel inside a router. Each input port implements routing logic seperately to select the requested output. So in maximum there can be five parallel connections. The crossbar signals, $XY_{decode}$, from each input channel get updated as given in following. The function $XY_{decode}$ computes the destination for the incoming packets in router by comparing the destination coordinates to the local router co-ordinates. $r_1$, $r_2$, $r_3$ and $r_4$ are the four requests for making connection to the decoded destination output channel. Each input channel can connect only to one of the other four neighbouring output channels and not to the same ports output channel. Each request is asserted based on the value decoded by the XY logic, otherwise, all requests are zero.

The request generated after the XY-decoding is defined in a fixed order in each input channel. The request depends on the value od $XY_{decode}$. The functions for these requests are given below:

$$\begin{aligned} {r_1}' \leftarrow \quad & if \quad (XY_{decode} = 1) \quad (r_1 = 1) \\ & else \qquad r_1 = 0 \\ {r_2}' \leftarrow \quad & if \quad (XY_{decode} = 2) \quad (r_2 = 1) \\ & else \qquad r_2 = 0 \end{aligned}$$

$$r_3{}' \leftarrow \quad if \quad (XY_{decode} = 3) \quad (r_3 = 1)$$

$$else \qquad r_3 = 0$$

$$r_4{}' \leftarrow \quad if \quad (XY_{decode} = 4) \quad (r_4 = 1)$$

$$else \qquad r_4 = 0$$

$$r_5{}' \leftarrow \quad if \quad (XY_{decode} = 5) \quad (r_5 = 1)$$

$$else \qquad r_5 = 0$$

Below we define a function for decoding the destination of a packet in each router. The routing scheme is XY-routing so the function is named as $XY_{decode}$.

```
$XY_{decode} \leftarrow if \ \{ (rp_{in} = 0) \wedge (R1 = data - in) \}$
        $if \ (cx < dx) \ 1$
        $\{elseif \ (cx > dx) \ 2$
        $\{ elseif \ (cy < dy) \ 3$
        $\{ elseif \ (cy > dy) \ 4$
        $\{ elseif \ (cy = dy) \wedge (cx = dx) \ 5$
        $else \quad 0 \ \}\}\}\}$
    $elseif \ \{ (rp_{in} = 1) \wedge (R2 = data - in) \}$
        $if \ (cx < dx) \ 1$
        $\{elseif \ (cx > dx) \ 2$
        $\{ elseif \ (cy < dy) \ 3$
        $\{ elseif \ (cy > dy) \ 4$
        $\{ elseif \ (cy = dy) \wedge (cx = dx) \ 5$
        $else \quad 0 \ \}\}\}\}$
```

```
$elseif \ \{ (rp_{in} = 2) \wedge (R3 = data - in) \}$
    $if \ (cx < dx) \ 1$
    $\{elseif \ (cx > dx) \ 2$
    $\{ elseif \ (cy < dy) \ 3$
    $\{ elseif \ (cy > dy) \ 4$
    $\{ elseif \ (cy = dy) \wedge (cx = dx) \ 5$
    $else \quad 0 \ \}\}\}\}$
$elseif \ \{ (rp_{in} = 3) \wedge (R4 = data - in) \}$
    $if \ (cx < dx) \ 1$
    $\{elseif \ (cx > dx) \ 2$
    $\{ elseif \ (cy < dy) \ 3$
    $\{ elseif \ (cy > dy) \ 4$
    $\{ elseif \ (cy = dy) \wedge (cx = dx) \ 5$
    $else \quad 0 \ \}\}\}\}$
$else \quad 0$
```

Each of the request signal to output channel is either asserted as 1 or 0, depending the routing logic decoded for the destination. Only one request can be asserted at a time by a particular input channel. Each input channel has maximum four requests, a port cannot send a request from its own input channel to the same output channel. We have shown five requests, since, we present a generic input channel model covering all request signals after decoding the destination.

### 6.3.2. Invariants of Router

We have defined the state variables and wire signals of our router design in previous section. This design has a very large state space but we need to verify only those states which are defined or of use in our design specification/model. Therefore, we need invariants to capture only the reachable states of our system. Invariants are either the range of values of state variables which cannot be exceeded, or they are the predicate conditions on state componenets which always remain true in each and every state. Invariants are used to restrict our verification effort to the reachable states of implementation. Unreachable states can often be inconsistent and create spurious bugs i.e the bugs which are actually not present in the system. Verification of a system actually amounts to correctly identifying all the essential/required invariants of a system. Our router model has a number of invariant which are described below.

### 6.3.2.1. FIFO Buffer Invariants

In a parallel FIFO buffer the control logic to update the read and write pointers is complex as compared to the series FIFO design. We need to show that the read and write pointers get updated in a right order while remaining consistent with the buffer status flags. A read pointer should always point to the entry whose flag is 1, except when the buffer is empty. A write pointer should always point to an entry whose flag is 0, unless the buffer is full. Both the pointers are equal only when buffer is either full or empty. The fifo registers get updated in order as R0, R1, R2 and R3. Therefore, based on this order the four flags f0,f1,f2 and f3 cannot have some combinations because read and write actions happen in order of first-in,first-out. Also, the write and read pointers cannot point to the wrong entries while violating the fifo order.

$$inv1_f : (f0^{'}f1f2^{'}f3 \ \vee \ f0f1^{'}f2f3^{'})'$$

As the FIFO is a first-in and first-out design, so the four flags cannot have the bit combinations of 1010 and 0101. $inv1_f$ describes the forbidden flag combinations and status.

The position of read and write pointers is related to the status of individual flags for each register. There are only certain allowed positions for the read and write pointers depending on the flags status. The next invariant $inv2_f$ captures those requirements. It makes sure to cover all possible conditions for the four buffer flags and it is applicable to the both the input and output FIFO buffers.

$$
\begin{aligned}
\text{inv2}_f : \{ \; & \{ \; f1'f2'f3'f4 \rightarrow \; (wp = 0) \; \wedge \; (rp = 3) \; \} \; \wedge \\
& \{ \; f1'f2'f3f4' \rightarrow \; (wp = 3) \; \wedge \; (rp = 2) \; \} \; \wedge \\
& \{ \; f1'f2f3'f4' \rightarrow \; (wp = 2) \; \wedge \; (rp = 1) \; \} \; \wedge \\
& \{ \; f1f2'f3'f4' \rightarrow \; (wp = 1) \; \wedge \; (rp = 0) \; \} \; \wedge \\
& \{ \; f1'f2'f3f4 \; \rightarrow \; (wp = 0) \; \wedge \; (rp = 2) \; \} \; \wedge \\
& \{ \; f1'f2f3f4' \; \rightarrow \; (wp = 3) \; \wedge \; (rp = 1) \; \} \; \wedge \\
& \{ \; f1'f2f3f4 \; \rightarrow \; (wp = 0) \; \wedge \; (rp = 1) \; \} \; \wedge \\
& \{ \; f1f2'f3'f4 \; \rightarrow \; (wp = 1) \; \wedge \; (rp = 3) \; \} \; \wedge \\
& \{ \; f1f2'f3f4 \; \rightarrow \; (wp = 1) \; \wedge \; (rp = 2) \; \} \; \wedge \\
& \{ \; f1f2f3'f4' \; \rightarrow \; (wp = 2) \; \wedge \; (rp = 0) \; \} \; \wedge \\
& \{ \; f1f2f3'f4 \; \rightarrow \; (wp = 2) \; \wedge \; (rp = 3) \; \} \; \wedge \\
& \{ \; f1f2f3f4' \; \rightarrow \; (wp = 3) \; \wedge \; (rp = 0) \; \} \; \wedge \\
& \{ \; f1f2f3f4 \; \vee \; f1'f2'f3'f4' \; \rightarrow \; (wp = rp) \; \} \; \}
\end{aligned}
$$

### 6.3.3. Specification of Router Output Channel

The output channel of a router port is similar to the input channel with a few changes. There is a same kind of FIFO at output channel and, instead of routing/decoding logic, it has the arbitration unit. We define the state components and signals, respectively, of an output channel in general as given below.

$$S_{out} = \{wp_o, rp_o, f0_o, f1_o, f2_o, f3_o, R0, R1, R2, R3\}$$

$$\mathcal{W}_{out} = \{full_o, empty_o, g_1, g_2, g_3, g_4, arb_{sel}, ack_o\}$$

Almost all the state components are same as input state components, only a few exceptions. The functions for updating the FIFO flags, read-write pointers and the registers are almost same as for input channel. When a read operation happens at the input channel, the write operation happens at the output channel at a same time. This corresponds to the movement of packet from the input channel to the respective output channel, establishing a path from input to output through a router. So, the events that increment the read pointer at input channel are same as the events that increment the write pointer at output channel.

#### 6.3.3.1. Arbiter

An arbiter unit is used to decide which input request should be granted if the multiple requests are sent to the same output channel. We use a fixed priority arbiter design. The aribiter circuit monitors the input requests and generates a grant signal in response to a particular request according to the assigned priority order. The arbiter logic is implemented in the output channel. First, we describe the functions for grant signals as given below:

$$\begin{aligned}
g_1 \quad &\leftarrow \quad if \;\; (full_{out} = 0) \;\wedge\; (r_1 = 1) \qquad (r_1) \\
&\quad else \qquad g_1 = 0 \\
g_2 \quad &\leftarrow \quad if \;\; (full_{out} = 0) \;\wedge\; (r_2 = 1) \qquad (r_1^{'} r_2) \\
&\quad else \qquad g_2 = 0 \\
g_3 \quad &\leftarrow \quad if \;\; (full_{out} = 0) \;\wedge\; (r_3 = 1) \qquad (r_1^{'} r_2^{'} r_3) \\
&\quad else \qquad g_3 = 0 \\
g_4 \quad &\leftarrow \quad if \;\; (full_{out} = 0) \;\wedge\; (r_4 = 1) \qquad (r_1^{'} r_2^{'} r_3^{'} r_4) \\
&\quad else \qquad g_4 = 0
\end{aligned}$$

In above functions, $r_1, r_2, r_3$ and $r_4$ are the four request signals asserted according to the XY decoding logic and are described previously. Only one of the grant signal among the four is asserted at a time by any output channel. Based on which grant signal is asserted, the control signals are updated to the crossbar. Each output channel send the control lines to the crossbar section. These control lines are the select lines of a multiplexer, indicated as $arb_{sel}$, which connect one of the input channel data lines to the output channel. If all the four grant signals are unasseted i.e. at 0 value, then the select lines disconnect all the input channels from the output channel. The function for $arb_{sel}$ is given below:

$$\begin{aligned} arb_{sel} \leftarrow \; & if \quad (g_1 = 1) \quad (arb_{sel} = \text{input}_{CH1}) \\ & elseif \;\; (g_2 = 1) \quad (arb_{sel} = \text{input}_{CH2}) \\ & elseif \;\; (g_3 = 1) \quad (arb_{sel} = \text{input}_{CH3}) \\ & elseif \;\; (g_4 = 1) \quad (arb_{sel} = \text{input}_{CH4}) \\ & else \;\; (arb_{sel} = \text{disconnect}_{all-channels}) \end{aligned}$$

After the arbiter selects the corresponding input line, the packet moves into one of the FIFO registers at output channel. The mechanism for output FIFO write and read operations is similar as for the input channel with only the difference of signals initiating these operations.

## 6.4. NOC : Case Study

NOC has an interconnection of different routers to connect IPs together to form a specific topology. Real-time applications use NOC to communicate and share the information between different nodes. The communication between several nodes should happen in a timely manner. We showed how to verify the timing properties when NOC is employed in a real-time application on FPGA. To avoid complexity, we considered a simple XY-mesh topology. The specificiation for a router is described in the previous section. Our case study targeted a distributed application implemented on FPGA through an NOC. Basically, the objective is to verify the correctness of NOC circuit implementing the distributed application in a real-time safety-critical systems.

In our case study, there is a processing node at one end i.e. an active node. At the other end, there is a passive node i.e. a memory location which gets the processed data from active node after a fixed interval of time. At first node, referred as node A, we connected a controller for stepper motor (SM) control logic and at node B, there is the actual motor connected and it acts as a passive node. In between these two nodes is a connection of routers making up an NOC. The information generated by node A should be transmitted to the node B at a right time and in correct manner . The traffic pattern for this setup is known ahead of time since we know about the data generated by SM controller. This is the case typically in safety critical systems, that the traffic pattern can be known at the design time. Our goal was to verify that a packet generated at the source node is safely transmitted to the destination node in a correct and timely manner.

### 6.4.1. NOC Components

Multiple processing units can be integrated with the help of a FPGA based NOC. The circuit of NOC under consideration is implemented on FPGA. The processing unit here, referred as node A, is also an FPGA controller that generates the fixed bit-pattern for stepper motor wires for example 0001, 0010, 0100, 1000,0001.... The node A generates this bit-pattern after a fixed time interval when a counter counts upto certain range i.e. $counter = Max$. This bit-pattern information is packed with the destination address and a request variable is generated by the controller and sent to the network router. The network router acknowledges this request $req_{in}$ and asserts $ack_{in}$ signal.

$$\begin{array}{llll} {req_{in}}' \leftarrow & if & (counter = Max) & req_{in} = 1 \\ & elseif & (counter \neq Max) \ \wedge \ (ack_{in} = 1) & req_{in} = 0 \\ & else & req_{in} & \end{array}$$

$ack_{in}$ signal in the router is described in section 6.3. The router decodes the incoming packet which resides in the input FIFO buffer and send the request to output channel and then the packet moves from one router to another in the network until it reaches the destination. Inside the NOC, every router architecture and functionality is the same. The path taken by a packet is determined by XY-decoding logic. The packet first moves in the X-direction. Once aligned in the X-axis direction, it then moves along the Y-direction.

### 6.4.2. Network Invariants

The state space of our proposed NOC case study is very large. But in actual, the circuit is assumed to perform in a certain way only. So we need to bound our verification efforts only to those states which are reachable in our system. We identified all the required inavriants before generating the correctness proof. An invariant is a predicate that remains true in the current state and in the next state.

$$\text{Invariant}_{\text{Current-state}} \longrightarrow \text{Invariant}_{\text{Next-state}}$$

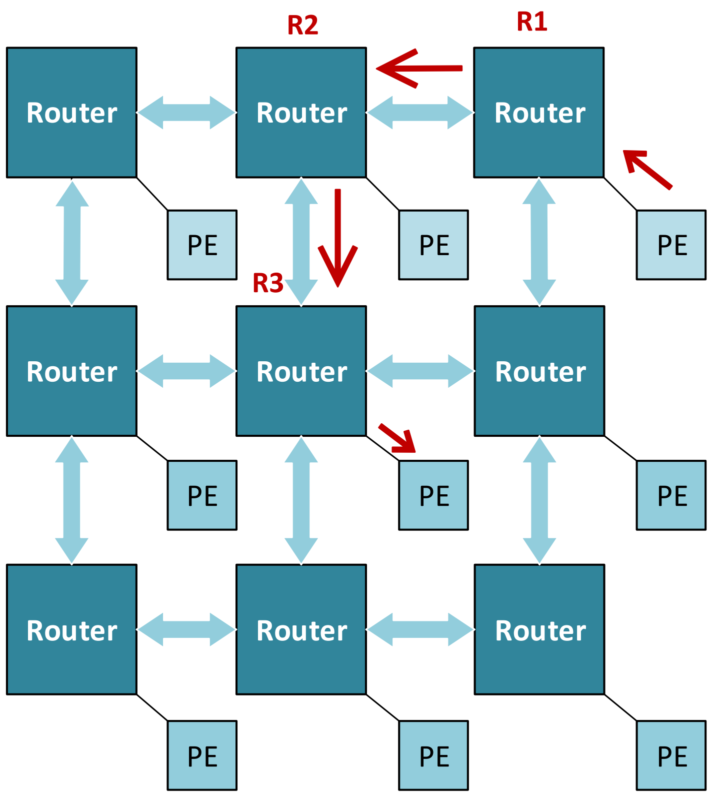

Figure 6.5. Depicting the Path Taken by Data in 2D NOC

Below we describe all the invariants that were constructed to perform the functional and timing verification of NOC step by step. There is a main timer to which all the operations are synchronized. In verification language we map that timer to a counter that can count only upto a certain maximum value i.e. $Max$.

$$\text{INV}_{counter}: \quad \text{counter} \quad \leq \quad Max$$

The SM controller sends the data packet into a buffer known as $Buffer_{in}$, before injecting it into the network. The $Buffer_{in}$ is a register and its width is equal to the packet size. The input buffer always holds only one of the packets generated by SM controller and no other data. It has a flag $b_{flag}$ which is asserted each time a new packet is sent into this buffer, otherwise this flag remain 0.

$$
\begin{aligned}
\mathrm{INV}_{buffer\text{-}in} &= (s_1\text{-}dxdy) \lor (s_2\text{-}dxdy) \lor (s_3\text{-}dxdy) \lor (s_4\text{-}dxdy)\\
\mathrm{INV}_{b\text{-}flag}{}' \leftarrow &\quad if \;\; (counter = max) \qquad b\text{-}flag = 1\\
&\quad elseif \;\; (counter \neq max) \qquad b\text{-}flag = 0
\end{aligned}
$$

We have considered a small path in 2D NOC from router R1 to router R3 as indicated in figure 6.5 with red color arrows. We define an invariant for the destination node, B, which is a passive node i.e. a register holding the packet transferred from node A. The invariant states that the destination register contains only one of the four states as sent by SM controller and not anything else.

$$\mathrm{INV}_{dest\text{-}reg} = s_1 \lor s_2 \lor s_3 \lor s_4$$

In order to track the packet movement within the network, there is a need to define an invariant that relates the counter value or range of counter values to the packet location inside NOC. We define two invariants for this purpose. The first invariant accounts for the overall range of counter tracing the packet movement in each stage inside routers and NOC. This invariant $\mathrm{INV}_{tracking}$ for the selected path is defined below:

$$
\begin{aligned}
\mathrm{INV}_{tracking}: \quad & (counter = 0 \;\rightarrow\; p1p2'p3'p4'p5'p6'p7') \;\land\\
& (counter = 1 \;\rightarrow\; p1'p2p3'p4'p5'p6'p7') \;\land\\
& (counter = 2 \;\rightarrow\; p1'p2'p3p4'p5'p6'p7') \;\land\\
& (counter = 3 \;\rightarrow\; p1'p2'p3'p4p5'p6'p7') \;\land\\
& (counter = 4 \;\rightarrow\; p1'p2'p3'p4'p5p6'p7') \;\land\\
& (counter = 5 \;\rightarrow\; p1'p2'p3'p4'p5'p6p7') \;\land\\
& (counter = 6 \;\rightarrow\; p1p2'p3'p4'p5'p6'p7) \;\land\\
& (7 \leq counter \leq Max \;\rightarrow\; p1p2'p3'p4'p5'p6'p7')
\end{aligned}
$$

In above definition $p1$-$p7$ are the predicate conditions. For each value of counter only one of the predicates is true and the others are false. $p1$ is the predicate for input buffer which gets updated by the SM controller before sending the packet into the network. The predicate $p2$ accounts for router R1 FIFO$_{in}$, that only one of the reisters is holding a valid packet and no other register is holding any other packet. Among other predicates, $p3$ accounts for output channel of R1. Similarly, $p4$ for input channel of R2, $p5$ for ouptut channel of R2, $p6$ for input channel of R3 and $p7$ for ouptut channel of R3 which is the final destination. These predicates are defined below.

$$\begin{aligned}
p1(\text{Buffer at node A}): \quad & b\text{-}flag = 1 \\
p2\ (\text{router}R1\text{- FIFO}_{in}): \quad & \{\ R0_{in} = SM_{pack}\ \wedge\ (f0_{in}f1'_{in}f2'_{in}f3'_{in})\ \}\ \vee \\
& \{\ R1_{in} = SM_{pack}\ \wedge\ (f0'_{in}f1_{in}f2'_{in}f3'_{in})\ \}\ \vee \\
& \{\ R2_{in} = SM_{pack}\ \wedge\ (f0'_{in}f1'_{in}f2_{in}f3'_{in})\ \}\ \vee \\
& \{\ R3_{in} = SM_{pack}\ \wedge\ (f0'_{in}f1'_{in}f2'_{in}f3_{in})\ \} \\
p3\ (\text{router}R1\text{- FIFO}_{out}): \quad & \Big\{\ \{\ (R0_{out} = SM_{pack})\ \wedge\ \ (R1_{out} \neq \text{dest-address})\ \}\ \wedge \\
& \quad (R2_{out} \neq \text{dest-address})\ \}\ \ \wedge\ \ (R3_{out} \neq \text{dest-address})\ \}\ \vee \\
& \{\ (R1_{out} = SM_{pack})\ \wedge\ \ (R0_{out} \neq \text{dest-address})\ \}\ \wedge \\
& \quad (R2_{out} \neq \text{dest-address})\ \}\ \ \wedge\ \ (R3_{out} \neq \text{dest-address})\ \}\ \vee \\
& \{\ (R2_{out} = SM_{pack})\ \wedge\ \ (R0_{out} \neq \text{dest-address})\ \}\ \wedge \\
& \quad (R1_{out} \neq \text{dest-address})\ \}\ \ \wedge\ \ (R3_{out} \neq \text{dest-address})\ \}\ \vee \\
& \{\ (R3_{out} = SM_{pack})\ \wedge\ \ (R0_{out} \neq \text{dest-address})\ \}\ \wedge \\
& \quad (R1_{out} \neq \text{dest-address})\ \}\ \ \wedge\ \ (R2_{out} \neq \text{dest-address})\ \}\ \Big\}
\end{aligned}$$

The definitions for predicates $p4$, $p5$, $p6$ and $p7$ are similar to the definition of $p3$, with the only difference that FIFO register names are changed for input and output channels of each router. $p4$ and $p6$ contain the input FIFO registers of routers R2 and R3, respectively. Similarly, $p5$ and $p7$ contain output FIFO registers of routers R2 and R3. These predicate conditions $p3$-$p7$ report that only one of the FIFO registers hold the desired data packet generated by node A, and all other registers do not have the same destination address. The later ensures that destination node does not receive anything from anyother source except node A. We define the two more invariants, the first one for the source router input FIFO flags INV$_{source\text{-}flags}$.

$$\begin{aligned}\text{INV}_{\textit{source-flags}}: \quad & \{counter = 1 \;\; \rightarrow \;\; (f0_{in}f1'_{in}f2'_{in}f3'_{in}) \;\; \vee \;\; (f0'_{in}f1_{in}f2'_{in}f3'_{in}) \;\; \vee \\ & \qquad\qquad\qquad\qquad (f0'_{in}f1'_{in}f2_{in}f3'_{in}) \;\; \vee \;\; (f0'_{in}f1'_{in}f2'_{in}f3_{in}) \; \} \;\; \bigwedge \\ & \{counter \neq 1 \;\; \rightarrow \;\; (f0'_{in}f1'_{in}f2'_{in}f3'_{in}) \; \}\end{aligned}$$

The second invariant is for the destination router R3's output FIFO flags $\text{INV}_{\textit{dest-flags}}$. These two channels in router R1 and router R3 are assumed to be receiving the packets at firmly defined time intervals.

$$\begin{aligned}\text{INV}_{\textit{dest-flags}}: \quad & \{counter = 6 \;\; \rightarrow \;\; (f0_{o}f1'_{o}f2'_{o}f3'_{o}) \;\; \vee \;\; (f0'_{o}f1_{o}f2'_{o}f3'_{o}) \;\; \vee \\ & \qquad\qquad\qquad\qquad (f0'_{o}f1'_{o}f2_{o}f3'_{o}) \;\; \vee \;\; (f0'_{o}f1'_{o}f2'_{o}f3_{o}) \; \} \;\; \bigwedge \\ & \{counter \neq 6 \;\; \rightarrow \;\; (f0'_{o}f1'_{o}f2'_{o}f3'_{o}) \; \}\end{aligned}$$

All of the above defined invariants are constructed to restrict the state space of implementation model. We successfully proved all invariants. These invariants were used together with some constraints to formulate the functional and timing proof obligations.

## 6.5. Functional Verification: Stuttering and Non-Stuttering Proofs

We perform formal verification of NOC circuit using the proof obligation (PO) templates defined in section 4.5 for stuttering and non-stuttering proofs. The path taken by a packet to move from source to the destination is a small one and it is known in specification. We can determine the time when a packet should be generated at source and update the destination memory. Based on this information we come up with the conditions for stuttering and non-stuttering phenomenon.

**Stuttering Proofs**

$$(r(w){=}s1) \;\; \wedge \;\; (counter \neq t_{ns}) \longrightarrow (r(v){=}s1) \;\; \wedge \;\; (rank(v) < rank(w))$$

$$(r(w){=}s2) \;\; \wedge \;\; (counter \neq t_{ns}) \longrightarrow (r(v){=}s2) \;\; \wedge \;\; (rank(v) < rank(w))$$

$$(r(w){=}s4) \;\; \wedge \;\; (counter \neq t_{ns}) \longrightarrow (r(v){=}s4) \;\; \wedge \;\; (rank(v) < rank(w))$$

$$(r(w){=}s8) \;\; \wedge \;\; (counter \neq t_{ns}) \longrightarrow (r(v){=}s8) \;\; \wedge \;\; (rank(v) < rank(w))$$

The refinement map $r(w)$ is identified here as the destination register-bits which show the SM bit-pattern received as a packet at the destination. The rank function is the difference between the counter value in state $w$ and the maximum value of counter $Max$. After reaching $Max$, the counter gets reset. These proof obligations show that the implementation system is not making progress with respect to the specification because the current and next both states are same. But at the same time it is also not caught in the deadlock situation which is justified by the rank function.

**Non-Stuttering Proofs**

$$(r(w){=}s1) \quad \wedge \quad (counter = t_{ns}) \longrightarrow \ (r(v){=}s2)$$

$$(r(w){=}s2) \quad \wedge \quad (counter = t_{ns}) \longrightarrow \ (r(v){=}s4)$$

$$(r(w){=}s4) \quad \wedge \quad (counter = t_{ns}) \longrightarrow \ (r(v){=}s8)$$

$$(r(w){=}s8) \quad \wedge \quad (counter = t_{ns}) \longrightarrow \ (r(v){=}s1)$$

The transition that updates a new packet at the destination register, is the non-stuttering transition. This transition takes place when the counter reaches a certain value $t_{ns}$. At the time other than $t_{ns}$, the system is stuttering. In our case, the packet traverses through 3 routers and it takes 6 cycles ($t_{ns}$=6clock-cycles) in overall to reach from source to destination. The time taken by the packet to travel through the network is determined by the router and NOC design plus the traffic patterns. We use some constraints which describe the conditions that the proposed system is not going to have those. Using these constraints, invariants and the current state conditions, we prove all the stuttering and non-stuttering proof obligations.

## 6.6. Timing Proofs of NOC

For timing proofs, we use the three PO templates defined in section 4.6. The first timing PO accounts for a constant rank difference through each transition in stuttering segment. The second PO accounts for the *entry-state-rank* value. During a stuttering phase, the maximum value of rank is given by *entry-state-rank*. This PO is encoded for each non-stuttering transition. The first two POs lay the foundation for proving the third PO which is the actual timing PO. The final PO verifies that the difference between the maximum rank value i.e. *entry-state-rank* and the *exit-state-rank* remains within the time bounds of the specification. We successfully prove the correctness of timing properties in our distributed application mapped onto the NOC. The verification statistics for current case study are given below in the table 6.1.

Table 6.1. Verification Statistics for NOC Case Study

| Particular Aspect | Count |
|---|---|
| # of Functional POs | 8 |
| # Timing POs | 12 |
| # of Invariants | 14 |

## 6.7. Related Work : NOC Verification

NOC is a very custom design architecture. The router in NOC is a distributed and complex system with non-linear behavior. Simple equations cannot chracterize the behavior of a router completely. The routing fabric is typically verified to prove some fundamental properties which include mutual exclusion, starvation freedom, deadlock freedom and liveness. NOC is employed in many-core system on chip (MC-SOC) platform to support critical operations in real-time and mission critical systems. These is much work about the verificaton of NOC used in MC-SOCs.

Y. Chen et al. [80] have proposed a methodology to analyze the formal models essential for verifying the above mentioned four fundamental properties. The proposed technique was applied on a specific bidirectional channel NOC using a popular model checking tool known as State Graph Manipulators(SGM). A. Zaman and O. Hasan [81] proposed a generic methodology for formal verification of any circuit switched NOC architecture. They modeled the NOC architecture using the PROcess MEta LAnguage (PROMELA language) and verified the desired functional properties using SPIN model checker. D. E. Holcomb et al. [82] have used compositional reasoning to prove the latency bound properties in NOCs. A single large proof is broken down into several smaller proof sub-goals. If all smaller proof sub-goals are proved then the original larger property is also proved to be correct. If the larger property is a latency bound of N number of cycles through a network, then a usual breakdown would be smaller bounds as a packet moves through the smaller paths or between the specific routers. These proof sub-goals are known as latency lemmas. Authors have developed the methods to discover and apply such latency lemmas. Their technique is based on model checking and it takes more time to get verification results. By contrast, our work is based on theorem proving and takes very short time to prove the verification results.

D. Holcomb [83] and V. A. Palaniveloo [84] have proposed the abstraction-based model checking approaches for latency verification in NOC. They explicitly decompose the bigger problem into discrete subproblems. Viktorov and Gotmanov [85] suggest a theorem-proving approach for latency verification in xMAS (executable microarchitectural) networks, which is based on ranking functions. Our approach also uses rank functions to account for the timing properties, but we target the RTL design of distributed applications in FPGAs. The concept of our verification approach is similar to ranking functions [86], i.e. the functions having numerical values indicating the progress of a state model. Generally, the rank function is used to prove the termination or liveness properties, but they can also be used to reason for latency bounds. We have applied the same idea in our approach.

## 6.8. Conclusion

We successfully proved the functional and the timing proof obligations for our case study. Some environmental constraints were defined to abstract the irrelevant details in the network which were not important to verify the particular path taken by the packets. We used XY- routing for 2D mesh topology. A packet can travel through many stages in a network among routers, until it reaches the final destination. At every time moment, the location of packet in the network is tracked and validated with the help of defined invaraints. The implementation states were identified to be either stuttering or non-stuttering states by applying the appropriate PO templates as defined in section 4.5. The timing proof obligation templates defiend in section 4.6 were used to verify the timing properties. We showed that the system with the NOC satisfies the higher-level specification under some assumptions. Our developed methodology can be used with some adaptation to verify any real-time application implemented on FPGA that uses NOC architecture.

# 7. CONCLUSION

We live in the era of digital world where we are surrounded with a variety of different digital systems. For example, our cell phones, automobiles, transportation, medical devices, and control systems in industries. Most of these digital systems are real-time in nature. Real-time systems essentially require processing the information and generating the response within fixed timeline known as deadline. The response generated after the deadline, even if logically correct can be equivalent to a wrong response or of no use. We have become dependent on real-time systems to an extent that if there is some defect in a system, it affects us largely as a consequent. Safety-critical systems are all real-time systems. If there is a failure in a safety-critical system, it leads to unacceptable consequences and irreversible impairment like a human life loss or a huge financial loss etc. Examples of safety critical system are nuclear power plants, automobiles, mission-critical systems, traffic surveillance systems, airplanes, medical devices and surgical robots.

It is indispensable to prevent any unsafe operating mode of operation in safety-critical real-time systems. The safe operation requires to verify the system before it is launched commercially. The traditional techniques used for verifying the system correctness are based on testing methods. When a design artifact is ready, it is checked against different inputs to monitor and evaluate the behavior. But in large and complex systems it is not possible to cover all possible scenarios for testing. It requires a lot of effort and testing also does not expose the corner-case-bugs. A more authentic approach is to apply formal verification to check the system response for correctness. Formal techniques employ rigorous mathematical reasoning. The real-world systems are manipulated as mathematical models by abstracting the system behavior. Formal proofs are constructed on the mathematical model of system and a correspondence exists between the mathematical model and the actual system by construction. Only formal verification can give guarantee that a system is bug free.

We focused our research to provide a formal verification approach to prove the correctness of real-time system applications. There is a shifting trend of using FPGAs, instead of software or microcontroller based solutions, in high end and complex systems because of many advantages offered by FPGAs. The specific notable advantages are the predictable timing behavior and scalability in

FPGA systems. In this regard, we devised a technique to formally verify the RTL circuit designs for FPGA based real-time applications. Our technique is based on deductive verification method in which we developed proof obligation templates to account for functional and timing correctness of a system. Any system should satisfy either a stuttering or a non-stuttering proof obligation at any time during its operation. Our timing proof obligations offer a more direct approach to reason for the latency bounds and timing correctness. We described the proposed technique with the help of some case studies in chapter 4. We extended this methodology to the applications which use network on chip(NOC) in their design. The case study for NOC based real-time system design is covered in chapter 6. For the case studies, the implementation model i.e. the RTL circuit was encoded in the SMT language and its satisfiability was checked using an automated theorem prover Z3. The results revealed that our devised technique can be applied to verify a certain class of applications which are implemented on FPGAs. The developed technique is capable to directly verify the implementation circuits by satisfying the proof obligations using an SMT solver. It offers efficient verification times.

In future work, we aim to discover the applicability of this technique to the Application Specific Integrated Circuits (ASICs). ASIC is implemented as a circuit and it also exhibits the nice property of having same delay, equal to the delay of the ASIC clock cycle, in each step like FPGAs.

# REFERENCES

[1] S. Malviya, H. Former, and M. FMS, "Five port router for network on chip," in *Proc. of ARRL and TAPR Digital Communications Conference*, 2010.

[2] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, "RASoC: A router soft-core for networks-on-chip," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 3. IEEE, 2004, pp. 198–203.

[3] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das, "Vichar: A dynamic virtual channel regulator for network-on-chip routers," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, Dec 2006, pp. 333–346.

[4] P. Schnoebelen, "The complexity of temporal logic model checking." *Advances in modal logic*, vol. 4, no. 393-436, p. 35, 2002.

[5] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal—a tool suite for automatic verification of real-time systems," *Hybrid Systems III*, pp. 232–243, 1996.

[6] S. Yovine, "Kronos: A verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 123–133, 1997.

[7] H. Christopherson, W. Pickell, A. Koller, S. Kannan, and E. Johnson, "Small adaptive flight control systems for UAVs using FPGA/DSP technology," in *AIAA 3rd" Unmanned Unlimited" Technical Conference, Workshop and Exhibit*, 2004, p. 6556.

[8] Y. Lin, "Using FPGAs to solve challenges in industrial applications," *EETimes http://www.eetimes.com/design/programmable-logic/4230820/Using-FPGAs-tosolve-challenges-in-industrial-applications*, 2011.

[9] G. Fichtinger, P. Kazanzides, A. M. Okamura, G. D. Hager, L. L. Whitcomb, and R. H. Taylor, "Surgical and interventional robotics: part ii: surgical cad-cam systems," *IEEE robotics & automation magazine/IEEE Robotics & Automation Society*, vol. 15, no. 3, p. 94, 2008.

[10] Y. Chen and V. Dinavahi, “Multi-FPGA digital hardware design for detailed large-scale real-time electromagnetic transient simulation of power systems,” *IET Generation, Transmission Distribution*, vol. 7, no. 5, pp. 451–463, May 2013.

[11] R. Dobias and H. Kubatova, “FPGA based design of the railway’s interlocking equipments,” in *Euromicro Symposium on Digital System Design, 2004. DSD 2004.*, Aug 2004, pp. 467–473.

[12] T. Huffmire, B. Brotherton, T. Sherwood, R. Kastner, T. Levin, T. D. Nguyen, and C. Irvine, “Managing security in FPGA-based embedded systems,” *IEEE Design & Test of Computers*, vol. 25, no. 6, 2008.

[13] E. M. Clarke, E. A. Emerson, and A. P. Sistla, “Automatic verification of finite-state concurrent systems using temporal logic specifications,” *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 2, pp. 244–263, Apr. 1986. [Online]. Available: http://doi.acm.org.ezproxy.lib.ndsu.nodak.edu/10.1145/5397.5399

[14] P. Manolios, “Mechanical verification of reactive systems,” Ph.D. dissertation, University of Texas at Austin, August 2001, http://www.ccs.neu.edu/home/pete/research/phd-dissertation.html.

[15] R. Milner, “An algebraic definition of simulation between programs,” Stanford, CA, USA, Tech. Rep., 1971.

[16] C. Kern and M. R. Greenstreet, “Formal verification in hardware design: a survey,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 4, no. 2, pp. 123–193, 1999.

[17] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS’08/ETAPS’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: http://dl.acm.org/citation.cfm?id=1792734.1792766

[18] ——, “Satisfiability modulo theories: Introduction and applications,” *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/1995376.1995394

[19] SMT-LIB, http://smtlib.cs.uiowa.edu/.

[20] I. Kuon, R. Tessier, and J. Rose, “FPGA architecture: Survey and challenges,” *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[21] Y. Hu, “Exploring formal verification methodology for FPGA-based digital systems,” *Sandia National Laboratories, New Mexico, California*, 2012.

[22] J. C. Godskesen, K. G. Larsen, and A. Skou, “Automatic verification of real-timed systems using epsilon,” *BRICS Report Series*, vol. 1, no. 19, 1994.

[23] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, “Timing verification of dynamically reconfigurable logic for the xilinx virtex FPGA series,” in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. ACM, 2002, pp. 127–135.

[24] J. Bengtsson and W. Yi, “Timed automata: Semantics, algorithms and tools,” in *Lectures on concurrency and petri nets*. Springer, 2004, pp. 87–124.

[25] C. Daws and S. Yovine, “Two examples of verification of multirate timed automata with Kronos,” in *Proceedings 16th IEEE Real-Time Systems Symposium*, Dec 1995, pp. 66–75.

[26] K. Havelund, A. Skou, K. G. Larsen, and K. Lund, “Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal,” in *Proceedings Real-Time Systems Symposium*, Dec 1997, pp. 2–13.

[27] C. Daws, A. Olivero, and S. Yovine, “Verifying et-lotos programs with Kronos,” in *Proc. FORTE*, vol. 94. Citeseer, 1994, pp. 227–242.

[28] H. E. Jensen, K. G. Larsen, and A. Skou, “Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL,” *BRICS Report Series*, vol. 3, no. 24, 1996.

[29] K. Kristoffersen, F. Laroussinie, K. Larsen, P. Pettersson, and W. Yi, “A compositional proof of a real-time mutual exclusion protocol,” *TAPSOFT’97: Theory and Practice of Software Development*, pp. 565–579, 1997.

[30] F. Wang, “Formal verification of timed systems: a survey and perspective,” *Proceedings of the IEEE*, vol. 92, no. 8, pp. 1283–1305, Aug 2004.

[31] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed i/o automata: A complete specification theory for real-time systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/1755952.1755967

[32] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *International Conference on Computer Aided Verification.* Springer, 2007, pp. 121–125.

[33] R. L. Eveleens, "Integrated modular avionics development guidance and certification considerations," *Mission Systems Engineering*, vol. 2, pp. 1120–1132, 2006.

[34] T. A. Henzinger, S. Qadeer, and S. K. Rajamani, *You assume, we guarantee: Methodology and case studies.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 440–451. [Online]. Available: http://dx.doi.org/10.1007/BFb0028765

[35] A. Cimatti, M. Dorigatti, and S. Tonetta, "Ocra: A tool for checking the refinement of temporal contracts," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2013, pp. 702–705.

[36] E. Endres, C. Müller, A. Shadrin, and S. Tverdyshev, "Towards the formal verification of a distributed real-time automotive system," 2010.

[37] L. C. Paulson, *Isabelle: A generic theorem prover.* Springer Science & Business Media, 1994, vol. 828.

[38] P. Conmy and I. Bate, "Component-based safety analysis of FPGAs," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 2, pp. 195–205, May 2010.

[39] J. D. Poole, "Model-driven architecture: Vision, standards and emerging technologies," in *Workshop on Metamodeling and Adaptive Object Models, ECOOP*, vol. 50, 2001.

[40] *DO-254 Design Assurance Guidance for Airborne Elecronic Hardware*, Radio Technical Commission for Aeronautics (RTCA), April 2000.

[41] H. Gall, "Functional safety iec 61508/iec 61511 the impact to certification and the user," in *Computer Systems and Applications, 2008. AICCSA 2008. IEEE/ACS International Conference on.* IEEE, 2008, pp. 1027–1031.

[42] J. Hammarberg and S. Nadjm-Tehrani, "Formal verification of fault tolerance in safety-critical reconfigurable modules," *International Journal on Software Tools for Technology Transfer*, vol. 7, no. 3, pp. 268–279, 2005. [Online]. Available: http://dx.doi.org/10.1007/s10009-004-0152-y

[43] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy *et al.*, "The coq proof assistant reference manual: Version 6.1," Ph.D. dissertation, Inria, 1997.

[44] H. Deng, "Formal verification of FPGA based systems," Ph.D. dissertation, McMaster University, 2011.

[45] N. S. Bjørner, Z. Manna, H. B. Sipma, and T. E. Uribe, "Deductive verification of real-time systems using STep," *Theoretical Computer Science*, vol. 253, no. 1, pp. 27–60, 2001.

[46] "Xilinx simulation tool," http://www.utdallas.edu/~poras/courses/ee3320/xilinx/upenn/FoundationtutorialFunctionalandTimingsimulation.htm, Accessed: 03-20-2017.

[47] I. Robertson, J. Irvine, P. Lysaght, and D. Robinson, "Timing verification of dynamically reconfigurable logic for the xilinx virtex FPGA series," in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, ser. FPGA '02. ACM, 2002, pp. 127–135. [Online]. Available: http://doi.acm.org/10.1145/503048.503068

[48] H. Mangassarian, B. Le, and A. Veneris, "Debugging RTL using structural dominance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 153–166, Jan 2014.

[49] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 10, pp. 1606–1621, Oct 2005.

[50] J. Urdahl, D. Stoffel, and W. Kunz, "Path predicate abstraction for sound system-level models of RT-level circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 2, pp. 291–304, 2014.

[51] N. Milivojevic, M. Krishnamurthy, Y. Gurkaynak, A. Sathyan, Y. J. Lee, and A. Emadi, "Stability analysis of FPGA-based control of brushless dc motors and generators using digital pwm technique," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 1, pp. 343–351, Jan 2012.

[52] S. D. Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta, "SA-FEMIP: A self-adaptive features extractor and matcher ip-core based on partially reconfigurable FPGAs for space applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2198–2208, Oct 2015.

[53] V. Kharchenko, A. Kovalenko, O. Siora, and V. Sklyar, "Security assessment of FPGA-based safety-critical systems: US NRC requirements context," in *2015 International Conference on Information and Digital Technologies*, July 2015, pp. 132–138.

[54] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive control using an FPGA with application to aircraft control," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 1006–1017, May 2014.

[55] C. Bernardeschi, L. Cassano, and A. Domenici, "SRAM-based FPGA systems for safety-critical applications: A survey on design standards and proposed methodologies," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 373–390, 2015.

[56] G. R. Jagadeesh, T. Srikanthan, and C. M. Lim, "Field programmable gate array-based acceleration of shortest-path computation," *IET Computers Digital Techniques*, vol. 5, no. 4, pp. 231–237, July 2011.

[57] A. Sanyal, K. Chakrabarty, M. Yilmaz, and H. Fujiwara, "RT-level design-for-testability and expansion of functional test sequences for enhanced defect coverage," in *2010 IEEE International Test Conference*, Nov 2010, pp. 1–10.

[58] E. Bareisa, V. Jusas, K. Motiejunas, and R. Seinauskas, "Path delay test generation at functional level," *IET Computers Digital Techniques*, vol. 9, no. 3, pp. 135–141, 2015.

[59] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.

[60] M. A. L. Dubasi, S. K. Srinivasan, and V. Wijayasekara, "Timed refinement for verification of real-time object code programs," in *Working Conference on Verified Software: Theories, Tools, and Experiments.* Springer, 2014, pp. 252–269.

[61] S. Shuja, S. K. Srinivasan, S. Jabeen, and D. Nawarathna, "A formal verification methodology for DDD mode pacemaker control programs," *Journal of Electrical and Computer Engineering*, vol. 2015, p. 57, 2015.

[62] H. Mangassarian, B. Le, and A. Veneris, "Debugging RTL using structural dominance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 1, pp. 153–166, 2014.

[63] J. Urdahl, D. Stoffel, and W. Kunz, "Path predicate abstraction for sound system-level models of RT-level circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 2, pp. 291–304, 2014.

[64] A. Boudjadar, J. P. Bodeveix, and M. Filali, "Compositional refinement for real-time systems with priorities," in *2012 19th International Symposium on Temporal Representation and Reasoning*, Sept 2012, pp. 57–64.

[65] P. Manolios, *Mechanical verification of reactive systems.* University of Texas, 2001.

[66] S. Jabeen, S. K. Srinivasan, S. Shuja, and M. A. L. Dubasi, "A formal verification methodology for FPGA-based stepper motor control," *IEEE Embedded Systems Letters*, vol. 7, no. 3, pp. 85–88, Sept 2015.

[67] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam, "Modeling and verification of a dual chamber implantable pacemaker," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 188–203, 2012.

[68] “Stepper motors keep surgical viewing system light-weight,” http://www.micromo.com/applications/medical-lab-automation-equipment/volk-optical-surgical-viewing, Accessed: 10-08-2014 .

[69] J. Churchill and W. Volmary, “Stepper motor throttle controller,” Apr. 2 1991, US Patent 5,003,948. [Online]. Available: http://www.google.com/patents/US5003948

[70] Y. Juan, H. Kang, Y. Wang, and G. Lu, “A control system of three-axis stepper motor based on the FPGA,” in *Mechatronic Sciences, Electric Engineering and Computer (MEC), Proceedings 2013 International Conference on*, Dec 2013, pp. 3334–3337.

[71] N. Dahm, M. Huebner, and J. Becker, “Approach of an FPGA based adaptive stepper motor control system,” in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, June 2011, pp. 1–6.

[72] K. Zaferullah, R. Bansode, S. Pethe, M. Vidwans, and K. Dsouza, “Speed control of stepper motor for collimator jaws positioning based on FPGA implementation,” in *Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014 International Conference on*, April 2014, pp. 353–357.

[73] M. A. L. Dubasi, S. K. Srinivasan, and V. Wijayasekara, “Timed refinement for verification of real-time object code programs,” in *Lecture Notes in Computer Science series*, vol. 8471, 2014.

[74] J. A. Rowson and A. Sangiovanni-Vincentelli, “Interface-based design,” in *Proceedings of the 34th annual Design Automation Conference.* ACM, 1997, pp. 178–183.

[75] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, Aug 2005.

[76] Q. Sun, L. Zhang, and A. Du, “Design and implementation of the wormhole virtual channel NoC router,” in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 01, Dec 2015, pp. 854–858.

[77] S. Shermi and C. S. Arun, “A novel architecture of bidirectional NoC router using flexible buffer,” in *2016 International Conference on Emerging Technological Trends (ICETT)*, Oct 2016, pp. 1–6.

[78] Kiran and K. Solanki, “Design of efficient NOC router for chip multiprocessor,” in *2016 International Conference on Inventive Computation Technologies (ICICT)*, vol. 3, Aug 2016, pp. 1–4.

[79] A. K. Chatrath, A. Gupta, and S. Pandey, “Design and implementation of high speed reconfigurable NoC router,” in *2016 International Conference on Inventive Computation Technologies (ICICT)*, vol. 3, Aug 2016, pp. 1–5.

[80] Y. R. Chen, W. T. Su, P. A. Hsiung, Y. C. Lan, Y. H. Hu, and S. J. Chen, “Formal modeling and verification for network-on-chip,” in *The 2010 International Conference on Green Circuits and Systems*, June 2010, pp. 299–304.

[81] A. Zaman and O. Hasan, “Formal verification of circuit-switched network on chip (NOC) architectures using SPIN,” in *2014 International Symposium on System-on-Chip (SoC)*, Oct 2014, pp. 1–8.

[82] D. E. Holcomb, A. Gotmanov, M. Kishinevsky, and S. A. Seshia, “Compositional performance verification of NoC designs,” in *Tenth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMCODE2012)*, July 2012, pp. 1–10.

[83] D. Holcomb, B. Brady, and S. Seshia, “Abstraction-based performance verification of NoCs,” in *Proceedings of the 48th Design Automation Conference.* ACM, 2011, pp. 492–497.

[84] V. A. Palaniveloo and A. Sowmya, “Formal estimation of worst-case communication latency in a network-on-chip,” in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on.* IEEE, 2012, pp. 15–20.

[85] Y. Viktorov and A. Gotmanov, “Latency analysis in microarchitectural models of communication fabrics,” *Problems of Advanced Micro-and Nanoelectronic Systems Development (MES)*, pp. 67–72, 2012.

[86] A. Turing, "Checking a large routine. paper for the EDSAC inaugural conference, 24 june 1949. typescript published in report of a conference on high speed automatic calculating machines," *Reprinted with corrections and annotations in L. Morris and CB Jones: An early program proof by Alan Turing, Ann. Hist. Computing*, vol. 6, no. 2, pp. 129–143, 1984.

# APPENDIX. LIST OF PUBLICATIONS

- S. Jabeen, S. Srinivasan, and S. Shuja. ”A Formal Verification Methodology for Real-Time FPGA.” IET Computers & Digital Techniques (2017).
- S. Jabeen, S. K. Srinivasan, S. Shuja and M. A. L. Dubasi, ”A Formal Verification Methodology for FPGA-Based Stepper Motor Control,” in IEEE Embedded Systems Letters, vol. 7, no. 3, pp. 85-88, Sept. 2015.
- S. Shuja, S.K. Srinivasan, S. Jabeen, and D. Nawarathna, ”A formal verification methodology for DDD mode pacemaker control programs,” Journal of Electrical and Computer Engineering, 2015, p.57.
- H. S. Kia, C. Ababei, S. Srinivasan and S. Jabeen, ”A new scalable fault tolerant routing algorithm for networks-on-chip,” 2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS), Fort Collins, CO, 2015, pp. 1-4.