FPGA Driven Synthesis Employing a Self-Testing VLSI Controller Implementation as a Case Study

Betina K. Hold,

(Bachelor Eng 1988)

McGill University, Montreal

November 1994

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering.

SHORTENED TITLE OF THESIS

FPGA Driven Synthesis Employing a Self-Testing VLSI Controller

Abstract

As the complexity of VLSI circuits increases and the time to market requirements become more stringent, the demands on the designer aggrandize while the challenges of achieving specified performance intensify. The skill of rapidly producing functional circuits for intra-system testing and prototyping is essential to the electronic market. Tools which aid the designer, thereby quickening the design cycle are indispensable. However, the methodologies employed by these tools, though maturing, still necessitate amelioration, as shall be shown through a conceivable case study.

Rapid prototyping in numerous digital application domains are served by synthesis and FPGA technology. Embedded controller systems touch one application area. The study of particular design examples can provide information on the current status of tools, their algorithms, and available hardware architectures and consequently furnish a list of deficiencies and efficacious enhancements towards further, automatically generated implementations. In fact, subsequent to in depth research into FPGA architectures, synthesis algorithms, and digital design techniques, one example application was realized. This particular case study lead to proposed specifications supporting foundations for an "Application Specific Design and Synthesis System", and their necessity particularly when design time is critical and real-time responses are in demand.

This thesis formulates, and implements an automotive Anti-lock Brake System (ABS), reporting on it's design simulation, synthesis, and eventual layout steps, from which extensions are drawn towards digital controllers onto FPGA technology, and the potential migration of the design onto ASIC technology. Implementation/environment fine-tuning of embedded controllers as such necessitate quickly prototyped circuit realizations. Examination of its functionality, real-time response, implementation, and testability is performed in an

attempt to measure the usefulness of higher level design entry facilities such as VHDL in a rapid prototyping environment. Continuous on-line testing is included using aperiodic sample injections where the resultant generated values are compared to signatures known a priori, without compromising functionality. The achievable area and timing aid in the determination of the efficiency of the process and provide fuel for an FPGA and/or ASIC migration path for eventual implementation. Commentaries and generalized methodologies are assembled from the design's simulation, synthesis and layout utilizing VHDL and FPGAs, illustrating CAD tool capabilities/requirements/limitations, with respect to real-time synthesis and rapid prototyping of general controller applications involving asynchronous elements.

Résumé

La complexité grandissante des circuits VLSI et la réduction de leurs échéances de mise en marché imposent une productivité croissante aux concepteurs dans le but de rencontrer les critères de performance demandés. L'expertise requise pour la production rapide de circuits prototypes complets et de prototypes fonctionnels en guise de substitution aux composantes non-conçues d'un système est essentielle aux tests de la phase de développement d'un produit du marché de l'électronique. Des outils sous forme de logiciels sont devenus indispensables pour accélérer le processus de conception. Cependant, bien que ces outils aient atteint un certain niveau de maturité, ils ont plusieurs défaillances qui doivent être améliorées. Cette constatation fera l'objet d'un cas d'étude pratique présenté dans cette thèse.

La conception rapide de prototypes dans plusieurs applications du domaine de l'électronique numérique emploient la synthèse digitale et les tableaux de portes logiques programmables (FPGA). Les contrôleurs digitaux encastrés à leurs systèmes représentent une telle application. L'examen de concepts digitaux pratiques fournira des données importantes sur l'efficacité de la synthèse, des algorithmes, et des architectures digitales disponibles à l'intérieur des outils en plus d'indiquer des voies menant à l'amélioration de la performance de ces mêmes outils. En effet, à la suite de recherches approfondies sur les architectures FPGA, les algorithmes de synthèse and la conception digitale, un exemple d'application à été conçu. Ce cas d'étude particulier à conduit à la proposition de justifications supportant la base dans le Domaine d'Application Spécifique (CSDAS) En particulier, la nécessité du CSDAS sera argumentée pour les produits ayant de courtes échéances de conception et les systèmes à reaction temporelle critique.

Cette thèse utilise la spécification d'un système de freins anti-blocage (ABS) pour illustrer le processus de simulation, synthèse, et la mise en circuit de contrôleurs digitaux sur des

FPGAs. En outre, la transformation possible de ces contrôleurs sur des circuits intégrés à application spécialisée (ASIC) est discutée. Les multiples étapes de raffinement dans la conception de contrôleurs digitaux encastrés requièrent la disponibilité rapide de circuits prototypes. Dans le cas du contrôleur ABS, la qualité des outils de haut niveau d'abstraction, comme VHDL, est mesurée selon les opérations fonctionnelles de l'implémentation résultante, la rencontre des contraintes de performance temporelles, et la capacité du circuit à être vérifié pour son bon fonctionnement.

Un méchanisme d'auto-vérification apériodique est intégré au système. Des vecteurs tests sont injectés dans le circuit et les réponses du système sont alors comparés aux signatures obtenues a priori d'un circuit sans défauts. Les tests se produisent de façon continuelle pendant l'opération normale du circuit et ce sans compromettre aucune des ses fonctions. La surface de silicium requise et la performance temporelle des circuits synthétisés contribuent à l'analyse de l'efficacité des outils de synthèse. De plus, ces données fournissent un cheminement vers l'implémentation éventuelle du contrôleur sur un FPGA ou un ASIC. D'autres résultats, acquis à partir d'outils sur le langage VHDL et de synthèse sur les FPGAs, génèrent des observations additionnelles. Ces résultats provenant de simulations, de synthèse et de mise en circuit démontrent les exigences, les limitations et les capacités des outils de conception assistée par ordinateur par rapport à la synthèse de contrôleurs ayant des composantes asynchrones et la rencontre de leurs contraintes temporelles.

Acknowledgements

The work presented has been supported partly by the Natural Sciences and Engineering Research Council of Canada (NSERC), by Dr. V. K. Agarwal, with the aide of the facilities of the McGill's Microelectronics and Computer Systems Laboratory (MACS) and the tools and workshops of the Canadian Microelectronics Corporation (CMC).

From a professional perspective, I would like take this opportunity to thank my supervisor, Prof. Vinod K. Agarwal for allowing me to pursue this area of study, for providing me with the means to accomplish whatever tasks were necessary, and for supporting my attendance to several conferences and workshops. I would also like to express thanks to Professor Bhatt for his co-supervision and continuous proof-reading of all my papers and thesis outlines. Owing to the strength of inter-university bonds, gratitude also goes to Professor Cerny, Université de Montréal, for offering me a chance to pursue PhD studies in an interesting and closely related field. And last, but definitely not least, I would like to thank Jindrich Zejda (Jindra) for his help in proof-reading and giving focus to this thesis, the fruit of my efforts.

On a personal note, I would like to express my sincere thanks to my parents, Anton and Anita Hold for their constant support and understanding throughout my Bachelor's and Master's, François Lambert for bearing with me, for providing unix administrative advice and for helping organize the figures and the final thesis format, and to my family, Markus and Susan Hold for understanding that I would resume my existence as an available family member once the never-ending days of lab-life would be finished. Je remercie Eric Masson pour son aide, ses réponses détaillées et bien pensées, sa traduction de l'abstract et son support moral, and Jacek Slaboszewicz, our indispensable system administrator for providing me with a superb working environment, and so many answers. I would also like to mention Mike Toner for his aide in scanning in the FPGA pictures, and most of all for his listening ear, his advising and his boosting. And lastly, I thank MOSAID Technologies Inc., my new employers, for offering me work in my preferred field of interest and providing that last incentive to finish.

Contents

List of Figures

Chapter 1

Introduction

In the 1940's the first electronic, programmable, general-purpose computer, the ENIAC, a breakthrough in its time, occupied an area 100 feet long, 8.5 feet high by several feet wide. Simple design took weeks, programming was manual, accompanied by cable connections and setting of switches, a task which took from half an hour to an entire day. Along analogous lines, in the 1990's, automatic design tools are flourishing and programmable hardware exists which can be reconfigured for an application in a matter of minutes occupying about 1 square inch of area. Hardware has evolved, circuit densities are increasing, permissible prototyping times have diminished, and tools which aid circuit realization have correspondingly been unfolded. From stand alone compilers and schematic entry packages, complete frameworks exist which take the designer and his/her specifications from design entry to implementation. But the questions arises, how efficient are they and is their maturity attainable? Additionally, as circuits grew larger and larger, testability became an issue, requiring controllable and observable in-circuit viewpoints. Ensuring that the resultant functionality of the hardware meets the original specifications upon fabrication and after in system utilization are difficult and grow notably with increasing circuit complexities. Conceptualization thus forth embodies testability, alongside design with its ever increasing constraint lists.

Controller designs were selected for the analysis as they represent a well-known and largely encompassing application domain. Furthermore, conjoining them with FPGA (Field Programmable Gate Array) technology, offers an alternative solution to the current *micro-controller-software* implementations. This additional motivation demands the assurance or surpassing of previously attained real-time effects while maintaining the ease of design en-

coding, re-encoding, compilation and testing for both faults and fine-tuning.

It is proposed in this dissertation that an investigation of *hardware* and *software* elements could produce a set of design, synthesis, testing, and implementation methodologies and recommendations for the electronics engineer and consequently facilitate future conceptualizations. Enhancements are proposed suggesting favorable evolutionary paths bridging the methodologies developed today with ones to be used tomorrow. Precluding an exclusive research-oriented study, an engineering approach to explore such hardware/software facilities/limitations was employed, utilizing case studies –*specify it, build it, test it, and document and evaluate the steps undertaken.* Tangible results could then be obtained, re-used, and then extrapolated to general circuit realizations and systems, and be labeled as application specific design methodologies forming the groundwork for *Application Specific Design, Synthesis and Layout* (ASDSL).

This engineering approach envelops the above methodologies employing *technology of the times*, which we refer to as Integrated Circuit (IC) technology, two of which are FPGAs and Synthesis tools. A schema, to optimize their exercise by designers in similar, future design and module undertakings, is developed, the evolution of which is best formulated through "hands-on" experimentation. Notwithstanding, the development of FPGA design and synthesis techniques requires analysis of their architectures and the tools which realize their circuits.

1.1 Background to the Real-time Controller Design and Synthesis onto FPGAs

The time-to-market expediency has invigorated the search for rapid circuit a..d system implementations with limited risks. FPGAs serve as excellent prototyping devices and additionally provide expeditious solutions in selected application areas. The evolution of FPGAs from Masked Programmable Gate Arrays (MPGAs) and Programmable Logic Devices (PLDs) must encompass not only architectural advances but must incite a proficiency in Electronic Design Automation, (EDA) tools as well. This thesis examines the above utilizing VHDL for design entry and separate tools for synthesis and layout onto FPGAs. Specification of timing and area constraints cannot be entered for a design in VHDL format, hence real time

constraints must be entered through a synthesis tool. Not all tools address this issue very well, many of them relying on the *design — simulate* iterative process.

Real-time implies an ability to respond to the fastest rate of arrival of the input events. Digital controller applications exhibit real-time qualities when their outputs are generated as soon as possible after the inputs are available within their sampling time interval. Real-time cannot be quantified in terms of time limits or intervals as individual system response times are highly application dependent. Furthermore, real-time control systems must be reliable, possess error detection capabilities and exhibit some kind of fault recovery mechanism such as an ability to restart or shut itself down. Traditionally, redundancy must also be built into the overall control system. Success in its design and implementation hinges on achieving this real time response utilizing a given technology. One of the motivations of this thesis is to explore applicability of FPGAs in real-time controller applications.

Some elements from Houpis and Lamont, [2], which receive attention in our design are: real-time task concurrency, synchronization, coordination and scheduling, response time, sectioning of internal and external timing (sampling), resource allocation, fault/error recovery, restart and system shutdown, and self testing. Though system controllers require control and data-path circuits and are most often of a synchronous nature, asynchronous factors do exist demanding, in some cases, particular treatment. Data-path logic includes operations for buffering, multiplexing, control, and data processing. Typical circuits require multiplexers, registers, buffers, FIFOs, counters, accumulators, ALUs, adders/subtracters/comparators, and are synchronous. The asynchrony emanates from the need to respond to externally sampled input and to realize the on-line self-testing mechanisms.

An ABS controller is one example of an embedded application class. A vehicle in motion is a complex, nonlinear and time varying system responding to its asynchronous environment. An ABS controller requires data from this vehicular system and its environs. Optimal control techniques and theories can provide suitable strategies for digital design and implementation, but the on-line computing requirements become excessive and somewhat time prohibitive. Hence, to obtain "real-time" responses, various assumptions, and simplifications of the ideal theoretic equations and models must frequently be proposed to reduce the on-line computing requirements while still retaining correct functionality.

Part of the implemented design comes from Kuo and Yeh [8], who describe mathematically a four-phase brake pressure control loop. No previous hardware implementation onto FPGAs for this theory is known to date, fueling the motivation for its selection for realization. Preliminary constraints entail ABS controller response time requirements which should surpass the human response time, in the order of milliseconds, while allocating adequate time to its actuators and test circuitry in addition to satisfying FPGA area and size limitations.

Rapid prototyping is one of the virtues resulting from FPGA technology. FPGAs present more restrictions to the design automation tool, while offering more flexibility to the designer. Their *Logic block* structures exhibit larger granular diversity than present ASICs with their gate granularity. Low level structures are fixed and a prelaid interconnection schema for routing signals and connecting blocks already exist, ready for circuit design or customization through the FPGA programmable capabilities. The resulting overhead suggests a need for circumventions and shortcuts using crafty approximations to achieve efficient utilization of the hardware. For an ABS controller, some of these include linearizing the tire-road friction versus slip dynamics about certain operating points and considering a simpler optimization problem. The vehicle velocity is assumed constant for short time intervals so that computations can complete their execution producing valid and current results. To further minimize the on-line computing time while maintaining precision, external (to the FPGA) hardware availability is assumed in the form of Read Only Memory (ROM). Quick to program, they can store large amounts of precalculated data, and competently complement the FPGA.

An additional goal of this thesis is to evaluate the applicability of FPGA technology in safety critical real-time applications. This study focuses on timing and fault/error detection in an FPGA environment. In the event of sudden braking and slippage under hazardous road conditions, the vehicle must respond quickly enough to regulate brake pressure to achieve optimal braking and steering. Should an error in operation occur, the ABS controller should be disabled to avoid incorrect braking interference and to guarantee safety of the driver.

Integrating all these elements through a case study, is an all-encompassing, difficult task. The specification and design of real-time systems not only requires extensive analysis, but also relies on modeling and simulation in order to develop the desired confidence in achieving system performance requirements before circuit realization. Even so, it is impossible to validate that the system performance requirements are achieved for all possible inputs given

the system resources. This applies to all of pre- and post-layout, and in-system renditions of the controller. Hence the need for intelligent designer controlled testing and validation are still necessary. Additionally, the shortcuts and design approximations which are required in the implementation phases must also be justified and accepted, a task which can be done through "special" simulations and mathematical analysis.

High level design languages provide a standard medium for communicating design data. VHDL was selected for design entry based on its ease of implementation, general acceptance and standardization, and its capability of allowing a designer to specify circuits at varying levels of abstraction. VHDL possesses traditional hierarchical level descriptions yet adds behavioral descriptions and user defined attributes. Though currently not the best for synthesis, as it was originally written for simulation purposes, VHDL is a good refinement of a hardware specification language which carries a potential for an efficient design entry method to evolving synthesis tools. Occasionally misleading in simulation, a clear understanding of the event-driven VHDL simulator will result in proper interpretation of results.

For vehicles with ABS, it is estimated that the quantity of some models will easily surpass 1000 units so that eventual migration to ASICs is feasible and thus considered in addition to FPGA implementation. The results of VHDL synthesis are judged on how well the timing requirements are met, testability, and the FPGA chip count required to implement the design, all of which affect the choice for possible migration to a smaller ASIC die.

1.2 Design Issues, Models, and Tools

Modeling a system's dynamics for eventual hardware implementation is analogous to writing a film script based on an existing novel. The director/designer must be able produce the best possible movie/design given the script/specifications and the actors/tools available. The actors/tools are interchangeable and replaceable while the script/specification provides the blueprints and fuels the strategy.

Circuit design embodies four main tasks: modeling, implementation encompassing both hardware and software elements, modification, and validation. Constraints are inherent to each task initially directing the synthesis process and ultimately to check correctness of the circuits realization. Modeling from initial specification consists of three elements: *structur-*

ing, encoding, and constraining. Structuring refers to the subdivision or organization of a design/application into smaller sub-blocks for design entry/definition. It also incorporates hierarchical organization, the number of levels of which depend on the original description. Encoding specifies how a design is entered into a Computer Design Aided(CAD) tool, for further processing such as simulation or synthesis. Constraining refers to the individual goals associated with each block or groups of blocks. Encoding and constraining depend on the choice of modeling language or library block representations and the implementation facilities whereby a subset of these elements, some general, some customized, are used.

An original description of a system for eventual modeling can come in several formats. Textual, pictorial, programming-oriented, mathematical, and petri-net or FSM type descriptions, to name a few, are common. The eventual modeling of the system described in this thesis was developed from textual descriptions and mathematical equations. The design platform for the realization of the above specification, focuses on VHDL descriptions and models as the avenue for design entry. The availability of hardware and software in the university environment rendered Xilinx XC4010 FGPAs as the chosen technology for circuit implementation, and Synopsys as the chosen tool for "automated circuit creation".

Accepting that there exist numerous paths from mathematical formulations to synthesis and eventual FPGA layout, we embarked on a search for mathematical formulas, or control dynamics which are better suited for FPGA implementation. Some useful approximations, optimizations, and implementation short-cuts have been unearthed through the construction of an ABS controller, and can be applicable to general controller applications.

In perspective, the focus of this dissertation rests on the exploration of VHDL design issues for FPGA synthesis of controller type applications using CAD tools. Constraint specifications, which are an integral constituent of any design tasks, can be used to direct synthesis and ultimately to check correctness of the circuits realization. Capabilities and limitations of the many facets in a design cycle, were sought to ultimately propose improvements in the areas of FPGA architecture, design entry, simulation, synthesis, and routing. These can then be formulated into a methodology for design and synthesis onto FPGA technology with particular emphasis on controller type applications, in attempt to parallel the functions of the traditional designer via automation. Producing a high performance and area efficient design meeting all constraints is an aspiration of designer and hence automation process

as well. Additionally, the analysis of testability procedures and their usefulness in FPGA implementations shall be performed.

1.3 Motivation and Research Goals

With the advent of programmable hardware, FPGA technology has since emerged. Avoiding costs and risks of traditional gate array design methodologies is one avenue towards which FPGAs can be directed. Exploration of the capabilities of such a hardware implementation is best performed through a design example, realized onto the technology, after which verification of timing and area can be done by the designer. The example chosen for realization, is that of an ABS Controller, which exhibits the desired application features, such as FSM control, synchronous and asynchronous functionality, datapath calculations, and embedded system circuit. Additionally, such an exercise will verify FPGA technology real-time capabilities in a safety critical control environment.

From a design entry and tool interpretation point of view, traditional schematic entry is less ambiguous than HDL encoding, as direct library-component matching during synthesis is possible and somewhat explicit. However, a fair amount of research remains to be done in order to resolve and produce an optimized and synthesized design with respect to VHDL and FPGAs, especially with commercial tools and the ever evolving FPGA architectures. Particularly, much work remains in exploiting technology specific FPGA architectures and pre-designed blocks (hard macros), and exploring synthesis, and design methodologies. Synthesis requires extensive design space exploration schemes and itself is an NP-hard problem. Creating an optimal implementation is, consequently, not a certainty and heuristics shall continuously be sought.

FPGAs are vendor and family specific possessing distinct architectures better adapted to certain design styles, or "niches". This thesis reports on one such niche by documenting the design process from specification to technology mapping.

Before embarking with background material, let us reiterate our goals:

- To explore VHDL design issues for FPGA synthesis of controller type applications using CAD tools. This embraces scrutiny of limitations and observance of features and

capabilities.

- To evaluate the suitability of CAD tools for such a design.
- To gauge the applicability of FPGA technology in safety critical real-time applications..

Likewise, our achievements can be summarized as follows:

- The development of a methodology for design and synthesis onto FPGA technology with particular emphasis on data-path/controller type applications. These applications revolve around processing externally received data in a controlled and ordered manner.
- Analysis of testability procedures, their feasibility and their usefulness in FPGA technology.
- List of limitations/capabilities of commercial synthesis tools.
- The implementation of a self-testing ABS controller onto FPGAs, based on recent four phase control loop theory and self-designed, complementary surrounding blocks and connecting logic.
- Realization of a controller onto programmable technology so that prototyping and in-system testing can be achieved with minimal cost.
- Presentation of an alternative solution to the traditional implementation of controller applications onto microprocessors and program memory (RAM:random accessible memory).

1.4 Author's Specific Contribution

The originality of this thesis stems not from an individual specific contribution but from an embodiment of diverse contributions fueling strategies and suggesting approaches and amelioration in future design and synthesis undertakings. Synthesis issues and algorithms from all levels -system level down to logic and boolean levels- were researched along with the numerous tools, both commercial and non-commercial. Considerable time was spent researching all available FPGA architectures, their construction blocks, special features and their synthesis tools, as well as implicit basics such as digital design techniques, programmable logic, synchronous and asynchronous systems, and Moore and Mealy machines. In addition, the extrinsic area of automotive mechanics and electronics was investigated to arrive at an implementable specification of an ABS Controller. A brief mention of all researched elements is included, and further details can be found from the extensive reference list located in the

bibliography. Subsequent to the research phase, the experimental phase ensued. The ABS Controller was specified and refined, succeeded by its encoding, simulation, synthesis, and eventual routing and layout. Results were then accumulated and interpreted, followed by the formulation of design methodologies and their documentation.

Briefly, the thesis is organized as follows: Chapter 2 presents background information and includes past work in the area of application specific synthesis. Chapter 3 introduces design and experimental methodologies and evaluation criteria. The Anti-lock Brake System (ABS) dynamics, the hardware specifications encompassing the self-testing features, are recited in Chapter 4, whereas implementation details, synthesis and layout results along with FPGA design strategies are located in Chapter 5. Ensuite, Chapter 6 summarizes and generalizes the approaches and methodologies used throughout the design and synthesis case study, with regard to VHDL controller design onto FPGA technology. Finally, concluding remarks are presented in Chapter 7. Additional ABS definitions, derivations and explanations are given in Appendix A, while sample VHDL code is available in Appendix B describing some interesting encoding styles for datapath/controller implementations. Lastly Appendix C touches briefly on synthesis needs, tools and supporting libraries and their structures as related to FPGAs, which is an important area of study encompassing entire new domains of research.

Chapter 2

Controller Implementation Fundamentals

The ideal synthesis tool will explore the largest design space and build an efficient design conforming to all user constraints. A comprehensive and complete *design space exploration*, or the study of various possible designs from a cost-performance trade-off perspective becomes more difficult at higher levels of design abstraction. Fixed hardware features useful for timely realizations may add proportionate limitations further complicating the synthesis process. *Application-specific* synthesis techniques or methodologies, provide inherent advantages, [48], and researchers have in the past investigated specific classes of circuits such as microprocessors, DSP Systems and control/datapath based methodologies. This thesis focuses on controller systems, yet many of the methodologies presented are applicable or can be extended to other application domains.

2.1 Theory behind the Controller Design and Implementation

Control system design characteristics, entry methods and implementation technology merit some introduction. For synthesis to be acceptable in tightly constrained systems, these three elements must be well understood by the designer and intelligently administered and processed by the tool, as will be seen in Chapters 5 and 6.

2.1.1 System Dynamics Definitions

A *system* is a synergistic combination of components which perform a specific objective. A *component* is a single functioning, atomic, unit of this system. A *dynamic system* is one where the present output depends on the past input, and as a result, the output changes with time [1]. A *real-time system* is one where the present output depends on the past input, and as a result, the output changes quickly enough so as to effect correct responses in the surrounding system, within the specified time. Modeling time and translating events and durations in time to hardware are difficult problems to address in synthesis, yet happen to be essential in the calculations of several real-time controller systems. Hence, having tools that correctly model such behavior on the chosen technology is imperative.

System design begins with a prediction of its performance and behavior. Such prediction is based on a mathematical description of the system's dynamic characteristics, called a *mathematical model*. For physical systems, differential equations are used for these mathematical models. *System dynamics* deals with the mathematical modeling and response analysis of dynamic systems. Linear and non-linear systems are modeled by linear and non-linear differential equations respectively. Due to the mathematical difficulty involved with non-linear systems, they are often linearized about the operating point. Many hydraulic and pneumatic systems exhibit non-linear relationships amongst their variables.

2.1.2 Asynchronous vs Synchronous Design and their Synthesis

Sequential circuits are composed of gates, flip-flops and latches interconnected in some complex configuration. A synchronous circuit employs an independent, periodic clock signal to synchronize its internal changes of state, whereas an asynchronous circuit does not. Set, reset, and interrupt signals occurring sporadically characterize such asynchronous circuits. State changes occur in direct response to signal changes on primary input lines, and different memory elements can change state at different times. This can give rise to critical race conditions which must be avoided. Asynchronous circuits offer substantial benefits in the design of digital control units or sequencers, particularly when many of the actions of the control units are based on externally generated signals that are not guaranteed to be correlated to an available clock signal [41].

The development of asynchronous design techniques for synthesis are beneficial in control systems where input events occur haphazardly, and in interfacing components where handshaking occurs or the individual blocks are not synchronized to a common clock. Asynchronous methods are also exercised for module or system interfacing. Request-acknowledge (negative acknowledge) protocol of the four-cycle or two-cycle type communications requires asynchronous procedures and careful circuit realization. Starting from as early as [42] and [41], where a "one-hot" row assignment and one feedback variable restriction was suggested for circuit synthesis, asynchronous control unit implementations were being analyzed. Simulation of asynchronous systems using HDLs is commonplace, [34], however the research invested in its implementation has not yet reached commercial tool levels. Control applications in complex embedded systems, such as the ABS realization, would benefit from such a tool.

The approach to partition the specification and then employ different synthesis tools for algorithmic and interface parts [44] is feasible, but not the integrated solution sought after. It assumes clock availability, a synchronous design and focuses solely on handshaking, omitting sampling and response time issues which are vital for asynchronous designs. An ideal synthesis tool would permit modularization at a functional rather than at a timing level, so that a language such as VHDL could specify order and time of events with simple sequential and "if ... then ..." statements. However, as advances in asynchronous synthesis currently undergoing research are not yet readily available in commercial synthesis tools, designers are forced to either a manual approach or a synchronous approach[1]. If the timing issues can be solved with the switching speed of the technology, then clocks can be used and an asynchronous design can and is most often transformed into a synchronous design. Alternatively, a synthesis tool and particular designer encoding can result in asynchronous circuits. The Design Compiler^{TM} by Synopsys can process VHDL which results in asynchronous type components, however, their realized behavior is not guaranteed to work as simulated. Certain elements are not allowed, and others are processed without guaranteeing correctness or immunity from hazards or race conditions. Designer discretion is advised and the automatic synthesis may not provide any advantage. Ultimately, more flexibility and

[1] append a *fast-enough* clock to the circuit and synchronize the entire design, inputs, handshaking and actuator control signal generation to the clock

more refined synthesis are expected in newer tool versions, while it is still understood that manual synthesis options must remain.

2.1.3 Control System Commonalities with Respect to Synthesis

Control systems can be divided into datapath and control logic blocks, the latter of which is often realized in the form of finite state machines (FSMs). FSMs prevail, due to the facility of depicting states of operation for tasks such as data processing or response generation and their transitions which effectually sequences the sub-tasks and applies order to them. FSMs comprise sequential elements for current (past) state storage and combinational logic for the determination of the next state. Datapath reflects the flow of data in a system under the management of control units such as sequencers or FSMs. Datapath infers a chain of operations which will be executed on one or several words of data of some pre-defined length. Again, both combinational and sequential logic are required for circuit materialization.

Typical control-type applications involve tasks such as sampling, data-processing, buffering, multiplexing, ordering and decision-making, and response generation. Functionality at the periphery is associated with analog circuits whereas data processing is often associated with digital circuits. Sampling often requires an analog-digital (A/D) interface, just as response generation will implicate digital-analog conversion. Data-processing leads to datapath/register-ALU-mux based hardware while decision making encompasses straightforward boolean type logic and combinational gate level logic. Typical datapath and control circuits are transformed into components such as: adders, subtracters, accumulators, counters, multiplexers, register files, FIFO buffers, three-state buffers, and busses.

Several facets exist within a digital control system; control laws (algorithms), hardware implementation, the conversion between analog and digital signal domains, the system performance, and the sampling process [2]. In continuous control systems, all system variables are continuous signals, regardless of whether the system is linear or not. The generation of a control model to be realized as a digital information processing device is of primary importance [2], and it is why much time was alloted to develop these models and why Chapter 4 is devoted to the specification of the example control system selected and employed.

Previously proposed techniques for synthesis of control systems involving FSM and *dat-*

apath structures [38], [44], [40] assume a similar succession of steps: the compilation of the input behavior into an intermediate graph representation[2] followed by synthesis tasks such as scheduling, unit selection, functional, storage and interconnection binding, and control generation. From an HDL starting point, such as VHDL, many lack specific guidelines in the use of timing constructs for synthesis, such as *after* and *wait* statements, since they have simulation semantics for scheduling future events or for suspending simulation execution. In synthesis these constructs have ambiguous semantics, without specific guidelines.

Control logic activates functional units, component blocks in a schematic and entity blocks in VHDL, in the datapath according to a given schedule. Synthesis of control logic is important because it affects the control flow of operations and hence directly impacts the overall performance of the resulting hardware [40]. Control logic can be implemented as microcode sequences (software solution) or finite state machines (hardware solution). With VHDL this would implicate explicitly encoding FSMs via process statements, as shown in Appendix B, or via wait statement sequences, Section 3.3.

Due to the common nature of control applications, their decomposition and subsequent realization leads to a communal set of logic and circuit elements. Typical datapath components such as adders, ALUs, registers, and interconnection units are used to realize operations such as +,-,*,/,>,<,=,&,—, to retrieve and store their operands, and to connect them via multiplexors. FSMs, which control the multiplexers, registers, bus drivers, ALUs, etc., materialize themselves through registers and combinational logic. Figure 2.1 illustrates simple datapath components with a *bus-mux* interconnection system. Due to the inherent nature of the *four phase control loop* which dominates the ABS controller design, notably complex calculations were required in short times. It was determined that they could be best realized through look up tables (LUTs) implemented in a ROM. Hence a ROM is included in addition to RAM and registers as datapath memory elements.

Efficient VHDL synthesis elicits a proximate coupling of VHDL to these components, which in turn must evoke an adroit *mapping* between them and FPGA inherent structures. Further discussion of VHDL design issues appears in Chapter 3, Section 3.3.

[2] often a dataflow graph (DFG) and a data structure, or control flow graph (CFG) for control flow

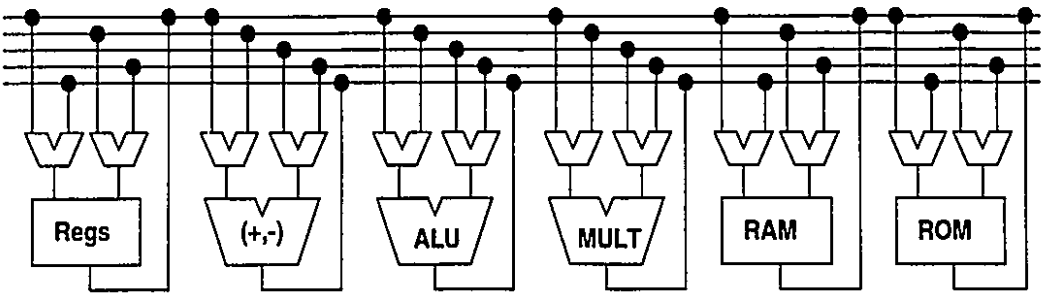

Figure 2.1: Typical datapath components and their interconnections

2.2 Automated and FPGA Specific Synthesis

The realm of synthesis embodies four main elements, the available hardware, a set of design styles or mechanisms, tools and designer. An FPGA architecture base can be multiplexor, gate, LUT, or PLA oriented. Design styles such as pipe-lined, data-path and FSM, random logic, or highly memory dependent, will have their corresponding components implemented with varying levels of efficiency on the uniquely structured FPGAs. Subsequently some FPGA types from different vendors or families better suit certain applications. Software tools link the designer to the hardware and attempt to perform automatic synthesis and map a supposedly generically entered design to the FPGA technology. Yet just as different design types will, and should, be diversely implemented on FPGAs, they also tend to adjust themselves to different design compilers and methods of encoding. Accommodation to the tools and hardware can be equated to *manual synthesis*, and results in customizations, which is often disadvantageous. By and large, there will always remain a certain amount of *designer synthesis* with its level dependent on the available tool suites and on the quality of final implementation specified by the design constraints.

In addition to the above, it is viewed that further classifications reside in the design and tool analysis; the *front end*, the *design constraint specification capabilities*, the *synthesis algorithms and heuristics*, the *layout tools* and their tool specific constraint inclusion capabilities, the *available technology libraries* for simulation and synthesis, and the *testability features*. While we capture the FPGA Design flow in Figure 2.2, a technology specific design flow shall be described in Chapter 3, Section 3.2. Certain simulation phases are optional

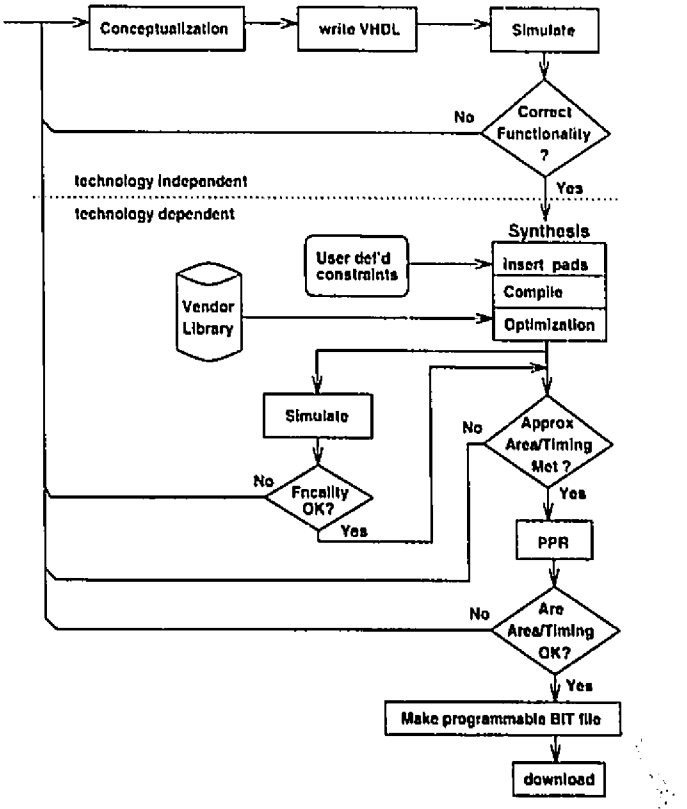

Figure 2.2: FPGA design flow chart for VHDL design and implementation

such as the block which appears after the synthesis step. Decision nodes are present in the design flow indicating that a step was performed and requires a positive confirmation before proceeding to the next step. Without confirmation re-conceptualization must be done. Two such tests are those for validation of correct functionality, and verification for correct timing and acceptable area. The remaining steps involve design partitioning and mapping to logic blocks, actual placement or selection of on chip blocks, and routing, collectively referred to as PPR(Partition, Place and Route). The layout version includes more accurate delay values permitting more precise validations of the implemented circuit. As soon as the layout version is acceptable, it can be transformed into a bit file specifically encoded for the technology chosen, and then downloaded onto the programmable gate array.

2.2.1 Synthesis Fundamentals

A taxonomy for design automation, the Y-chart, introduced by Gajski [38] consists of three different domains: the *behavioral domain*, the *structural domain*, and the *physical domain*. Each domain embodies numerous levels from the transistor level to the system level description which manages items such as CPUs and memory elements. The ultimate goal of design automation is to automate the transformation of a specification given at the highest level of abstraction in the behavioral domain into a description at the lowest level in the physical domain [38]. Software systems which provide this transformation are called *silicon compilers*. Contingent to the level of design entry, a corresponding synthesis procedure will be employed. Our ultimate focus was on *high-level synthesis* where a behavioral description at the algorithmic level would serve as the starting point. In effect, encoding and implementation began with higher level specifications but terminated with the inclusion of *register-transfer level synthesis* where initial design descriptions were in terms of elements such as ALUs, registers, multiplexers, etc. VHDL, the choice for design entry is applicable to all these levels of description.

Synthesis embodies a diverse set of algorithms, many of which are NP hard. The purpose of utilizing high level methods of design entry are manifold: the reduction of design effort and design time, the creation of circuits which are correct the first time, *correctness by construction*, thus eliminating the need for time consuming engineering changes and redesign. Design specification and synthesis tools should be uniform so that VHDL sections which are behaviorally described, structurally written or asynchronously encoded can be compiled as a conglomerate with at most different algorithms being applied automatically to their synthesis. Aside from the type and level of the systems description, constraints on the design behavior weigh heavily in importance during the synthesis process. They serve to guide the synthesis algorithms toward feasible realizations in terms of performance, costs, testability, reliability, and other physical restrictions [38].

Controller VHDL synthesis issues and datapath synthesis algorithms are addressed in [44], [46], [38], [43], [40], and [48], to list a few. Interface behavior specification where synchronization and handshaking, and inter-process timing issues were addressed through "special" VHDL subsets, is discussed in [45] and [44] with a more thorough discussion of

synthesis algorithms found in [46]. Their first step generates a data flow graph (DFG) and a data structure for the control flow for a coherent intermediate representation on which to perform the design transformation to realization. Along similar lines, an algorithm for the allocation of functional units from RT component libraries for the abstract operations within a behavioral description is presented in [43]. The synthesis steps flow from HDL input parsed to a CDFG, to an RT Data Flow Graph, and finally to the RT structures and instantiations. Further mention of a variety of interface representations, and strategies for allocation of library components and scheduling of operations are given comprehensive narrations by Gajski, Michel, and DeMicheli, in [38], [39], and [40] where they cover an extensive range of synthesis algorithms. A common element with these algorithms is that they are well developed in theory, sometimes implemented, but their integration in commercial FPGA synthesis tools from a VHDL entry point, is not readily advanced nor well-documented. To exemplify such a situation, Synopsys, a leading edge synthesis tool, cannot yet manipulate high level behavioral descriptions, cannot guarantee functional timing with behavioral encoded process statements, contains a strict VHDL subset and still requires RTL level descriptions including planned register inferences and explicit clock circuitry encoding. It does however contain an *FSM Compiler*^{TM} to create highly optimized FSM structures and encodings, whereas no such design specific tool exists for datapath/control type circuit implementations.

In an FPGA environment, an automatic design tool, will similarly transcend levels of circuit representations during its synthesis algorithms, the number of which depends on the design entry format. Higher levels of input descriptions abstract the user from low level FPGA component considerations, and produce correct[3], but not so efficient circuit realizations. Low level synthesis, which requires FPGA block structure knowledge, can process register transfer level blocks via algorithms in closer proximity to the hardware, and will produce more efficient FPGA realizations. Regardless of the front end, the final objective is to minimize the number of Logic Blocks (LBs) and I/O Blocks (IOBs) in the final circuit when either area or timing constraints, or both, are under regard.

Descending the FPGA synthesis ladder, ultimately yields a technology mapping endpoint where the design description, which has been reduced to logic level equation format, is mapped to the arrays of equal blocks and special feature elements. Specifically, boolean logic

[3]as per vendor/designer measure and criteria, not as per verification

equations are transformed into programmed *logic blocks*. A number of such LUT technology mappers exist including: Chortle [26], mis-pga [27], [28], Asly, Hydra, Xmap, and VISMAP. All of these map a Boolean network into a circuit of K-input LUTs, attempting to minimize either the total number of LUTs or the number of levels of LUTs in the final circuit. Recent work by Murgai, [29], has included mention of FPGA mapping of sequential circuits. Two approaches are discussed: (i) mapping combinational logic and flip-flops together or (ii) mapping combinational logic and flip-flops separately. Ascending the synthesis ladder towards higher levels of design entry demands additional algorithms and better strategies at each level to produce a favorable FPGA implementation.

2.2.2 Controller Synthesis Techniques

As a precedent, control systems have often been implemented on various computing engine based machines such as microcontrollers/microprocessors, transputers, and larger computing systems. Software was written, compiled and then executed and tested on the hardware. Algorithmic modifications inducing implementation alterations could be performed easily and repeatedly. The lack of speed and the difficulty with ensuring testability and reliability are two disadvantages which accompany this approach. Room for performance enhancements always exists, and correct operation was often assured through multiplicity of microprocessor and its associated RAM. ASIC implementation offers a hardware solution and increased execution speeds but lack in the ease of design alterations, improvements, and upgrades. FPGAs, on the other hand, present a favorable alternative in offering both the speed of hardware and the re-programmability of software. Much has been written about software approaches to controller applications [2], and many software packages exist to simulate controller type applications. Performance can be measured, operation can be tested in-system, but testability and reliability are difficult to build into a single processor. Our aim was to approach controller applications from a hardware viewpoint, analyze the feasibility of design refinement and performance attainment, and introduce a form of concurrent testability employing FPGAs as the hardware solution for circuit materialization.

Rao and Kurdahi [47] proposed a synthesis method which exploits inherent characteristics of a particular class of VLSI systems, to build efficient models or abstractions which simplify the synthesis process. *Modeling* or *abstracting* the problem domain is unlike alternative tech-

niques which concentrate on the development of complicated heuristics for decision making in the various synthesis stages. We suggest analogous modeling at the design entry stage, such that additional effort is spent with design specification and entry, and present a case study to support our proposals and exemplify the extraction of common features and design stages in the design process of a class of applications. We define *class regularity extraction* as the exception of templates to serve as library elements, synthesis and design flow steps, or aggregated design constructs for applications which exhibit particular uniformity and comparableness. This inherent regularity in description and in design methodology should be exploited in FPGA synthesis to furnish better implementations and reduce design time.

2.3 FPGAs: Technology of the Times

An engineering approach in research demands implementation in addition to pure research. One known implementation of and ABS controller was by BOSCH [9] where a microcontroller was used in conjunction with some RAM, supporting a datapath, and this precision of 10 bits of data. In our case, the technology of the times encumbers FPGAs, VHDL and commercially available synthesis and layout tools. Constant evolution of the tools must follow the architectural changes for them to remain useful. One of the intents of this thesis is to recommend some refinements to the tools. It is also accepted that at publishing date, many of the suggestions may already be implemented, newer hardware structures will exist and new tool versions will be released.

FPGAs are changing the world of ASIC design by providing fast turnaround and negligible non-recurring engineering costs. But they suffer from lower logic density and speed compared to Masked Programmable Gate Arrays (MPGAs) because the programmable switches used for logic configuration and routing take up more space and produce higher resistances and capacitances than metal wires.

At present, FPGAs have emerged as the prevailing solution to the traditional time-to-market and risk problems in the electronics industry, because they provide immediate manufacturing and very low-cost prototypes. The final structure can be directly configured by the end user without the need for IC fabrication facilities. FPGAs combine the user programmability of a Programmable Logic Device (PLD) and the scalable and flexible inter-

connection structure of an MPGA. These attributes complement the design environment, by allowing on-site implementation and immediate testing in either a workbench test-jig or the destination embedded system. Synthesis tools can accommodate design changes for exploration of alternatives rather quickly so that altered circuits can be downloaded in minutes, and multiple implementations can be tested without any additional hardware costs.

Similar to ASICs, FPGAs can be used for some higher density circuit needs such as: random logic, glue logic, interface units, embedded controller systems of varying levels of granularity, small to medium sized computing machines, and for decoder or sequencer logic, to list a few.

2.3.1 FPGA Classification and Architectures

An FPGA consists of an array of uncommitted elements that can be interconnected in a general way. Each vendor assumes a proprietary name for their two-dimensional array components, but they all possess similar features: they contain a programmable routing structure, they implement multi-level logic, and they are user-programmable, which characterizes them as FPGAs. Logic circuits are designed and realized onto FPGAs by partitioning the logic into individual *logic blocks* and then interconnecting the blocks through a network of switches and multiplexers. The structure and configuration mechanism of the *logic blocks* and the *interconnection resources* vary greatly amongst the individual modern day vendors of FPGAs.

FPGA Taxonomies

Four major architecture classifications exist for FPGAs. They are: (i) symmetrical array, (ii) row-based, (iii) hierarchical PLD, and (iv) sea-of-gates [21]. FPGAs can additionally be grouped according to their granularity and programmability. Commonly used structural classifications are: (i) minute-grained such as those with transistor-level programmability, (ii)(a) fine-grained: LUT based, (ii)(b) fine-grained: MUX based, (iii) coarse-grained: Logic Block Based (LUT AND MUX) with higher logic densities, and (iv) large-grained: PLD Based FPGAs. FPGA families of the above types can be found from respective commercial

vendors such as: (i) Crosspoint Solutions, (ii) Xilinx INc., Quick Logic, Actel Inc., Concurrent Logic, (iii)Xilinx Inc., and (iv) Altera.

Fine-grained FPGA architectures have a large number of small logic cells. Due to the higher density of these logic cells on one chip, larger routing channels are necessary to handle a massive amount of inter-cell connections. Consequently, a higher percentage of the total area is consumed by routing lines than with a large-grained architecture. Large-grained FPGAs architectures have a smaller number of logic blocks and thus have a less dense and more manageable routing channels per chip. Consider the infamous design tradeoffs: for each advantage created, a disadvantage of some sort will result. The above architectures are not optimal since a fixed large grain structure cannot be fully utilized by the mapping software. The more strict/complicated a design, the less ease for full utilization of the block, but the less need for routing resources and cluttering of interconnection resources.

Floorplans and routing interconnections

Numerous floorplan types exist for the layout of the logic blocks, the I/O blocks, the extra features and the interconnection networks. Some specific features of some major FPGA manufacturers are shown in Figure 2.3. Both horizontal and vertical, as well as hierarchical routing facilities exist, and vary amongst vendors. Logic blocks can contain say simple 2-input NAND gates, some are complex, and contain multiplexer, LUT or PLA type structures, and most contain flip-flop elements to aid in the implementation of sequential design. The diversity amongst a collection of logic block configurations is illustrated in Figure 2.4. Repeated logic blocks of type (a), (c), (e), and (f) fit into the numerous floorplan varieties. Larger blocks of type (b) and (d) contain smaller blocks and are characterized by hierarchical routing networks such as those in figures 2.3(e) and (h). Located near the periphery and connecting to the routing network are I/O blocks. Varying their internal structures and contents, they can be analogously diverse amongst the numerous FPGA vendors. Their uniqueness arises from the individual bidirectional, register, slew rate, boundary scan, and tristate capabilities of the blocks. Additional details can be found in the corresponding FPGA data books.

The routing architecture of an FPGA, encompassing both the structure and content of

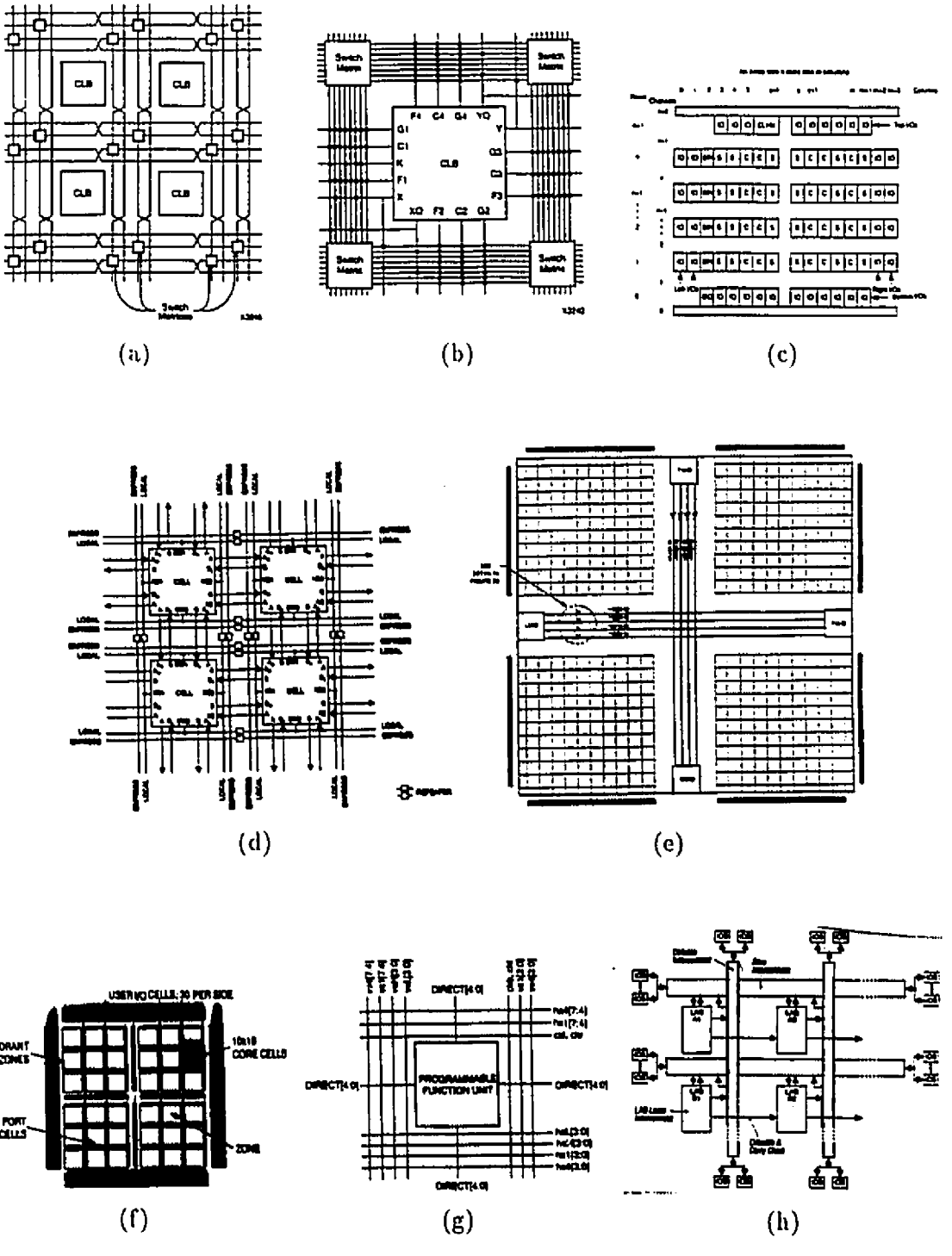

Figure 2.3: Floorplans and Interconnects: (a) **Xilinx's** Double-Length Lines, (b) **Xilinx's** Adjacent Single-Length Lines, (c) **Actel's** Generalized Floor Plan of ACT 3 Device, (d) **Atmel's** Cell-to-Cell and Bus-to-Bus Connections, (e) **AT&T's** Interquad Routing, (f) **Motorola's** MPA1036, (g) **AT&T's** Single PLC View of Inter-PLC R-Nodes and (h) **Altera's** FLEX 8000 Device Interconnect Resources

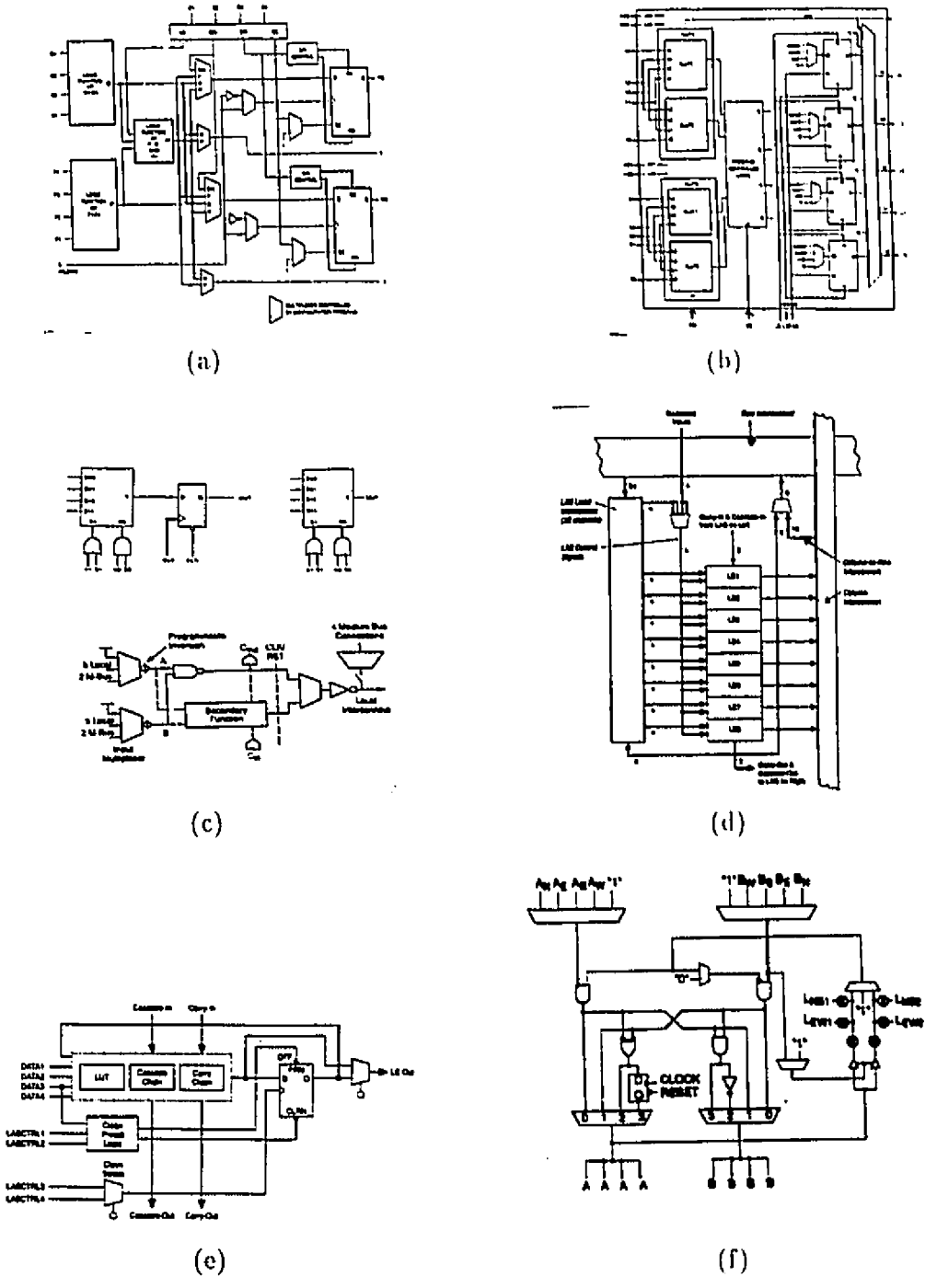

Figure 2.4: FPGA Logic block architectures: (a) **Xilinx'x** XC4000 Configurable Logic Block, (b) **AT&T's** Simplified FPU Diagram, (c) **Actel's** S-Module & C-Module Diagram *(top)*, **Motorola's** Core cell *(bottom)* (d) **Altera's** Logic Array Block (LAB), (e) **Altera's** FLEX 8000 Logic Element (LE) and (f) **Atmel's** Cell Structure

the interconnection, incorporates both the wire segments and the programmable switches. Commercially available programmable switches currently in use fall into the following forms: (i) static RAM cells, (ii) anti-fuses, (iii) EPROM transistors, and (iv) EEPROM transistors. Each shares common property two-state configurability: *on* and *off*. In (i), programmable connections are made using pass-transistors, transmission gates, or multiplexors which are controlled by SRAM cells. These types of FPGAs must be configured on power-up and exhibit a volatile programming technology. As a consequence, some kind of permanent storage mechanism must exist in the system to charge up the RAM cells. Compared to other techniques, the chip area required by the SRAM approach is relatively large. On the other hand, here is an FPGA which can be re-configured in circuit very quickly. In (ii), the anti-fuse normally resides in a high-impedance state but can be *fused* into a low-impedance state when programmed by a high-voltage. Less area is occupied but these are only one-time programmable and require larger current and voltage driving circuits to program them. Both (iii) and (iv) exhibit programmable capabilities with (iv) offering in-circuit re-programmability.

Xilinx Architecture

Xilinx FPGAs, our chosen medium for implementation, also known as Logic Cell Arrays (LCAs), consist of an interior matrix of logic blocks and a surrounding ring of I/O interface blocks. Interconnect resources occupy the channels between the rows and columns of logic blocks and between the logic blocks and the I/O blocks. The core of the LCA device is a matrix of identical configurable logic blocks (CLBs). Each CLB contains programmable combinational logic and storage registers, permitting the implementation of both combinational logic and sequential logic. The combinational section of the block is capable of implementing any Boolean function of its input variables, explainable in terms of its LUT-filled structure. Three function generators, several multiplexors, two flip-flops with clock enable circuitry and asynchronous set/reset capabilities, and carry logic for fast arithmetic operations are all housed in the CLBs as can be seen in Figure 2.4(a). Interconnect network containing neighboring, short, and long (global) routing capabilities are illustrated in Figures 2.3(a) and (b). The functions of the LCA configurable logic blocks, and I/O blocks, and their interconnections are defined by a configuration program stored in a on-chip memory, which must be

loaded on power-up. LCA performance is determined by the logic speed, storage elements, and programmable interconnects. These three features are commonly used to characterize the individual FPGA types.

2.3.2 FPGA Propitious Technology-specific Features

FPGA architectures are tailored for synchronous applications, with their clock distribution networks and multiplicity of registers. In most FPGAs, dedicated global networks distribute low-skew clocks and control signals throughout the array. Many data path applications require multiple clocks, so multiple global networks often are provided to distribute multiple clocks without consuming other routing resources or introducing clock skew. Additionally, with register rich capabilities in FPGAs, 1-hot encoding[4] would seem to, and did in fact produce very efficient FSM realizations for the controller example.

Similarly, global set and reset nets usually are provided to easily force all registers into a known starting state. FPGA logic blocks typically include a flip-flop and some combinational logic. This provides for fast and convenient registering of logic signals. Many FPGAs also include dedicated clock-enable inputs for the flip-flops in the logic cells. These dedicated enable signals permit clock control without consuming valuable logic or routing resources or introducing additional delay into the circuit. Clock polarity controls in these devices allow registers to be independently configured to trigger on the rising or falling edges of the clock, again without consuming other logic resources.

Weighing the above, success in the domain of FPGA synthesis requires clever modeling of the elements. Two domains of hurdles classified in [47] were applied to FPGAs materializing to: (i) the incorporation of realistic physical design details into the synthesis process most particularly routing delays and LB structure, and (ii) the ability to conquer the complexity of the synthesis problem itself, often resulting in heuristic techniques producing suboptimal solutions or exhaustive search techniques which are expensive and time consuming.

[4] One register or flip-flop is dedicated to a single state of a finite state machine. In terms of encoding, this method is the least register efficient, however the logic determining state transitions is often simplified.

2.3.3 Delay Modeling and Estimation

A *timing model*, presents a methodology to calculate *propagation delays* from inputs to outputs, of single logic blocks, aggregates of blocks, and clusters of aggregates in the respective FPGAs. Exact or near exact, timing values cannot be generated without post-layout back-annotation. However, conservative estimates can be calculated given the regular structure and configurability of the FPGAs. With modular library support systems, vendor-specific timing models can be included. Be it synthesis tool or FPGA vendor who designs them, concerted efforts are vital as is highlighted in Chapter 5, where one caters to algorithms and database management and specification while the latter knows the hardware. Combining the two is useful to accurately model the time delays, connectivity, and density of all the building blocks in order to calculated the final, overall time delays resulting from and due to the programmable nature of the FPGA. Development of these models is a research topic of its own.

FPGAs operate at circuit speeds where a complete operation, including data set-up and computation times, can be executed in one clock cycle, unlike with microprocessors where an operation traditionally takes several clock cycles of assembler generated code. Hence, FPGAs compute much faster than pure software functions but wiring and logic delays make them slower than equivalent masked gate arrays, or ASICs. However, timing is quite predictable due to the fixed hardware so that a conservative estimates can be made, simulated and ultimately reflected in the resulting circuitry. Several models exist for measuring the delay of a circuit, each of which is heavily dependent on blocks affiliated with the FPGA architecture. Singh et all, [32], suggest the speed of a circuit implemented in an FPGA with given logic block structures is a function of the combinational delay of the Logic block, the number of logic blocks in the critical path, and the delay incurred in the routing between logic blocks. A reasonable assumption where each stage of block incurs one logic block delay and one routing delay, [21], results in a total delay of $D_{TOT} = N_L \times (D_{LB} + D_R)$. The number of levels of logic blocks N_L can be determined for each circuit after the technology mapping step, the logic block delays (D_{LB}) are readily available since the chip is already fabricated, but the routing delays (D_R) are more difficult to estimate. It depends on routing architecture, fanout (capacitive loads), length of connections which are determined by the placement and routing

of the circuit, the process technology, and the programming technology. The Xilinx FPGA is one example befitting such timing predictions and analyses.

Alternatively, [30] and [16] offer timing models based on stricter routing structures, the former on the row-based architecture of Actel, and the latter on the hierarchical PLA format of Altera. It is claimed in [16] that device-wide routing provides predictable performance even in complex designs, while segmented routing requires switch matrices to connect a variable number of routing paths, increasing the delays between logic resources and reducing performance. Longer routing delays with respective RC time constants versus switch delays is one trade-off to consider. Other models, [18], [33], and [21] focus on the diverse logic block architecture types and sizes. Each method of delay measurement seems to model the logic blocks and interconnect capabilities very well and leads to a high level of confidence in the resulting timing analyses from simulations and synthesis, yet is not a determining factor in the selection of the FPGA type for implementation.

2.3.4 Technology Considerations

Space limitations and performance demands require integrated circuit technology such as ASIC and FPGA. Due to its many advantages, FPGAs are excellent for rapid-prototyping and serve as a logical starting point. Hence the goal of the case study materialized to initially realizing controllers with asynchronous features onto programmable hardware while reserving the option to migrate to the denser ASICs. Real-time constraints would still have to be satisfied while maintaining the above flexibility. Depending on the technology and the manufacturing process, the gate logic and routing delays will vary across ASIC-ASIC boundaries, and FPGA-FPGA boundaries as seen above. Furthermore, circuit timings across different implementation technologies, FPGA-ASIC, will differ even more so.

Although circuit timing analyzed with FPGA simulators and synthesis tools will most often predict acceptable results to proceed with their programming, the corresponding timing and responses with ASIC technology cannot be guaranteed. Functional validation is equitable but timing analyses are not comparable. While certain elements are predictable in FPGAs, some such as parasitic capacitances, resistances, interconnect delays and loading in ASICs are not. Worst case delays can be given for FPGAs, however the same is not necessarily true with

ASICS. Hence if ASICs are the final implementation, circuit timing from FPGA simulations are not accurate enough and re-simulation of the entire circuit is needed. In fact, post layout characteristic values will be needed for the back annotation and re-simulation. Even though the circuit will most likely be faster, the relative timing is no longer similar and functionality can fail. Nonetheless, an alternative exists but it bears drawbacks. Predictability can be achieved to a certain degree if the circuit is completely synchronous. Such circuits are more easily transferable, process and technology-wise, than asynchronous circuits. Given the nature of the FPGA architecture, much of the asynchronous elements will have to be synchronized to ensure correct functionality and predictable behavior should ASIC migration be a potential route in the implementation phase.

2.4 VHDL for Design Entry

For high-density FPGA designs, gate-level entry tools often are cumbersome, and the use of logic, RTL, or high-level synthesis and high level description languages (HDLs), such as VHDL or Verilog, can raise designer productivity [54]. Simulation of HDL descriptions can test alternative decisions early in the design cycle and can hence refine specification. Additionally, with design changes facilitated, more extensive experimentation and trade-off exploration in the architectural design are permitted. Elements which deserve attention while reviewing VHDL synthesis issues are:

- The synthesizable subset
- Behavioral versus structural encoding advantages
- Time to market versus design specifications
- Area versus timing tradeoffs
- How to design with VHDL for controller applications, (hints for good syntax dependent synthesis)
- Limitations/capabilities of VHDL language, synthesis and layout tools.

VHDL was chosen for design entry due to its ability to accept hierarchical design input specifications, and capture designs at various levels of abstraction permitting both behavioral and structural specifications. These can be expressed through *sequential* and *concurrent*

statements, respectively. Encoding can be algorithmic (behavioral) or it can be structural (register level and schematic like, with component instantiations). In addition, VHDL can adequately model concurrently executing processes, a necessary feature for reliability checks and for hardware descriptions where fine- and coarse-grain parallelism exist and must be realized expressly.

Processes were employed wherever possible as they are a natural starting point of high-level synthesis. They are described with sequential statements and impress higher levels of design entry. *Blocks* contain concurrent statements and impress lower structured levels of design entry. As with any textual design description, certain encodings produce more optimal implementations. Well written VHDL code for simulation, may not be so useful for synthesis. The pros and cons of low and high level VHDL, which effectively translates to the calling of low and high level synthesis procedures, respectively, must be weighed. Synthesis of structural descriptions is less ambiguous, more predictable and hence less time consuming, algorithmically speaking. Behavioral descriptions are less predictable, not as efficient, but portable across technologies, often more readable, and easier for the designer to encode.

2.5 Evaluation of Synthesis Quality

Benchmarking, quantifying the area utilization and runtime of blocks, aggregate blocks and systems through simulation or execution, circuits for performance measures is a classical way to compare hardware. Design methodologies are not so easy to evaluate and their utilization on different technologies is further difficult to gauge and equate. One of the first analysis of design methodologies known to date, appeared in 1983 under Obrebska [42]. Several methodologies were presented and compared in the design of control parts of microprocessors where layout quality and facilitation of the designers' tasks ranked in importance. The term "synthesis" scarcely existed yet the underlying requirements, goals and theories were present. Design methodologies were weighted on efficiency in terms of hardware cost, which was defined as a function of the total area, design time, and performance. Design time was estimated based on the percentage of structures which could be generated automatically like ROMs or PLAs, and was labeled *percentage regularity factor*. Today synthesis tools must

handle many more structures and work at regularity factors of close to 100% since otherwise the design's implementation would not be automatically generated.

Another difficulty for achieving equitable comparisons arises from the diverse nature of FPGA technology. Currently it is viewed that, to effectively compare technologies is to synthesize the same top level design specification, resorting to generic modeling if necessary, with the different synthesis tools. Though the FPGA architectures and synthesis algorithms will be significantly diverse, no customizations should be allowed to better the results. However, the ideal and desired approach of finding a common VHDL subset may prove impossible and its inclusions may be somewhat biased. Technology specific features will only be utilized if the vendor specific synthesis algorithms can realize them from the VHDL starting point. A disadvantage from a tool developer's viewpoint[5], but an appreciable advantage from a designer's perspective arise. In fact, a current set of FPGA qualifiers, the PREP^{TM} benchmarks attempt to measure the utilization and implementation of a list of specified circuits. However, the circuits were realized by the vendors themselves exploiting their features to the fullest not necessarily using their synthesis algorithms and VHDL encoding, precluding the original intent of unbiased implementation comparisons, leaving their worth somewhat prejudiced and questionable. If, on the other hand, a generic VHDL description was used and synthesized on the individual architectures, comparisons of the realizations would be informative. However, this is once more challenged, as some of the customized features will not be exploited due to VHDL encoding limitations or vendor tool status.

2.6 Testability Measures

Off the shelf, programmable FPGAs preclude designing tests for manufacturing faults, in their design and field environments[6], fundamental in ASICs and full custom chips, yet admit the need for tests for field reliability. A subset of criteria for evaluation of the quality of testability includes some of the following: the area overhead and the performance degradation

[5]They will be forced to develop tools exploiting their proprietary features while catering to some generic platform which will likely be ill-specified in their eyes eliminating the ease of adding "patches" which are common for customizing software

[6]FPGA die manufacturing, logic and interconnects are tested by the manufacturing when the wafers are created

incurred by extra circuitry, the fraction of faults/functionality tested/covered by the circuitry, and the design time for its implementation. Additionally, the time required to apply the tests, the applicability to a general class of systems, and the technological independence all contribute to the merits of a testable design. Section 3.4 furnishes additional details.

Chapter 3

Controller and FPGA Design Considerations

The constantly evolving design characteristics and system variables, and the need for in-system (embedded) testing provides justification for the design of a controller application onto FPGA technology. The availability of fast prototyping and programmable capabilities of FPGA technology offer a favorable dyad. VHDL design specifications as opposed to schematic entry implicate higher level synthesis techniques where the implementation feasibility can be determined. This study is likely to open up a high level of automated design methodology for safety critical applications with real-time constraints. In this chapter design considerations, in context of VHDL for design entry and FPGA technology for implementation, are elaborated upon. The generic design flow diagram of Chapter 2 has been instantiated to a Xilinx FPGA design flow.

3.1 Requirements for Optimal Controller Design

FPGA realization of controller type applications is driven by several objectives. Many are qualified as engineering issues and must therefore be solved or concluded upon after the research and experimentation. A wish-list follows:

- Rapid prototyping feasibility - easily alterable designs and embedded testing.

- Real-time validations - can the hardware respond to input events, explicitly, to the fastest rate of change of the input values, which itself is highly system and application specific.
- Parameter programmable - easily adaptable to parametric alterations and changing system variables.
- Field upgradeable - can the implementation be altered in its embedded system without having to install a new component or revert to a new manufacturing product.
- Error correcting, fault tolerant.
- Self-testing.
- Compact, time efficient.
- Error detection, system shutdown capability
- Safety - both from reliability and human safety viewpoints.

In addition to the above set of goals, their evaluation criteria deserve attention. Several merits of CAD tool usage for design and implementation on FPGA technology can be measured by their ability to excel in some, or all, of the following evaluation criteria:

- Exploitation of technology (the different FPGA architectures, logic blocks)
- Rapid prototyping facilities (imperative for *implementation fine-tuning*)
- Area versus timing tradeoffs
- Designer synthesis versus tool synthesis tradeoffs
- Tool capabilities from a real-time implementation perspective
  - input sample rates: sufficiently high
  - calculations: sufficiently fast
  - output results: adequate response times dependent on the application.
  - safety criticality of the controller system.

3.2 FPGA Design Flow

Precluding final implementation and focusing on a prototype implementation where the technology is known, the generic design flow as depicted in Figure 2.2 can be customized

to the Xilinx design flow as in Figure 3.1. Commencing with a VHDL description, the synthesis platform followed initially produced a list of generic FPGA CLBs and IOBs, taking some design constraints into consideration. These were subsequently translated to logically equivalent primitive combinational, sequential, and tristate gates from Synopsys' defined version of a Xilinx FPGA technology specific library. The resulting netlist, in ".xnf" format[1] was then used by FPGA vendor's (Xilinx') Placement, Partition and Route (PPR) Tools for an XC4010 implementation. Consequently, it was determined that the Synopsys-XACT layout was not a direct technology driven path, as the ".xnf" files were created. Hence, FPGA CLB clustering and implementation before the *replace_fpga* command, will not necessarily be preserved. A transformation to logic gates after FPGA synthesis, where CLB and IOB programming were previously created, was required, distancing the router from clustering and input design constraints. Fortunately though, some hard macros[2] were and can be passed in the ".xnf file" format.

3.3 VHDL Constructs

With an appetite to substantiate higher levels of design entry, the VHDL *process* and its corresponding sequential statements were used wherever possible in the description of the controller. Similar research by Gutberlet and Rosenstiel, [44], involving high level synthesis and algorithmic specifications also advocates process statement utilization. The resulting target system architecture is then composed of interacting hardware processes in which communication is achieved using signals connecting the control units and the data paths. For purely synchronous designs, this communication can be embedded within the process description where the user can explicitly describe the communication using standard VHDL constructs which force events (read, write, calculate, set, reset) on correct clock cycles. When communicating processes are not synchronous, a specific set of protocols to achieve synchronization of the interacting designs is needed. There are two basic methods for achieving this synchronization: shared medium and message passing [38]. In VHDL the former becomes interconnections between processes via straight port descriptions. In the latter, primitives

[1] this **.xnf** netlist file does not support CLBs and IOBs but rather includes a small subset of primitive gates, and some hard macros, ADSU16 and ADSU8.

[2] Vendor specific blocks with specific functionality and predictable CLB utilization and some routing .

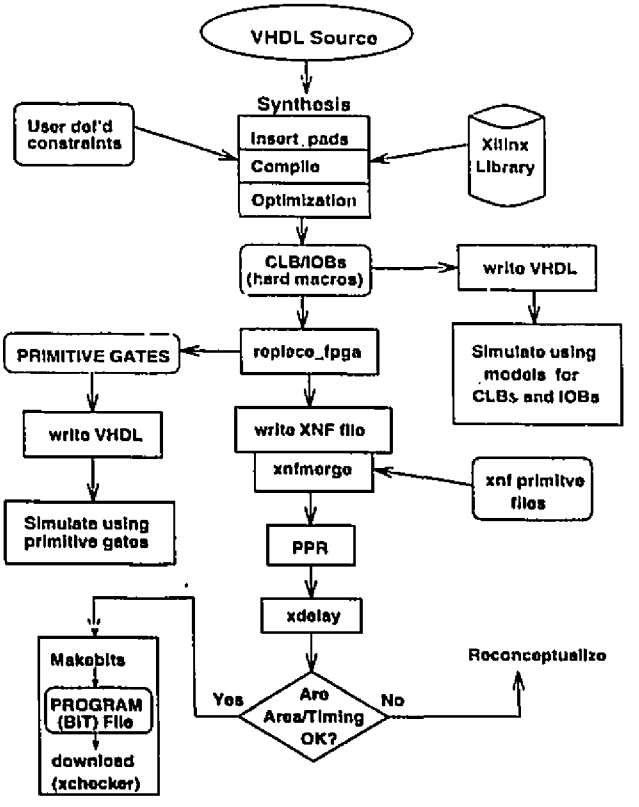

Figure 3.1: Design flow for VHDL design and implementation using Xilinx FPGAs

are used to describe and control the synchronization between communicating processes.

A subset of preferable and reusable VHDL encoding styles for inter-process communication are included below. Their syntactic form is beneficial for FPGA types which exhibit built-in clock enable circuitry. A typical VHDL control for execution of a functional block follows:

```
process:
  ...
  wait until (enable='1');
  initialize some signals;
  initialize some variables;
  may or may not want to synchronize to a signal;
  do work, calculations;
  (re)assign signals;
  (re)assign variables;
end process;
```

A typical situation where one process exhibits control over another process is demon-

strated by the VHDL code below. One process can be master over an other, using wait statements.

```
process;
  ...
  initialize some signals; initialize some variables;
  do work;
  set slave_cntrl_job_enable;
  wait slave_cntrl_job_complete;
  do work;
  (re)assign signals; (re)assign variables;
end process;
```

A typical handshaking example between processes, each of which drives the intercommunication signals, is outlined for a general case. Customization is straightforward and merely follows suit.

```
process;
  ...
  set resource_request;
  wait for resource_ack;
  use resource;
  reset resource_requests;   -- release resource
  wait for resource_ack_off;
  (or wait for a clock event to synchronize this process;)
  ...
end process;
```

Care must be taken when employing the above VHDL construct so as to avoid an accidental and unwanted *deadlock* situation, where two processes reach a point in time where they must wait for each other. Additionally, aside from handshaking, the *WAIT* statement can aid in the creation of a control flow for the design. Averages can be calculated over many clock cycles easily with the *WAIT* statement. Additionally, time or event durations can be synthesized with multiple, consecutive uses of a wait statement.

3.4 The Testable Design

One purpose of integrated circuit testing is to reject those parts that do not function due to imperfections in the manufacturing process. Since SRAM FPGA programming is reversible[3],

[3]repetitive options to load both 0 and 1 into each memory cell permitting circuit modifications

the FPGA can and should be tested at the factory to ensure that it is correct regardless of the programming pattern. Therefore, the part does not require testing part (in the usual IC terminology) for each design implemented on it — the factory testing guarantees that the part will work regardless of the programming. This property still holds regardless of the size of the FPGA. In fact, most Xilinx, Altera, Atmel, Actel, etc. claim 100% fault coverage.

Another aspect of testing deals with reliability, security, and the circuit's reaction to error detection. It is almost impossible not to have faults somewhere in a system, be it in the controller itself or in the surrounding elements, at any given time. A fundamental problem in estimating reliability is whether a system will function in a prescribed manner in a given environment for a given period of time [5]. There are many solutions to fault-tolerant and detection circuits. One solution would be to duplicate the circuit, add in comparative circuitry and busing for the interconnections and arrive at a self-checking circuit with more than 110% overhead. An alternative proposed in this thesis, is to implement a partial solution employing a self-testing value injection circuit described in Chapter 4, Section 4.1.4.

A remaining area for testing arises when design complexities intensify beyond single components to boards, sets of boards and systems. Interconnects, routing, timing, drive capabilities, and short circuits can lead to faulty operation even if the ICs themselves are fault free. In addition to the testability concerns for hierarchical system designs such as these, particular time-sequential testability concerns surface with PGAs. Consider the nature of programmable devices wher it comes to board tests. Their logic is changeable and preparation of tests for such components must be delayed for the duration of these changes. Confusion and diagnostic difficulties caused by board faults can arise from the time the board is powered up, to the time the programmable chip has settled.

A partial solution with a Joint Test Action Group (JTAG) Boundary-Scan Test (BST) architecture[4] can be applied to one case, but providing testability over time intervals is not straightforward. When it comes to testing these field programmable elements, two device types exist: (i) those that have no pre-defined logic and (ii) those that have a small amount of logic built in, and certain I/O pins assigned, and three methodologies exist: (i) inclusion

[4] Which allows the isolation of a device's internal circuitry from its I/O circuitry with the intent that JTAG boundary-scan testing can offer the capability to efficiently test components on circuit boards with tight lead spacing.

of JTAG BS features, (ii) no inclusion and (iii) power-up inclusion with no-inclusion during operation. There are advantages/disadvantages in each type. Those that have no pre-defined logic, can always be programmed to include boundary scan logic, but then, not all logic cells will be free to implement the intended design for the IC. Otherwise the FPGA can be programmed twice with JTAG features loaded first and the IC design implementation loaded during operation offering no subsequent testability options unless power down. However, added circuitry, control and memory to support two FPGA programming is required and the secondary programmation of the chip is not tested. On the other hand, a hard-coded, boundary scan logic in a small area of the chip, and in the I/O blocks, can reduce the area taken for BS and thus offers more logic blocks for the intended circuit. But some FPGA area is removed resulting in fewer overall gates per die.

Some FPGAs offer built in hardware [20], [16], others don't but provide application notes on JTAG implementation indicating that the cost of the added circuitry is not justifiable as their resources become depleted and the performance degradation is not worth the JTAG implementation. Xilinx's XC4005, contains a hard-coded boundary-scan facility. It contains a hard-coded 1149.1 shell comprising of a TAP, an "Instruction Register" and a "Bypass Register" superimposed on the IC infrastructure. What remains to be added is the "Boundary Register" itself, which is made part of the I/O blocks. The idea is to have a resulting configuration which is that of a 1149.1 compliant component. In addition to Xilinx, both Altera and Crosspoint carry support in their latest programmable devices, for the JTAG boundary test scan standard.

3.4.1 Fault-Detecting and Fault-Tolerant Designs

There exist many techniques to construct fault-tolerant and fault-detecting systems, a few are listed below:

- *triple modular redundancy* (TMR) with three of the same modules where the result can be an average of all three or it can be the majority value (2 out of 3).
- *N-module redundancy* (NMR), a generalization of the TRM for N-modules.
- *duplex system*, where a single back up unit exists and their results are compared.
- *biduplex system*, a double duplex system where two sets of two operation results are compared and four modules are required.

Of those, individual customizations are possible. A mismatch at an output stage can trigger an error flag which may shut the entire system down or just the faulty module. In the case of only two modules, the faulty unit must first be determined. A back-up system may or may not then resume system operation. Two main types of faults exist: permanent and intermittent. Intermittent faults can be corrected with a TMR system where the system response assumes majority result. Permanent faults must shut the system down or resort to a backup unit.

Vital to controller systems is inclusion of such circuitry which is *failure tolerant*. An error of any kind must be detected, and the system must respond accordingly so that further malfunctions do not occur producing hazardous consequences. Faults (which at their lowest level can be modeled as *stuck-at-0* and *stuck-at-1*) must either be compensated or effect system shut-down. As with all testable systems, this also applies to the logic which performs the system functionality. In the ABS, a *duplex system* is employed without the added overhead of module duplicity. The approach taken was to exploit integrated circuit speed of operation and perform comparisons at different time intervals in lieu of simultaneous output matching. Hardware time-sharing and added control and storage logic to test the functionality with previously stored values was incorporated into the specification for the ABS design. Fault detection became available, however its tolerance would be limited to a certain error margin resulting from the calculations themselves. As a result, if the error was large enough, the system would shut itself down and request maintenance. The option of a back up unit does exist, yet was not implemented due to the original specification of FPGA resource limitations (a restriction as limited by this case study and not by the ABS embedded environment, information of which was not known).

3.4.2 Test Circuitry Evaluation Criteria

Just as the design, the methodologies, and the performance undergo judgment, the merits of test circuitry can be evaluated by consideration of the following criteria. This list enumerates a possible set of parameters which can be used to evaluate the overall quality of a given built-in self-testing, self-diagnosing circuit design.

- Area overhead incurred by the extra circuitry.
- Effect of extra circuitry on performance of the original circuit.

- time required to apply tests and perform the diagnosis.
- Engineer's design time required to implement self-test overhead.
- The fraction of faults located by built-in self-diagnosis
- The *self-testability* of extra circuitry which performs the self diagnosis
- The flexibility of design with respect to the following issues:
  - Independence of the fabrication technology used
  - Ability to accommodate to new and different fault types as the technology evolves and
  - Applicability with minor changes to different types of FPGAs such as those which are PLA, naud, or mux based.
- Length of the testing sequence and the fault coverage.

The test sequence length determines the time required to test the actual device. Fault coverage is defined as the percentage of all faults detected given that the faults could exist, by the vectors in the test sequence.

3.5 Design Procedures

Knowledge of the design process was acquired by selecting a specific controller example and progressing through the steps from design specification to implementation. The nature of an engineering control-design problem can be split into seven stages according to Houpis and Lamont [2]. The steps originally referred to a software solution but can be applied for hardware solutions. Four have been assimilated here for their direct applicability while the third element in the list was appended for its benefit towards hardware implementation. The last two ascribe particularly to the work which produced several of the design methodologies covered in subsequent chapters.

1. Establish a set of performance specifications relating system input to output based upon given criteria (tracking response, sensitivity to system variations/uncertainties, etc.).
2. Generate a linear model that describes the basic or original physical system.
3. Exploit available Computer Aided System Technology (CAST), to help formulate the implementation specifications.
4. Simulate the overall control system, *iterating* the design until performance specifications are achieved.

5. Emulate, test, and *iterate* the implemented design. Technical knowledge of contemporary digital hardware and software architectures has critical influence on cost and real-time operation.

FPGA technology is very well adapted to step 5, and our control problem can be summarized to that of designing the system to obtain the desired performance. The *synthesis approach* embodies an iterative design procedure. Initial attempts permit the tools from Synopsis, to perform the synthesis for hardware realization and conform within their own limitations to the original specifications. Subsequent iterations require the design engineer to become more involved using known or learned synthesis strategies. Ultimately the definition of levels of design specification in VHDL which can be employed for controller system type applications, while conforming to and satisfying the design constraints, is the goal. Specifically, what is, or what can be defined, as a synthesizable set of VHDL for an externally asynchronous, internally synchronous controller design. At a low level this translates to efficient specification, design partition, synthesis, and implementation of controller constituents such as FSM, data path or random (control) logic onto FPGA technology.

Chapter 4

The ABS Control Model

Our evolutionary understanding of control synthesis arose from the analysis of two experimental applications. A telephone answering machine controller and an ABS controller were specified, designed and simulated in VHDL. In this chapter we describe the more complex and more complete of the two, the ABS controller, which was eventually synthesized and routed onto a Xilinx FPGA. The circuit's specification follows while details of its' realization, and the formulation of application specific design and synthesis methodologies follow in Chapters 5 and 6, respectively.

The **experimental aspect** of this thesis thus focuses on developing a schema for an ABS Controller application which continually self tests its operation, implementing it, verifying and validating its operation. And, in the process, fashion FPGA design methodologies and re-usable, well-constructed VHDL components specific to FPGAs and real-time controller applications written to conform efficiently to both area and timing constraints. These are to then serve as a means for designing larger modules and systems and be used by higher level synthesis and simulation tools, and will become part of the ASDSL for FPGAs.

4.1 ABS Dynamics

4.1.1 Motivation for ABS Design and Implementation

Originally developed for airplanes, where the high cost of development was justifiable from both monetary and safety fosterings, ABSs have transgressed to the automotive industry

and today are commonplace in several vehicle models. An ABS is used to prevent wheel lock-up during an emergency braking maneuver, by controlling the solenoid valves which modulate brake pressure, and thus provide directional stability and steerability of vehicles, while reducing the stopping distance. The motivation for automatic ABSs arose from the need to reduce speed during a "panic" brake on slippery surfaces, requiring the driver's removal of continual application of the brake, or pumping. It is claimed that an electronic system/controller would be more reliable and responsive than human reactions and would therefore contribute to the safety of vehicle operation.

4.1.2 The ABS Functional Model

Much research exists in the modeling of ABS and in its corresponding control theory [9], [14], [10], [11], as well as in related anti-slip regulatory (ASR) systems for acceleration control, [12], and [13]. The underlying fundamentals of operation of an ABS system involve the detection of optimum braking conditions and the application of corresponding brake pressures to maintain the optimum conditions. Details of these two elements along with notational definitions, terminology, and mathematical equations and derivations can be found in Appendix A.

In the single wheel vehicle model, the equations: $M\dot{V} = -F_x$, $I\dot{\omega} = -(T_b - rF_x)$, $S = 1 - \frac{V_t}{V}$, where $V_t = r\omega$, describe the fundamental vehicle dynamics. S is defined as the slippage where **slip** is the measure of the sliding component during a rolling movement. F_x represents the frictional force between tire and road, I and r the moment of inertia and radius of the wheel, and T_b is the wheel torque resulting from braking. V_t is the tangential wheel velocity, and V is the actual vehicle velocity. Further definitions of the complete set of variables can be found in Appendix A. The ABS along with the manual braking of the driver interact with the solenoid valves whose hydraulic pressures are being controlled inducing varying amounts of wheel braking. Many hydraulic and pneumatic systems involve nonlinear relationships among their variables. In our case, an ABS controller is highly dependent on a tire-road model, the characteristics of which depend on weather conditions, vehicle parameters and dynamics, and hydraulic system parameters. In an ABS, tire force varies non-linearly with slip S. As in [6], [7], and [8], a piecewise linear approximation of the tire-road characteristics will be used.

ABS controller requirements embody certain asynchronous design techniques due to unpredictable input arrivals and desired immediate actuator response. A potential demand for multiple clocks or inter-clock activities to support the different timings required for input sampling, calculations, system response and on-line continuous testing, add to the complexity. A deviation from single clocked synchronous controller designs surfaces, yet a conceptualization can be formed by limiting the system to one main clock, a "fast" clock with several offspring clocks created by on chip clock dividers requiring inter-clock synchronizations techniques using this one main "fast"clock. Encoding many of these characteristics in a VHDL model is a major part of this thesis.

The ABS design implemented in this thesis was formulated from a culmination of algorithms, theories, and known physical automotive dynamics, as mentioned above. Complexities arise due to the non-linear dynamics, and the large number and diverse nature of input variables. ABS dynamics behave differently under the varying road conditions, vehicle speeds, nonlinear tire models, and different values of pressure increasing and decreasing rates due to inherent differences in a hydraulic system itself. Appendix A briefly outlines the overall functionality and evolution of the ABS controller while methodically developing the models used for implementation of the central *pressure control loop*, which dominates in the realization of the ABS case study . Measuring continuous vehicle variables, performing calculations, comparing results, evaluating time expirations and responding accordingly with directives to the solenoid valves (pressure actuators) comprise the pressure control loop. One path in the development of its theory, hinges on the evolution from a two-phase controller to a four-phase controller. Proven in theory to be better, facility of its implementation remained to be evaluated.

Two phase control loops employed *pressure increasing* and *pressure decreasing* stages to control braking by circulating around the optimum braking force, or tire-road friction coefficient [10], [6]. Kuo and Yeh [8], suggested four-phase control to improve the ABS performance by adding a *high-pressure* and a *low-pressure holding* mode. Previously, no analytical studies existed describing the fundamental design principle for ABS with four-phase control, until [8] where systems with this four-phase control were investigated and the threshold values for the ABS control laws were determined. Figure 4.1 depicts the pressure control loop during ABS operation. The *ab* arc indicates the pressure increasing phase, the

bc segment indicates the high pressure holding phase, and so on. Approximations to this theory with some extensions form the complete control loop, further described in this chapter and in Appendix A.

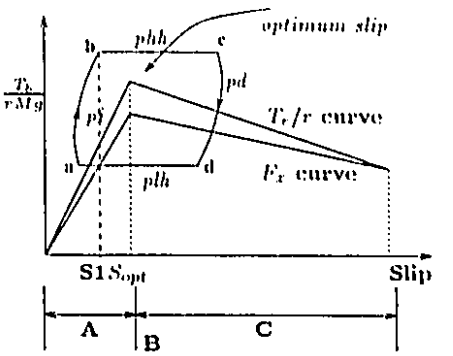

Figure 4.1: Piecewise linear approximation of the tire-road characteristic and the desired path to follow under the ABS Control Loop.

FPGA implementation of the above control laws was investigated along with supplementary modules forming much of the aggregate ABS controller, demanding real-time operations. Having met a satisfactory number of design constraints, current and future work concentrates on the expansion of the single wheel model of past researchers [6], [7], [8] to a four wheel system where each wheel assumes the appropriate distribution of the mass of the vehicle and carries an additional driven/non-driven attribute, and individual braking force percentages.

4.1.3 ABS Design Specifications and Structure

Global constraints for the design were set considering the circuit's eventual environment. Minimal area and system response times faster than human response times (in the sub-*ms* range) were sought. Real-time relative to sampling, calculations and responses as defined in Section 3.1 is anticipated. Additionally, an on-line self-testing/monitoring system capable of error detection for system shut-down falls into the design specification. Lastly, it was desired to allocate minimal time in the design phase for the implementation of the above system onto programmable hardware using an HDL.

A block diagram of the proposed aggregate system appears in Figure 4.2, where all blocks

and sub-blocks are controlled by Finite State Machines (FSMs). Complementary blocks such as the *Wheel velocity generator*, *wheel velocity calculator*, *wheel acceleration calculator*, and *slip calculator*, and FSMs such as *master FSM*, and *pressure control FSM*, supplement the actuator controller labeled as the *ABS pressure regulating 4-phase loop*. This block embodies the four phase ABS controller of [8], and consists of four sub-blocks each activated during one phase of the pressure regulating cycle, and controlled by the *pressure control FSM*, subsequently referred to as the *pres cntrl FSM*.

Off-chip elements such as a ROM, which holds pre-calculated data, is also part of the system but not part of the implementation on FPGAs. FPGA resources must be used cautiously so only calculations requiring the sampled incoming data have explicit hardware resources. A large Look-up Table (LUT) of values cannot be realized in an FPGA, but as it is vital to the design, it was included in the system specification, is referred to as the ABS-ROM and was used in all simulations. Not included is the added circuitry for the on-line system monitoring. The complexity and size of the design necessitates design partitioning, one of which is proposed by the designer and indicated by a dotted line in the ABS block diagram of Figure 4.2. Upon actual circuit realization through *tool* and *manual synthesis*[1], an initial partition is most likely to change, as it did with the ABS controller (Chapter 5).

Master Controller: Top Level FSM

Controller applications naturally descend into FSM cycle control units and data path circuit implementations. The top level FSM, referred to as the *master FSM*, determines the overall state of the vehicle, in the ABS controller. There exist five major states of vehicle operation: (i) Stopped vehicle position; (ii) Rest to sliding action; (iii) Normal vehicle movement; (iv) Regular braking of vehicle; (v) ABS controlled state. Appendix B lists the VHDL code for the *Master_FSM* implementation which demonstrates the conditions for vehicle state transitions and FSM encoding. The calculation and storage of values such as velocity and acceleration will differ, depending on the state of the vehicle. Angular wheel dynamics can be obtained from wheel rotational measurement instruments and are central to most calculations.

A *frequency generator* generates the wheel rotational frequency (velocity) from values

[1]This refers to designer direction and explicit hardware instantiations or partitions through compiler directives or definitive and unambiguous encoding during an iterative synthesis process.

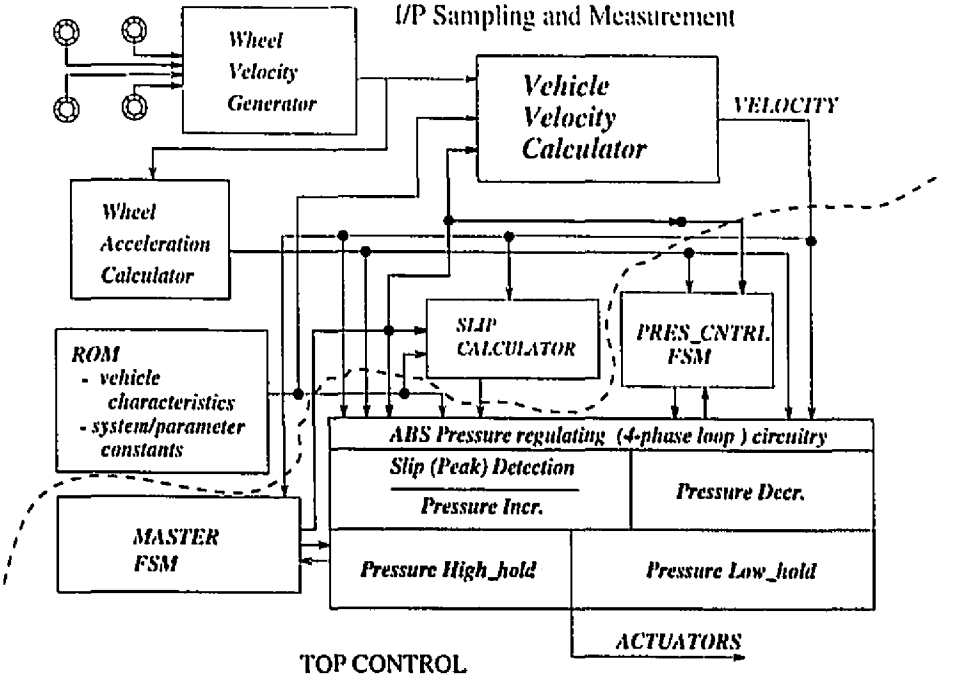

Figure 4.2: Block diagram of ABS Controller : Top Level

generated from individual wheel sensors. Four of these exist. The continuously measured velocities, or samples, are accumulated by the *average velocity calculator* resulting in an average velocity value at a lower sampling rate. Contingent on the state of the vehicle, particularly whether it is braking or not, the velocity is calculated from an average across diagonals with more weight given to the non-driven wheels since they better represent the wheel velocity. Diagonals are compared, with the four wheel average, and when braking occurs, past velocity values are taken into consideration for generation of the final velocity value. Using an external, *proposed with specification*, clock of 10MHz, frequency values are generated every 256 clock cycles. From this, the averaging velocity module takes 16 samples producing rounded velocity values every 0.4 ms. This rate is within range to successfully furnish values to the other modules and affect the necessary system responses, and thus it proves real-time capabilities of some controller on FPGAs.

The *vehicle velocity calculator* and storage unit, calculates, broadcasts, and registers[2] the

[2]When braking is present and slipping occurs, previous velocity values are more reliable and are required

velocity of the vehicle which is needed for numerous computations and subsequent operations by neighboring modules. Vehicle velocity is derived from the angular velocity, radius, and drive type[3] of the individual wheels. Velocity storage employs first-in-first-out data management as incoming values arrive continuously and order must be preserved. Other modules like the angular and vehicle acceleration calculators, the slip calculator, the timer, and the peak/slip detector all calculate their values from both measured, calculated, and previously stored quantities. The central one being the instantaneous angular wheel velocity.

Ideally, an exact, embedded vehicle velocity unit is desired, such that the module could reliably, detect the velocity by facilities such as *motion detection* through image processing techniques, laser beam measurements on road surfaces, radar techniques, or with road side external units which measure and transmit vehicle velocities on a "request" and "send to owner" protocol. ABS circuitry would be reduced with direct vehicle velocity availability.

Pressure control FSM: the 4-phase loop

Once in the ABS enabled state, a sub-controller takes over and monitors the 4 phases of ABS control which involves regulating the pressure applied to the braking system, through dually controlled valves. A *limit cycle* is comprised of four substates: pressure increasing, high-pressure holding, pressure decreasing, and low-pressure holding and is used to keep vehicular operation during slippage around the peak friction coefficient of the F_x curve, B in Figure 4.1, so as to secure optimal braking force. Each state is represented by a block whose function is to generate its respective *deceleration peak* or *time duration* threshold when activated. The control parameters for every mode are designed to achieve phase transitions under all road conditions[4] for all the possible ranges of slope, subsequently denoted as m, in regions A and C, and to follow a curve circumferencing S_{opt}. The mode threshold calculators are individually enabled by the *pressure control loop* FSM, which indirectly authorizes their access to neighboring hardware resources through multiplexers. Figure 5.4 illustrates the

in future velocity predictions and calculation of slippage conditions.

[3] A wheel is said *driven* when gas pedal pressure affects the wheel differential directly effecting a turning movement. In a front wheel drive vehicle, the rear tires merely roll along the road with no gas-pedal pressure torque forces.

[4] Varying road conditions result in different slippage values which result in different slopes for the curve in the friction-percentage slip graph. Thus, the slope m numerically equates to a ratio of friction over slip yet reflects the road conditions.

entire pressure control block with subcomponent elements, the timer, the multiplexers, and test circuitry and the external ROM which houses the LUTs.

As recommended by Kuo and Yeh [8], the transition of states from the **pressure increasing mode**, or initial braking, to the high pressure holding mode occurs when the wheel deceleration exceeds a certain value H_1, i.e. $-I\dot{\omega} > H_1$. In lieu of selecting a threshold quantity which exists between two calculated boundary values [8], one lower bound threshold is calculated and a predetermined offset is added to it[5]. H_1, the threshold value triggering the high-pressure holding mode, is obtained from H_1^* which is determined from the largest value of h_1^*, the critical value of torque differential to pass over the peak of the T_r curve for a given road condition, for all road conditions, so that $H_1 > max\{h_1^*(V,m)\}$ over the range $(10 \geq m \geq 1)$. h_1^* can be found by solving the simultaneous equations (4.1) for S_1 and h_1^*, from [8], which are the slip value and $[T_b - rF_x]$ respectively.

$$\begin{aligned} S_1 &= -\frac{h_1^* - mlg(1-S_1)S_1/r}{mrMg} - \frac{IVU_i}{m^2r^3M^2g^2} \times ln\left[1 - \frac{mr^2Mg}{IVU_i}\left\{h_1 - \frac{mlg(1-S_1)S_1}{r}\right\}\right] \\ S_1 &= S_{opt} - \frac{h_1^* - mlg(1-S_1)S_1/r}{mrMg} \end{aligned} \tag{4.1}$$

Analysis runs using MATLAB[6] over extensive ranges of m, V, and system parameters were attempted to produce a safe incremental offset of h_1^*, which satisfies the condition for H_1, and which translates well into hardware as seen in Equation (4.2). Additionally, VHDL encoding and hardware implementation limitations engender certain adjustments since many of the above operations are not readily available, particularly on FPGAs. Note that the equations (4.1) can be reduced to the set in (4.3) and (4.4).

$$H_1 = h_1^* + \frac{1}{8} * h_1^* \tag{4.2}$$

$$S_1 = S_{opt} - \frac{k_1 V}{m^2}\left[1 - e^{-\frac{S_{opt}m^2}{k_1 V}}\right] \tag{4.3}$$

$$h_1^* = m[\gamma_1(S_{opt} - S_1) + \beta_1(1 - S_1)S_1] \tag{4.4}$$

[5]When a range of values exists for the threshold calculations, the minimum value can be chosen as a base, knowing it will be first attained. The precalculated offset (calculated with the aid of MATLAB simulations) must be valid over the entire range of input variables and when added, must ensure adequate timing or deceleration values for correctly timed state transitions.

[6]a registered trademark of The Mathworks Inc.

where, $k_1 = \frac{IU_1}{r^2M^2g^2}$; $\gamma_1 = rMg$; $\beta_1 = \frac{Ig}{r}$

The initial detection of critical slippage/angular deceleration during braking has been incorporated into the four phase loop controller. Consequently, *phase 1*, the pressure increasing mode, where a threshold value is calculated and compared to the wheel angular acceleration, will be activated during both of the *ABS control enabled* and *braking* states. The former will result in a change of pressure state but not effect a change in the vehicle state while the latter will.

Once the **high pressure holding mode** is triggered, a time interval T_2 elapses, and a slip change of S_2 is permitted, before the flat trajectory of Figure 4.1 is changed into the pressure decreasing mode. The time interval T_2 must be long enough so that the trajectory during this mode passes over the peak of the F_x curve for all road conditions, obtains maximum brake force and avoids wheel lock-up, but not too long to cause a greater than allowable slip change of ΔS_2. A necessary and sufficient ΔS_2 of 0.4 is recommended [8]. To satisfy both conditions, an effective T_2 must be chosen such that, $T_2^* < T_2 < T_2^{**}$, where the limits are determined in [8]. We propose sufficiency in the calculation of T_2^*, Equation 4.5, with a corresponding predetermined offset, to determine T_2.

$$T_2^* = \underset{0.5 > m > 10}{\text{MAX}} \left[t_2^* : t_2^* = \frac{-IV}{mr^2Mg} \times ln \left\{ 1 - \frac{mrMg(S_{opt} - S_1)}{H_1 - mIg(1 - S_1)S_1/r} \right\} \right] \qquad (4.5)$$

For our ABS design, both the calculated T_2 and known S_2 values are used to detect the end of the state. Each condition then acts as mutual back-up, where the first condition met induces a change of state. Slip comparisons can be done immediately whereas T_2 must first be calculated, followed by the start of a timer which signals expiration of time interval T_2. Reduction of Equation 4.5 based on the approximations of Equation 4.2 results in Equation A.16 as derived in Appendix A.

Both holding phases exist for a duration of time calculated from the present state of the system variables. Expiration of the time is measured by an on chip timer. Calibration and normalization must be done both in theory and in hardware. The time count value generated is fed to the timer which counts clock cycles of known period. Hence the input clock frequency must be fixed and worked into the equations, which in turn will have to be

normalized for the hardware. MATLAB aided in these calculations as well as for the pressure control loop calculations.

Several constants are known a priori, being either vehicle specific or directly dependent on road or temperature conditions. For the mathematical operations which must be calculated on-line, look up tables (LUTs) are best employed as implementation aids to generate equation results and complex computations such as logarithms. Section 4.2.1 clarifies the usage of MATLAB for LUT generation. Hence both threshold and time-duration values for velocity V, and road conditions m, can be determined from one or more *read cycles* of the LUT, realized through a ROM. The value obtained for the angular deceleration α (or $\dot{\omega}$) is compared to the calculated value for H_1 to determine whether the next pressure controller state becomes the pressure holding state. Implementation of the pressure increasing phase of the four phase loop is illustrated in, Figure 5.2.

Similar assumptions and calculations were made for the remaining two states. Enabling of the **pressure decreasing mode** initiates generation of the threshold value $H_3 = 0.08rMg$, which must be compared to input wheel deceleration to determine the state termination. With such an approximation, further outlined in Appendix A, no LUT is required. The **low-pressure holding mode** must cater to more possibilities and hence demands supplementary cases which introduce additional internal sub-states of operation. Three threshold quantities to be determined in this mode have been reduced to that of H_3 from the previous phase and equations A.19 and A.20, derived in appendix A. Sample numerical values for time intervals in the holding phases were found in [8] at T_2=0.08sec and T_1=0.25sec.

4.1.4 The Self-Testing Design

The specification requires that the system continually test its functionality. Incorrect operation must be detected due to the severity and potential hazards of malfunctions. Consequent to the nature of the design, its timing, space limitations and desired responses, a time-sliced "hardware re-usable" solution for on-line system monitoring was employed. Tolerance for the resulting overhead is reflected in establishing it not merely as a part of the test circuit but also as a part of the design.

Clarification on the motivations for online testing is required. The purpose is not to

detect manufacturing faults. Chips received from the FPGA vendor are assumed to be tested and working. ABS controller testing monitors *field faults*: those errors which occur during the lifetime of a circuit upon leaving the site of programming. Detection of errors caused by physical elements such as transient faults, field distortion effects, mechanical wear, radiation effects, and aging, etc is imperative for correct and reliable operation. The interaction between embedded controllers and their surroundings must be continuously monitored since certain alterations or variations, be they electronic or magnetic, may well effect circuit functionality at some point in time. Vehicle wear, for example, change of tires, an accident, installation of heavy equipment on the vehicle, falls under this category. Additionally the weather may have detrimental effects. At some point in time the vehicle may undergo sufficient changes to warrant re-programming of the original controller.

A control system such as the ABS, which depends on external variables to control its operations and change of state, will not exhibit a predictable operation. Subsequent states of control, their duration, and sequences of operations depend on the input conditions, (the environment, user, and vehicle) and on timer expirations, an additional internally generated input. These elements, classical in asynchronous systems, contribute to a degree of nondeterminism of the controller, for even though a *next state* can be defined in terms of the inputs and *present state*, its duration is not known, and it will only change dependent on asynchronous changes in the environment or the internal signals.

Such a circuit cannot be tested for all situations as there exist a myriad of ways, times, and durations for which a test circuit would have to be activated. However, by definition, controller applications do have a fixed number of states and a known set of state sequences which can be tested even if their duration is not known. Hence, test suites can be organized to cause state transitions by injecting "prepared" inputs and thenceforth effect calculation sequences. Responses can be checked for correctness by accumulating the partial results and measurements along the sequences of test state transitions and comparing them with signatures known and stored *a priori*. Testing a subset of wisely chosen state transitions can validate functionality of both the ABS controller and the self-testing circuitry itself. The level of coverage will depend on the number of test sequences stored on chip which in turn depends on the amount of free space on the FPGA after circuit synthesis.

For ABS, such *test injection* refers to the injection of velocity values into the *pressure*

control loop modules where threshold values will be generated as in a regular iteration of the controller, state transition will occur, necessary complementary inputs will be generated, further transitions will occur and all the results will be accumulated and compared with the FPGA stored signatures. During test mode, some blocks must have optional feedback paths to input their calculated thresholds, so that transitions can or cannot occur. Physical ailments will affect the interaction between the ABS and its surroundings which in turn will affect the measured values. Incorrect measurements, out of range samples or responses can be detected by a circuit which exhibits this *test injection* behavior, as erroneous results will be propagated to incorrect accumulations.

4.2 Design Engineering and Mechanics

Three phases comprised the progression from controller specification to FPGA implementation: (i) the theory/mathematics, (ii) the approximations, (iii) the implementation. Theoretical considerations involve exploring the mathematics of vehicle dynamics along with analyses of the system and its environment. Approximations in the form of linearizations, system parameter and variable assumptions, to name a few, can be employed to simplify system representation. The implementation encompasses the circuits realization and the further approximations required to accomplish its task. Mindfulness of precision needs versus sufficiency will greatly affect the circuit's final implementation. A hierarchy of decisions is required. For example, selecting integers for the datapath operations mitigates a further decision of range, which in turn determines the datapath width, and subsequently requires choice of procedures for rounding and normalization.

Computer Aided System Technology (CAST) facilities are highly useful for the above type of approximations. MATLAB analyses aide in ranging (finding the maximum and minimum points), word-length delimiting (integer size for desired precision), and verification (to ensure reduced equations still produce desired results and match or approximate well the system dynamics). Particular attention to I/O approximations is imperative for controller applications. Sampling rate, clock(s') rate, calculation rates, and response frequency decisions must all be made to suit the design specifications and adhere to the limits of technology. Simulated analyses of the various options and approximations aid in circuit specification for

eventual implementation.

One remaining area for estimation inheres within the digital implementation domain. Analysis of the continuity and sufficiency of the power-of-two number space can lead to acceptable approximations and simpler implementations, if operations on such data were to transpire. Making use of *shift operations* to aid in normalizations or to replace division with multiplication, or performing calculations on power of two multiples as opposed to consecutive integer or continuous real values for approximation purposes are two such examples. Several such techniques were used, but not being specific to FPGAs they will not be discussed further. Such approximations are very specific to the controller in question and future work could focus on such a class of circuit reduction techniques.

4.2.1 Tooling of Design

MATLAB is credited with its analysis and verification of mathematical equations and their simplified versions. Specification of hardware boundaries and realizations of threshold values, bit-lengths, time interval calculations in addition to the generation of LUT values for the ROM entries were all facilitated by MATLAB, and MATHCAD^{TM}. Determination of maximum (minimum) integer values for register size delimiting, floating point rounding for fixed point implementation by the multiplication of powers of two, and truncation steps are some examples. Synthesis requires integer ranges to delimit register sizes, otherwise a 32 bit default size is allocated. Likewise, ROM data was created using MATLAB and placed in an array structure for circuit simulation. Fixed point values were calculated and multiplied by powers of 2 to create integer values which could be handled by all ABS blocks with their data defined in terms of integers.

Some sample MATLAB equation analyses are shown in Figures 4.3 and 4.4. Each contour represents a different vehicle velocity. Both graphs provide boundary and contour information, for their respective mathematical equations. Both state transition threshold h_1 and the state duration time T_2 vary over the slope m, where m represents the tire-road traction. Higher slopes indicate less friction and more slip which directly depends on the road conditions and can be equated to wet or icy surfaces. A peak in the threshold graph indicates the point of maximum vehicle deceleration and equivalently maximum friction coefficient and

braking effectiveness. It is around this point that the pressure control loop will cycle during emergency braking maneuvers.

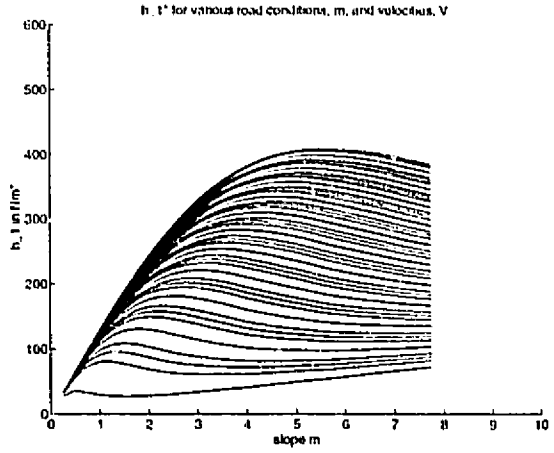

Figure 4.3: Thresholds for various road conditions and a range of vehicle velocities

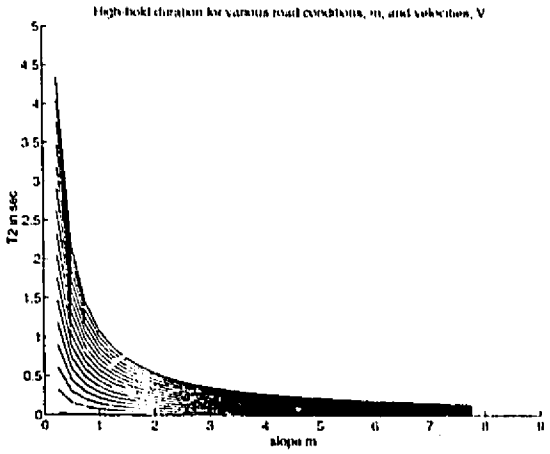

Figure 4.4: Time duration to maintain brake pressure in the high and low holding states

4.2.2 Preamble and Approximations for an ABS Realization

Circuit realization evolves not only through synthesis tool iterations but through design trials as well. Prior to addressing the final implementation, a few prelusive conceptualization facets deserve presentation. Ideally a specification of a control system should determine hardware resources and circuit implementation. In some cases the reverse may occur. Given a set of hardware limitations the design, while still meeting response times, may require adjustments. Control systems like ABS can afford to undergo some approximations and normalizations, which facilitated the fitting onto FPGAs. The level of estimations acceptable can be determined by simulation and synthesis iterations: steps which programmable FPGAs are well-suited for, compensating for their lack of density, abundant in their ASIC competition. Hence trade-offs in addition to design tricks such as LUT usage for pre-calculated results surface. Though feasible, such "hardware adjustments" carry disadvantages; (1) lack of precision, (2) block module interconnections issues if individual blocks have their equa-

tions normalized, (3) result interpretation, (4) loss of generality if too specific, and (5) extra memory requirements, to name a few. The problem to solve now becomes that of finding the level of approximation which can be tolerated to achieve the required response times with the available hardware resources, without causing too much added design-time overhead in their incorporation..

Some mathematical approximations or algorithmic simplifications found in the encoded sections of the design can be applied to other controller types, generalized and classified as *generic*. For example, fractions limited to a representation with denominator power of 2's, continuous function ranges represented by discrete ranges, and holding certain continuous values constant for a few clock cycles so that some calculations can complete before input variables change, are some examples. With particular reference to the ABS design, some of the following specific cases originated, while additional examples are cited in Chapter 5.

Most blocks in the ABS controller depend extensively on vehicle velocity, a parameter which depends on continuously changing values and, hence, one which must be constantly updated. Sampling at intervals is equivalent to latching the value in hardware, and holding it over successive clock cycles in order for calculations to be performed. The faster the processing capabilities, the more frequent sampling can occur, and the more accurate the responses generated. A minimum sampling frequency, on the other hand, will be determined by the precision requirements not by hardware limitations. As with most circuits, FPGA throughput ties in closely with accuracy. Exact choice of range and bit precision can be approximated in hardware yet will be finalized in the course of the embedded system tests to validate that sufficiently precise controller functionality is attainable once in the destination environment. Selecting a range of 0-63.75m/s for velocity proved acceptable offering a large enough range for the threshold calculations as per MATLAB, and subsequent comparisons. In hardware this translated to a normalized integer range of 0 to 255 where a fixed point notation[7] of [6,2] was ultimately chosen for its representation. Accordingly, the velocity generation and utilization modules must respect this range of values.

Some specific cases depicting the strength and usefulness of power-of-two operations come from module alignment and division avoidance. Numerous adjustments of multiplica-

[7] $[n,d]$: n bits for integer part, d bits for the fractional part.

tion/division by powers of two were incorporated through shifting, bus slicing or alignment, where successive bus connections were purposely misaligned. Division can be replaced by multiplication, LUT operations, or consecutive additions by initially left-shifting the fixed-point or right shifting the integer (dividing) the number by a large power of two value.

The search for a good value for the H_1 threshold came from analysis of the equations of h_1^* and h_1^{**} [8], and the assumption that calculating one limit will suffice and thereupon reduce hardware. Both equations were computed using MATLAB over discrete ranges of V and m. Based on the h_1^{**} limits, an offset ratio can be calculated which works for all h_1^* values. Accordingly, only h_1^* was used in the final calculation to be implemented. The formula, $H_1 = h_1^* + \frac{1}{8}h_1^*$ exemplifies the resulting estimate.

Though continuous time variables, velocity, V and friction-slip ratio, m, reside in the set of input variables, hardware implementation requires discrete ranges, warranting in our controller resolution to fixed point ranges with decimal powers of two. Correspondingly, calculated values, such as h_1 and t_2, are continuous in theory but acquire fixed point representations once destined for a hardware realization. Surveillance of the deceleration threshold equations over the ranges of the m parameter[8] for fixed values of V, resulted in an h_1 *maxima* between range of $0.5 \leq m \leq 7$. Hence a numeric range for m with power of two fractions, represented as normalized integers, could be determined along with the number of bits for its allocation. A similar analysis is done with MATLAB for the calculated threshold values. Countenance on a ROM implementation, and on utilizing V and m as indices, affects choice of ROM size and of the level of precision possible for the indices (m and V), and the calculated values. Eventually, a normalized integer range from 0 to 31, 5 bits with the two bit fractional part, was allocated to represent the above range, a fixed point equivalent of [3,2]. Additionally, a normalized integer range from 0 to 4095 was chosen for LUT entries stored in the ROM.

Some elements which do not translate well from theory to implementation are limits and operational ranges, boundary value calculations, and special cases. Equations executing at a full range of inputs, measurement of values which result in 0 outputs can produce undesired effects. These elements, both from a design and an implementation perspective,

[8] m actually represents the road conditions and is mathematically represented as the ratio of the friction coefficient, or a linear multiple of it, and the slip as seen in Figure 4.1.

were not covered by any papers within our research, including those [6], [7], [8], which directly influenced our realization. Excluding the "out-of-bound" cases with restricted loop or "if" statement clauses, duplication or time sharing of certain hardware blocks, if possible, to account for varying processing procedures for full input ranges are some means of managing indeterminacies found within equations and theory.

Time, ease of data retrieval, space, flexibility and precision concerns were conducive to the acceptance of LUT implementations via an external ROM, in place of computational hardware. Once again, numerical estimations were necessary particularly with respect to ROM and individual LUT size, as hardware is not limitless. Threshold calculations depend on iterative computations and comparisons which can all be related to V and m. Hence, they became the obvious choice for indices for the LUTs holding equation computations. For each phase requiring a LUT operation (a computation), a LUT of size 8K must be alloted in the ROM. Hence we will be saving time, for memory read operations shall be performed instead of real-time calculations.

The *low-pressure holding mode* requires two wheel decelerations thresholds in addition to the time duration calculation. Both H_3 and H_4 can be calculated a priori. H_4 marks a safe guarding region for the H_3 condition in this mode, so that the oscillations of pressure which can take place at this time have some room to move in, i.e. between the two threshold values. Because H_4 was not defined in the paper by [8], an approximation was defined with the help of MATLAB simulations. So to date depending on the accuracy of the value needed, H_4 will be a fraction of H_3, and will be one of two expressions: either $H_4 = \frac{5}{8}H_3$ or $H_4 = \frac{9}{16}H_3$.

A possible future extension to the deceleration slippage control of an ABS implementation could be to incorporate acceleration slippage control. Such an addition elicits motivation from a desire for engine efficiency, vehicle handling, minimizing stress on vehicle components, etc. in contrast to the predominant safety precautions for ABS. Basically, a system for regulating wheel slip at the driven wheels () possesses similar requirements to those for a system designed to regulate brake slip (ABS); steerability, directional stability and effective transmission of longitudinal forces onto the road. Even further, as current research suggests, lateral slippage can also be incorporated into an ABS.

Chapter 5 addresses the final phase, the encoding in VHDL, the circuit realization for syn-

thesis, FPGA hardware analyses and design methodologies. Technology dependent concerns are required if most available resources and features of an FPGA chip are to be exploited by way of the tools. At the present time, synthesis algorithms have not yet matured to fully exploit all architectural attributes of the many FPGA types.

Chapter 5

FPGA Synthesis

Chapter 4 aided by Appendix A, introduced the ABS controller theory and presented building blocks and design angles. With the groundwork of theory laid, and the specification of the ABS controller completed down to the sub-block, encodable elements, that which remains and which is more conducive to our research, is the implementation. The steps undertaken from design entry to semi-detailed implementation procedures and results, to the self-testing circuit of the FPGA realization are all described in this chapter. The experimental design goal can now be stated as: *To ascertain whether real-time asynchronous control type applications can be implemented in FPGAs and withal conform to the necessary real-time constraints, and to provide methodologies for application specific design, synthesis and layout (ASDSL).*

5.1 The FPGA Design Cycle

FPGAs have successfully instituted their way into the realm of rapid prototyping. Their success can be attributed to their ability for *implementation fine-tuning* due to their field programmable features. Hardware and software (tool) availabilities contributed to the choice of technology for this circuit's realization.

5.1.1 Tuning an Application to FPGA Structures

The design phases for the development of a rapid prototype are depicted in Figure 5.1. VHDL design, simulation and synthesis were performed using the CAD Framework from Synopsys. XACT tools from Xilinx were used for partition, place and route (PPR) and bit-stream creation for FPGA downloading onto XC4010s. One key aspect in this FPGA design takes root in the *tooling of a design*, which refers to the use of Computer Aided System Technology (CAST) to facilitate some of the specification and design phases. MATLAB was used to analyze certain equations for range limits, for variations in system parameters and variables, and for pursuit and verification of approximations which can be made while still achieving correct controller functionality.

The original design goal was to fit the entire controller onto a single FPGA. A Xilinx XC4010 consists of 400 CLBs and 160 IOBs, and carries an equivalent gate count of about 10K. Due to technology limitations, multiple FPGAs were considered with the potential of using on-chip RAM facilities for storage of constant data. *Constant data* can be determined a priori and may consist of physical constants, calculation results (multiplication, division, logarithm), and normalization results. With the lookup table[1] (LUT) sizes required, FPGA logic blocks programmed as RAMs proved insufficient and external memory elements were sought, such as RAM or ROM. Perhaps with the availability of application specific memories (ASMs) and the creation of programmable ASMs, such logic and memory implementations, i.e. one chip solutions, will be possible. The conceptualization of compact memory elements combined with standard logic is difficult due to the different manufacturing processes required for each. However design strategies are surfacing which can create efficient memory structures combined with logic, and the area is open for further research.

Eventual implementation involved manual design partitioning, optimization, and resource sharing. For the controller circuit, two FPGAs were required along with one dedicated memory chip, a ROM, to hold the bulk of the LUTs created using MATLAB. Aperiodic testing was then annexed using a built-in signature analysis scheme and resource sharing. The same hardware is used for both the controller operation and on-line testing. This is accomplished with the added overhead of multiplexor inputs, and off-state cycle stealing.

[1] Can be regarded as a large array of constant values

When the bulk calculating circuitry is not in use, during off/normal cycles of the vehicle, the test circuitry controller awakens and proceeds to verify the circuits operation with injected test values, measuring, accumulating and tabulating the results.

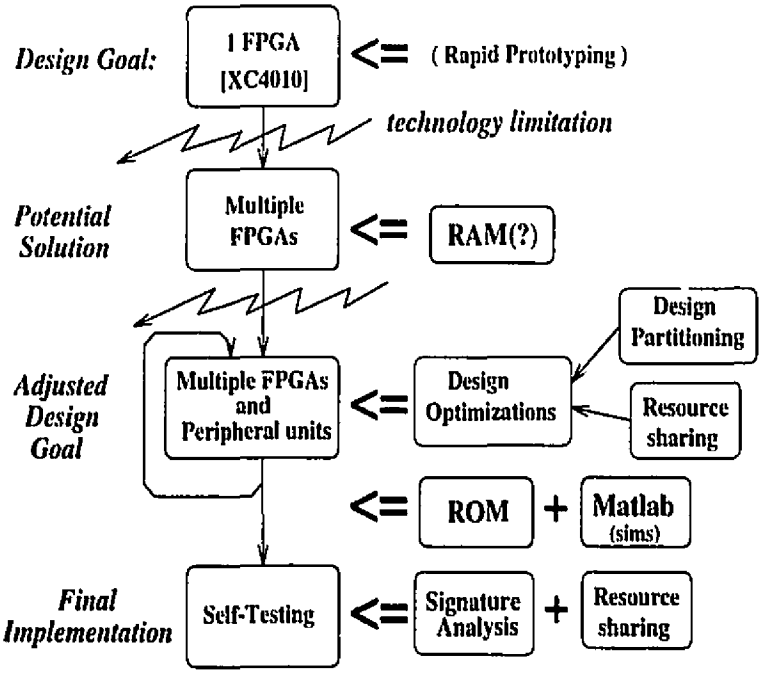

Figure 5.1: FPGA design steps and realization cycle

Fine-tuning the design in its embedded environment may alter some of these hardware realizations. The advantages of FPGAs for both rapid prototyping and final circuit realization surface during the validation of circuit implementation approximations. Additionally, parameters change with time, and the controller implementation may need to be altered with time. FPGAs can easily be re-programmed in the field, the vehicular repair shop, as opposed to the factory, if an ASIC was the choice for implementation. Thus far, superior circuit density capabilities are the main support for ASIC implementation.

Most of the design originated from mathematical models describing ABS dynamics. Implementation of equations, variables and their units and normalization across functional blocks is not forthright. Logical interconnecting and normalization of values while maintaining their real world implications for external connections and responses in addition to

equation implementation or rather datapath realization were difficult tasks in the design phase. Selection of precision is another matter and simulations are required to settle on a final data length for the individual modules. Between 10-12 bits of precision are deemed adequate and normalization of input and output block values are based on the maximum and/or minimum expected values. For example velocity values (V), road condition (m), and LUT entries are represented by normalized integer ranges which translate to a fixed-point hardware formats of [6,2], [3,2], [6,5] respectively. With 2 bits for the fractional part, the precision is held at ± 0.25. Shifting of the datapath is required to align interacting data for correct processing.

5.1.2 Encoding Primitives

A synthesizable VHDL subset along with some encoding guidance, found in one of the manuals of the Synopsys' tool set [52] became an aide for circuit description. Careful attention was given to the hardware inference resulting from VHDL synthesis. Standard VHDL syntax, such as *if*, *case*, *loop* statements, variable and signal assignments were utilized, while all syntactic timing elements, (*after* and *wait*, statements), were avoided. VHDL format also enables direct control over combinational and sequential realizations, as well as satisfactory synthesis of certain behavioral descriptions. For example, depending on the context, variable or signal assignments may infer either registers or latches. FPGA architecture harbors certain kinds of flip-flops and dependent on completeness of a signal assignment (whether all cases of inputs are covered), either sequential (flip-flop usage) or combinational circuitry will result. Additionally, a statement such as $C <= A + B$ will initiate a search in the technology specific library for specific blocks which are capable of supporting the addition function. Design constraints subsequently effect final block or macro selection.

In controller applications, most inputs occur asynchronously. Outputs can be synchronized to a clock, but as most actuators are expected to respond asynchronously, this is not really necessary. Careful VHDL encoding can generate and control both synchronous and asynchronous events, as seen in section 3.3. VHDL examples were included illustrating handshaking protocols where entities wait for events to happen before continuing their execution. Synthesis tools easily support events which portray clock signals. Difficulties arise when

an event is not a derivative of a global clock signal. They are often reflected by errors in synthesis or if the circuit can be synthesized, by requiring closer scrutiny during simulation if the circuit can be synthesized.

Our stratagem requires catching asynchronous signals and synchronizing them to a clock, and permitting sub-clock handshaking but eventual clock synchronization similar to what was mentioned in [44]. The statement `WAIT UNTIL clock <edge>` is used throughout the code to synchronize events with one of the system clocks. It can also be employed more than once in a process statement to "wait" for multiple clock cycle times, so that events in time can be ordered/scheduled. A `WAIT UNTIL` can also be used for handshaking to control inter-process communication, a procedure which is somewhat asynchronous and requires careful design and simulation to ensure correct functionality. Interface action can then occur with a finer granularity than clock cycles with the use of the above `WAIT` statement and a compiler which can synthesize them correctly. VHDL encoding of the above would thence require that interface procedures terminate on a clock transition (*wait until clock <edge> statement*). Global time, and control can then be maintained throughout the design. The ABS controller employs this type of timing control, executing sequential statements and then synchronizing to either the fast clock (clk_in) or the medium clock (pipe_clk). Process statements, generated responses will all appear synchronized at the periphery even if there are slower or faster events from within. An alternative finer-grain solution worth exploring is to use a faster clock, as limited by the technology, to trap all signals and make responses quick enough so the circuit becomes completely synchronous.

A sample encoded control FSM and datapath implementation from two blocks of the ABS design exists in Appendix B. The FSM is broken into two processes, the combinational process which determines the next state and should define all signals for all input combinations, and the sequential process which infers registers to hold the current state via inclusion of VHDL `WAIT until clock <edge>` statement. Both datapath and control can coexist as seen by ROM access requests and multiple use of `WAIT` statements in the same process. Several of the VHDL models contain `WAIT` statements to control the multi-cycle flow of events using the medium speed clock. As FPGAs are register rich, one may use a pipelined style of design to obtain increased throughput, should the demand be there.

5.2 FPGA Implementation of ABS

The dotted line in Figure 4.2 separates the design into the two original partitions or modules, one containing the "input sampling and measurement blocks", and the other the "top control and action blocks". In the current realized version the slip calculator has become part of the "top control" modules, as there was space available and its result was being processed directly by the four phase control loop. Each module, requires a single 5-10K FPGA chip for realization, and when larger FPGAs become readily available, samples are already out, it is estimated that the design will fit on a single chip. All of the initial specification, the top level ABS block diagram, and the resulting VHDL encoding encapsulate hierarchy. Encoded VHDL entity were written to agree with the block diagram and its inherent hierarchy of blocks. The corresponding *architectures* with *process* statements of the entities describe the block's functionality. The block interfaces to neighboring blocks translate to entity ports. Synchronization amongst the blocks, or entities, follows from individual event interpretation as exhibited in the VHDL FSM description listed in Appendix B. Recall that high level synthesis concerns were the driving forces for the use of sequential statements within the VHDL process configuration. Depending on the complexity of the block, its constituents and levels of hierarchy will vary.

5.2.1 Control Circuitry

Both of the above modules with their subblock functionality were specified in Chapter 4. A subset of them from the "top control and action" module which exemplifies unique, common, or special feature progression from original specification and VHDL encoding to final implementation and testing were selected for further mention. Five blocks of interest: the *Master_FSM*, the *Pres_level_cntrl* FSM, the *Pressure-_incr_cntrl*, the *Pres_high_hold*, and the *Slip_ calculation* block shall have their implementation specifics described here.

The *Master_FSM* block determines the vehicular state at all times and will be disabled on an error signal event generated by the self-testing circuitry. Its four major states of operation are *stopped*, *normal* (moving without braking), *braking* and *ABS_enabled*. The vehicle state will change from the initial stop state to normal when the vehicle moves, and subsequently to braking if the brake pedal is pressed. The minute this state is entered a threshold calculation

sub-block is enabled and if critical slippage or optimum braking is attained, the ABS four phase control loop is enabled putting the vehicle in *ABS_enabled* mode. Appendix B lists a concise VHDL description illustrating how combinational and sequential logic inferences and a synchronized FSM can be synthesized.

The *Pres_level_cntrl* block, also known as the minor FSM, works only in one sub-state of the *Master_FSM*, and functions similarly. It is charged with controlling, enabling, and absorbing results from the external modules and from its four interacting sub-modules which are activated at different times in the *limit cycle* of ABS enabled operation. One of these states activates the *Pressure_incr_cntrl* block, which calculates threshold values based on the current (or sample) value of the velocity. From the theory, the Equation 4.1 led to the calculation of the threshold angular deceleration value. Subsequent to some hand approximations and simplifications, the Equations 4.2, 4.3, and 4.4 resulted. A block implementation of the pressure increasing phase of the four phase loop is shown in Figure 5.2. A LUT holds all the potential S_1 and h_1 values indexed by both the m and V values. The comparator checks the results and tracks the maximum value of vehicle deceleration to attain before the next state must be initiated. A loop generator was required to cycle through a limited range of road conditions. Though one value could be stored instead of several, this approach was taken to account for future options where an external m value could be generated by external vehicle or road-side generators. The synthesized schematic representation of the *Pressure_incr_cntrl* block in Figure 5.3 depicts the instantiation of two hard macros, one for the "integer" comparison, and the other for an addition. Each box represents a programmed CLB containing some mapped combinational and sequential circuitry within.

Three similar blocks and one further sub-calculation block which exhibit ordering of operations, control, read from the ROM, work in multi-cycles and perform some kind of handshaking with other sub-blocks, were also encoded in VHDL, along with their complementary and secondary blocks required for full ABS operation. A block diagram of the implementation of the four phase control loop is outlined in Figure 5.4. The existence of multiple concurrent events requiring multiple clocks, makes the task of synchronization a delicate task. Nonetheless, each sub-module was synchronized producing deterministic responses and outputs. `WAIT` statements were strategically employed to permit efficient capturing, storing and processing at alternate times of block input data in addition to supplying output data

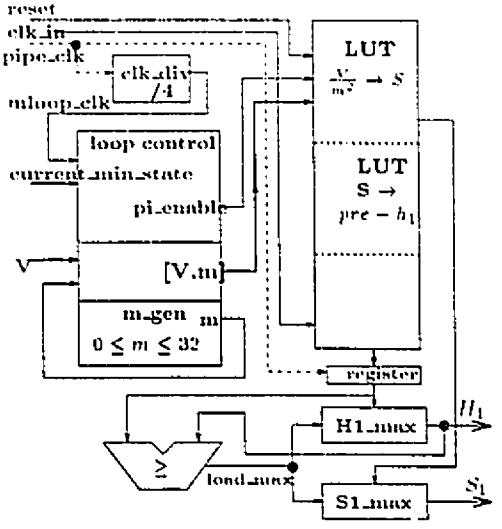

Figure 5.2: Block diagram implementation of the pressure increasing state.

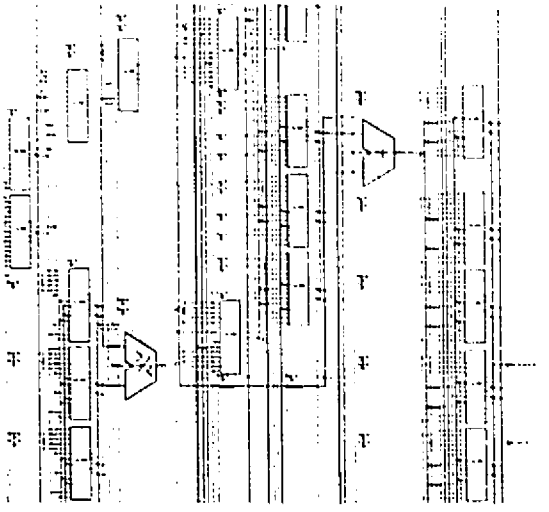

Figure 5.3: Synopsys' synthesized version of the pressure increasing state and its calculations.

to external actuators at reasonable times with respect to the hydraulic response times. Wait statements were also employed to materialize handshaking protocols within and among the various modules.

The overall picture, Figure 5.4, includes these blocks as well as some multiplexer blocks to aid in hardware re-usability and some testability resources. Presently, the entire ABS controller does not yet fit on a single FPGA. As this was one of our targets, any advantageous simplifications should be exploited. In particular, potential resource sharing should be attempted by the designer as the synthesis tool cannot globally implement resource sharing. (In fact, the Synopsys compiler restricts sharing to within process statements.) The calculation of ROM addresses using individual offsets for LUT access in some of the four phase modules, exemplifies such forethought. Though each read request could have calculated its own address, encoding this (effectually an addition for each state, with different offsets) in VHDL across *processes* would result in instantiation of several adder units. Accordingly, having one unit performing the addition of an offset contingent on the pressure state during the ABS_enabled state would save on area so long as pressure control state information can be made public, which is the case. Figure 5.4 illustrates the addition of an *add* operation to

the *multiplexer* unit which does not merely serve as a *multiplexer* but as a *MUX+add* unit. Manual synthesis had to be performed across processes.

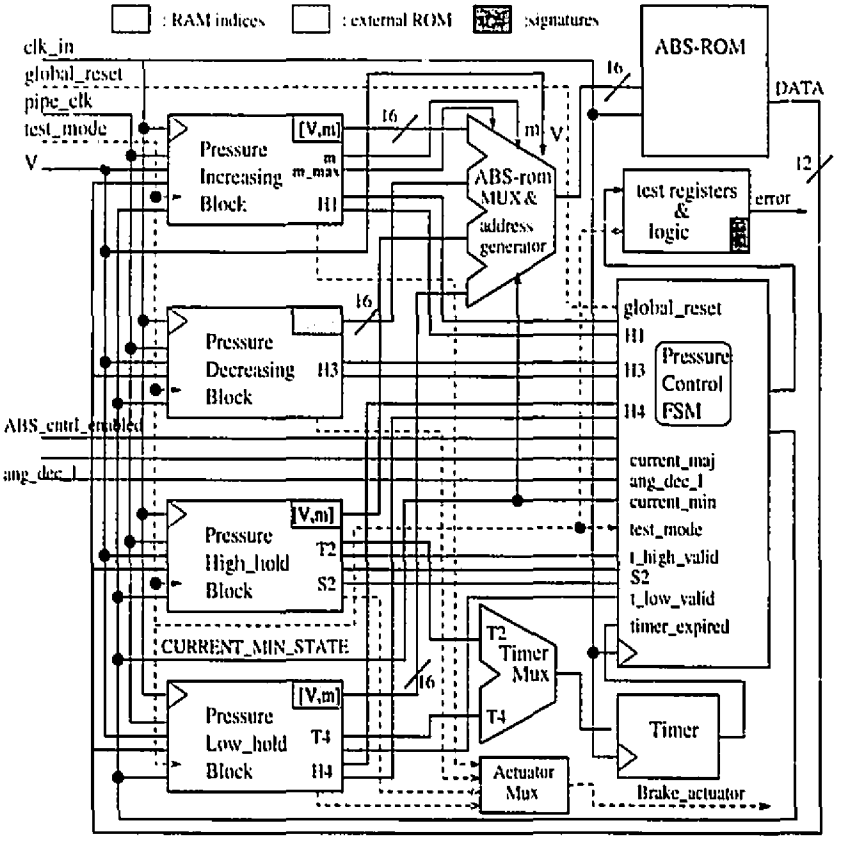

Figure 5.4: Implementation of Pressure control Blocks.

The *Pres_decr_cntrl* block exhibits analogous behavior as its sibling *Pres_incr_cntrl* block. However additional conditions must be analyzed which is reflected in the minor FSM (pressure control FSM). Longer VHDL code with more clauses, *if..then...else statements*, surfaces but otherwise, its realization follows suit. The *Pres_high_hold* block differs in that it requires access to the timer which is a simple downcounter with reset capabilities. When time expires, the top level pressure controller detects the need for a state transition. As with pressure high holding mode, a *Pres_low_hold* block behaves similarly in that a timer is required to track the duration of the pressure holding state. Unlike the high mode, after the time limit has expired, the next state may or may not be the pressure increasing as one would expect,

other conditions ($I\dot{\omega} >$ or $<$ H_3 and $I\dot{\omega} >$ or $<$ H_4) must be tested, see Appendix A. Subsequently, the timer circuitry may have to be enabled more than once in this state, and it is shared by both pressure holding states.

The *Slip_calculation* is as close to a datapath description as possible with the tools available. Encoded in a process statement, the datapath of the slip calculation involves the vehicle velocity, the wheel's tangential velocity and access (address and data paths) to a LUT or ROM. Explicit control using state variables for operation enable, variable (as opposed to signal) usage to sequence operations and synchronization of the entire process to a clock using a closing WAIT statement are depicted in Figure 5.5. Figure 5.6 illustrates the resulting synthesized circuit onto CLBs, housing the control and data path logic, and hard macros, representing the multiplication, addition and subtraction operations.

```
USE WORK.ABS_DEFS.ALL;
entity slip_cal is
  port(global_reset : in bit;
       pipe_clk . IN bit;
       V : IN integer range 0 to 255;
       freq_adj : IN integer range 0 to 255;
       CURRENT_MAJ_STATE:in vehicle_state_type;
       CURRENT_MIN_STATE:in pres_valve_state;
       rom_data : IN pres_value_word;
       si_rom_addr : OUT abs_rom_range;
       slip_cal_norm : OUT pres_value_word);
end slip_cal;
architecture behav_slip_cal of slip_cal is
begin
s_cal: process
  VARIABLE vel_adj : integer range 0 to 255;
begin
 IF (global_reset = '1') THEN
    slip_cal_norm <= 0;
 ELSIF ((CURRENT_MAJ_STATE = abs_enabled) AND
 ((CURRENT_MIN_STATE = high_hold) OR
 (CURRENT_MIN_STATE = low_hold ))) THEN
   si_rom_addr <= slip_offset + V;
   vel_adj := rom_data;
   slip_cal_norm <= SLIP_IDENT*(freq_adj+vel_adj);
 END IF;
 WAIT UNTIL pipe_clk = '1';
end process s_cal;
end behav_slip_cal;
```

Figure 5.5: VHDL code for slip calculation.

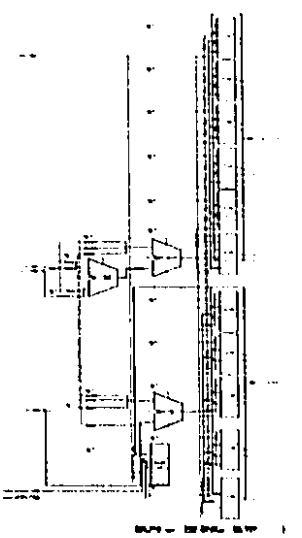

Figure 5.6: CLB representation of slip calculation.

5.2.2 Self-testing Circuitry

Design for testability was accounted for beginning with the specification stage and traversing through to the implementation. Functionality testing was done through simulation. Event timing was validated through the monitoring of input signals, state transitions, and generation of output signals. Back-annotation of technology specific timings remains upon avail-

ability of the respective tools. A built-in reliability check was implemented by the inclusion of self-monitoring circuit control blocks, multiplexed inputs, and accumulation capabilities.

During off cycles, a test clock (enabled through the test circuitry and derived ultimately from the main clock) is generated enabling the re-use of hardware for testing purposes. With a high-speed clock driving the computation logic, a medium speed clock driving the controlled pipelined stages, adequate time should exist to perform on-line system monitoring, within the system's specified response times. The slower *test* clock is only enabled during the *test* mode and monitors the global test events so that all calculations and state changes still function with their regular clock inputs. Recall, values are injected whereby this test clock will produce velocity values (stored on chip if the number is small and off-chip otherwise) which are normally calculated during *regular* mode operation. Such testing by stealing clock cycles, makes use of existing modules by initiating calculations and comparing results to signatures to verify correct functionality. Figure 5.4 showed the inclusion of test structures in the overall design. A test mode exists where the individual blocks carry out their regular operation yet register the results in separate test structures for subsequent accumulation and signature analysis.

A more detailed diagram of the test injection circuit coupled with the top level controller appears in Figure 5.7. Velocity, and wheel acceleration values can either arrive from the other blocks where they are calculated from external measurements or from the test circuitry block. Continual ABS malfunctions will turn the ABS controller off disengaging all brake actuators and inform the driver through a dashboard LED.

A little extra cost, for example the area overhead for on-line testing, can be alloted for a higher level of reliability and security, so its initial specification was implemented. Eventually the chip will fail, at which point detection is imperative and the ABS must shut itself off, indicating malfunction so that the user/driver is aware and can have it checked! The error can then lie in the chip itself or in a surrounding element of the braking system or vehicle. The source of error at this point is well outside the realm of an ABS controller, and is up to the vehicle repair person. However, further research can be invested to create a newer ABS version which contains additional circuitry for post-failure fault search and repair. In such a case a larger FPGA or, alternatively, an ASIC may be required in the final system realization.

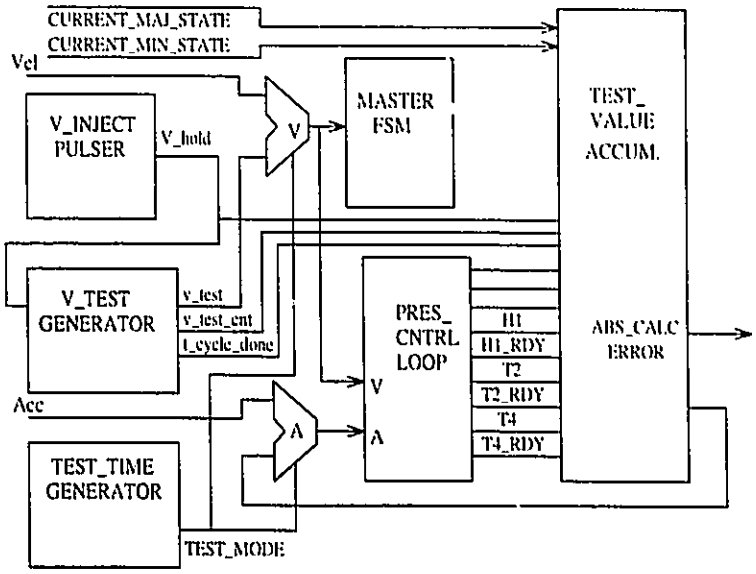

Figure 5.7: Self-testing circuitry added to the pressure control block

Self-checking, achieved with aperiodic circuit monitoring, ensures reliability and security through added test circuitry which detects functionality errors potentially occurring in the field. [2]. *Signature analysis* and *compaction* methods, renown built-in self test (BIST) techniques, along with *test injection*, constitute part of our ABS self-test circuit. An accumulation scheme was chosen over RAM for the obvious reason of space. RAM is expensive in FPGAs and accumulators are less area consuming. FPGAs are designed for such circuits. In fact, Xilinx implements fast carry circuitry which can be used to build even more efficient accumulators. For small test vector sets, and a single signature, the cost of an accumulator is not justified, but for this controller, 32 velocity vectors with a single signature require storage, an additional ALU is warrantable. The option to reverse the vectors, thus testing functionality with 64 vectors, can be readily accommodated by flipping a data bus halfway through a test cycle. Easily implemented in hardware, such a design concept requires manual instantiation as it is not assuredly synthesizable. Any VHDL encoding describing such an

[2] It is acknowledged that the design is not fully fault-tolerant but there is no problem or obstacles to extend it so it will become one. Similarly, 100% fault detection is currently not available, but its inclusion is achievable as long as the added area, and timing overheads (both in circuit and design) can be tolerated.

operation, would be sequentially listed inferring a circuit with many shifts and bit checks, as opposed to simply twisting the data bus.

The rate of injection depends on the vehicle state. Self-checking can consume more time-spatial use of the available hardware outside of an ABS control enabled state. Less time is allocated for controller testing during computation intensive states to ensure that real-time requirements of the system are satisfied. An attempt of maximum hardware re-usability for the above test features is targeted at less than 40% area overhead. Note that self-checking is not merely BIST on start-up. It is part of the design so additional area can be allocated to the circuits implementation.

Figure 5.7 indicates the blocks involved with the continual monitoring of the ABS enabled control section. The *test_time_generator* determines when the pressure control loop hardware can be employed for self-checking operation. When the *test mode* is enabled, the corresponding test blocks are permitted to perform their respective functions. For instance, the *V_test generator* will generate sample velocity values to be injected into the *pres_cntrl loop* block. The values are part of a predetermined list stored as constants on the FPGA. Subsequent to traversal of the entire list, an *end of cycle* signal is initiated to indicate completion and permit the accumulator block to verify the results.

The *V_inject pulser* block determines the duration of time for the velocity value to be held so that a complete cycle of pressure state transitions can transpire, and results can be properly accumulated. The *test_value accumulator* accumulates all the threshold values, and time duration calculations. Aliasing can occur, but not due to overflow since all inputs are of the same length and the bit widths were designed to accommodate all max values with adequate margins. One extra feature of the accumulator is that it will also generate sample input deceleration values in order for the *pres_cntrl_loop* block to function correctly and undergo state transitions. The simplest case would be a direct feedback path, taking the calculated value, incrementing it and returning it to indicate that the respective threshold or time expiration value has been attained.

The testing cycle was designed to be time multiplexed (on the shared hardware) with regular operation of the ABS enabled controller circuit. Hence if test mode is interrupted, it will resume its state of operation sometime later with the correct test velocity value

last injected into the *pres_cntrl_loop* block. Similarly, when the regular mode of operation is interrupted, it too, will resume operation with its continuous sampling of inputs and calculations. Such *context switching* is possible due to the individual storage units for the respective modes of operation: test mode owns its own registers for data accumulation and state recording, and regular mode had its own velocity storage and calculation modules, acceleration and slip generation units providing accurate and updated inputs to the hardware resources.

5.2.3 Simulations

VHDL simulations validated ABS functionality and timing. Control system testing necessitated generation of input signal sequences, as sequential logic testing procedures were utilized. A succession of alterations on the inputs is required to fully test functionality as a controller application monitors events in time, operates with current inputs and past states, and must consequently respond in time. Waveform viewers visually aide such tasks.

Careful VHDL modeling when simulating to account for eventual synthesis was required. Apprehension that VHDL semantics execute in zero time, while only *wait* statements consume time is necessary to preempt false simulation interpretation. While simulation verifies functionality, timing cannot be assumed. For some synchronous circuits and FPGA implementation, pre-layout simulation sufficed whereas for some asynchronous blocks, re-simulation with back-annotated values was required.

Figures 5.8 and 5.9 exhibit the transitions through all the ABS pressure controlled states during an emergency braking maneuver. Synchronization between various states (blocks) is illustrated by correct state changes which result from event occurrences either external, such as a sudden braking or release of braking, or internal, such as time expiration. State dependent calculations are also triggered, creating thresholds for subsequent comparisons which may trigger internal events. The numbers appearing in the timing diagrams reflect the normalized integer values representing ABS variables such as ROM data and address, slip values, angular acceleration, calculated thresholds, and timer (counter) states.

The controller begins in a startup state, proceeds to *normal* operation when a velocity is detected, and enters *braking* when the brake pedal is pressed. When a critical deceleration

valued is measured to be larger than the threshold, "HI_THRESH", then the *ABS_enabled* vehicle state, along with the *high_hold* pressure controlled sub-state become active enabling their corresponding computational modules. All four pressure states are transgressed verifying the functionality of the threshold calculation and timer units.

5.2.4 Timing Considerations

The accuracy of Synopsys versus Xilinx timing is quantized in Section 5.3.1. Timing models in FPGAs account for such values as: set-up times, hold times, propagations delays, clock to output delays for the logic blocks, and routing delays for the interconnects. Traditionally, simulations use these local temporal values to generate global timing values which in turn are analyzed by the designer to verify that the specified constraints have been met. Two sets of values, post-layout and pre-layout are obtainable through the tools. For FPGAs the need for back-annotation is questionable and not all-together necessary. Realistically, back-annotation of post-layout timing delays is vital for accurate simulations. ABS timing values were compared before and after layout, proving this point, as will be seen in the next section. However, the differences were not extremely large, and with programmable hardware the need for this added simulation can be avoided, particularly for completely synchronous designs and contrary to ASIC circuits where such back-annotation is a necessity. However, for strict asynchronous designs on FPGAs, it is proposed to import the post-layout timing values into simulation to verify both functionality and timing.

Determination of the maximum clock speed and critical path are areas where circuit timing must be well approximated. The *XACT PPR* utility which took the ".xnf" netlist and partitioned the logic, placed the circuits into CLBs and IOBs, and routed the interconnecting signals is one such tool. Timing results, which include critical path analysis, individual path traversals for multiple clock circuits, and maximum clock speed potential, obtained after the *XACT xnfppr*, are close to what can be achieved in the laboratory environment, and sufficient to begin embedded system testing. Synopsys' timing model is conservative and thus good for estimations, but merit more accurate FPGA technology dependent timing values, and routing information from Xilinx.

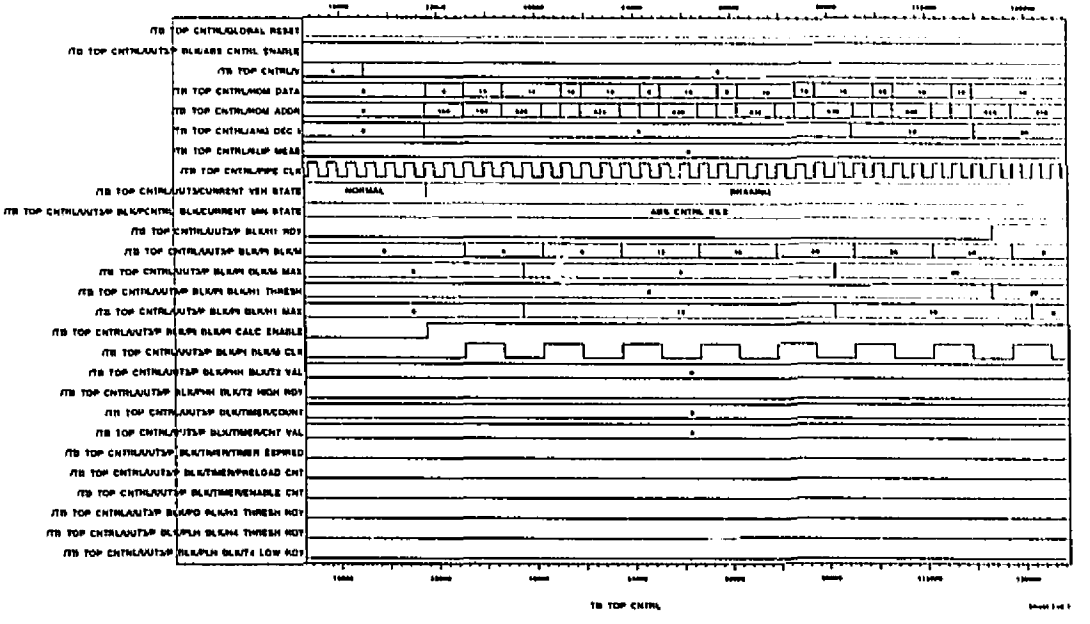

Figure 5.8: Simulation results: Timing diagram of start-up and Normal to Braking transition of 4-phase ABS control loop.

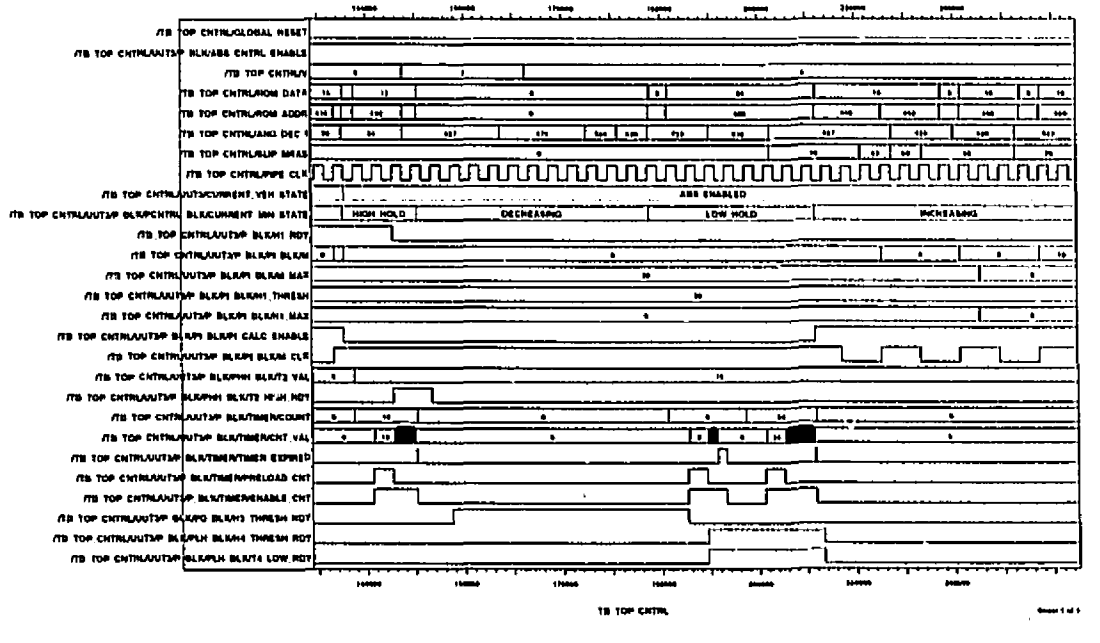

Figure 5.9: Timing Diagram cont'd: *high hold* - decreasing - *low hold* - increasing transition of 4-phase ABS control loop

5.3 Synthesis

Once desired functionality was defined, and longest critical paths were accounted for by constrained clocks and specified circuit timings, VHDL files were *read* and checked for syntax errors, I/O pads were inserted, clock pads were defined, and the design was subsequently compiled producing a CLB-IOB design. Design iterations followed, leading to the accumulation of evolutionary data, highlighting the growth of the hierarchical design.

5.3.1 Implementation Specifics and Results

VHDL design, simulation and synthesis for the ABS controller were performed using the CAD Framework from Synopsys. Design compilation was accomplished using the Synopsys FPGA Compiler. The XACT Development System from Xilinx was used for PPR and bitstream creation for downloading circuit programming directives onto XC4010s. One single XC4010 FPGA was not large enough for the entire ABS controller even though numerous global theoretical approximations were attempted. Circumventing the hardware limitations, required foremost, design partitioning as shown in Figure 4.2, and subsequently varied design strategies, optimizations, further approximations using MATLAB simulations for confirmations, and the addition of an external ROM. Primarily, effort was made to accommodate both the *Master_FSM* and the *Pressure regulating 4-phase loop* onto a single FPGA. Gradually, with improved synthesis techniques and adjusted design strategies, the slip calculation with its integer multiplication was included in the single Xilinx FPGA.

The remaining calculations and control logic could be synthesized onto a second FPGA, or with the advent of the new XC4025, it is expected that this single chip will accommodate the entire controller design. It remains the choice of the designer whether to implement the design on one or more FPGAs, and whether to wait for new hardware or employ "off the shelf" technology. There remain many ways in which a synthesis tool could be improved, especially with respect to circuit partitioning in the gate count range of 10-20K, and chip selection. Currently, we must accept exploiting tool capabilities manually with iterative attempts on circuit partitioning. So much as it were within Synopsys's grasp to select Xilinx FPGA parts, based on gate counts, the ultimate decision must should rather embody broader considerations such as FPGA vendors and their product lines and cost/performance

tradeoffs.

The design flow involved two paths of result accumulation. One path followed incremental design formulations for only Synopsys tools, while the other used the results of the first and applied FPGA vendor specific tools producing more accurate timings and layout information. Imperative at any stage of VHDL synthesis is boundary optimization, i.e. the flattening of blocks across borders so that more compaction and optimizing can be done during synthesis, and more specifically, mapping. When the blocks were joined, ungrouped and boundary optimized, the resulting circuitry occupied less CLBs and area. Depending on the number of sub-blocks a circuit may contain, the area utilization can almost be cut in half if the design hierarchy is flattened and optimizations are made across block boundaries.

The results for synthesis of the circuit using Synopsys' compilation tools, the **Design Compiler**^{TM} and the **VHDL Compiler**^{TM}, and its Xilinx libraries are captured in Table 1. The modules built were designed and synthesized in chronological order as they appear in the table, illustrating the accumulative design flow. The area values provide comparative descriptors as opposed to quantitative ones and correlate almost linearly with chip utilization. For all the timing analyses, an input clock was specified with a period of 200 ns, or a frequency of 5MHz. However, this and the FPGA logic block timing values, which Synopsys utilizes for its delay calculations, are also approximations. The slack time is determined by subtracting either an 8 or 6 ns set-up time and the maximal path delay through the circuit. The IOBs reflect the number of ports on the top level entities of each module and do not provide indications as to their relative sizes.

Design Description	CLBs [Util.%]	IOBs	Area [Unitless]	Hard Macros	Max. delay arrival time	Slack time[ns]
Master_FSM	11	52	64	0	68.30	123.70
Pres_cntrl_block	207[52%]	112	320	5	110.20	83.80
Self-Testing Pres_cntrl_block	292[73%]	108	362	8	118.20	74.80
Top_cntrl_pres	216[54%]	107	324	5	114.40	77.60
Self-Testing Top_cntrl_pres	309[77%]	108	418	9	123.20	68.80
Top_pres_slip	290[72%]	108	398	9	131.40	61.60
Self-Testing Top_pres_slip	385[96%]	109	494	10	140.20	51.80

Table 5.1: Synopsys' implementation results synthesizing different ABS partitions/modules.

The *Top_cntrl_pres* block includes both the *Pres_level_cntrl* and *Master_FSM* blocks and occupied only 54% of the FPGA. With test circuitry added, the CLB count went up to 309 CLBs or 77% utilization, understandably so, and attempts to route the testable design were successful. The testing circuit added a timing delay of between 9 and 10 ns and a 20-24% area overhead to the circuit. These values are conservative coming from the synthesis tool's approximation of delays before actual routing and exact interconnect delays were known. With space remaining on the FPGA, the next step was to incorporate additional blocks into this FPGA design partition until we fill the utilization quota. We define this as the point at which the CLB quota is at its maximum from logic and routing. One such block contains the slip calculation logic. With the *slip_calculation* block added, the CLB count reached 385 with 96% utilization. As will be mentioned below, after using the customized layout tool, this module was also routable once its logic was mapped to the FPGA logic blocks.

With the FPGA vendor tools, precise layout and timing results could be obtained. The synthesis tool translated the design to a file format (.xnf) acceptable to the layout tool. The XACT *XNFmerge*, *PPR*, and *XDelay* tools[3] were used to generate more accurate design implementations and timing analyses with the inclusion of more precise interconnect and routing information. The results of applying the XACT software onto the .xnf file produced by the Synopsys *Design Compiler* after synthesis can be seen in Table 5.2. Timing analyses from *XDelay* were more accomplished and exhaustive so that all of Clock to Pad, Pad to Setup, Clock to Setup, Clock to Clock (same edge and other clock edge), Clock to Setup (same edge and other clock edge) timings for each clock or clock-related signal were calculated, resulting in a higher accuracy and confidence in the resulting timing delays. The critical path was identified from the maximum over these values and this determined the *fast*, sampling and processing, clock speed. Both the *medium*, and *slow* clock circuits exhibited critical paths of comparable parameter timing to the system (fast) clock so that dividing the clocks for our sub-state computations by 4 and 32 respectively is acceptable and produces sufficient slack.

[3]Registered trademark of Xilinx, Inc.

Design Description	CLBs [Util. %]	IOBs [Util. %]	Hard Macros	Critical Path (max.)	CLB Feedthrus
Pres_cntrl_block	201 (50%)	137 (85%)	5	115.6	41
Self-Testing Pres_cntrl_block	285 (71%)	70 (43%)	8	155.9	51
Top_cntrl_block	204 (51%)	108 (67%)	5	123.7	29
Self-Testing Top_cntrl_block	294 (73%)	109 (68%)	9	138.8	51
Top_pres_slip	268 (67%)	108 (67%)	6	136.0	40
Self-Testing Top_pres_slip	370 (92%)	109 (68%)	10	146.2	62

Table 5.2: Comparison of synthesis results for testable and non-testable chronologically designed ABS Models

The above timings are indicative of limitations on the system (fast) clock which is used for global system synchronization. The slack time can be calculated by subtracting the above timing values from a given clock period given in nanoseconds. For a comparison with the previous table, a value of 200 ns can be used. The above timing values take setup time into consideration. Additional information was available from the PPR tool with respect to CLB utilization: some CLBs were used for direct circuit implementation, while others, *feedthrus*, were used for routing in addition to the interconnect network. Feedthrus, thus offer incite into the actual LCA utilization and potential density improvements. As is, using a base clock of 7-10MHz, a slower pipelined clock of 1-2MHz, and a threshold calculation phase of 1/32MHz, produce adequate response times of 0.032 ms which satisfies our real-time requirements. Furthermore, the critical path produced a slack of 25% in the timing analysis, providing reassurance that alternative or additional routing can be tolerated within this margin and still satisfy the necessary controller response times.

The testable circuit added a 22-25% area overhead (CLB count), close to the above, and a 10-15 ns timing delay, somewhat more realistic than that of the pre-layout results since it is known that at least two extra multiplexers lie in the critical path. This is rather high but it can well be tolerated since (i) it is part of the design and serves to fulfill a reliability specification, (ii) the test vectors are included in chip, even if presently there are only 16, and (iii) in control-type applications, it is customary for hardware duplicity to provide the redundancy needed to verify circuit operation, so in comparison the overhead here is small.

The difference from Table 5.1 and Table 5.2 in timing and area exist due, in part to different algorithms which map primitive gate representations to the FPGA CLBs and IOBs,

in addition to imprecise routing information, and library component module representation used by the Synopsys' tools, a deficiency which the Xilinx' XACT toolset does not exhibit. With synthesis tools like the Design Compiler and the VHDL Compiler, worst case analysis is often used and the timing results tend to be over-compensated. In general, the timing differences between pre-layout and post-layout for the Xilinx FPGA fell in the 6-17% range, while the CLB utilization differed by at most 5%. Hence fairly reasonable approximations for implementation can be made with the more generic tools, however final validation still requires the technology dependent layout tools. In fact, sufficiently accurate timing was achieved from technology specific tools, without back annotation, so that the next stage in the design phase was the programmation of the FPGA and consequently prototype testing. Hence, the importance of collaboration between synthesis tool vendor and the FPGA architecture vendor becomes apparent to shorten the design cycle to hasten the testing of the realized design on a a programmed FPGA.

The timings are rather high for the *Pres_cntrl_block* as compared to the *Top_cntrl* block which contains one more internal block. During the routing stage, PPR experienced difficulties with the *Pres_cntrl_blk*. We attribute the lack of routing efficiency to the large number of I/O connections as all of the entity inputs and outputs were translated to ports, which in turn had to be bonded (as defined by the experiment with respect to comparisons for synthesis). The *Top_cntrl* block had less I/O ports in its top level entity description as most connections between the incorporated modules were internal so that the number of external signals translated to PORTS and I/O pads was considerably less. In general, the IOB count is only an indication of how the block connects to the outside world and are not valid for comparison of self-testing versus non self-testing circuits. The area of routing does increase with the number of I/O pads required since additional nets must be allocated to connect the internal signals and entity ports to the IOBs. A key factor in FPGA synthesis is to remember that the process is algorithmic and not optimal. At times a small change in code, can drastically change the area and timing values or dramatically affect the routing. These problem areas are very difficult to locate and are not always easy to solve. Awareness of such possibilities is imperative, even if the solution is to "haphazardly" change the VHDL code while retaining the same desired functionality.

Alternative design methodologies, design partitioning, FPGA vendor types, and encoding

strategies, with perchance built-in-self-test with internal scan cells and JTAG implementation, to name a few, are areas of future research. Due to non-availability of scan cell models for the Xilinx cells, a version which implements scan technology was not synthesizable. The awareness of the added overhead of BIST and the programmable nature of the chip were sufficient to discourage the inclusion of this type of chip testing facilities in the implementation. However, B/S does merit future consideration, as its focus is not only on single chip but rather board and system level testing, which is a must even if the chip itself can be reprogrammed. As a further note for expansion, it was determined that the Field Programmable Interconnect Components (FPIC[4]) and Field Programmable Circuit Boards (FPCB[5]) from Aptix for fast wiring and interconnection could be beneficial for the prototyping of the aggregate ABS controller, until larger FPGA chips are available.

5.3.2 VHDL Synthesis and Tool Support

Two intelligent *design-for-synthesis* techniques lie with *trial and error* and *modularity*. Meeting constraints is an iterative process, especially in VHDL where their direction specification is not straightforward. One approach is to try several compiler options, i.e. exploit the synthesis tools to get diverse results from which the most favorable one can be selected. The is true for "all" levels of synthesis from logic up to system where different module entry formats, ordering, and encoding along with variations in the ordering of the compilation utilities and their directives (such as command line arguments or special in-line coding directives) will result in different circuit realizations. The ABS/Synopsys case study resulted in several convictions for pragmatic compilation. With Synopsys, specific blocks such as FSMs can be optimized with the *FSM compiler*, FPGAs can be optimized with the *FPGA Compiler*, and testability features can be added with the *Test Compiler*[6]. Designing small, modularized blocks, synthesizing them individually with different optimization facilities, and subsequently connecting them and resynthesizing them produces better results than synthesis of a flattened design.

Straight sequential VHDL behavioral code tended to produce excessive combinational

[4]a registered trademark of Aptix Corporation

[5]a registered trademark of Aptix Corporation

[6]all registered trademarks of Synopsys Inc.

logic and a somewhat asynchronous design with occasional indeterminacies. Manipulation of logic gate and register connectivity is, on the other hand, time consuming. Fragments of each entry format, behavioral and structural, were employed to achieve quick and efficient synthesis. Once the desired area and timing was achieved, the VHDL description was kept, otherwise more structural features were brought in. Hence, achieving the desired response times and area is an iterative procedure. Refinements for circuit improvement, such as real-time constraints placed on clock(s), input and output points, the addition of area constraints or more RTL level constructed elements can be employed if the resultant timings and area are unacceptable. Additionally, ensuring that certain encodings are already explicit, such as control logic into FSMs as illustrated in Appendix B, also breeds efficient realizations.

Implementation directives in design entry and compilation phases can be employed to communicate design constraints to the synthesis tools. If this is not sufficient, the designer will have to intervene and direct some of the synthesis stages. Of the VHDL hardware compilers tested, resource sharing across processes was not performed. If it is known that two separate processes perform the same functionality and they will never be run concurrently, the designer can arrange to have one single process be accessed by multiple blocks or processes. In the ABS design this can be seen with the timer module, and its mutually exclusive operation in different vehicle states. Using a structural approach, one instance of the timer was necessary implicating additional multiplexor circuitry for its sharing. Tree versus cascaded implementations are not inferred naturally by the tool. Care mus. be taken when encoding in VHDL to convey this type of information. Pipelined designs are not automatic. Optimizations are done on the combinational logic only, register retiming has not yet made its debut. The designer must understand what makes the synthesis tool infer combinational and sequential circuits from VHDL code. Even at a higher behavioral level of circuit description such circuit types can be conveyed through the *process* or *block* descriptions. Further reflections on "tight" FPGA designs which deserve observance, can be found in Chapter 6.

In summary, commercial synthesis tools are not yet that far advanced to produce a design equivalent to a user entered schematic. VHDL synthesis may not be sufficient for constraint strict designs and some manual intervention is vital for circuit realization. Additionally, care must be taken when optimizing, some circuitry may be discarded, and some ports or nets may be deleted producing an incorrect circuit. Once again we re-emphasize the need for

designer surveillance during synthesis.

5.3.3 FPGA Design Specifics

As FPGA resources lack slightly in density and timing, more attention should be taken by the synthesis tool and the designer during circuit implementation. A partial list of clever design approaches, many of which require special encoding or compiler directives and most of which are *area conservers*, is presented. The list will continue to evolve as FPGA architectures improve, synthesis tools better interpret VHDL yielding better architecture exploitation, the applications directed towards them diversify, and additional simplifications/approximations are found.

- Power of two implementation: in calculations, and integer ranges.
- FSM realization: explicit VHDL encoding of state transitions for control logic for best extraction.
- Data path width versus approximation: larger more accurate data path may require ROM usage, and can avoid the instantiation or creation of operational units **(*, /, exp, ln)** and the inaccuracies which may result from operations on them.
- LUT conceptualization: may save time with usage of fast RAM as LUTs.
- Explicit encoding to utilize built-in clock and global reset circuitry, through either block instantiation or compiler directives, in the event where the synthesis tool does not automatically instantiate such elements.
- Architectural specific features: It is imperative to be aware of the FPGA technology features and ensure that they are being exploited. Exploit technology specific architectural features such as the LUTs, RAM cells, registers, tristates, buffers, multiplexers, global reset circuitry, global routed signals, etc. Some FPGAs are multiplexor rich, for example Actel and Xilinx, and for multiple drivers of a signal, *multiplexer* usage may be preferred over *tristate* implementation due to the difficulty of the additional routing required in their (tristate) instantiation. One particular case with Xilinx stems from the CLB structure supporting explicit gated clocks. Synopsys' VHDL Compiler

currently takes VHDL code which infers a gated clock and transforms it into a flip-flop with feedback. Not exactly the desired implementation when dealing with FPGA technology. The inherent architecture of a *logic block* accounts for a gated clock, and hence the synthesis tool must be capable of exploiting the FPGA features while allowing a high level description such as VHDL. The circuitry is usable, built-in and functional so it may as well be exercised, even if it must be forced.

- Sequential features: It is also essential to know the flavor of the registers/flip-flops of the FPGA. The programmable logic blocks will often predetermine the allowable encoding. Recall, inferences of asynchronous or synchronous load or preload, synchronous with asynchronous reset/set, etc., latches can be controlled by the VHDL description. Figure 5.10 depicts a mandatory encoding alteration specific to the Xilinx FPGA. An asynchronous set or reset circuit required transformation to a synchronous set/reset as Xilinx CLBs only support synchronous loads. Note however, that simulation will permit all of the above statements whereas synthesis will flag component availability, and the beginnings of technology dependent VHDL are apparent.

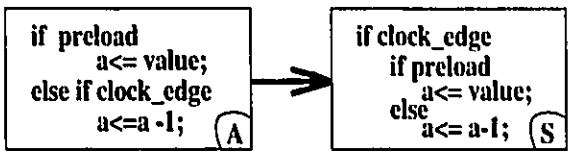

Figure 5.10: VHDL transformation from asynchronous code to implementable synchronous sequential code.

- Synchronous design: Synchronous designs are by far the most synthesizable, and predictable, and it is advisable to design with such a style in mind regardless of the level of specification. FPGA architectures contain large numbers of clocked registers which cater to synchronous conceptualizations. While one clock is preferable, often multiple clocks are needed, particularly for controllers. In such a case child clocks should be created on chip and they should be derivations of the parent input clock (external clock) using a generically encoded clock divider.

- Asynchronous design: If the design is asynchronous, a fast clock can be employed to synchronize aspects such as the input sampling, subsequent calculations, and result generation commonly present in controller type applications, avoiding circuit hazards. However, hardware permitting, the clock rate must be sufficiently high to trigger the circuitry quickly enough to respond to the input rate and eventual generation of asynchronous output signals, and perform the necessary computations. If physical limitations prohibit a complete synchronous approach, asynchronous encoding may be beckoned. Synthesis of such VHDL, not merely on FPGAs, often produces unpredictable circuit implementations. Unless carefully designed and simulated, final behavior of synthesized asynchronous VHDL code will not be known, and even then glitches and race conditions will most surely result. Furthermore, deterministic behavior will not be guaranteed should the design traverse a migration path to an ASIC, as timing is not equivalent between the two technologies.

- The types (behavioral and structural) or levels (logic, RTL, abstract behavioral and system level, etc.) of encoding can be varied in VHDL. Iterations are commonplace, with the synthesized version which best meets design constraints being chosen, either by the tool or the designer. The tool will carry out iterations of its own recourse, but if not satisfactory, manual intervention is necessary.

A brief summary of some design and implementation considerations:

- VHDL descriptions: If encoding is behavioral (and to some degree structural), it is advisable to become aware of the synthesis tool capabilities, learn the FPGA architectural features, obtain a reasonable understanding of FPGA synthesis (partitioning and mapping) algorithms (or heuristics), and visualize the hardware. Complete abstraction deteriorates FPGA realization efficiency. With structural descriptions, only instantiate FPGA specific components if technology dependence at the design entry level can be accepted and ASIC migration is a small possibility.

- FPGAs are built with repetitive structures, hence try to organize, if possible, the circuit correspondingly. If a synthesized module produces an ingeniously implemented circuit (upon reflection of parameters such as CLB or IOB counts and timing results or upon

viewing schematics with *design_analyzer*), the format of the code should be retained. If possible, the parameterization of such well-constructed blocks should be attempted with the help of *generics*.

Noteworthy of mention regarding FPGA technology in general: there are features and design styles which should be utilized just as there are those which cannot be supported. Fitting a VHDL design to different FPGAs will not result in the same design realizations, timings and parameters, some of which appear in Tables 5.1 and 5.2, and are certain to generate new measurement parameters and criteria.

Chapter 6

Design and Synthesis Methodologies

After studying several high-level synthesis techniques [46], [38], [39], [40] producing RT-level descriptions, VHDL encoding strategies, FPGA architectures and capabilities, and performing an ABS case study, it was determined that many of the heuristics and synthesis systems have not been directed to the commercial FPGA domain. Two major stumbling blocks were found. One deals with theory, its realization and application, while the other centers on the type of application being computation intensive. In theory, insufficient application of the large quantities of research (papers, texts) has been directed towards FPGA architectures with their structures and limited resources. Larger logic blocks, unlike primitive gates, maintain particular configurations which synthesis algorithms should cater to. As is, the abstraction of the synthesis algorithms towards a decomposition of a circuit into control and data flow and then the reduction to an unlimited number of registers and busses will not necessarily find optimal results for FPGA implementations. Such techniques are more directed to ASICs. Secondly, the case study chosen required more than straight ALU operations, for example logarithms and exponentials in the ABS threshold calculations. Such equations would need to be written differently, and in simpler behavioral format to be acceptable to some of the proposed synthesis systems. In fact, for the VHDL Compiler from Synopsys, the encoding was more RT-level than behavioral, as this was the only alternative to fit the

controller design onto the programmable hardware. To summarize, this study recommends the inclusion of standard HDL pattern definitions, more optimization features particularly for these HDL patterns (such as for FSMs), better resource allocation where resources can be primitive logic gates or the unique building blocks of the FPGA, and resource sharing capabilities and their detection. This chapter mentions a few of the most notable FPGA design hints and strategies.

Though much research has extended into the domain of high level synthesis, tools have yet to efficiently implement some of these algorithms from a VHDL design entry format accounting for FPGA architectures. Simply porting of ASIC tools for FPGA implementation will not produce optimal realizations. At the time of this experimentation, often, only the simplest form of heuristics could be expected from a synthesis tool such as a generic VHDL compiler. FPGA specific synthesis tools are needed yet in these, localization to hardware types can be both a strength and an hindrance. Ideally, a generic VHDL compiler for design portability which is many-hardware focused would be most beneficial. Some of the most obvious measures are taken while selecting hardware resources, but their localized approaches bread small scale solutions. Ideally one would like to see globally efficient scheduling of operations, resource sharing, and appropriate hardware selection (each of which have received extensive research). Additionally, while VHDL compilation produces predictable synchronous hardware, asynchronous implementation, already difficult from a design viewpoint, is still in its infancy from a synthesis standpoint. Much work remains for successful matching of an asynchronous circuit's functional and timing specification to both pre- and post-synthesis results through simulation. Once again, embodying asynchronous design techniques and research into synthesis tools remains to be exercised.

Through the implementation of the real-time controller, various design hurdles were overcome, and in the process, application specific, tool-oriented, and VHDL encoding methodologies were developed. Defining practical design methods and synthesis procedures bestows a pragmatic, as opposed to purely theoretical, engineering anchorage. Methodologies can be formulated from regularities in the design cycle, re-used and updated when theory changes or transgresses application boundaries. The following sections highlight mechanisms for design within specified constraints, exploitation of architectural features via synthesis tools, improving VHDL encoding etiquette, and comments on commercial tool capabilities/limitations.

6.1 Well-Constructed Procedures for FPGA Conceptualization and Implementation

Just as 10 years ago, from a simple study into design methodologies for the control parts of microprocessors [42], regular structures, time and facility of design, modular architecture approach, appropriate CAD tools were all deemed mandatory to handle the complexity of future circuits and to ensure optimal solutions using automatic analysis. Similar needs, tasks and goals are reflected in FPGA synthesis, where time and facility of design carry more importance than area, and regular structures in both the design models and the selected hardware must be correlated to aid in decreasing global PPR procedures. This thesis aims to show that the best correlation exists among FPGA architectures geared for specific design methodologies and classes of applications.

preferred Unless the task of design entry is amongst one of specification refinement (such as was the case for the Telephone Answering Machine Controller (TAM)), peripheral unit description for simulation purposes[1] and, or a test bench (where the use of the *after* statement is indispensable), then only synthesizable VHDL should be used. With synthesis as the ultimate target, simulation of VHDL designs to test for functionality and specification refinement, can benefit further stages in the design flow if constricting synthesis rules for encoding and compilation are considered up front. If possible, selection of the FPGA type is recommended first-off, followed by technology specific simulation using a synthesizable FPGA-VHDL subset, then circuit realization. However, in some situations it is desired to refrain from vendor choices until that latest possible so that alternative options can be examined and compared.

The proficiency of the synthesis tool as evaluated by the resulting circuit implementation, was determined through a series of steps: VHDL encoding, compiling (synthesizing), and viewing the resulting architectures and output files all of which generated CLB count, area, hard macro, and timing information. Both tool and designer took part. There will always remain features which are better exploited manually than with the synthesis tool, and to a small degree, vice versa. Currently manual intervention is necessary for dense designs.

[1]The modeling of the ROM was behavioral and used a large array with simple indexing to mirror read operations. Due to its size it could not fit on the FPGA, so its encoding was less strict.

Whether a synthesis tool can be created to remove this dependency, currently no, but an initial step is to offer better tool-hardware integration, so that more FPGA features could be exploited. Ideally, as little manual intervention as necessary is , but the capability to exert some control over block selection, placement and routing to maximize device speeds and densities must remain.

6.1.1 FPGA-Specific Hardware Considerations

A well known conceptualization and realization tactic is to employ *hardware re-usability*, both in time and space. Both are easily implementable in VHDL and on FPGAs. Time re-usability involves multiplexing units which seek to control the same hardware block. Spatial re-use refers to straightforward entity block copies with a unique set of of input signals driving the input ports and output signals being driven. FPGAs have repetitive hardware structures to support spatial re-use and sufficient multiplexer elements to support time re-use.

Currently FPGA architectures are fairly regular. The core logic blocks in the array and on the periphery are identical. However, certain augmented features such as: specialized I/O buffers, clock and reset circuitry, carry logic, decoders, tristate buffers, along with the unique programming mechanisms and interconnect networks, etc., further characterize the individual FPGA. Even now, memory capabilities are being integrated within some of the FPGA structures (Xilinx) and more tristate capabilities are being incorporated into the individual logic blocks (Atmel). Room for feature development, architectural expansion, and creativity exist and are needed as design requirements and application areas are continuously evolving. In addition to pondering over application-specific synthesis, consideration should be given to Application Specific FPGAs (ASFPGAs). Atmel designs FPGAs for datapath applications with its partial reconfiguration on the fly and abundant registers and tristate buffers for efficient implementation of bus and data-flow functions. A recommended next step for this thesis would be to realize the ABS design on Atmel FPGA technology and compare the density and performance capabilities with that of Xilinx technology. Additionally, with the partial chip re-programming option, self-repairing circuits could be designed and implemented complementing the existing self-error-detection circuitry.

Vendor logic blocks (LBs) could be customized to the different application areas and

design styles, and diversity in the LBs could be introduced as well so that the array of elements becomes heterogeneous as opposed to homogeneous. Experimental evidence [23] has indicated that several heterogeneous architectures are more area-efficient than the best homogeneous ones, a viable alternative, and innovation. Section 5.3.3 lists some strategies to exploit FPGA architecture.

6.1.2 VHDL Encoding Styles

VHDL provides abstraction and ease of design modifications. Altering the design, and making incremental changes in VHDL are less arduous than with schematic entry. In addition to the two approaches for intelligent synthesis mentioned in Section 5.3.3, a third became implicit: top-level manual synthesis. The designer should not ease up on hardware implementation considerations when synthesis tool aides are available. They serve to reduce the design time, not to remove the thinking process. The VHDL designer must still keep in mind testing procedures and design methodologies, and strategies such as scheduling and resource allocation. One must "think hardware" as "he or she" writes HDL code, and if a logic architecture is known that would yield a good implementation, "communicate it to the tool" [36].

Writing synthesizable VHDL code differs from encoding for simulation only. Even though the specified functionality has been attained, the realization may not be acceptable. Dependent on the mapping library chosen, and the design constraints, the VHDL to hardware translation will vary. The current process is *iterative*. When results are favorable, they are kept, otherwise another encoding, such as more explicit FSM or RTL level elements is attempted often resulting in more structural level or technology dependent VHDL. Favorability is often detected by analysis of a generated schematic, something which was to be avoided through an HDL design entry. Visual inspection is often easier than netlist or code viewing and less cryptic. Hence, a valid case study conclusion using a synthesis tool such as the Synopsys VHDL Compiler, is that avoidance of low level considerations is not yet possible if both ASIC and FPGA hardware is to be exploited to its fullest. A path for direct designer manipulation and override must always exist, no matter how high level the synthesis tool input is.

Just as with schematic capture, the *best* designs are created from the lowest form of technology components, where best is defined in terms of area and timing. A comparative analogy applies to VHDL as an alternative design entry mechanism. The lower the description, i.e. structural VHDL using gates and simple primitives, the better the area efficiency and timing for programmable logic. The higher the level, moving up towards RTL and ultimately to behavioral, the less area and time efficient the design realization becomes. Regrettably, the duration of the design cycle moves in the opposite direction with the complexity of logic primitives used in the design process.

The manner in which VHDL code is written, program structure, encoding, modularization, affects synthesis. For example, control can be attained implicitly with the knowledge of what type VHDL creates combinational logic, and what form produces sequential logic. Encoding and implementation of finite state machines has received much attention, and now an automatic synthesis path is being served by the research community. FSMs are the backbone of control circuits. Appendix B lists a cleanly, synthesized VHDL description of what was determined to be the best way to implement FSMs on FPGAs. In the ABS case study, one hot encoding offers beneficial circuit realization due to the register rich nature of the Xilinx FPGA, where each CLB has two registered elements in addition to the possibility of a programming a wide variety of combinational logic. In fact, it is our belief that writing VHDL descriptions for FSMs can be standardized, so that a tool can easily decode the input description. Another option is to use a specialized compiler, as Synopsys does, which can focus on FSM extraction. The use of an FSM compiler is suggested for controller systems to best cater to FSM implementation since a tool like this could sample various state encodings for circuit minimization on the FPGA architecture in question. The advantage of application specific synthesis tools is self-explanatory. With system modularization and the affiliation of different tools with these individual modules, global synthesis can become specialized. A small designer overhead is involved to include the tool-associations but the resulting implementation will be well directed.

Local resource sharing within VHDL process statements and single nested blocks is possible whereas global resources sharing, intra-process and inter-block statements, is not yet entrenched in commercial tool products. However, not all sharing, even at the sub-process level will be exercised as seen with the following example: Writing the following code,

```
v_offset := (V*32);
   IF ((current_min_state = increasing) OR (current_maj_state = braking))THEN
   ..out_addr <= addr1 + v_offset + m;
```

results in two instantiations of adders whereas the following code:

```
v_offset := (V*32) + m;
   IF ((current_min_state = increasing) OR (current_maj_state = braking))THEN
   ..out_addr <= addr1 + v_offset
```

results in one. Careful encoding, or *manual synthesis* has optimized the circuit. Integrating these skills into a synthesis tool should be unquestionable.

The main disadvantage with behavioral synthesis, is the need for much tighter constraints to accompany the VHDL description, leading to a need for particular encoding styles and standards. Often the resulting circuit is not necessarily what is desired, does not match timing specifications, and may not function correctly. Behavioral encoding does *not* produce glitch free implementations unless a design is completely synchronous. As mentioned in [53], a design synthesized with complex logic driving the gate of a latch rarely works. Something such as *IF (A=B) THEN* will not work in asynchronous designs if A and B are complex types such as integers or bit vectors. The comparison is made at low levels and each bit-pair of the vectors will not necessarily respond simultaneously. Once again, hardware design concepts cannot be neglected.

6.1.3 Design Problems: FPGA Solutions

Many of the previous commentaries apply not only to FPGAs but to the process of automatic synthesis in general. It is acknowledged that deficiencies exist with VHDL compilation tools and often this can be reduced to the problem of design space exploration. If unlimited time is allowed, the optimal solution can be found, however the designer cannot afford to wait for eternity. Many problems are theoretically solvable, but practical solutions are the ones in demand. A few general *design for synthesis* pointers which generate more efficient realizations are listed.

- Learning the hardware is useful to ensure that synthesis tool generates efficient designs.

- Writing explicit VHDL code is beneficial. FSM structures are well studied and their encoding can be cleverly synthesized when written correctly. Latch and register inferences are also manipulatable from VHDL descriptions.

- Grouping constructs, VHDL blocks, and VHDL expressions attentively can elicit architecturally optimized realizations.

- In designing synchronous circuits, one should avoid writing VHDL code which infers asynchronous sequential logic. Certain asynchronous encodings are usable, but attention to their effects on implementation and consequently FPGA hardware knowledge will be a prerequisite. Section 6.1.3 on asynchronous design and handshaking provides more details.

- Generally, a single clocked design is the ideal choice for circuit design. It is safe, simple, structured, easily implemented and predictable. Multiple clocks may be required, in which case the designer must take extra precautions in encoding and simulation. Event driven simulators do not always perform as expected, and their operation must be clearly understood.

- While usage to tool optimizers is imperative, their results must be verified manually to ensure continual correct circuit realization (applicable outside the realm of FPGAs as well). Synopsys provides such tools as the *FPGA Compiler* and the *FSM Compiler*, both of which were used, the latter only occasionally.

- Tool directives must be mastered. If the VHDL compiler alone does not infer the correct FPGA features, then the compilation must be directed for the desired technology, so that technology specific features are optimally instantiated.

- If design portability across FPGA technologies, or future implementations are a potential, then customizations should be avoided. For example with the Xilinx XC4010, though all on-chip flip-flops contain a built-in clock enable, the version of the synthesis tool utilized could not instantiate the proper hardware and created a feedback path to the flip-flop inputs instead. In lieu of customizing the VHDL code to use the built-in clock enable, it was decided to leave the code as is and accept the results knowing that in future versions of the compiler the correct hardware features would be exploited.

- If the final technology is known, and portability can be sacrificed, then lower levels of design and "customizations", such as hardware instantiation in VHDL, particular synthesis (compiler) options, and more structural entity descriptions, are alternative design options when constraints are tight.

- Designer synthesis techniques, referred to as *manual synthesis* can be invoked by the following methods and approaches should the resulting circuit implementation prove unsatisfactory.

  - Resource sharing: time and space multiplexing.
  - Design small modules, ones that fit compactly in a CLB(s).
  - Manual partitioning and modularity. Plan and organized circuit with the size and structure of the FPGA in mind. Create individually optimized modules for those circuit parts which are time/area critical.
  - Employ tree versus cascaded structures.

- Solution to *bus conflicts* or *multiple drivers on a signal* is unearthed most easily through tristate buffers or multiplexers. VHDL *bus resolution functions* are not synthesizable unless they are of the *WIRED_AND* and sometimes the *WIRED_OR* type. Often this may not suffice. As the synthesis tool will not correct such a problem, it is up to the designer. In such a case the implementation is then application specific. For bussing of wide signals, multiplexers should be employed, as they are readily available and reside in the CLBs alongside the CLBs. For single signals, tristate buffers are suggested and can be instantiated but they do add routing overhead, since they are external to the CLBs, and do not exist in byte or word aggregates. The *FPGA compiler* version used, could not instantiate tristate buffers, so it could not infer one over the other based on selected integer lengths of the signals in question.

A few summarizing key points are worthy of mention. Though the front end of the design phase involves a VHDL compilation, and a synthesis tools exist, one must still think hardware. The viewpoint that VHDL is technology independent is a myth for any realistic design. If synthesis and a technology is chosen, then the likelihood of the designer customizing

code is unquestionable. Acceptance of technology dependent VHDL will result in better circuit implementations.

Synchronous/Asynchronous Behavior

Asynchronous design is delicate as events happen independently from a system clock. Multiple clock interaction, two communicating processes synchronized to separate clocks, sampling haphazard input data at relatively high frequencies, or handshaking between two processes are all examples of asynchronous elements. Describing these events in VHDL, automatically synthesizing them, and attaining the desired functionality is not always immediately possible. Asynchronous design principles are specific to the synthesis tool and to the technology. Tool specifics, particularly VHDL synthesis subsets and encoding styles, and technology limitations must be reviewed before design. For example, the Xilinx XC4010 technology would require asynchronous set/preset flip-flops to assist asynchronous design realizations. Without their availability, work-arounds had to be found, and VHDL encoding was restricted as was illustrated in Section 5.3.3.

The ABS design process in this thesis' case study, explored several asynchronous methodologies but repeatedly encountered solutions which required synchronizations with a clock or strict monitoring of handshaking (ack-nack) protocols. A *wait* statement can synchronize asynchronous elements to a system clock, supervise handshaking, and provide control flow. Its use for general synchronization, control and management over the asynchronous control signals is highly endorsed. One approach was to always synchronize communicating processes to a fast, global clock, through higher level state machines as seen in Chapter 5 with the *Master* and the *Pres_cntrl* FSMs, also listed in Appendix B. With respect to handshaking protocols, Section 3.3 gives examples of encoding styles for VHDL description of inter-block communications. The timer block interacting with the FSMs and the two pressure-holding subblocks is indicative of inter-process communication, whereby the wait for clock edge was replace by a wait for an enable signal or an event generated during one of the states. Asynchronous events were translated to "enable" signals, and either internally synchronized to a clock, or externally synchronized through another process, one of the two main FSMs. Appendix B lists the VHDL encoding, indicating process interaction during ABS operation

of the above mentioned blocks, and methods of global clock synchronization.

Other techniques used to time, synchronize and control events was the use of clock divider theory. On one hand events could be staggered along several clock cycles dependent on the application needs. Analogous to a pipe-lined design approach, this strategy permits event sequencing and may solve some synchronous or asynchronous timing issues. Alternatively, waiting for multiple occurrences of changes of a signal, such as a clock, can produce a clock divider. Nevertheless, synchronization of individually clocked processes had to be done explicitly, and further stresses the RTL encoding style required with contemporary synthesis tools. Additionally, FPGA usage favors design where all synchronization should be done on chip. Ideally one clock is preferred, but if multiple clocks are needed, on chip clocks dividers are advised to ensure logic conformity.

Dependent on whether a process is constructed synchronously, with a global clock, or asynchronously with enable signals inferred from external or neighboring process events, the preparation and interpretation of the simulation should reflect the design methodology.

6.1.4 FPGA Test Philosophy

Several of the different stages of testing and and mechanisms of test inclusion were discussed in Section 3.4. During construction, a design is not complete and can not be branded functional until tested. During operation, the design must remain fault-tolerant or possess compensation alternatives or system shut-down capabilities. Controllers constantly monitor and manage systems, and are required to be continually self-checked for correct functionality. Methods to perform this efficiently and within the area and time limits of FPGA technology were considered. A renown test philosophy would be to utilize more than one controller chips and compare the outputs in time. Should one of the controller chips fail, then another could resume master operation with some sort of *back-up in effect* flag or the system would shut down with an *error* flag. With such multiplicity, the burden of additional chips, added comparators, some verification circuitry and control logic must be tolerated in addition to newly created datapaths since the operations are multiplied. As mentioned earlier two or three-unit systems are commonplace. To avoid this area overhead, the proposed *cycle-stealing, hardware time-sharing* solution was utilized, successfully implemented and extension

to other control applications is encouraged.

Hardware sharing with value injection as a concept can elicit a generic application specific testing procedure, so long as the hardware performance can accomadate the real-time needs of the conglomerate system. In the ABS controller application, where the timing requirements were in the order of human response times, timings in the order of *ms* are sought. FPGA technology can be clocked in the order of tens of MHz, so that some leeway remains when the cycle stealing is enforced. Implementation of on-line sampling for testing purposes is therefore a workable preposition. However monitoring the results of operations and ensuring a number of fixed sequences will require some customizations which requires knowledge of the design itself.

Ideally we want continuous monitoring of the controller circuitry. The moment a failure is detected somewhere in the circuit, its operation must either be adjusted or halted, as incorrect controlling can be hazardous. For an ABS, a failure could hinder overall braking, precipitate brake failure, and in turn cause an accident. However during constant monitoring of the system, normal functionality must not be sacrificed, and the controller *must* continue to supply real-time responses to its actuators. Currently, the ABS specification mandates fault-detection and partial fault-tolerance. A design improvement would encompass the implementation of theoretical ABS dynamics which is directed towards more exhaustive fault-tolerance and coverage. However, the percentage of thorough *fault-processing* will remain limited by the FPGA technology limitations.

The Test Compiler (TC) from Synopsys is capable of adding in scan chain elements, connecting them together, and adding a few extra pins (IOBs) for scan chain serial input and output. A Xilinx library which furnishes the scan chain equivalents for the flip-flops was not available. The possibility of writing proprietary ones in VHDL and then including them during the Test Compiler's component inferal phase exists. It was decided that these features did not merit inclusion as the chip is programmable and chip density after partitioning was greater than 90%. The TC is also capable of affixing state of the art testing facilities known as Boundary Scan, JTAG, to the circuits periphery. Already embedded in some FPGA architectures, this feature can help create more global or board level testing procedures for a controller application which subsists in a larger system and interacts with other chips possessing the same style of testing. Due to previously mentioned unavailability

of a particular Xilinx library, such a feature could not be implemented, but it is of future interest.

6.2 FPGA Synthesis Issues and Proposed Solutions

Certain hurdles exist for FPGA circuit realization using synthesis tools, and HDL entry methods. Todays synthesis tools have a number of limitations. To cite a few, they require many iterations or test cases; they do not provide much feedback, or control, over placement and routing; and they may produce results that fall short of those attainable from schematics. As stated in [25], even successful synthesis users say that the dream of technology independence – the idea that one can write a single high-level description and simply re-target it to any device family – cannot be achieved today.

Both the synthesis tool and the FPGA device architecture must be well understood to obtain favorable results. Just as hardware structures and logic blocks can be customized to specific applications one can also customize hardware to synthesis tools and HDLs. As mentioned by Rose, [23], two angles for direction exist: "Architecture must be synthesis friendly, and synthesis must be architecture smart".

An immediate limitation of synthesis tools materializes in terms of device performance and area efficiency. The key to using FPGA synthesis successfully is knowing where it works and where it doesn't. For example with respect to FPGAs it was noted that logic synthesis for random-logic elements such as state machines, address decoders, lookup tables and straight boolean type equations is very efficient, but synthesis for arithmetic or dataflow functions necessitates supplementary improvements, as depicted with the ABS case study.

Library support systems merit investigation as they are the back bone of a synthesis system and its inherent heuristics. Technology specific libraries will obviously produce better realizations, however device independence is becoming increasingly important as numerous FPGA vendors exist and continue to enter the market with new devices. However, frequently, the libraries supplied by the synthesis vendor are not sufficient as they are too generic and will not be optimized to exploit the technology's features. Abiding by traditional approaches, as mentioned in [61], the libraries can be considered analogously to hardware *off-the-shelf*

components, where each FPGA vendor is responsible for guaranteeing their programming or implementation in the most optimal way on their FPGA families. Nevertheless, some non-standard components must reside in the library to cater to particular architecture families. Settling on a standard set of hardware elements, such as the Library of Parameterized Modules (LPM) [57], across FPGA vendors with the option to easily integrate vendor and user specific components would aid optimization of synthesis algorithms and help make designs portable across vendors. LPM defines higher-level modules for synthesis tools to pass to back end tools. An attempt at their integration into design aid tools has been materialized through XBLOX^{TM} from Xilinx. Hence to support technology-transparent design methodology and to obtain the best circuit implementations, a generic FPGA library of blocks (which all FPGA vendors would agree upon and support) along additional vendor specific blocks (which can be accesses through hooks in the synthesis system) should be reviewed. A more in depth discussion of synthesis support systems and library creation can be found in [56].

Furthermore, from an algorithmic perspective, it is proposed that technology specific synthesis utilities deserve more observance, for superior circuit realizations. Some vendors such as Synopsys and Exemplar have already taken steps in creating specialized optimizations for FPGAs: Exemplar catering to many FPGA types while Synopsys catering to Xilinx and Actel FPGAs through its *FPGA Compiler*^{TM}. Measurement of Exemplar's system merits was, unfortunately, not possible, however both systems dealt more explicitly with RTL and logic level synthesis and were lacking in higher levels of design entry and behavioral synthesis. If they could process some higher levels, the realizations will not be optimal or tuned to application specific features. Hence, as an underlying theme, synthesis tools should become more design style specific. Subsequent sections will highlight certain application specific synthesis steps which could be integrated into a design style specific compiler to further optimize the tool-automated circuit realizations.

With respect to synthesis and design size, further limitations surface. It seems 1000 gates is about as high as one should dare to go in a single pass [25]. The idea is to synthesize small blocks and then pull them into a hierarchical schematic. Using reduced blocks permits certain passes of the synthesis tools to succeed in some local optimizations. Varying the numerous compiler directives and utilizing the specialized compilers can also be used at this granular level (for the individual modules) resulting in overall (global) circuit optimizations. Different

circuit modules may be better served by application or design-style specific algorithms which can be served by the specialized tools.

For many of the above to achieve fruition, more regard to the need for a tighter coupling between synthesis and layout (placement and routing) must be alloted. A synergy between tool and technology must exist with adept communication protocols. This would involve the tools understanding the hardware features, providing queries, processing feedback and passing timing constraints to layout and bringing accurate wire delay estimates back into synthesis, i.e. back annotation.

6.2.1 FPGA Structures and Design Methodologies

Exploitation of the FPGA architecture via synthesis algorithms is lacking. Most FPGA families have dedicated hard macros for handling arithmetic functions, but often synthesis tools pass collections of gates, not hard macros, to the placement and routing tools. It is usually only the technology specific (vendor specific) tools which pass hardware specific macros and much more technology specific information. In this design experiment, Synopsys was able to pass one type of hard macro, an ADDSU16 bit unit, which is the extent of the technology specific library available.

Chapter 2 introduced the significance of regular structures and distinctive design procedures for particular application types and hardware resources. With controller applications instantiations of Moore and Mealy type FSMs are abundant. One such case study proved the efficiency of their implementation on Xilinx FPGAs heedful to register abundance and software capability to handle known structures (VHDL models in this case) and exploit hardware. Datapath operations like addition, subtraction, control, data transfer are recognized and synthesized to units such as adders, ALUs, registers, multiplexers, bus drivers and busses, by the better synthesis tools. The better high level synthesis tools will extract FSMs to control the multiplexers, registers, bus drivers, and enable certain processes with correct input data during the correct state and on the correct clock cycles unlike RTL level synthesis where conscience FSM encoding is required. However, in either case, once the application traits are recognized, their implementation should fall onto hardware which caters to their kind. Our FPGA served for the most part, well but was lacking in the area of drivers, busses,

and fast and easily usable memory banks.

Most datapath functions are synchronous in nature, matching their timings and operation with the system's clock or controller. Availability of programmable logic devices which provide for easy clock distribution, predictable timing, multiplexer, register, and bus driver availability, in addition to bus layout features would be highly beneficial. Additionally direct mapping of ALU operations, counters, and register files with fixed higher level blocks would be favorable for FPGA synthesis.

6.2.2 Design Entry and VHDL

High level design languages provide a standard medium for communicating design data between vendors and customers, in industry, and among designers working on various aspects of the same project. VHDL is a hardware specification language, a recognized standard, and it encompasses many interesting features, such as hierarchical level descriptions, behavioral descriptions and user defined attributes. It is a straightforward and versatile method of entering design specifications and if the code is generic enough migration to an ASIC is faster.

VHDL is currently more acceptable in the industry environment, so much as a corresponding versatile library system should cater to both the hardware platforms and the HDL for optimum synthesis performance. The synthesis system which cultivates the common elements of the entry and implementation of a design, is superior.

Repeatedly, the steps undertaken by a synthesis tool to realize an HDL encoded circuit are either not sufficient or they produce undesired effects. In such a case *manual synthesis* can be attempted to alleviate unwanted circuit realizations. One occasion arises when an addition operation of unequal size operands (different integer ranges) has to be carried out. Synopsys automatically expands the smaller addend to the length of the longer addend, and infers a large adder. An implementation using an adder the size of the smaller addend, then adding the carry of this operation to the remaining part of the larger addend, and concatenating the two proves to be faster, and less area consuming[2]. Moreover, at times, addition is required

[2]If the operands are comparable in size this is true, however of one is less than one half the other such methodology may not provide better realizations, but will not result in a poorer implementation

which disregards the upper bits, such as a circular counter or an address generator. Inferal of the smallest adder to correspond to the smaller addend is then desired. Additionally better concatenation and slicing features could be employed. Defining an integer range of 0 to 255 should not infer a 9 bit adder for a potential sign or carry bit when area is critical in FPGA designs even though it may be acceptable for ASIC implementation.

The creation of a synthesis language would be beneficial such that both a synthesis paradigm and functional components become standardized through a specification language. It would be generic yet parameterizable and would fit with all levels of synthesis. For example, just as with VHDL, there are reserved words for language constructs describing the hardware, perhaps elements such as CLOCK and RESET could become fixed. At any level of a design implementation you often need a clock to synchronize your circuit no matter the level. The term CLOCK would then imply at say a high level, synchronize this part of the circuit, and at the low level it could be describing the clock signal which are feeding into the flip flops. Similarly with a RESET signal. In fact, more than one could be defined (reset2, reset3, etc) with similar functionality at the various levels of circuit description.

There are additional problems when employing an HDL, and synthesizing to FPGA hardware. The following points are noteworthy:

- Loss of technology independence. One has to manipulate the VHDL so that the targeted architecture flows smoothly.

  Synthesis does accelerate circuit realization once a methodology is in place, however better performance estimates during the early stages of synthesis are desired. For the 20-50MHz frequency range, synthesis provides satisfactory performance for parts running in this ranges.

- Better timing control is needed. Synopsys predicting interconnect delays. Often too optimistic results or too conservative results are obtained. So it was decided to ignore delays and concentrate on area, and use the timing from the XACT PPR and *xdelay* tools.

- A synthesis tool often required additional help to meet design requirements. Occasionally one has to take the VHDL down to a very low level to force the synthesizer to

generate desired realizations which meet specifications. If timing or area specs are not met, several iterations are needed to clean up blocks which consume to much space or reduce critical paths.

- Certain VHDL constructs must be avoided which generate inefficient designs. "It comes down to a technology sympathetic design" [22].
- The designer is not removed from technology concerns as the importance of locating constraints within the FPGAs themselves proved vital. It was not possible to write VHDL code without knowing the target technology.

6.2.3 Synthesis/Layout Suggestions

The Synopsys to XACT layout was not a direct technology driven path. CLB clustering, and FPGA CLB implementation before the *replace_fpga* command, will not necessarily be preserved. A change in the current design flow would be beneficial. Ideally a netlist format which could accept logic block elements specific to the FPGA would be desired. The synthesis tool resolved the circuit algorithmically into technology specific components, yet the transfer to the layout tool required translation to logic gates. CLB and IOB clustering was somewhat lost in the process, and design constraints could not be passed to the layout tool. Much of this is known, and consequently the synthesis tool vendors and the hardware people are working to improve the tools and hardware to form a true synergy and reformed tools.

In Chapter 2 we introduced the synthesis and layout platform from which it can be deduced that a need for a tighter integration between synthesis front-end design tools and FPGA layout tools arises. To bring out the full power of the technology [22], synthesis tools must be capable of passing hard macros to the placement and routing tools, and layout tools must readily accept constraints (timing, area, etc.) from logic synthesis or schematic entry to produce fast, dense designs with automatic placement and routing. However, some guidance and manual intervention whenever necessary should remain accessible. For example, FPGA pin-out placement capability is vital for prototype testing otherwise its surrounding test environment would continually require alterations (quite time-consuming). Additionally, better timing control on interconnect delays, could be accomplished if timing constraints could be passed to the layout tools and accurate wire delays could be brought back into

synthesis. Furthermore the preservation of logic element clustering as the design passes to the layout (XACT) tools should be enforced, and can be implemented with meaningful constraints being passed to the layout level from synthesis tool. Lastly, full incremental design, synthesis, and routing capabilities should be attainable. One thing missing from the Xilinx 4000 tools, is a re-entrant router, i.e. the ability to change a few logic elements without rerouting the entire chip. Synopsys has an incremental compiler but this is for synthesis, and it is more a time saver than a design tool.

An efficiently synthesizable VHDL subset which is technology independent, yet which contains language constructs specific to FPGAs, is also worthy of mention. The VHDL subset tested in this case study should be expanded, and uniformity amongst other synthesis tool vendors which support VHDL is also recommended. Common elements akin to circuit reset, signal passing and synchronization can be dexterously supported by keywords such as *clock*, *start*, *stop*, *reset*, *global_reset*, *state* , *edge* and etc, precluding the need for explicit and custom encoding prescribed by the synthesis tool capabilities.

Currently, the VHDL code must be manipulated for the target architecture. Some VHDL constructs produce inefficient designs and unwanted structures while others are illegal. Sometimes, the VHDL code has to be taken to lower structural levels to force the synthesizer to generate the desired circuits. It was noted that the VHDL Compiler from Synopsys was unable to instantiate certain hardware features automatically, the decoders, memory elements, and clock enable circuitry from VHDL descriptions. At these levels some technology independence is lost. Synthesis tools should target an approach which uses generic, technology specific, and user components effectively at higher levels of synthesis, as opposed to manually instantiating them which becomes just another form of schematic entry in words as opposed to symbols.

Ideally, using a generic VHDL subset and synthesis tool which contains device-specific optimization heuristics to provide efficient utilization of logic within the FPGA could avoid the above manual work. Optimization for the technology can also be performed for specific applications or design styles under the influence of dedicated compilers, directives, and libraries (as mentioned earlier). Just as an FSM Compiler exists, a new conjecture suggests additional tools: one such as a DP Compiler for a strict datapath VHDL encoding, a Pipe Compiler for implementation of pipelined hardware structures, or a MEM Compiler,

to best explore large data structure realizations. For each, optimal VHDL encoding can be performed knowing the particular VHDL constructs which favor certain design styles.

Additional capabilities sought encompass: timing-driven placement and routing, the ability to take constraints from synthesis, manual floor-planning capabilities, high-level macro functions, direct exploitation of specific hardware technology features. Deadlock detection, as is flagged by some software compilers, would be an interesting feature to have. It is more of a simulation wish but could alleviate time spent simulating and debugging a circuit where two hardware units are waiting for each other. Such an error occurs frequently in asynchronous designed circuits with handshaking protocol.

It is expected that the synthesis tool explore and adopt some of the available algorithms which are so favorably described in some of the literature. At the same time direct application of them to FPGA technology is desired. For example, due to the nature of some FPGAs, a global memory block (shared memory) versus local register (distributed memory) tradeoff should be explored in the respective FPGA's context. Xilinx in particular has flip-flops evenly spread out as they reside within the CLBS which are the regularly repeated structures of the logic cell array. This arrangement should be considered during memory element binding and allocation. Knowledge of the hardware can permit synthesis tools to formulate FPGA specific optimizing design strategies. Employing tools which use sophisticated algorithms which converge on the best solution, and a rules driven approach where requirements are set up front to reduce the amount of clean up required at the end will keep the FPGA design cycle to the fewest iterations possible. To further decrease prototyping time, the shortest time per iteration is needed, hence support for incremental design and synthesis capabilities would be quite practical.

A closer link between the hardware and the synthesis tools would contribute to the achievement of a fast and dense design which meets the designer's constraints. FPGA architectures are constantly evolving and an alliance would open a forum for design exchange between VHDL design entry and FPGA logic block structures so that more direct and efficient synthesis inferences are made. This would permit better exploitation of the FPGA specific hardware resources, and direct input of synthesis constraints into the layout tools. It seems this soon shall be achieved with the alliance of our two selected EDA vendors (Synopsys and Xilinx). As mentioned in [22], knowing where the architectures are heading, will

quicken the design of respective synthesis algorithms and hence the availability of these efficient tools. Some of these requests, improved support for Xilinx 4000 features, and ability to drive XACT with Synopsys timing constraints, and the ability to back-annotate technology specific logic and routing delays from XACT to Synopsys's simulator, are slated to come out within the next months according to [22], and their effectiveness is yet to be determined.

Timing Constraints and Validation

Timing driven analyses are imperative. The incorporation of timing constraints in VHDL is imperative. Synthesis of VHDL code with straight inclusion of *after* statements is not yet possible. An alternative would be to support a subset of timing constraints. One idea is to allow *after* statements at only certain points in the VHDL code. Adding in a timing parameter or a *timing map* along with the port definitions is another alternative, would provide some timing information for a block or process, or an entity description, respectively. Another alternative is to include relative timing techniques. This can be equated to the inclusion of single or successive wait statements among processes or within a single process. An analogous feature was introduced by Gutberlet et all [44] using the *after* statement to incorporate relative time into their synthesis algorithms. Wait statements with timing values which were multiples of a clock cycle were permitted and hence contained in the synthesizable subset. For example, a wait of 200 ns (a multiple of the main clock which is 100 ns) is synthesizable given that the base clock has a cycle time of say 100 ns. As an extension, the synthesis of VHDL containing multiple clocks should also be ensured. With such a strategy, it would be unnecessary to specify a cycle time and all the timing values as they are merely multiples of each other, and more optimization schemes could be explored within the synthesis system with respect to clock speeds.

For validation purposes and perhaps verification, automated timing analysis with the ability to compare the completed design against user specified requirements and report back on potential problems, would decrease the development cycle time. For a synthesis tool to adequately process design constraints and match them with the implementation results, all elements must be modeled strategically so as to provide sufficient coverage and testimony of this to acquire the designer's confidence in the tool's ability.

6.2.4 Functional Self-Checking

For the ABS application, driver safety is vital so fault detection must be a priority and the system must either tolerate faults, still ensuring safety, or shut the system down completely. The testing must be done on-line, which is possible with the cycle-stealing test procedures described, and ideally it should not double the size of its hardware implementation while providing a high degree of reliability. Furthermore, a proficient mechanism is sought to evaluate the merits of the self-checking circuit based on its efficiency of error detection.

It was assumed an FPGA is fault-free before programming, and that circuit verification for manufacturing faults has previously been performed by the vendor. Consequently, the testing procedure proposed in Chapter 5 is a functional test for the detection of field errors. The ABS controller is expected to function as an aid in emergency braking maneuvers without endangering the vehicle occupants, and if this functionality if not being met, the ABS controller must be halted from further operation. The necessary tools to evaluate such a functionality-testing procedure were not available, however a procedure to do so will be proposed. Based on the previous assumption, a standard procedure of comparing fault-free and faulty circuits can provide information as to the testing capabilities of a test circuit. If the test circuit can detect a difference in functionality, often measured by the circuit's outputs, between the two circuits, then that particular fault in the circuit is testable by the test-circuitry.

Injection of faults into the ABS circuit followed by resimulation of the circuit checking whether the functional test circuitry detects the fault and shuts down the ABS controller is one way to measure the coverage of the on-line self-monitoring mechanism. The evaluation process depends on the number of faults to be accounted for, and can be a lengthy process. To achieve the highest fault coverage one would have to consider all possible faults, i.e. stuck-at-0, 1, wire shorts, open circuits, delay faults, and multiple faults which can be combination of all the above. Realistically, as time to market is a central issue in the design cycle this is not possible, and testing of stuck-at-0 and stuck-at-1 faults is considered sufficient [4], [5]. With a list of faults, the next step is to find a set of vectors which cover 100% of the faults. Being a rather difficult task, a lower coverage can be accepted, the value of which would depend heavily on the application. Alternatively, for area sensitive designs, a given

set-size of test vectors can be specified with the coverage calculated correspondingly by the above method. If either measure is unacceptable, the testing circuitry, or vector set can be enhanced.

The selection and encoding of faults into a VHDL description is not the most feasible solution as the synthesis tool will not guarantee circuit implementations, and faults like the above must be added at lower circuit levels. However, this is not always possible with an FPGA implementation, for the exact low level architecture is not always known and the designer would find it difficult to inject faults on an "unseen" circuit. A viable solution then rests with the FPGA and the synthesis tool vendors adopting the responsibility of injecting faults into a circuit, and permitting the designer to evaluate her/his design and test methodologies. Some commercial tools support this functionality, even generating test vectors for the circuit under test. Alternatively, the designer can concentrate on functional tests which is the what the ABS self-monitoring circuit attempts to address.

6.2.5 Mapping onto FPGAs

High level synthesis requires algorithms to function at the various levels of circuit modeling. Some of the higher level methodologies were mentioned in Section 2.2.1, but ultimately the lowest level of circuit representation or modeling will pass through the technology mapping phases. Like ASICs, every stage of synthesis necessitates optimizations, but unlike ASICs, this stage is very particular to the FPGA vendor as fixed structures exist in lieu of primitive gate forms. Certain algorithms exist for mapping, Chortle-crf, Mis-pga, Amap, Xmap, [21] are no longer sufficient due to the number of additional special features which are surfacing on the newer FPGAs. The heuristics require modifications to cater to these specific features appertaining to each FPGA. Assurance that VHDL compilations and translations exploit the available hardware directly, instead of going through several design phases which convert VHDL to first standard logic and library components, is desirable.

The FPGA logic blocks are now designed to support multiple types of functionality. Often the combinatorial and sequential sections can be assigned individual and separate design blocks. Straightforward mapping of logic no longer exists and more technology specific algorithms are required. The move to pre-planned designed units containing an aggregation

of logic blocks offers exploitation of target architecture and potential for good synthesis.

Though clustering does provide a degree of efficiency, it still remains a problem to decipher the correct building block size, at what level technology dependence should step in, how to interconnect the aggregates, and at how to handle the interconnection of blocks with their varying levels of specification.

Chapter 7

Closing Remarks

7.1 Accomplishments and Results

The initial goals propagated research in this thesis to master:

1. Research into *application specific synthesis* and its merits. With FPGA synthesis, it is well worth considering both hardware and software application specific features and design methodologies to best exploit technology. Timing critical and higher density requirements demanded from FPGAs provide motivation to produce efficient and satisfactory circuit realizations from such synthesis tools.

2. Success in the implementation of a *controller type system using FPGAs* instead of the traditional microcontroller/RAM/software approach.

3. Introduction of a new method for *system reliability tests in controller applications* precluding the module multiplicity of previous methods. A self-checking circuit was realized with knowledge of design specifics, employed hardware re-usability, and provided a general methodology for area-limited controller testing.

4. Proposed evaluation scheme for the self-checking functional tests on the FPGA, which involves injecting faults and verifying whether the test circuitry detects them.

In general, with the use of VHDL for design entry, the performance will be substantially below that which is achievable through schematic capture. The resulting maximum clock

rate is design, designer, and encoding-dependent. Our design falls in the 7-10MHz maximum clock rates. Highly pipelined (i.e. registered) designs with minimum levels of logic between flip flops will run significantly faster. Testability, through cycle stealing, can be incorporated into the design without sacrificing the required system's response times. Xilinx maintains that real-time performance can be achieved if a design's requirements fall in the 20MHz area, and VHDL iterations are performed. Achieving 5MHz performance was possible without considering the architectural structures of the 4000 series, whereas several iterations were required to achieve the 7-10MHz range. For above the 12-16MHz range, more manual intervention and iterations in the synthesis process such as compiler directives, component instantiations, carefully encoded VHDL and RTL structured VHDL to efficiently program the logic blocks, may be required. To run at full 20MHz, it is estimated that a full structural approach in VHDL encoding would be mandatory.

From a marketing viewpoint, for approximately \$1K, not including tools, and hardware required to test the prototyped circuit, an FPGA (of high logic block utilization), and one ROM can be built to accommodate an ABS controller. With production quantities of greater than 1000, the cost can be reduced to under \$200. To ensure safety, inherent self-checking circuitry exists, however if all possible faults are to be detected, there is room for improvement in its design and evaluation. However with the FPGA technology, alterations are always possible, so that upgrades are untroublesome.

7.2 Conclusions

Success of an FPGA implementation for a real-time controller suggests its potential as an alternative to the software programming–microprocessor approach with added hardware performance. Rapid prototyping with FPGAs is quick and efficient. Great potential exists for *fine-tuning* embedded controllers within multi-variable, safety critical systems which are highly dependent on environmental side effects. For final implementation on programmable logic, technology dependent VHDL is a viable option to ensure satisfaction of both area and timing constraints and subsequently generate controllers with acceptable real-time responses. Real-time performance is easily achievable on FPGAs if the controller design requirements fall in the 10MHz area. Furthermore, self-checking features can be incorporated without

affecting the system's reponse times.

Controller testability for *reliability* was developed from the nature of the controller. Hardware re-use permits cycle stealing and on-line system checking with a smaller overhead than that found in traditionally implemented control systems (often over 100%). As controllers exhibit a certain subset of characteristics, this test methodology can be extended to the general case. A standard way to re-use hardware, add in the test injection circuitry and the multipexed inputs can be developed to facilitate the incorporation of the self-checking features for the general controller. However, though are for the testable circuit is reduced, 100% fault coverage of field errors is not guaranteed, particularly with such a functional test. Evaluation mechanisms of the functional testing accomplished by the vector-injection schema were mentioned in section 6.2.4, yet require additional research, as measuring fault coverage is not an easy task for on-line field testing of FPGAs.

Commercial synthesis tools add area and timing overheads during circuit realization in addition to the inherent architectural costs of programmable hardware. Hence a balance between performance issues, design time, design features and testability concerns must be found whereby the design becomes self-testable, real-time and concise. As FPGA complexities evolve, an HDL design entry method becomes a viable solution for quick designing. FPGA architectures are already vendor specific with certain architectures fitting in better with some design styles. It is expected that they will further "niche" themselves towards application specific FPGAs (ASFPGAs) due to more strenuous design constraints on the programmable devices, the growing design complexities which favor concise realizations, and the effectiveness of encoding customizations. Consequently, it becomes imperative to develop efficient VHDL synthesis tools which exploit the new and improving FPGA architectures, so that designs with specified real-time constraints can be implemented quickly, intelligently, and correctly.

Synthesis is not a complete solution but a productivity aid. It does not provide push-button results. Planning, regular structure extraction and creation, and a modular architecture style in order to manage the complexity of the current controller and to extend the design methodologies to future controller designs were strongly emphasized in this thesis. Synthesis merits analysis as the time and facility of designs are rising in criticality and allowance for less dense circuit areas is tolerable so long as circuit timing is achievable. For

asynchronous circuits, contrary to [37], it is not supported that the synthesized circuit will always be, that which was simulated, and it is suggested that these authors were refering to datapath-control dominated synchronous systems rather than to any general circuit.

As hardware design styles emerge and proliferate, complementary design aides must ensue. The ideal tool must be technology independent from the designer's viewpoint and technology dependent for the synthesized circuitry, producing optimal designs which best exploit the technology for a given design application or style. CAD tools must undergo continual evolution, playing hardware "catch-up". It is our opinion that the best synthesis tool will be the one which is written/designed for the hardware, and has the best techniques to exploit this hardware. It is proposed that better integration of tools and hardware will produce the best design-aides for the design engineer. What then remains is for the designer to learn the HDL syntax and its capacity for circuit design, understand the technology, learn the tool, and use all concurrently.

Although VHDL supports high level design entry, attention to the details of technology implementation and digital design methodologies cannot be avoided if a certain level of efficiency in enforcing constraining parameters such as timing and area are vital in the final circuit. With the present synthesis tools, VHDL design entry does not eradicate all gate level design considerations. If constraints cannot be met, the designer will be forced to resort to hand-crafting portions of the design. Nonetheless, a disadvantage of trying to make the design efficient by exploiting the hardware is that added details and accuracy are required in the original specification (design entry format will tend to be technology specific). Thence, though we advocate technology specific tools and in some cases, languages, the bulk of time spent designing circuits may just be shifting from implementation time to design and specification time, and learning of the constantly evolving tools. Nevertheless, with the mission to realize the circuit and facilitate the designer's task, respective tradeoffs will always surmount.

If ASIC migration is to follow an FPGA prototype, the intent must be reflected in the VHDL code, perhaps implicating certain restrictions. With the current tools, migration onto non-programmable hardware is supported for VHDL designers if the encoding is not technology specific (no special feature instances). Currently only functionality can be equated between FPGA and ASIC designs. Synthesis tools should provide some correlation between

the two technologies so that adequate foresight is gained from an FPGA design, especially with respect to timing. Proper timing models would alleviate compromises in timing validation, facilitating the migration. Ideally the synthesis tool should also automatically partition the design to fit on multiple FPGAs so the entire circuit's functionality could be simulated, implemented and tested, without restructuring and timing compromises.

Ultimately high-level and behavioral synthesis are envisioned. Exploring architectural alternatives, hardware/software partitioning, automated functional partitioning into memories, processors and buses, VHDL generation are undertakings of the former while scheduling operations into clock cycles, allocating and sharing resources across multiple cycles, and pipelining fall into the latter. In both, fast metrics/performance estimators and both manual and automatic scheduling, resource allocation and intervention in general, are desired, with a conjoint drawback that success in this area will come at the expense of less efficient hardware realizations. Analogous to logic synthesis where optimizations are sought across clock boundaries, VHDL synthesis should concentrate on optimizations across processes and entities.

As for future areas of study, more quantitative evaluation mechanisms for the functional testing schemes of controllers on FPGAs remain to be researched and developed. Additionally, the examination of alternative FPGA architectures, and commercial synthesis tools while preforming equitable design, synthesis, and testing steps for the comparison of circuit realizations merits further study. Circuit density, total area requirement, circuit timing, testability, fault coverage of test-circuits, reliability, etc..., can collectively serve as control variables for such a study. With the evolution of technology and tools, such a synthesis study will never terminate as the nouveautés require analysis, the gamut of application areas invariably grow and new evaluation criteria perpetually surface.

Bibliography

[1] K. Ogata, *System Dynamics*, Prentice Hall, New Jersey, 1978.

[2] C. H. Houpis, G. B. Lamont, *Digital control Systems: Theory, Hardware, Software*, McGraw-Hill, Inc., New York, 1992.

[3] K. Parker, *The boundary-Scan Handbook*, Kluwer Academic Publishers, 1992.

[4] P. H. Bardell, W. H. McAnney, J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, New York, 1987.

[5] P. K. Lala, *Fault tolerant & Fault Testable Hardware Design*, Prentice Hall, 1985.

[6] E. C. Yeh, C-Y. Kuo, P-L. Sun, "Conjugate boundary method for control law design of anti-skid brake systems", *Int. J. of Vehicle Design*, pp. 40-62, 1990.

[7] E.C. Yeh, G. C. Day, "A parametric study of anti-skid brake systems using the Poincare map concept", *Int. J. of Vehicle Design*, pp. 210-232, 1992.

[8] C-Y Kuo, E. C. Yeh, "A four-phase control scheme of an anti-skid brake system for all real conditions", *Proceedings of the Institute of Mechanical Engineers*, 1992.

[9] Bosch, "Braking systems for passenger cars",*Technical Instruction Manual*, 1992.

[10] DARCOM PAMPLET, *Engineering Design Handbook: Analysis and Design of Automotive Brake Systems*: Dec 1976, US Army Material Development.

[11] R. T. Fling, R. E. Fenton, "A Describing-Function Approach to Anti-skid Design", *IEEE Transactions on Vehicular Technology*, August 1981.

[12] L. Lind, "The Volvo electronic tractions control (ETC) system" , *Int. Journal of Vehicle Design*, 1983.

[13] B. G. Schulze and E. Lissel, "Anti-slip regulator system (ASR), *Int. J. of Vehicle Design*, 1987.

[14] Thomas W. Birch, *Automotive Braking Systems*, Harcourt Brace Jovanovich College Publishers, 1988.

[15] Actel Corporation, *Actel FPGA Data Book and Design Guide*, Actel Corporation, 1994.

[16] Altera Corporation, *Data Book*, Altera Corporation, San Jose CA, August 1993.

[17] AT&T Microelectronics, "Optimized Reconfigurable Cell Array(ORCA) 2C Series Field-Programmable Gate Arrays", *Preliminary Data Sheet*, AT&T Microelectronics, May 1994.

[18] Atmel Corporation, *Configurable Logic Design & Application Book: PLD · FPGA · Gate Array*, Atmel Corporation, San Jose, CA, 1994-1995.

[19] Motorola, Inc., "The Motorola MPA1000 Fine-Grain Family", *MOTOROLA Semiconductor Technical Brief*, doc: BR1340/D, REv 0, March 1994.

[20] Xilinx, Inc., *The programmable Gate Array Data Book*, Xilinx Inc, San Jose, CA, 1994.

[21] S. D. Brown, R. J. Francis, J. Rose, Z. G. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992

[22] R. Goering, "Users seek better FPGA layout", *Electronic Engineering Times*, January 1994, pp. 1, 35.

[23] J. Rose, "Field-Programmable Chips, Systems and Starships", Keynote presentation, *Proceedings, 1994 Canadian Workshop on Field-Programmable Devices*, Kingston, June 1994.

[24] B. Hold, P.C.B. Bhatt, V.K. Agarwal, "Rapid Prototyping and Synthesis of a Self-testing ABS Controller onto FPGAs", *1994 Canadian Workshop on Field-Programmable Devices*, Kingston, June 1994.

[25] Richard Goering, "FPGA synthesis offers benefits, but limitations remain", *Electronic Engineering Times*, April 11, 1994.

[26] R.J. Francis, J. Rose, Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", *Proc. 28th ACM/IEEE DAC*, June 1991.

[27] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, A. Sangiovanni-Vincentelli, "Logic synthesis for Programmable Gate Arrays", *Proc. 27th ACM/IEEE DAC*, June 1990.

[28] R. Murgai, R. K. Brayton, A. Sangiovanni-Vincentelli, "An Improved Synthesis Algorithm for Multiplexor-based PGAs", *Proc. 29th ACM/IEEE DAC*, 1992.

[29] R. Murgai, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Synthesis for Table Look Up Programmable Gate Arrays", *Proc. 30th ACM/IEEE DAC*, 1993.

[30] J. Kouloheris, A. Gamal, "FPGA Performance vs. Cell Granularity", *Proc of Custom Integrated Circuit Conference*, May 1991. pp. 6.2.1-6.2.4.

[31] J. Kouloheris, A. Gamal, "PLA-based Area versus Cell Granularity", *Proc. of Custom Integrated Circuits Conference*, 1992. pp. 4.3.1-4.3.4.

[32] S. Singh, J. Rose, P. Chow, D. Lewis, "The Effect of Logic Block Architecture on FPGA Performance", *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 3, March 1992, pp. 281-287.

[33] J. Rose, R. Francis, D. Lewis, P. Chow, "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency",*IEEE Journal of Solid-State Circuits*, Vol.25, No 5, October 1990, pp. 1217-1225.

[34] E. Sternheim, R. Singh, R. Madhavan, Y. Trivedi, *Digital Design and Synthesis*, Automata Publishing Company, 1993.

[35] B. Hold, P.C.B. Bhatt, V.K. Agarwal, "Rapid prototyping and Synthesis of and ABS Controller Using CAD Tools", *Proceedings 2nd IEEE Workshop on Real-Time Applications*, July 21-22 1994, Washington DC, pp. 151-156.

[36] P. Malardier, "HDL Programming For Synthesis", *Synopsys Methodology Notes*, pp. 33-50, Sept.1991.

[37] A. Stoll, P. Duzy, "High-Level Synthesis from VHDL with exact timing constraints", *Proc. of the 29th ACM/IEEE DAC*, 1992, pp. 188-193.

[38] D. Gajski, N. Dutt, A. Wu, S. Lin, *High-Level Synthesis, Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[39] Edited by P. Michel, U. Lauther, P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, 1992

[40] D.C. Ku, G. De Micheli, *High level synthesis of ASICs under timing and synchronization constraints* , Kluwer Academic Publishers, 1992.

[41] L. Hollaar, "Direct Implementation of Asynchronous Control Units", *IEEE Transactions on Computers*, Vol.C-31, No 12, December 1982, pp. 1133-1141.

[42] M. Obrebska, "Algorithm Transformations Improving Control Part Implementation", *Proc. ICCD-83*, pp. 307-310, 1983

[43] R. Ang, N. Dutt, "An Algorithm for the Allocation of Functional Units from Realistic RT Component Libraries", *7th International Symposium on High-level Synthesis*, pp. 164-169, May 1994.

[44] P. Gutberlet, W. Rosenstiel, "Interface Specification and Synthesis of VHDL Processes", *Proc. of the 2nd EuroDAC*, pp. 152-157, 1993.

[45] P. Gutberlet, W. Rosenstiel, "Specification of Interface Components for Synchronous Data Paths", *7th International Symposium on High-level Synthesis*, pp. 134-139, May 1994.

[46] H. Kramer, W. Rosentiel, "System Synthesis using Behavioral Descriptions", Proc. of the 1st EuroDAC, 1990.

[47] D.S. Rao, F.J. Kurdahi, "Hierarchical Design Space Exploration for a Class fo Digital Systems", *IEEE Transactions on VLSI Systems*, Vol.1, NO.3, September 1993, pp. 282-294.

[48] D.S. Rao, F.J. Kurdahi, "Controller and Datapath Trade-offs in Hierarchical RT-Level Synthesis", *7th International Symposium on High-level Synthesis*, pp. 152-157, May 1994.

[49] J-M. Berge., A. Finkoua., S. Maginot, J. Rouillard, *VHDL Designer's Reference*, Kluwer Academic Publishers, 1992

[50] V. Nagasamy, N. Berry, C. Dangelo, LSI Logic Corporation, "Specification, Planning, and Synthesis in a VHDL environment", *IEEE Design & test of Computers*, June 1992.

[51] D. Perry, *VHDL*, McGraw Hill, Inc,1991.

[52] Synopsys Inc., "VHDL Compiler Reference Manual", Version 3.0, December 1992.

[53] Synopsys Inc., "Design Compiler Reference Manual", Version 3.0, December 1992.

[54] B. K. Fawcett, "FPGA Development Tools: Keeping Pace with Design Complexity", *1994 Canadian Workshop on Field-Programmable Devices*, June 1994, Kingston, Ontario.

[55] Synopsys Inc., "4000-Series(Xilinx) Interface Using FPGA Compiler^{TM} Application Note", Version 3.0, Feb. 1993.

[56] B. Hold, V.K. Agarwal, P.C.P. Bhatt, "An Object Oriented Approach to an FPGA Based Library System to Support High Level Synthesis", *Technical Memo*, McGill University, June 1993.

[57] EDIF LPM Subcommittee, Electronic Industries Association, *EDIF (Electronic Design Interchange Format) Library of Parameterized Modules (LPM) Version 2.0*, EIA/IS-103, Electronic Industries Association 1993, Wash. DC.

[58] S. Trimberger, Xilinx, Inc. "A Small Complete Mapping Library for Lookup-Table - Based FPGAs", 2nd Intl. Workshop on Field-Programmable Logic and Applications, Aug 1992.

[59] R. Dekker, M. Litgthart, "Module generation for VHDL Synthesis", VIUF Proceedings, April 1993.

[60] S. H. Kelem, J. P. Seidel, Xilinx. "Shortening the Design Cycle for Programmable Logic Devices", IEEE Design & Test of Computers, December 1992.

[61] S. Kumar, J.H. Aylor, B.W. Johnson, W.A. Wulf. "Object-Oriented Techniques in Hardware Design", Computer, IEEE 1994.

Appendix A

ABS Theory, Notation, and Dynamics

A brief evolution of ABS dynamics follows. System dynamics are explained through mathematical equations which in turn depend on system variables and parameters. The following list of symbols is included to explain the variables used in the equations utilized throughout this section.

F_x	tire force between wheel and road (N)
g	gravitational acceleration (9.8 m/s^2)
I	moment of inertia of a wheel (kgm^2)
m	slope of friction coefficient versus slip curve
M	mass of the vehicle (kg)
r	rolling radius of the wheel (m)
U_i	brake torque change rate; increasing mode (Nm/s)
U_d	brake torque change rate; decreasing mode (Nm/s)
V	vehicle velocity (m/s)
S	wheel slip (ratio and %)
S_{opt}	slip at peak of F_x curve or T_e curve (optimum)
T_b	brake torque applied to wheel (Nm)
T_e	equilibrium brake torque (Nm)
ω	angular velocity of wheel (rad/s)
$\dot{}$	derivative with respect to time
$C_S S$	tire longitudinal stiffness
μ	friction coefficient
F_z	normal load on the tire (i.e. the weight of vehicle)

Control of events is largely dependent on system variables and dynamics. In the prevention of wheel locking, the relationship between the rotation of a wheel, its linear velocity and its adhesion to the road surface predominate. No slippage occurs only when a wheel

transmits no force in the horizontal plane – when the vehicle rolls freely in a straight line. As soon as acceleration, steering or braking of the vehicle transpires, the generation of the necessary horizontal force at the road/tire interface results in slip between the tire contact path and the road surface. The amount of slip depends upon the force transmitted, and in turn on the deceleration. A piecewise linear approximation of the tire-road characteristics is illustrated in Figure A.1, with the notation defined on the following page.

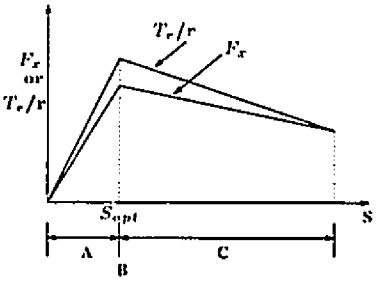

Figure A.1: Piecewise linear approximation of the tire-road characteristic

It is shown in [6], [7], [8], [9] that the braking force approaches its maximum value at slippage of 15%. Approaching the maximum braking friction coefficient value, the slippage increases rapidly, whereas after the peak the braking force declines considerably. The significance of this, is that while the vehicle is undergoing severe braking, the angular deceleration of the wheel is increasing as slip increases. Just as the maximum braking force is being approached, a maximum deceleration value is similarly being approached.

The first requirement for wheel lock prevention system is to detect excessive deceleration of the wheel, or wheels, concerned so that, as a second requirement, action can be initiated to reduce the braking force. Once the braking force is within the capability of the available adhesion to sustain, the tire will again grip the road, the wheel will accelerate and directional control will be restored. However, one cannot just remove braking completely or the vehicle will never stop. A further general requirement is for braking to be restored on an appropriate cyclic basis so that the deceleration approximates as closely as possible to the maximum theoretically available. Maintaining brake pressure exactly at the maximum peak friction coefficient (Figure A.1) is rather impossible, even if vehicle velocity was continuously measured and braking constantly monitored due to the dynamics of fluid in the hydraulic

braking system. More reasonably, the aim of an ABS is to encircle the optimum slippage point in the most proficient manner via a *limit cycle*.

Some elementary, yet fundamental terms for vehicle dynamics require definition. *Torque* is the product of a force and the perpendicular distance from a point of rotation to the action line of the force. It tends to produce a change in rotational motion of a body on which it acts. *Slip* is the measure of the sliding component during a rolling movement. As a wheel rolls under the effects of driving or braking forces, complex physical phenomena occur in the tire's contact with the road. The tire's rubber elements become distorted and are exposed to partial sliding movements, even if the wheel has not yet locked. The measure of the sliding component of the rolling movement is the slip denoted by $\lambda : \lambda = (V - V_t)/V$, where V is the vehicle velocity and V_t is the tangential (circumferential) wheel speed. Brake slip occurs as soon as the wheel starts to rotate more slowly than the wheel speed which corresponds to the driving speed [9].

Previous ABS systems operated with two-phase control, [6], [7], where the brake pressure is increased with a fixed rate until a criterion (P condition) is satisfied predicting the danger of the wheel lock-up; then the pressure is decreased until another re-selection criterion (R condition) is triggered when the danger of lock-up is averted. Typically the prediction and re-selection conditions selected for threshold analyses, as mentioned in [6], are: (P2): $-\dot{\omega}/\omega > k_2$ and (R4): $\ddot{\omega} < 0$ (deceleration). These conditions are considered both robust and serve as good threshold conditions for ABS dynamics [7]. To prevent the lock-up of wheels and further providing large brake force, ABS will modulate the brake pressure to produce a stable limit cycle around the peak of F_x curve (almost the peak of T_x curve as seen in Figure A.1) according to the criteria adopted in the P and R conditions of the control law of ABS.

The complete system involves velocity calculations which are dependent on the wheel velocities of all four tires. Taken from previous studies, [6], and [7], simplified equations of motion of the vehicle system for a single-wheeled model are introduced and form the basis of subsequent calculations.

The *vehicle model*, neglecting air resistance and the suspension dynamics comprises the

following two equations for the motion of the vehicle and the wheel respectively:

$$M\dot{V} = -F_x \tag{A.1}$$

$$I\dot{\omega} = rF_x - T_b \tag{A.2}$$

The *tire force model* is predominantly contingent on wheel slip. Wheel slip is defined as in (A.3) where $r\omega$ is V_t, the tangential velocity.

$$S = 1 - \frac{r\omega}{V} \tag{A.3}$$

The tire brake force F_x, under zero slip angle assumption (no centrifugal slip), is described by the following nonlinear equation:

$$F_x = \begin{cases} \frac{C_s S}{1-S} & \text{when } \frac{C_s S}{1-S} < \frac{\mu F_z}{2} \\ F_z\left[\mu - \mu^2 \frac{F_z(1-S)}{4C_S S}\right] & \text{when } \frac{C_s S}{1-S} \geq \frac{\mu F_z}{2} \end{cases}$$

According to [7], the value of μ on wet asphalt is given by the nonlinear equation:

$$\mu = \mu_0 exp\left(-\frac{pngVS}{100}\right) \tag{A.4}$$

where μ_0 represents the nominal friction coefficient at zero velocity, and png stands for percent normalized gradient which depends on the root mean square texture height of the road.

Differentiating both sides of Equation A.3 and substituting equations A.1 and A.2 to eliminate $\dot{V}$ and $\dot{\omega}$, the following equation for slip dynamics is obtained:

$$\dot{S} = \frac{r(T_b - T_e)}{IV} \tag{A.5}$$

where,

$$T_e = rF_x\left[1 + \frac{I(1-S)}{Mr^2}\right] \tag{A.6}$$

Note that $\dot{S}$ is zero when $T_b = T_e$. Thus T_e is called the equilibrium torque which is different from rF_x by a small quantity, as seen in Figure A.1.

A brief look at the *brake model* produces an additional set of equations. Using the assumption in [7], that no hysteresis and time delay occur in the brake mechanism, the relationship between the brake torque T_b and the brake pressure p in the brake cylinder, is simply assumed as a linear function:

$$T_b = K_t p$$

where K_t is a constant gain. Brake pressure dynamics can be approximated by a first order system. For the high switching frequency of solenoid valves, the values of dp/dt can be approximated to be constant for both increasing and decreasing pressure modes [40]. Differentiating both sides, the brake torque changes accordingly as:

$$\dot{T}_b = K_t \frac{d}{dt}(p) = U \tag{A.7}$$

where

$$U = \begin{cases} U_i = & K_{pi}, \text{a constant in pressure increasing mode} \\ U_d = & K_{pd}, \text{a constant in pressure decreasing mode} \\ 0, & \text{in pressure holding modes} \end{cases}$$

Thereupon, the nonlinear dynamic equations of the single-wheel brake system, from equations A.2- A.7 become:

$$\dot{V} = -\frac{F_x(V,S)}{M} \tag{A.8}$$

$$\dot{S} = \frac{r[T_b - T_c(V,S)]}{IV} \tag{A.9}$$

$$\dot{T}_b = \frac{d}{dt}(p) = U \tag{A.10}$$

Originally two stages were needed to control the vehicle braking. One to release the brake pressure, and the other to re-apply brake pressure both of which mimic the desired pumping motion. Kuo and Yeh [8] introduced two additional stages, the high and low pressure holding stages to better encircle the peak of optimum braking force, i.e. frictional force. The resulting four-sided trajectory of the controlled braking force can be seen in Figure 4.1 which illustrates the dynamics of the four phase pressure control loop.

For simplification, both in theory and implementation, the velocity was assumed to be constant during a few cycles of computation in the study. A set of equations developed for "the four phase control scheme of an anti-skid brake system for all road conditions" can be found in [8]. These were derived from the dynamics above, and were further simplified for implementation. The final set of equations created through assumptions and some approximations follows suit.

During the **pressure increasing mode**, and the vehicle braking state, the set of simultaneous equations in [8], which solve for system parameters associated with the threshold

decelerations, were reduced producing the following equations . The threshold H_1 is calculated for comparison with the vehicles deceleration to determine when the high-pressure holding mode must be triggered.

$$H_1 = h_1^* + \frac{1}{8} * h_1^* \tag{A.11}$$

$$S_1 = S_{opt} - \frac{k_1 V}{m^2}\left[1 - e^{-\frac{S_{opt} m^2}{k_1 V}}\right] \tag{A.12}$$

$$h_1^* = m\left[\gamma_1(S_{opt} - S_1) + \beta_1(1 - S_1)S_1\right] \tag{A.13}$$

In the **high-pressure holding mode**, time interval,T_2 must be calculated, the value of which defines the time to remain in this state and hold the break pressure constant in order for the system to pass comfortably over the *optimum slip*, the maxima in the tire friction coefficient-slip curve of Figure A.1. According to [8] T_2 exists between a calculable upper and lower bound. A method of evaluating a single lower bound and an offset is employed here and suffices to determine a valid time duration. Equations A.13, A.11, and 4.5 contribute to the derivation of T_2, the latter of which leads to the following equation for t_2^* after variable substitution. Recall, t_2^* is the value use to calculate the lower bound for a given road condition, while T_2^* is the maximum lower bound over all road conditions, and contributes to the value of T_2, which determines the holding time for the high-pressure holding mode.

$$t_2^* = \frac{-Vk_2}{m} \times ln\left\{1 - \frac{m\gamma_1(S_{opt} - S_1)}{H_1 - m\beta_1(1 - S_1)S_1/r}\right\} \tag{A.14}$$

From equation A.13 and A.11, H_1 is estimated to be:

$$H_1 = \frac{9}{8} m\left[\gamma_1(S_{opt} - S_1) + \beta_1(1 - S_1)S_1\right] \tag{A.15}$$

From which t_2^* becomes:

$$t_2^* = \frac{-Vk_2}{m} \times ln\left\{1 - \frac{\gamma_1(S_{opt} - S_1)}{\frac{9}{8}\gamma_1(S_{opt} - S_1) + \frac{1}{8}\beta_1(1 - S_1)S_1}\right\} \tag{A.16}$$

The H_1 (threshold value triggering high-pressure holding mode) dependency was removed via substitution and t_2 became a function of the velocity and road-condition ratio, $\frac{V}{m}$, and the slip, S_1. Similarly, S_1 can be calculated from the previous state (pressure increasing or braking) and is dependent on the ratio of $\frac{V}{m}$, so that one lookup table with indices formed from joining V and m, suffices to generate the t_2 and subsequently the T_2 values.

During the **pressure decreasing mode**, brake pressure decreases quickly so that brake torque, T_b drops below the rF_x curve, until a re-selection condition *R3* [8] is satisfied, that is when $rF_x - T_b$ is beyond a threshold value H_3, the low-pressure holding mode will be triggered. Moreover, when $rF_x - T_b > H_3$ the trajectory (brake torque versus slip) falls below the peak in the curve of Figure 4.1. From Equation A.2, this condition translates to $I\dot{\omega} > H_3$, so that the measured deceleration value is compared to the calculated value of H_3.

The determination of the threshold value H_3 is closely related to its effect on its successor state, the low-pressure holding mode. Consequently, two situations must be avoided: (a) a small brake force when H_3 is too large; (b) the trajectory becomes stuck on the T_c curve in region C, Figure 4.1, for positive slope m, which may yield a large slip value and undesirable directional stability. In [8], H_3 is determined to be $m(rMg)\Delta S$. In most ABS systems, the pressure decreasing rate is very fast, so that the slip value triggering the *R3* condition is proximate to the one leaving the high-pressure holding mode. Hence, ΔS is almost equal to the slip change in the high pressure holding mode, ΔS_2, which is the maximum allowable value for the slip change after passing over the peak of the F_x curve. H_3 is then chosen to be at least $(0.2)(rMg)\Delta S_2 = (0.08)(rMg)$ applying the maximum value of 0.2 for m in region C [8].

Unlike the high-pressure holding state, during the **low-pressure holding mode**, the pressure is held constant for a fixed time interval T_4, after which comparisons are made checking $rF_x - T_b$ with two threshold values H_3 and H_4. If $rF_x - T_b$ is less than H_3 the pressure decreasing mode will be turned on and if $rF_x - T_b$ is less than H_4 the pressure increasing mode will be triggered; otherwise the low-pressure holding mode will be active again for another time interval, T_4, until the next comparison. In this way, the trajectory is able to enter the region A, Figure 4.1 before switching to the pressure increasing mode.

Similar to T_2, the time duration of the low hold state was estimated to be $T_4 > T_4^* + \Delta T_4^*$. Calculation of T_4^* will lead to T_4, grace to MATLAB analyses. From [8],

$$T_4^* = \mathrm{M}IN\{t_4^*\} \quad for \ \ \mathrm{m} : 0 > \mathrm{m} > 0.2,$$

and t_4^* depends closely on H_4, giving rise to the equation:

$$t_4^* = -\frac{IV}{mrMg} \times ln\left\{\frac{H_4 + (T_c - rF_x)}{H_3 + (T_c - rF_x)}\right\} \tag{A.17}$$

After some substitutions and approximations, the low-pressure holding mode description is reduced to the following set of calculations, where some freedom is permissible due to changing precision requirements:

$$H3 = 0.08(rMg) \tag{A.18}$$

$$H4 = \frac{9}{16}H_3 \text{ or } \frac{5}{8}H_3 \tag{A.19}$$

$$t_4^* = -\frac{Vk_3}{m} \times ln\left\{\frac{H_4 + (0.85\alpha_1 I/(r^2 M))}{H_3 + (0.85\alpha_1 I/(r^2 M))}\right\} \tag{A.20}$$

where $k_3 = k_2 * r$.

This concludes the derivations of the ABS equations which were encoded in VHDL, synthesized, implemented on an FPGA and supplemented by an external ROM.

Appendix B

Examples of VHDL Code

A typical FSM encoded in VHDL follows suit. Its circuit realization is highly efficient with respect to a minimum number of flip-flops for the state encoding. It was used to determine the vehicle state for the ABS controller and thus determined which and when calculation-intensive modules requiring order preservation were enabled.

```
COMBIN: process(CURRENT_MAJ_STATE, Bat_power, ignition, trigger_pulses,
brake_pedal, V,  wheel_vel, current_min_state)
begin
  next_state <= stopped;                --   put this to avoid latches.
  case CURRENT_MAJ_STATE is
    when stopped =>
      IF (V /= 0) THEN
        NEXT_STATE <= normal;
      ELSE
        NEXT_STATE <= stopped;      --  need, else past next state takes over
      END IF;
    when normal =>
      if brake_pedal = depressed then
        NEXT_STATE <= braking;
      elsif (V = 0) then
        NEXT_STATE <= stopped;
      else
        NEXT_STATE <= normal;
      end if;
    when braking =>
      IF (V = 0) THEN
```

```
          --  double check the wheel velocities
          IF (wheel_vel = (0,0,0,0)) THEN
            NEXT_STATE <= stopped;
          END IF;
        ELSIF (brake_pedal = released) THEN
           NEXT_STATE <= normal;
        ELSIF (current_min_state /= ABS_cntrl_idle) THEN
            NEXT_STATE <= abs_enabled;
         ELSE
           next_state <= braking;
        END IF;
      when abs_enabled =>
        IF brake_pedal = released then
          NEXT_STATE <= normal;
        ELSIF (V = 0) THEN
          NEXT_STATE <= stopped;
          --  ABS has made car stop:: assuming other circuitry knows when
          --  to say that the vehicle velocity if zero and car has
          --   successfully stopped.
        ELSIF current_min_state = ABS_cntrl_idle then
        --   ABS has been turned off
          IF (brake_pedal = depressed ) THEN
              NEXT_STATE <= braking;
          ELSIF (V /= 0) THEN
              NEXT_STATE <= normal;
          END IF;
        ELSE
          next_state <= abs_enabled;      --   don't want to infer latches
        END IF;
    end case;
  end process;
-- Process to hold synchronous elements (flip-flops)
SYNCH: process(CLK_IN, global_reset,NEXT_STATE,abs_cntrl_enable )
--   asynchronous reset, but synchronous finite state machine.
  begin
    -- ( consider this as reset state: asynchronous. )
    IF  (global_reset = '1') THEN
      CURRENT_MAJ_STATE <= stopped;
    ELSIF (CLK_IN'event AND  CLK_IN = '1') THEN
      IF (abs_cntrl_enable = '1') THEN
          CURRENT_MAJ_STATE <= NEXT_STATE;
      END IF;
    END IF;
  end process;
```

The following code shows the use of WAIT statements to create a multi-cycle control sequence which synchronizes two blocks together. Its particular function was to sequentially perform the operations needed during the pressure increasing state of the pressure control loop which was enabled during the *ABS_enabled* vehicle state.

```
multi: process
  VARIABLE h1_star, s1_crit : pres_value_word;
  VARIABLE rom_addr_tmp : abs_rom_range;
begin
  IF (pi_m_enable = '1') THEN
     IF (m=0) THEN
       H1_max <= 0;
       --H1_thresh_rdy <= FALSE;
       WAIT UNTIL pipe_clk = '1';          --   skip reading ROM
     ELSE
       --WAIT UNTIL pipe_clk = '1';
       --create address to ROM and read the data
       pi_rom_addr <= incr1_offset;
       --WAIT UNTIL pipe_clk = '1';
       s1_crit := rom_data;              --   wait for data
       WAIT until pipe_clk = '1';
       --   now read LUT for the value of H1_Thresh
       pi_rom_addr <= incr2_offset;
       --WAIT UNTIL pipe_clk = '1';
       h1_star := rom_data;
       WAIT until pipe_clk = '1';
       --   select a max value
       IF (h1_star > H1_max ) THEN
         H1_max <= h1_star;
         S1 <= s1_crit;
         m_max <= m;
       END IF;
       IF (m=28) THEN
         H1_Thresh <= H1_max + H1_max/8; -- output correct thresh value
         --   will have to wait for the above instruction to finish adjust
         --   that in the clock!!!!
         h1_thresh_rdy <= TRUE;
       END IF;
     END IF;
     --  for now  do these two to synch to a slower clock, the m_clk which
     --  can generate values only every four of the pipe clock. So we hope to
     --  split our function here into three clock cycles, the last one just
     --  waits for now, may need it later!
```

```
    WAIT until pipe_clk = '1';
    WAIT until pipe_clk = '1';
    --  when calculation is enabled, and hit m=31, our max output is ready
    --   change the max to 28 for now, otherwise have to wait too long
    --  ATTENTION: this comparison could lead to a glitch
  ELSE
    h1_thresh_rdy <= FALSE;
    WAIT until pipe_clk = '1';
  END IF;
end process multi;
```

The VHDL code for the high- and low-holding pressure blocks follow. Both processes interact with the two above FSMs and the *Timer* block, whose VHDL description is also included below. The `wait` statements can be used to implement pipe-lining as shown.

```
--      File: pres_hhold_cntrl.vhd
--      Author: bHol
--
--      Description: Calculate the duration,T2, for the pressure holding
--      mode. It depends on velocity, slope, and the last values of S1 and H1
--      from the increasing state. So they must be registered.
--
--      Implementation note: assuming V is constant and m is  latched
--      then these two can be used as an indice into this LUT as well.
--
--      Modification history:
--
USE WORK.ABS_DEFS.ALL;

ENTITY pres_hhold_cntrl IS
  Port (CURRENT_MIN_STATE : IN    PRES_VALVE_STATE;
--        V: IN integer range 0 to MAX_VEL;
--        m: IN integer range 0 to 31;
        s1 : IN pres_value_word;
        rom_data : IN    PRES_VALUE_WORD;
        pipe_clk : IN    BIT;
        phh_rom_addr : OUT   ABS_ROM_RANGE;
        s2_limit : OUT pres_value_word;
        --slip_rdy : OUT   BIT;
        t2_val : OUT   pres_value_word;
        t2_preload_cnt : OUT bit;
        t2_enable_cnt : BUFFER bit;
```

```
        t2_high_rdy : OUT   boolean);
end pres_hhold_cntrl;

architecture behav_pres_hhold_cntrl of pres_hhold_cntrl is
  begin
--   calculate the time in stages of a slower clock
process
begin
  IF (CURRENT_MIN_STATE = high_hold)  THEN
    --   only want to calculate (use LUT) stuff once, so trigger on edge.
    IF (t2_enable_cnt = '0') THEN
      --WAIT until pipe_clk='1';
      -- may want to play around with the mapping, ie could still use m and V
      -- from the previous state to serve as indice in the lut.
      phh_rom_addr <= hhold1_offset;
      s2_limit <= s1 + MAX_SLIP_RANGE;
      --WAIT until pipe_clk = '1';
      t2_val <= rom_data;                --   data is ready at this point
      WAIT until pipe_clk = '1';
      t2_enable_cnt <= '1';
      t2_preload_cnt <= '1';
      WAIT until pipe_clk = '1';          --  make a pulse for the loading
      t2_preload_cnt <= '0';
      t2_high_rdy <=TRUE;
      WAIT until pipe_clk = '1';          --
    ELSE
      WAIT until pipe_clk = '1';
    END IF;
  ELSE
    t2_high_rdy <= FALSE;
    t2_preload_cnt <= '0';
    t2_enable_cnt <= '0';
    WAIT until pipe_clk = '1';
  END IF;
end process;
end behav_pres_hhold_cntrl;
```

```
--      File: pres_lhold_cntrl.vhd
--      Author: bHol
--
--      Description: Calculate the duration,T4, for the pressure low holding
--      mode. This value depends on the Velocity, m and H3. Access to rom is
--      necessary. H3 is a constant and is determined a priori so this value
--      once again depends on m and V.
--
--      Modification history:
--
--
-------------------------------------------------------------------------------
--  pres_lhold_cntrl:
-------------------------------------------------------------------------------
USE WORK.ABS_DEFS.ALL;

ENTITY pres_lhold_cntrl IS
  Port (CURRENT_MIN_STATE : IN    PRES_VALVE_STATE;
--        V: IN integer range 0 to MAX_VEL;
--        m: IN integer range 0 to 31;
        rom_data : IN    PRES_VALUE_WORD;
        pipe_clk : IN    BIT;
        timer_expired : in BIT;
        plh_rom_addr : OUT   ABS_ROM_RANGE;
        t4_val : OUT   PRES_VALUE_WORD;
        t4_preload_cnt:  OUT bit;
        t4_enable_cnt : BUFFER bit;
        h4_thresh_rdy : OUT boolean;
        t4_low_rdy : OUT   boolean);
end pres_lhold_cntrl;

architecture behav_pres_lhold_cntrl of pres_lhold_cntrl is
  begin
--   calculate the time in stages of a slower clock
process
begin
  IF (CURRENT_MIN_STATE = low_hold) THEN
    --   if expired, then load a new value
    IF (timer_expired = '1') THEN
      --  load next value, and reset enable_cnt to start down counting again
      t4_enable_cnt <= '0';
      WAIT until pipe_clk='1';
    ELSIF (t4_enable_cnt = '0') THEN
      --   may want to play around with the mapping, ie could still use m and V
```

```
      --   from the previous state to serve as indice in the lut.
      plh_rom_addr <= lhold1_offset;
      --WAIT until pipe_clk = '1';
      t4_val <= rom_data;
      WAIT until pipe_clk='1';
      t4_enable_cnt <= '1';
      t4_preload_cnt <= '1';
      WAIT until pipe_clk = '1';
      t4_preload_cnt <= '0';
      t4_low_rdy <=TRUE;
      h4_thresh_rdy <= TRUE;
    ELSE
      WAIT until pipe_clk = '1';          --  do nothing
    END IF;
  ELSE
    t4_low_rdy <= FALSE;
    h4_thresh_rdy <= FALSE;
    t4_preload_cnt <= '0';
    t4_enable_cnt <= '0';
    WAIT until pipe_clk = '1';
  END IF;
end process;
end behav_pres_lhold_cntrl;
```

The two entities, the *down_cntr* and the *timer_mux* which are affiliated with the *Timer* block functionality follow. Different encoding styles which produce both asynchronous and synchronous code are shown, with the latter one being favored by both the synthesis tool and the author, for reasons listed in the VHDL encoding's comments.

```
--      File: down_cntr.vhd
--      Author: bHol
--
--      Description: Countdown given a preloaded signal and an enable signal.
--      Taken from edge_cntr module and modified. Count down value is
--      expected to come in terms of clock counts. For the timer we may want
--      clock to be a larger clock than the input clock.
--      NOTE: May have to change to a synchronous preload. TEST THIS
--
--      Modification history:
--------------------------------------------------------------------------------
--  down_cntr:
--------------------------------------------------------------------------------
```

```
USE WORK.ABS_DEFS.all;

entity down_cntr is
        port(clk_in, preload_cnt, enable_cnt : in bit;
             count : IN pres_value_word; --   make a 10 bit counter
             timer_expired : OUT BIT;
             cnt_val : BUFFER pres_value_word);
end down_cntr;
--
-- According to a synopsys app note, this is a dangerous asynchronous design
--   and must be simulated carefully for the setting of the output must occur
--   before, the change of the clock.
--
architecture behav_down_cntr of down_cntr is
  signal gated_clk : BIT;
begin
--  gated_clk <= clk_in and enable_cnt; -- only cnt when clk and enabled
--  process(preload_cnt, gated_clk, cnt_val)
--  begin
--    --  timer_expired <= '0';  THIS produces and error, non clock edge????
--    IF (preload_cnt = '1') then
--      cnt_val <= count;                  --   this load must be fast enough
--    ELSIF(gated_clk'EVENT and gated_clk = '1' ) then
--      IF (cnt_val = 0) THEN
--        timer_expired <= '1';
--      ELSE
--        cnt_val <= cnt_val -1;
--        timer_expired <= '0';
--      END IF;
--    END IF;
--  end process;


process(enable_cnt, preload_cnt, clk_in, count)
begin
  IF (enable_cnt = '1') THEN
     IF ((clk_in'EVENT) and (clk_in = '1')) THEN
       IF (preload_cnt = '1') then
         cnt_val <= count;      --   this load must be fast enough
         timer_expired <= '0';
       ELSIF (cnt_val = 0) THEN
         timer_expired <= '1';
       ELSE
         cnt_val <= cnt_val -1;
```

```
        --timer_expired <= '0';
      END IF;
    END IF;
  ELSE
    timer_expired <= '0';
  END IF;
end process;

END behav_down_cntr;



--      File: timer_mux.vhd
--      Author: bHol
--
--      Description: lets more than one block be connected to the
--      timer or rather down_cntr entity. This approach uses muxes instead of
--      tristates.
--
--      Modification history:
--      Mar 11, 1994: Decided to go with Muxes to avoid routing problems
--
-------------------------------------------------------------------------------
--  timer_mux:
-------------------------------------------------------------------------------
USE WORK.ABS_DEFS.all;

entity timer_mux is
  port( current_min_state : in pres_valve_state;
        preload_cnt1, enable_cnt1 : in bit;
        count1 : IN pres_value_word;
        preload_cnt2, enable_cnt2 : in bit;
        count2 : IN pres_value_word;
        preload_cnt, enable_cnt : out bit;
        count : OUT pres_value_word);
end timer_mux;

architecture behav_timer_mux of timer_mux is
begin
process(current_min_state, preload_cnt1, enable_cnt1, count1,
             preload_cnt2, enable_cnt2, count2)
begin
  IF (current_min_state=high_hold) THEN
    preload_cnt <= preload_cnt1;
```

```
    enable_cnt <= enable_cnt1;
    count <= count1;
  ELSIF (current_min_state=low_hold) THEN
    preload_cnt <= preload_cnt2;
    enable_cnt <= enable_cnt2;
    count <= count2;
  ELSE
    preload_cnt <= '0';
    enable_cnt <= '0';
    count <= 0;
  END IF;
end process;

--    now lets try using concurrent logic, Synopsys produces similar circuits
--    for both type of encodings.
--preload_cnt <= preload_cnt1 WHEN current_min_state = low_hold ELSE
--               preload_cnt2 WHEN current_min_state = high_hold ELSE '0';
--enable_cnt <= enable_cnt1 WHEN current_min_state = low_hold ELSE
--              enable_cnt2 WHEN current_min_state = high_hold else '0';
--count <= count1 WHEN current_min_state = low_hold ELSE
--         count2 WHEN current_min_state = high_hold else 0;
END behav_timer_mux;
```

Appendix C

Synthesis and Supporting Libraries

An entered design, schematic or VHDL, transgresses to its realization via a library or hierarchy of libraries, employed by the synthesis tool, which attempt to best mirror the technology features. Analogous to design entry methods, libraries can be generic or technology specific. A schematic can contain both technology specific components and higher level blocks just as VHDL encodings can be behavioral (generic) or can instantiate technology specific primitives. Whatever the description, the mapping to an interconnection of components from a library during the synthesis will transpire. Logic synthesis will map logic gates to logic gate primitives. RTL synthesis will initially map to higher level blocks, then then to lower level logic primitives. The mapping steps during synthesis continue as one progresses higher up the specification ladder. High level synthesis implicates more abstracted modules which will go through many levels of refinement before final realization on the given technology. A library, which may also be hierarchical, must support such a synthesis process so that the best circuit realization results.

Synthesis attempts to replicate and improve the traditional conceptualization of the designer through its numerous algorithms using deductive reasoning and pre-constructed, sometimes pre-mapped, hardware components. Hardware design exhibits a certain degree of commonality which aides in constructing a generic, hierarchically integrated circuit database.

Technology distinct hardware features, known a priori, could also be translated into library representations so that all architectural features both standard and custom can be exploited. The incorporation into commercial tools of well-constructed and standardized library support systems composed of both fixed and parameterizable logic blocks which accommodate the many layers in a hierarchical design is proposed.

VHDL synthesis, the translation of VHDL encoding to a library of hardware blocks, and its eventual layout is not without complications and ambiguities. Hardware mapping performed as early as possible will produce optimum results. If the first stage accompanies generic blocks, then the lowest stage (often layout) which deals with logic gate primitives, will have lost certain hardware specific optimizations and clusterings due to the previously executed generalizations. A VHDL application specific and/or technology specific library would aid higher levels of design, optimize circuit realizations and consequently ease the job of the designer due to the facility of component re-use, hard macro exploitation and special feature or design style exploitation. In a library, one can encode (encrypt) design styles and exploit architecture and technology through well-constructed non-atomic entities. Various levels of genericity and specialization could also be incorporated for block parameterizations to customize a design as per the designer's specified functionality and constraints.

The objective of a module library is to provide a generic, technology independent set of logical primitives with which one can construct a design and achieve efficient performance from a wide array of technologies and obtain efficient access to unique architectures. Additional research remains to be explored in the domain of tool capabilities and their respective library support systems, with respect to design space exploration and library support system traversals.

C.1 FPGA Library Motivation

For FPGAs, it is desired to have clever explicit and implicit liaisons to the hardware features of the destination technology from a VHDL design entry point. *Inference*, the selection of a hardware equivalent or a logic equivalent circuit, is a difficult task. Because the architectures of FPGAs have evolved into many different structures and will continue to evolve as better architectures, and newer inherent structures are constantly being developed, it is sub-

sequently difficult to come up with VHDL inferences which can exploit all hardware features, when some are unique, some change, old ones die, and new ones are created. Maintaining compilers current with architecture is a hard task. It will be quite a challenge for hardware vendors and tools vendors to coordinate their efforts in keeping the design engineers supplied with tools which transform their designs into technology specific circuit realizations which meet area and timing constraints and require little low level design work. A well designed FPGA vendor created library can provide these facilities.

The goal to keep in mind is the desire for a technology independent design style supported by synthesis tool which maps an entered design "as best as possible" onto the given hardware. The problem is NP hard, providing justification for the unavailability of such a tool. Creating FPGA vendor specific modules seems to be a viable alternative. Instantiation is a simple measure to obtain desired results immediately. Having both, support for technology specific macros, and support for technology dependent structured logic synthesis in FPGAs is our proposed solution.

Recollect the nature of the FPGA architecture. Unwanted area overhead and propagation delays caused by the programmable switches, entail the need for more succinct mappings. Less synthesis slack than could be accepted for MPGAs can be tolerated. Hence, the exigency to support vendor specific well-constructed *blocks* which include one or more logic blocks and their strategically designed interconnections. From an engineering HDL design viewpoint creating a library of parameterizable and reusable components and a design methodology can serve as building blocks for additional system modules and future designs.

A *library support system* is needed with a high level design entry method in order to exploit the FPGA architectural features, support standard off-the-shelf components and utilize already designed blocks of past designers[1]. These blocks could include, *hard, soft, special feature macros*, and *user-created macros*, and would require alliance between synthesis and technology vendors. Hard macros consist of a fixed number of logic blocks, and some pre-assigned routing to form partially prelaid and prerouted larger blocks or aggregates. They are specially designed to exploit the target architecture to its fullest. A larger number of soft macros exist from each vendor supporting their version of standard blocks, some of

[1] who most likely work with the FPGA hardware and have mastered some efficient usage of the logic blocks and the routing structure and consequently created quite succinct blocks efficient both in area and time.

which are similar to standard TTL gates and SSI circuits. The hardware specific detail accompanying a soft macro is the number of its vendor specific logic blocks. Its functionality is easily defined and each vendor is responsible for the implementation. One remaining vendor specific type of macro to be supported are the *special feature macros*. Each FPGA vendor will possess special hardware features which offer performance or area advantages. For example, fast carry circuitry, low skew clock lines, and registers are all elements each FPGA vendor supports. The last type are *user-defined blocks* designed to serve their requirements. They can be created to comply well with user-specified constraints of minimum area and timing trade-offs and to best fit the technology, and/or facilitate module generation through repeated usage of application specific primitive blocks.

C.2 Existing Concepts

Some work which addresses FPGA based library support systems for synthesis is reported in [59], [60], [61] and [57]. Dekker [59], presents a methodology to utilize technology specific macros for arithmetic and relational operators while, allowing technology independent specification using VHDL for design entry. The approach useful but additional logic configurations and operators at the RTL structural and higher levels of design specifications is suggested.

Kelem et al, [60] propose a module synthesis tool which maps architecture-independent designs into architecture-specific implementations. Both schematic capture and HDLs are supported for functional design entry. In the current version, only the XC4000 block architecture is supported. There is mention of extensions to other families in the future.

A library of 32 general symbols exists representing the generic modules. Their parameters can be defined both during the schematic capture, and via generic construct usage in HDLs. The main difference between a conventional netlist and an X-BLOX netlist is that in an X-BLOX netlist the sizes and implementation styles of modules and buses do not have to be completely specified. They can be determined by context, data-types, and bus-types. Thus for each datapath only one value must be given, the remaining operators and buses will benefit from type-inference.

The knowledge base which supports the synthesis tool contains types and number of resources available for each member of the XC4000 family, and resembles a rule-based library. Its information is used to determine which types of architecture-specific optimizations can be performed and which styles are appropriate for synthesis. This way the special features can be exploited, and elements such as fast carry circuitry, can be used for mapping arithmetic functions.

The synthesis technique is similar to our proposal, yet little, if any, mention is furnished regarding rules governing a library system and how it can be extended across the numerous FPGA architectural varieties. We propose to delay the technology dependent phase of synthesis so that any, and more than one, type of FPGA may be used.

Underlying concepts and principles of object-oriented techniques have already found roots amongst hardware designers. The construction of systems using reusable library components, a highly renown advantage of object-oriented techniques is analogous to using off-the-shelf building blocks in the hardware world [61].

Using C++ constructs, classes, object and transformation functions, Kumar et all [61], indicate a potential for developing C++ models for hardware components. Just as their resulting model resembles a data path/control decomposition, we analogously emphasize the formation of design blocks, VHDL code, synthesis and implementation tricks and design methodologies for control/data-path circuits. Similar VHDL design techniques can be developed for other design styles and for other FPGA technologies.

Sample library component lists already exist [58] but they are described at the logic level in the form of boolean equations. A start, but the need for more parameterizable, standardized components which exhibit both simple and complex boxed functionality would be useful. Library samples [57], [60] with parameterizing capabilities have been proposed. From a synthesis perspective, an ensuing challenge demands support of these blocks through techniques such as easy component instantiation with different parameters, quick composition of new components, component re-usability, and tailoring general-purpose components. From a VHDL perspective, it is expected that such components will be properly instantiated with correctly specified parameters by the synthesis tool. Furthermore, in order to cater to application areas, particular blocks created with application or design style specific charac-

toristics would also be beneficial to a synthesis tool which itself possesses application specific features.

C.3 Proposed Library Component Types

Our work focuses on blocks of a particular application type, controllers, and technology type, FPGAs which can best be used by the synthesis tool for its creation of implementations and for the designer to select if the need arises, blocks closely related to the desired functionality. Exhaustive studies of design methodologies, application specific repetitive primitives gives incite to such blocks. And inspection in the analogous features between FPGA hardware and possible blocks and the input formats and viewpoints, is essential for the creation of such a set of cleverly constructed blocks which can be either directly accessed by the user or indirectly inferred by the synthesis tool.

Of the many types of library components mentioned earlier, we will focus on two: well-constructed functional blocks (user customized) and technology specific feature exploiting blocks, and justify the need for their existence. "Blocks" can represent both functional units, functions/operations which do not necessarily exist in component form, and design methodologies. For example, global circuit reset and register initialization are vital elements of digital design, but do not exist in component form. Implementation of reset capabilities (both synchronous and asynchronous) through VHDL statements directly (or even from an instantiation) exploiting known hardware resources would be beneficial. Having them inferred from the design description (VHDL in our case) rather than having to adjust the compile process, or even instantiate a "block" to represent this type of functionality is even better. Similarly being able to parameterize a finite state machine block would also be advantageous.

As an application specific simple block(component) example, controllers often contain different circuit sections executing at different clock rates. Hence, the need for a clock divider(s), and the usefulness of a generic, efficiently structured FPGA implemented, clock with a parametric division factor. The end-users sees an easily programmable block while the synthesis tool implements a well-engineered version on the technology. From a methodology angle, writing VHDL code with a regular `IF` statement before a clock edge query results in

asynchronous set/reset capabilities, useful for the event driven circuit behavior.

C.4 Synthesis Tool Functionality

There is much room for improvement within synthesis tools and library support systems. A library support system is most useful, if a synthesis tool is capable of exploring design implementation options and drawing out the block(s) which best realize the input description, which in the ABS case study was VHDL. Together they must offer optimum design space exploration.

Manually instantiating a block through HDLs is a small task, whereas conveying the presence of such a block (particularly ones for hard macros or those which represent special features) from encoding to the synthesis tool for inference is difficult, and will require a pre-learned sequence of input commands. Such elements affect the formalizations of the strategy behind a library structure, its access and the employment of genericity and parameterization.

With direct association between synthesis tool developers and FPGA technology, a functional unit (FU) could be implemented by first selecting a hard wired version, with pre-routing, if such a module exists. Next soft macros could be searched followed by user macros to see if the functionality and the design constraints are met. Should the above explorations above fail to produce a satisfactory component match, the FU could be constructed from lower level building blocks, which in some cases, may be primitive gates. Either way the requests of the synthesis tool must be transferred to the technology resources and libraries. Alternatives are essential and should be expressly available, with last resort availability of lower level building blocks. Ultimately, if a match is found between a functional request, and an available library module, the library module is instantiated, otherwise a boolean definition of the operation/functionality should be utilized for implementation. Finer grain components are just as important in a library system as are larger, cleverly constructed coarse-grained blocks.