2004

Fpga-based Design Of A Maximum-power-point Tracking System For Space A

Todd Persen
University of Central Florida

FPGA-BASED DESIGN OF A MAXIMUM-POWER-POINT TRACKING SYSTEM FOR SPACE APPLICATIONS

by

TODD EDWARD PERSEN
B.S. University of Central Florida, 2002

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2004

ABSTRACT

Satellites need a source of power throughout their missions to help them remain operational for several years. The power supplies of these satellites, provided primarily by solar arrays, must have high efficiencies and low weights in order to meet stringent design constraints. Power conversion from these arrays is required to provide robust and reliable conversion which performs optimally in varying conditions of peak power, solar flux, and occlusion conditions. Since the role of these arrays is to deliver power, one of the principle factors in achieving maximum power output from an array is tracking and holding its maximum-power point. This point, which varies with temperature, insolation, and loading conditions, must be continuously monitored in order to react to rapid changes.

Until recently, the control of *maximum power point tracking* (MPPT) has been implemented in microcontrollers and *digital signal processors* (DSPs). While DSPs can provide a reasonable performance, they do not provide the advantages that *field-programmable gate arrays* (FPGA) chips can potentially offer to the implementation of MPPT control. In comparison to DSP implementations, FPGAs offer *lower cost* implementations since the functions of various components can be integrated onto the same FPGA chip as opposed to DSPs which can perform only DSP-related computations. In addition, FPGAs can provide equivalent or *higher performance* with the customization potential of an ASIC. Because FPGAs can be reprogrammed at any time, repairs can be performed *in-situ* while the system is running thus providing a high degree of *robustness*.

Beside robustness, this reprogrammability can provide a high level of (i) *flexibility* that can make upgrading an MPPT control system easy by merely updating or modifying the MPPT algorithm running on the FPGA chip, and (ii) *expandability* that makes expanding an FPGA-based MPPT control system to handle multi-channel control. In addition, this reprogrammability provides a level of *testability* that DSPs cannot match by allowing the emulation of the entire MPPT control system onto the FPGA chip.

This thesis proposes an FPGA-based implementation of an MPPT control system suitable for space applications. At the core of this system, the Perturb-and-observe algorithm is used to track the maximum power point. The algorithm runs on an Alera FLEX 10K FPGA chip. Additional functional blocks, such as the ADC interface, FIR filter, dither generator, and DAC interface, needed to support the MPPT control system are integrated within the same FPGA device thus streamlining the part composition of the physical prototype used to build this control system.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Abdel Ejnioui, for his tireless assistance and endless contributions. With his help, I have learned tremendously from this experience.

I would like to thank my committee members, Dr. Issa Batarseh and Dr. Takis Kasparis, for their support and advice throughout this project. It is their work and vision from which this thesis is derived. Without them and the time they have devoted, I would not be able to complete this thesis.

I would like to thank my girlfriend, Genna, for her patience and compassion while keeping me on track. Thank you for always taking care of me and keeping me focused on the things that are truly important. I love you.

I would like to thank Chris Douglass for always being there for me. Thank you for being there with the answer I need and the willingness to help me see everything through. You never let me down.

I would like to thank my family and friends for helping me through my continual lack of sleep. I couldn't do anything without all of the great people around me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

1.1 Photovoltaic Systems

The *photovoltaic* (PV) *effect*, discovered by Edmond Becquerel in 1839, is the process by which a voltage is generated when photons strike an appropriate material [1]. The first PV cells consisted of either AgCl or AgBr upon an electrode in the presence of an acidic solution. Subsequent solar sells were developed either with selenium and platinum, or copper and cuprous oxide. As silicon development progressed in the 1940s and 1950s, the photosensitivity of doped semiconductors became apparent. In 1954, the first silicon-based solar cell was developed thus laying the road for decades of research [2].

In the 1970s, the oil crisis spurred an intense investigative effort into the development of PV power generation. Hundreds of millions of dollars were provided by the Department of Energy to help foster research in many areas of PV engineering [3]. Although the discovery of the PV effect preceded these events by more than a century, significant research remains to be completed in order to bring the efficiency level of the devices of then to the level seen in the products available in the current marketplace.

The first silicon-based cells had efficiencies around 6% [2]. With extensive development in this field, cells produced in research facilities can reach 35.2% efficiency. While their efficiency is increasing steadily, production-level PV arrays can currently produce efficiencies that are 50% to 65% of this value [3]. With peak solar levels at the earth's surface near 1,100 W/m^2, these cells make it possible to obtain several kW from a PV

array on the roof of a house [1]. As efficiencies increase, PV will have a profound impact on the marketplace.

1.2 Use of Photovoltaics in Space Applications

Space applications represent one of the most active areas of research using PV in general, and particularly in satellite spacecrafts. In fact, satellites are non-serviceable systems that need a source of power throughout their missions to help them remain operational for several years. The power supplies of these satellites must have high efficiencies and low weights in order to meet stringent design constraints [4].

Presently, PV cells constitute the primary power source of *geo-stationary earth orbit* (GEO) or *low earth orbit* (LEO) satellites [4]. GEO satellites are presently averaging 15kW to 30kW power utilization while lower-powered LEO satellites, such as those used for scientific purposes, are averaging around 2kW. At the high end of the spectrum, the International Space Station is targeted to have a maximum power output of 250kW with a nominal utilization of 75kW [5]. Based on these utilization levels, even a small increase in efficiency can yield significantly higher output, reduce the number of required PV cells, and help meet the end-of-life power requirements [4].

These satellites are powered by a limited number of arrays. Depending on the application, arrays can be body-mounted or deployable. Body-mounted arrays are physically fixed to the satellite, but limit the area available for gathering incident solar energy. Deployable arrays, on the other hand, are capable of unfolding in space to

generate more power at the expense of an increase in design complexity [4]. Regardless of the construction, increasing efficiency allows lower costs, and simpler design and implementation in some cases.

1.3 Power Control of Photovoltaics in Space Applications

Since the role of PV arrays is to deliver power, one of the principle factors in achieving maximum power output from an array is tracking and holding its maximum-power point. This point, which varies with temperature, insolation, and loading conditions, must be continuously monitored in order to react to rapid changes [6]. Due to the output resistance of the array and its inherent nonlinearity, the output power of a PV array has a natural maximum [7]. The curve containing this maximum can be characterized by the following equation:

$$i_s = I_{PH} - I_0 \exp(\frac{q}{kT} e_s - I_0) \qquad (1.1)$$

where i_s is the array output current, e_s is the array output voltage, I_0 is the array saturation current, T is the array temperature in K, I_{PH} is the light-generated current and K/q is Boltzman's constant [8]. As such, it is highly desirable to regulate the output voltage of the PV array whenever possible if maximum efficiency is desired. These maximum power points are shown in Figure 1.

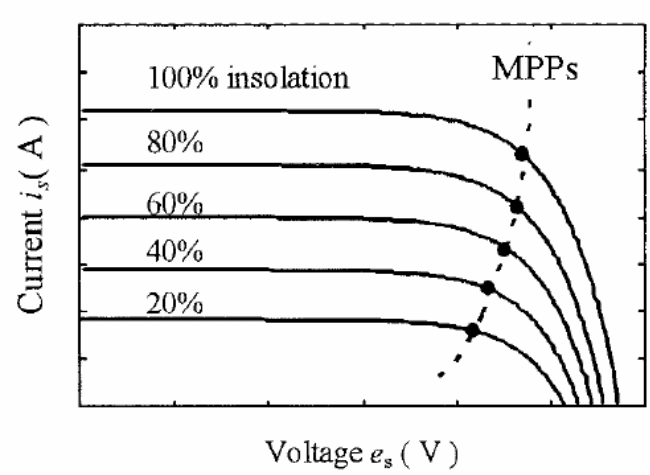

Figure 1: Characteristic V-I curves for a PV array [8].

In space applications, the loading and solar intensity experienced by the array can vary widely. For example, the lifetime planned for the Stardust spacecraft, launched in 1999, depicted an almost tenfold factor in the encountered solar intensities throughout its journey. The peak power demand was calculated to fall during a time when about 30% of the maximum energy would be available [9]. For this reason, it is crucial to track the maximum power point during times of high availability in order to charge batteries to service periods of high demand.

1.4 FPGA-Based Power Control in Space Applications

Until recently, the control of *maximum power point tracking* (MPPT) has been implemented in microcontrollers and *digital signal processors* (DSPs) [10-12]. While DSPs can provide a reasonable performance, they do not provide the high degree of flexibility that *field-programmable gate arrays* (FPGA) chips offer today. In comparison to DSP implementations, FPGAs offer equivalent or higher performance with the

customization potential of an ASIC. From an implementation perspective, FPGAs provide designers with a significantly finer grain of control at the design level, which can lead to highly optimized implementations. As a case in point, it is possible to do more with less by running the system at a slower clock frequency to conserve power.

As FPGA chips migrate to nanometer-range CMOS processes, they gain considerably in gate capacity, clocking speed and programming flexibility [13]. For example, the Xilinx Virtex-4 family of chips is fabricated using Triple-Oxide 90-nm process technology packaged as complete reconfigurable system-on-chips targeted toward high performance signal processing, communication applications, and embedded computing [14]. These chips can contain (i) up to 200,000 logic cells which can pack over 2 million gates, (ii) up to two *reduced instruction set computing* (RISC) PowerPC processors that can run at 450 MHz, (iii) provide DSP functions that can run at 500 MHz, and (iv) consume up to half the power consumed in previous Xilinx high capacity FPGA chips. As part of the growth of the FPGA industry, major FPGA vendors are producing low-cost, high-performance FPGAs designed to compete where ASICs are infeasible [13]. By targeting this market, it is now possible to replace DSPs with a customized controller capable of outperforming almost anything in an equivalent price range. Beside these advantages, companies such as Altera are offering their own microcontroller cores, as vendor-tested and guaranteed softcores, that can be implemented in and FPGA and customized – a feature DSPs cannot claim. As power control and regulation become even more important, it seems likely that FPGAs will begin to replace traditional microcontrollers and DSPs.

1.5 FPGA Technology

Field-Programmable Gate Arrays (FPGAs) are a member of the Programmable Logic Device (PLD) family. They are characterized by an array of logic cells surrounded by a highly configurable interconnect structure [15]. Depending on the particular FPGA family, the size and complexity of the logic cells will vary, as will the number and type of interconnects throughout the device. Some devices also include RAM blocks which can be configured as part of a design. The remainder of this section will focus on Altera devices since the FPGA implementation described in this thesis is based on an Altera FPGA chip.

The basic architectural approach in any FPGA is to duplicate numerous copies of a small, configurable logic element, and connect them together using programmable interconnects. These resources can be used to implement any design that does not exceed space constraints. Combined with a synthesizable Hardware Description Language (HDL), this advantage provides a powerful tool for implementing complex logic devices. Although FPGAs will never be as fast as ASICs, their relative high density and reconfigurability make them the perfect solution for many applications [16].

The Altera FLEX 10K series, used in this project, is comprised of a "sea-of-gates" architecture featuring a combination of *Logic Array Blocks* (LABs) and *Embedded Array Blocks* (EABs). These blocks are surrounded by row and column interconnects, which can be used to transport signals through the device. Each row of LABs contains a single

EAB. The LABs and EABs are connected using a specialized FastTrack Interconnect designed for direct, high-speed communication [13]. A diagram of the block layout is shown in Figure 2.

1.5.1 Logic Resources

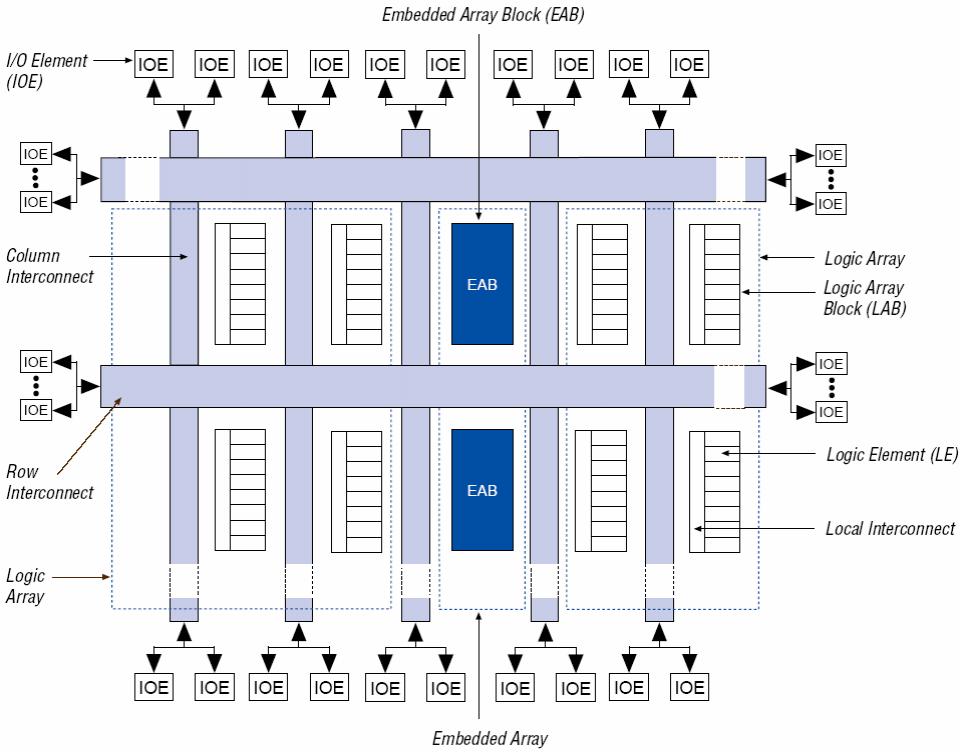

Figure 2: Altera FLEX 10K device block diagram [13].

The EABs inside the FLEX 10K devices are important because each one contains a 2,048-bit RAM array. As shown in Figure 3, the EABs can be used as traditional memory, or configured to implement complex logic functions, or other memory and computationally intensive functions. When used as memories, they can be configured as

256 x 8, 512 x 4, 1,024 x 2, or 2,048 x 1 memories depending on the application. Multiple EABs can also be cascaded together if necessary [13].

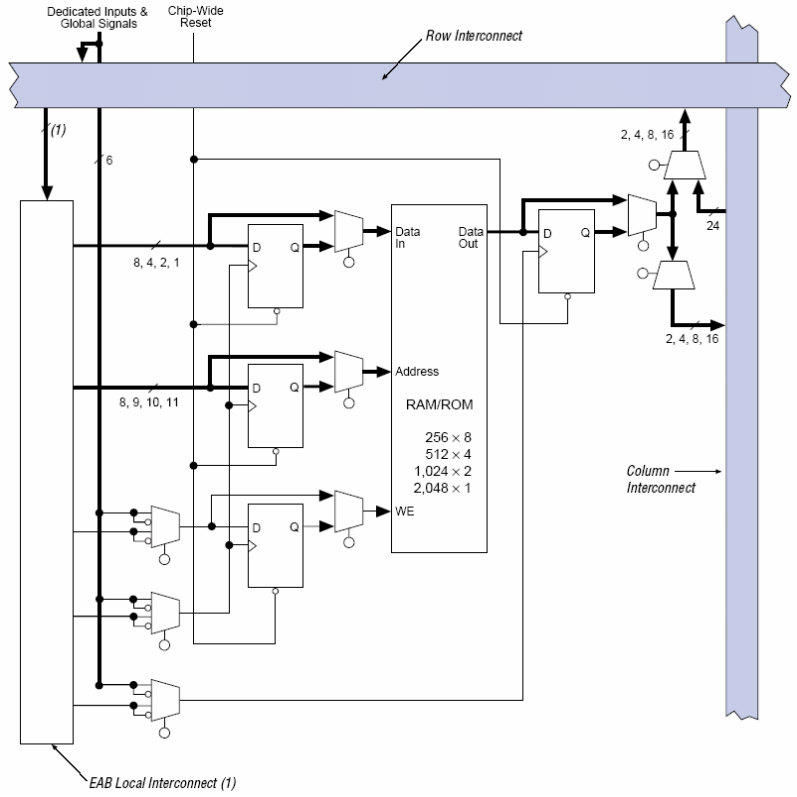

Figure 3: Altera FLEX 10K Embedded Array Block (EAB) [13].

The LAB within the Altera FLEX 10K is a coarse-grain logic resource suitable for reducing the overall demands on the interconnect resources. As indicated in Figure 4, eight LEs are clustered together into a single LAB, which makes it fast and easy to group the LEs into a single element for implementing medium-sized logic devices, such as an 8-bit counter. Carry and Cascade I/Os are also provided for chaining multiple LABs

together for high-speed arithmetic operations. A single LAB is equivalent to approximately 96 logic gates [13].

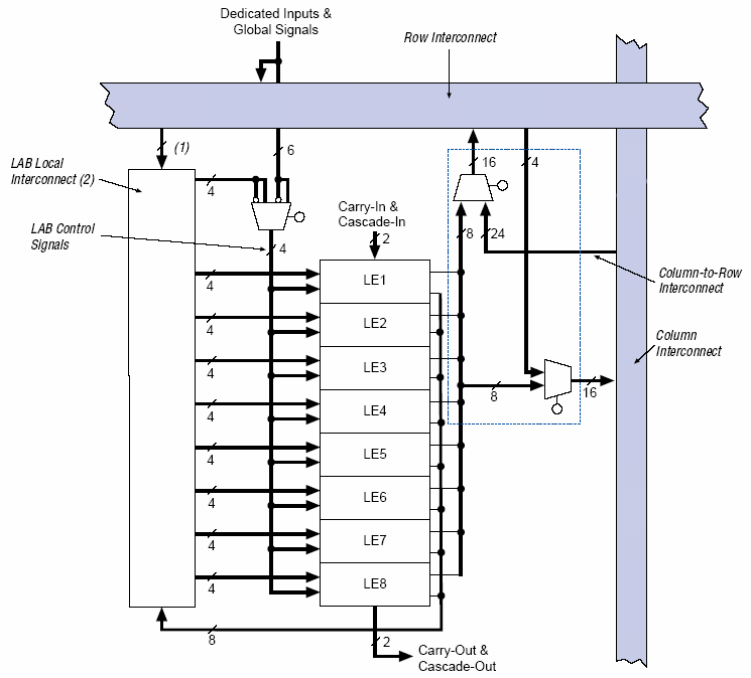

Figure 4: Altera FLEX 10K Logic Array Block (LAB) [13].

The smallest block within the FLEX 10K is the Logic Element (LE). As shown in Figure 5, the core of each LE is a 4-input Look-Up Table (LUT), coupled with some multiplexers, gates, and a programmable flip-flop to allow synchronous operation. The control lines going into these devices support the reconfigurable capability of the FPGA. Each control line is attached to a simple configuration SRAM which is either set or reset when the FPGA is programmed. The values of these configuration pins wholly determine the function of the LE [13]. Data and control lines are shown in Figure 5.

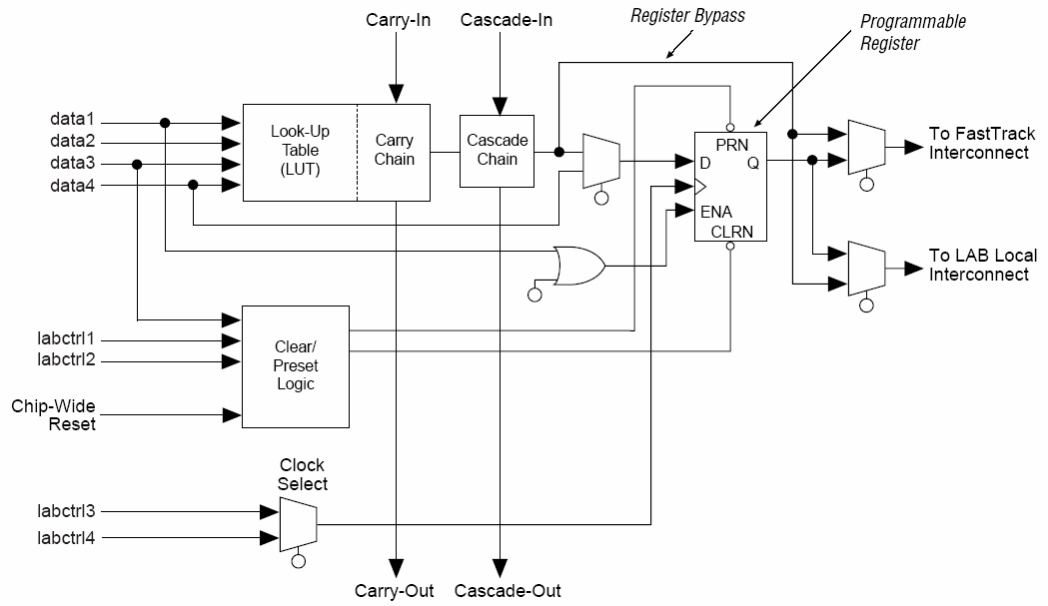

Figure 5: Altera FLEX 10K Logic Element (LE) [13].

1.5.2 Routing Resources

In the Altera FastTrack Interconnect architecture, row and column channels span the entire length of the device in order to connect the LEs and I/O pins. A row channel can be driven by an LE or one of three column channels. A 4-to-1 multiplexer brings these four signals together. Each LE has two multiplexers and is therefore capable of driving two different row channels [13]. A diagram of the FastTrack Interconnect architecture is shown in Figure 6.

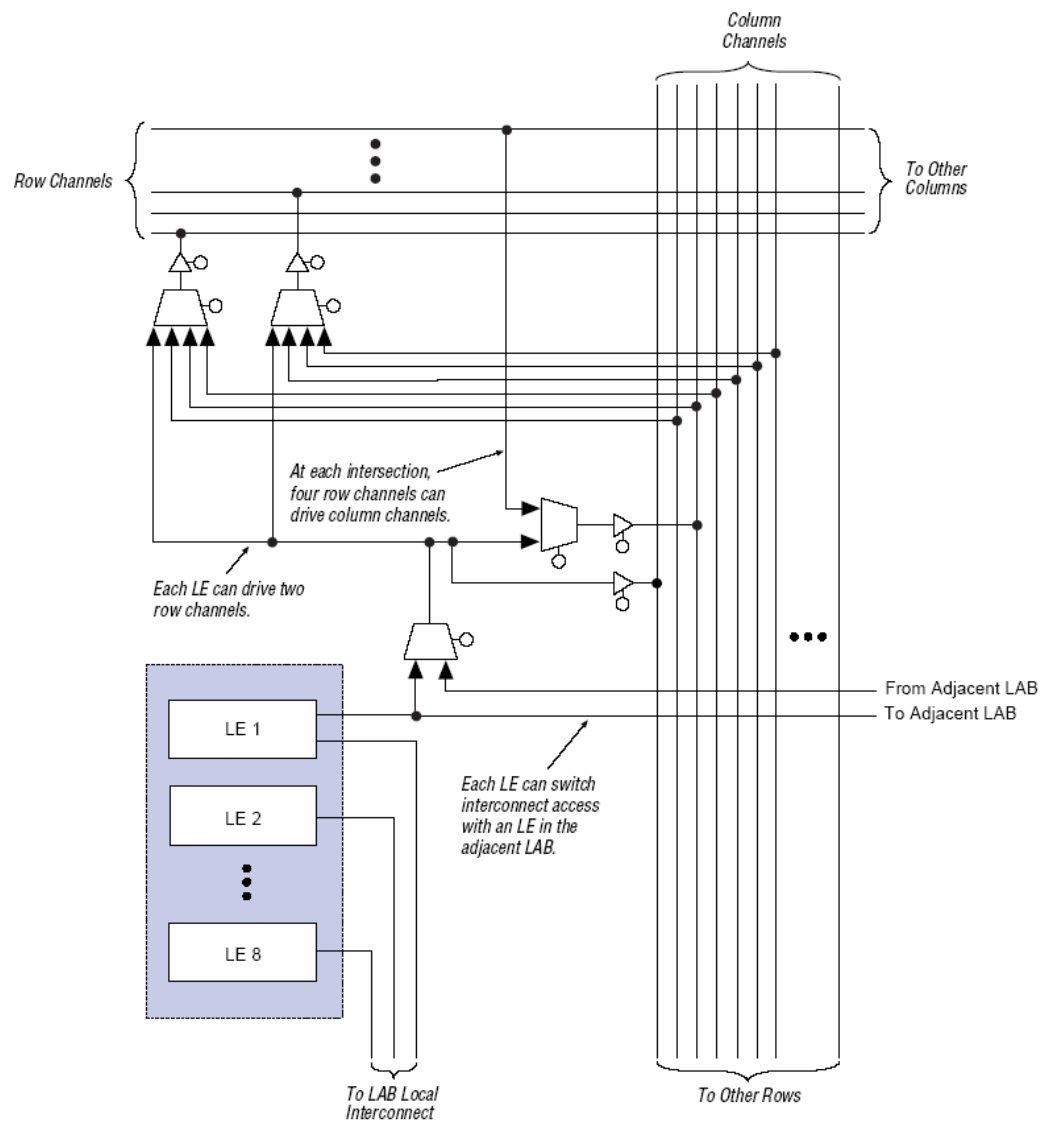

Figure 6: LAB connections to row and column interconnects [13].

Every LAB is given its own column interconnect which can then be routed to another LAB via a row interconnect, or can go directly to an I/O pin. Figure 7 shows a general view of the FastTrack layout throughout the device. In addition, the row interconnect contains some half-length channels designed for nearby LABs. This allows each half to be separately utilized where a full channel would have been needlessly monopolized by a

single signal. As a result, routing efficiency and design density is improved throughout the device [13].

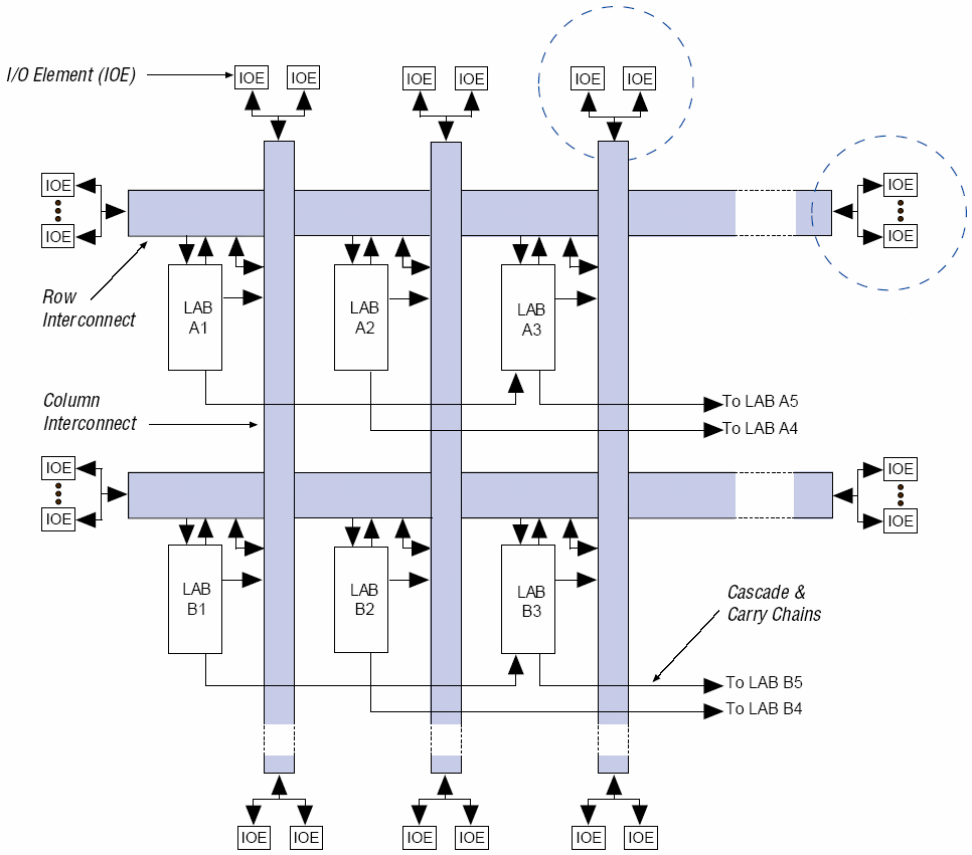

Figure 7: Global interconnect resources [13].

1.5.3 I/O Resources

I/O Elements (IOEs) are located at the periphery of the FLEX 10K chip. As shown in Figure 8, an IOE contains a bidirectional I/O buffer and a register that can be configured for either input or output. In addition, the IOE allows its own register to be bypassed so that an LE register can be used instead for faster setup times. Moreover, the IOE allows programmable inversion of signals so that the compiler can ensure the correct value is always available directly from the row and column interconnects [13].

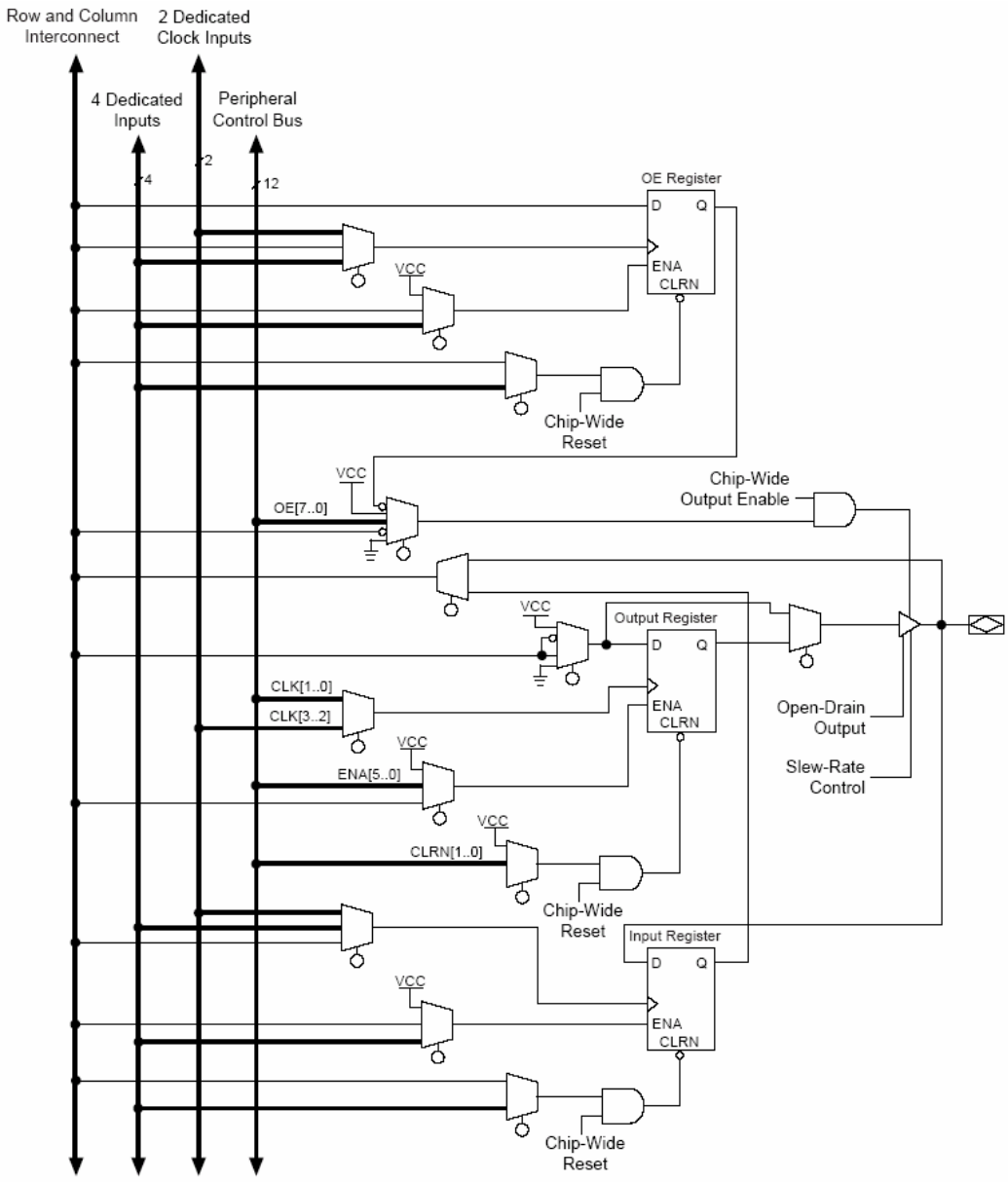

Figure 8: Bidirectional I/O registers [13].

At both ends of every FastTrack row and column, there are row-to-IOE and column-to-IOE connections, respectively. These simply allow the interconnect to be connected to the pins in a bidirectional fashion. Both row and column IOE connections allow an input signal to drive two row channels or two column channels, respectively [13]. Figures 9 and 10, respectively, show row-to-IOE and column-to-IOE connections.

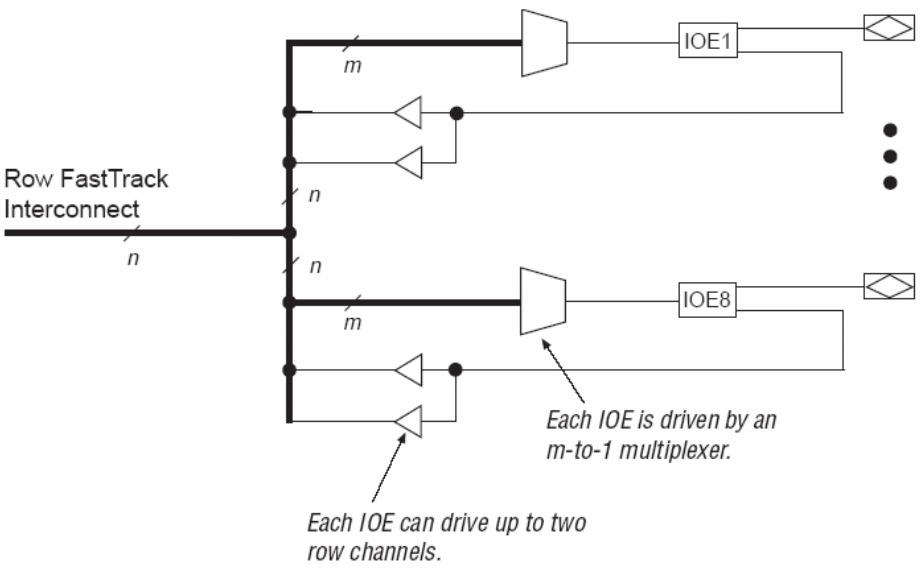

Figure 9: Row-to-IOE connections [13].

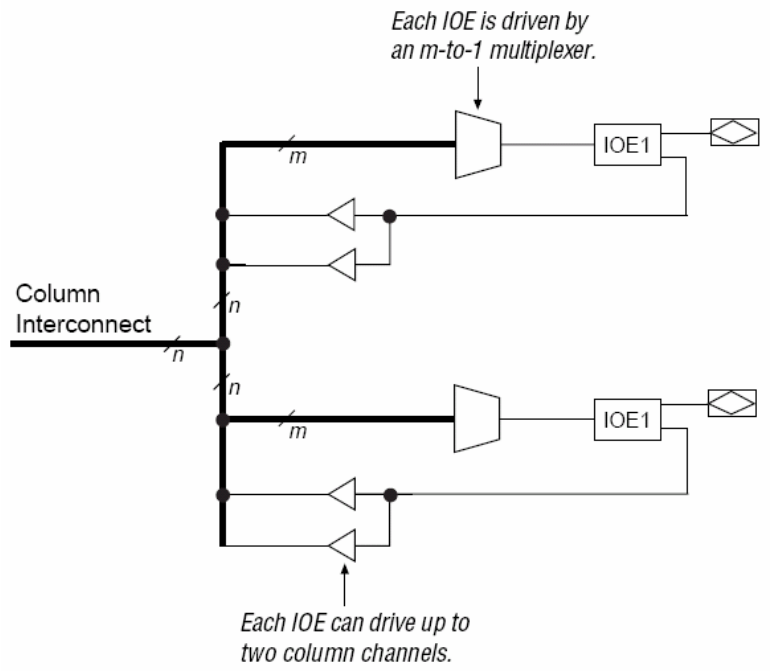

Figure 10: Column-to-IOE connections [13].

1.5.4 Device Reconfiguration

Because all the control signals within the FPGA are reconfigurable at any time, these configuration bits are stored as SRAM inside the FPGA. At configuration time, which can be at power-up or while the device is running, these bits are loaded into the FPGA. Since this storage is volatile, the FPGA can perform this configuration process during every power-up sequence. This data is created when a design is compiled, and can be downloaded from a computer over the JTAG interface or from an on-board EEPROM. A complete reconfiguration of a FLEX10K250A takes less than 320 ms, and smaller devices can be expected to take proportionally less time [13].

1.6 Future Photovoltaic Applications

In addition to their use in space applications, PV arrays are finding a home in terrestrial applications. The thrust to make PV a major component of the world's energy production requires the balance of several major factors [3]. First, the cost of manufacturing the arrays must be low enough to compete with more conventional methods, such as nuclear, coal, and natural gas. Moreover, the implementation and maintenance costs must also be kept to a minimum. In addition, the efficiency must be high in order to harness as much solar energy as possible per unit area. This is particularly important in space-constrained locations, such as residential installations. In every case, the underlying focus is to increase the efficiency of PV technology.

Whether the PV array is in an off-grid or grid-connected system, it is usually coupled with an inverter to provide the AC for either residential usage or current injection into the

grid [17]. As with any system, ensuring that every component is operating at maximal efficiency is fundamental to achieving high overall efficiency. In the case of PV arrays, their characteristic V-I curve display a maximum-power point which varies with numerous factors including temperature and insolation. In order to insure that the inverter never attempts to draw more than the maximum power, a condition which would cause the power output to collapse, a MPPT device is used to regulate the PV output [10].

MPPT implementations can vary widely although they all are based on sampling of a combination of current, voltage, and power. A MPPT algorithm is applied to locate and track the global maximum. As the algorithm becomes effective, the control time spent at the top of the characteristic power curve increases. This ultimately determines the efficiency of the MPPT system. However, in real-world implementations, numerous challenges must be overcome in order to perform reliable MPPT. The most daunting problem is the preponderance of noise, which can lead to sampling of false and subsequently incorrect power calculations. In addition, characteristic power curves can contain local maxima beside the desired global maxima. A robust MPPT implementation must be able to overcome these problems in an accurate and robust manner.

1.7 Thesis Contributions

While several MPPT approaches have been proposed before, this thesis presents several additions to a basic MPPT algorithm which lead to an effective MPPT design and implementation. The algorithm and its additions are implemented on an FPGA platform.

By adopting this platform for MPPT control, this thesis makes the following contributions:

(i) Until recently, all digital MPPT implementations have been based on DSP or microcontroller platforms. To our knowledge, this is the first FPGA implementation for MPPT control based on our search of the literature. This implementation presents a new development path which could yield lower costs and higher performance.

(ii) In general, a conventional microprocessor is capable of performing only a single instruction at any given time. In FPGA implementations, however, multiple copies of the MPPT algorithm can be running simultaneously, allowing multiple channels to be easily handled with a single device. This greatly simplifies communication and drastically reduces part counts.

(iii) In the pursuit of reduced costs and increased economic efficiency, some A/D and D/A concepts will be considered to see if they represent a feasible means of further utilizing the processing power of the FPGA to eliminate the external ADCs and DACs.

(iv) Little technical documentation exists regarding the optimum or necessary resolution of the ADCs and DACs available on the market. This documentation is necessary to integrate FPGA chips with ADCs and DACs on prototyping boards. In the absence of such documentation, the FPGA chip is instead used as a testbed to explore the resolution of the ADCs and DACs. By using a part with a 16-bit resolution, the ADC or

DAC's resolution can be decreased as a means of trial-and-error experimentation until the decreased bit width begins to affect performance. Ultimately, honing on the right resolution can yield the right implementation thus eliminating the possibility of producing an over- or under-designed implementation.

(v) Since it is not well understood how dither affects the tracking effectiveness of the MPPT algorithm, an FPGA platform, being highly flexible, can be used to conduct significant experimentation and testing on new dither variants as alternatives to purely sinusoidal dither as proposed in the Hybrid MPPT algorithm [10]. Among these variants, triangular dither will be tested to see how it handles the noise in the channel [18].

(vi) By using the available computational resources and flexible programmability available in FPGA chips, a series of digital filters will be tested on the inputs in order to improve the effectiveness of the MPPT algorithm. It is anticipated that a in a noisy environment, digital filters can reduce the amount of noise fed into the algorithm thus allowing cleaner samples and requiring a smaller amplitude dither.

(vii) In the overall, this thesis seeks to build a case for the of use of FPGAs in power control applications by demonstrating how their reconfigurability, customizability, and low cost allow them to be ideal for a wide range of implementations.

1.8 Thesis Outline

Chapter 2 discusses previous work related to MPPT control and explains some historical developments while chapter 3 describes the concepts behind MPPT, and the relevant additions that can improve MPPT control. Chapter 4 provides detailed information about the proposed FPGA implementation of the MPPT algorithm while chapter 5 presents the obtained experimental results. Finally, chapter 6 summarizes the findings in this thesis and provides directions for future research.

CHAPTER TWO: RELATED WORK

In this chapter, an overview of previously proposed MPPT implementations is presented. Previous MPPT studies are based primarily on implementations based on DSPs and microcontrollers although implementations based exclusively on analog components have been proposed also. A set of criteria, which can give an MPPT design noticeable presence in the commercial market if the latter meets the former, is proposed as a basis for evaluating the approach proposed in this thesis and previously proposed approaches. The chapter concludes by a summary of this evaluation based on these criteria.

In [19], the authors present an MPPT design for parallel-connected PV arrays. The design is verified with simulation in which the obtained results show that it can operate with non-ideal and non-linear sources. By monitoring the rates of change in both the average input current and average input power from the source, they propose to dynamically regulate the dc-dc converter system to track the peak power point of the source. Although the design topology is carefully laid out, the paper does not provide any implementation details. Moreover, the proposed design consists of entirely analog components, which can easily be affected by the presence of noise in the operating environment.

The authors in [20] present a power supply for the Stellenbosch University Satellite with a strong emphasis on MPPT. Due to the requirement stating that a satellite must be as light as possible in order to carry the maximum payload, MPPT becomes important in

this context in order to ensure that PV arrays are maximally used. The authors propose a controller-centered design based on a UC1846 IC chip and a converter. Their design uses the I-V characteristics of a PV array with positive feedback to achieve MPPT control. Although the proposed design is highly suitable for satellite applications, it lacks robustness and subsequently cannot be expanded to parallel-connected converters.

In [21], the authors propose a microcontroller-based solution intended for PV applications and alternate energy sources such as wind generators. The proposed design is relatively low in cost. However, it is not sufficiently elaborate as designs proposed very recently [10, 22]. While the authors argue for a low power solution, the Intel 80C196KC microcontroller they selected to implement their design does not lead to a low cost implementation, nor does it offer the flexibility of comparably-priced DSPs. As such, it is not highly adaptable to the changing requirements of the growing PV industry.

In [11], the authors propose to implement a basic MPPT algorithm by using a Texas Instruments TMS320C25 DSP intended primarily to provide a low cost solution. The authors' MPPT approach is based on the Perturb-and-Observe algorithm. In the paper, they use the principle of energy conservation to derive small- and large-signal models for analysis and simulations which ultimately correspond well to experimental results. Although the authors suggest possible extensions to increase the performance of their design by using high sampling rates, they do not provide sufficient details on how it can be achieved. Given their suggested sampling rates, most low-end DSPs will not be adequate for their demanding application.

In [23], the authors propose an MPPT algorithm based on a design in which a buck converter is coupled with a PIC16C72 controller. Implemented in this controller are series batter voltage regulation, sensorless short circuit protection of the converter, and intelligent shutdown and wakeup at dusk and dawn. Whereas the design is intended to be a low cost solution, arrays of no more than 110W are tested with the controller. To expand power handling, the authors suggest a configuration of parallel multiple controllers. However, such a configuration would seem inadequate to handle a demand of several Kilowatts. In addition, the authors do no provide any description of components necessary to support current sharing.

In [10], a significantly robust system is proposed using the Perturb-and-Observe and the Incremental Conductance algorithms. The system design is based on a 500W-capable, dual-channel controller in which a Texas Instruments TMS320C24 DSP handles both channels. The design uses commercial-off-the-shelf DC-DC converters under DSP control with a novel current-sharing concept to simplify the control requirements. The papers explores at length current sharing via paralleling without addressing potential opportunities to improve the basic design. In addition, no discussion was provided about the utilization of the DSP's internal hardware for A/D and D/A conversion. This utilization necessitates an increase in control hardware beyond handling two channels. If the DSP-based design is expanded to a multiprocessor configuration to handle multi-channel control, additional components such as busses have to be integrated within the expanded configuration. Using busses for communication add complexity to inter-

processor communication channels and tax these processors by stripping a large portion of their clock cycles from computation and sacrificing it to communication. At the end, a conclusive evidence on the adequacy of DSPs in handling multi-channel MPPT control is not provided.

While these previously proposed approaches represent significant findings on MPPT algorithms, they do not offer convincing arguments with regard to their suitability for the commercial market. It is expected that future markets will likely drive the need for ultra-high efficiency and massively-paralleled MPPT solutions. Based on this rationale, this thesis presents an FPGA-based implementation of a known MPPT algorithm. By using an FPGA implementation, the MPPT control becomes scaleable through the selection of an FPGA device with a suitable logic capacity. As the MPPT control scheme is expanded to handle multi-channels, an FPGA chip with a larger capacity can be used in order to house the logic necessary to implement multiple instances of the MPPT algorithm used for controlling these channels. Packing multiple algorithm instances within the same device completely eliminates the need for any bussing structure on the prototyping board, thus reducing hardware requirements and reserving FPGA clock cycles solely for computing. Even in the worst case where multi-channel control may require more than a single FPGA chip, several chips can be (i) wired directly to each other without any bus in between [24], or (ii) connected to each other through a programmable interconnection chip [25]. The latter chips are similar to FPGA chips in reconfigurability with the exception that they are exclusively used for routing rather than computing. Both interconnection approaches streamline the resulting implementation,

simplify the testability, and increase the expandability of the design. These considerations make the case for investigating the full potential of FPGAs in power control.

To offer a reasonable comparison of these approaches with the approach proposed in this thesis, a set of criteria is put forth to motivate the comparison. This set consists of the following criteria:

(i) *Cost*: How far does an MPPT design meet the low cost constraints demanded in the commercial market? The penetration of MPPT control technologies in this market depends heavily on their cost.

(ii) *Performance*: What performance is the MPPT design capable of providing? Future control applications may require nimble and swift power control.

(iii) *Robustness*: Is the MMPT design capable of operating in a wide range of environmental conditions such as noisy environments? Robustness is highly valued in the commercial market.

(iv) *Flexibility*: Can the MPPT design be easily upgraded or modified to meet a new design constraint? New efforts of research and development may require the design to be quickly and easily modified.

(v) *Expandability*: Can the MPPT design be expanded to accommodate increasing control requirements? It is anticipated that future MPP control systems will require handling of multi-channels.

(vi) *Testability*: Can the MPPT design be easily tested, verified, and validated against a set of requirements? The commercial market is highly aware that testability has a significant impact on cost.

In this thesis, it is believed that meeting these criteria by a given MPPT approach can boost its potential in gaining some foot ground in the commercial market. To this end, Table 1 shows a summary of the previously proposed MPPT approach and the approach proposed in this thesis based on the listed criteria.

Table 1: Summary of previously proposed MPPT approaches.

	A	**U**	**T**	**H**	**O**	**R**	**S**
Evaluation Criterion	Siri et. al	Buekes et al.	Koutroulis et al.	Hua et al.	Swiegers et al.	Wu et al.	This thesis
Cost	N/A	Low	High	High	Low	High	Low
Performance	Low	Low	Low	Low	Low	Low	High
Robustness	Low	High	Low	High	High	High	High
Flexibility	Low	Low	High	High	Low	High	High
Expandability	High	Low	Low	Low	Low	High	High
Testability	Low	Low	High	High	High	High	High
Comments	No proposed hardware	Compact Design Application Specific	Multiple Applications High Power	Low Power	Simple Design	Multiple Channels High Power	Multiple Channels Multiple Applications

The remaining chapters in this thesis present a design which is based on a popular MPPT algorithm and is highly suitable for a commercial market given its cost savings, its robustness, its flexibility, and its expandability to handle multi-channel control.

CHAPTER THREE: MPPT SYSTEM FUNCTIONALITY

This chapter presents an overview of two basic MPPT algorithms, namely the *Incremental Conductance* and the *Perturb-and-Observe* algorithms in addition to tradeoffs considerations to be made when designing an MPPT control system. Section 3.1 introduces the MPPT algorithms while section 3.2 discusses how noise can be reduced. Section 3.3 shows how to handle dither to improve the performance of the MPPT algorithm while section 3.4 summarizes the chapter.

3.1 Basic MPPT Algorithm

MPPT can be performed by using any algorithm which can adequately detect and follow the maximum power output of the solar array. Therefore, numerous algorithms have been proposed to support MPPT. This thesis considers the case of two algorithms, namely the Incremental Conductance (IncCond) algorithm and the Perturb-and-Observe (P&O) algorithm [11]. Others proposed approaches based on fuzzy logic, neural networks, and pilot cells are in general too complicated and expensive to implement. However, it is anticipated that they may become cost-effective to control power in larger PV arrays [12].

3.1.1 The Incremental Conductance Algorithm

The IncCond algorithm tracks the maximum power point by comparing the incremental and instantaneous conductances of the PV array. By sampling changes in the array voltage and current, the incremental conductance can be varied [11]. This essentially

correlates to finding the point where $\frac{dP}{dV} = 0$ without applying a dither to the system. The IncCond algorithm is advantageous because it has no oscillation about the power point, thereby increasing efficiency. In general, it tracks the maximum power point effectively under rapidly changing conditions. However, this algorithm is usually disregarded as requiring too much computation for the amount of time allotted, thereby reducing the sampling frequency of the system [12]. Figure 11 shows the flowchart of the IncCond algorithm.

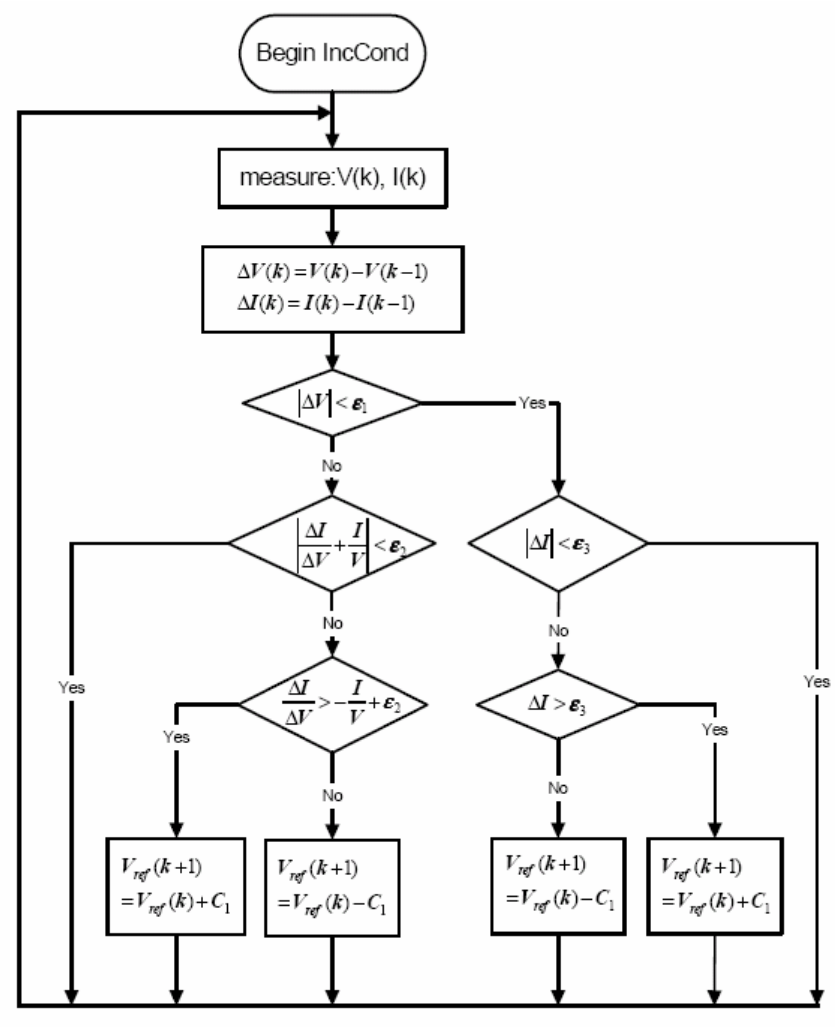

Figure 11: Flowchart of the incremental conductance algorithm [10].

3.1.2 The Perturb and Observe Algorithm

Although it is regarded as less efficient than the IncCond algorithm in some studies, the P&O algorithm is the most commonly used algorithm for MPPT [10, 12]. The P&O algorithm illustrated in this section is based on the MPPT system developed in [10]. Essentially, a traditional hill-climbing approach is used to track the peak power output. Based on this point, it is known that on the left hand side of the curve, $\frac{\Delta P}{\Delta V} > 0$. However, on the right hand side of the curve, $\frac{\Delta P}{\Delta V} < 0$. Therefore, this differential will only be equal to zero at the peak of the power curve. Whenever the controller drifts away from the maximum, the reference voltage can be adjusted by a constant factor in the opposite direction. This only requires the values for voltage, current, and power from the previous iteration in order to calculate the differentials. This yields the flowchart in Figure 12. Although there is no agreed upon ways to evaluate the performance of an MPPT algorithm, the effectiveness of the MPPT will be computed by

$$\eta_{MPPT} = \frac{\int_0^t P_{measured}(t)dt}{\int_0^t P_{actual}(t)dt} \qquad (3.1)$$

This equation can serve as a straightforward yet simple metric. Figure 12 shows the flowchart of the P&O algorithm.

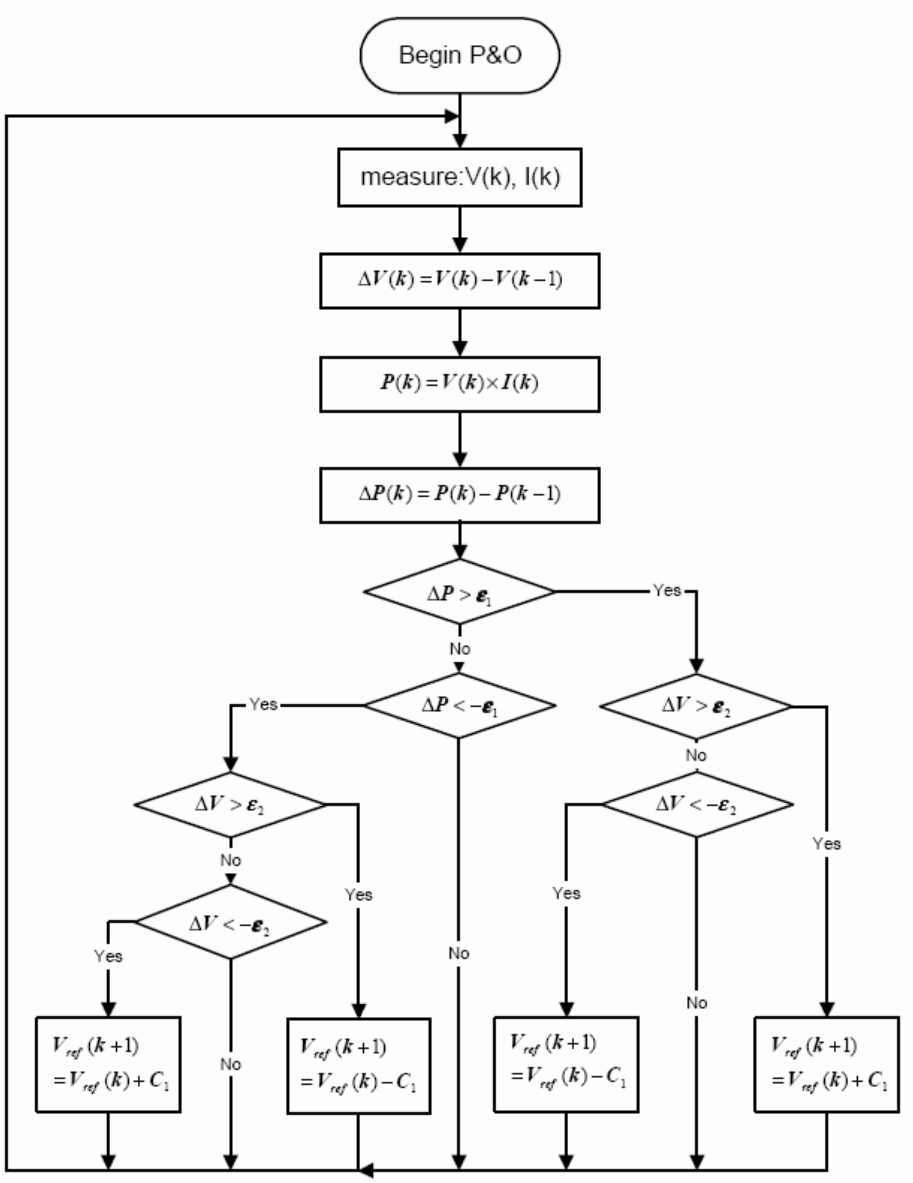

Figure 12: Perturb-and-observe algorithm flowchart [10].

3.2 Noise Cancellation and Reduction

Since MPPT implementations are usually part of larger systems containing DC-DC converters, a considerable amount of noise can be picked up. This can potentially cause

the system to track in the wrong direction or detect a false change in position. Therefore, several stages of constraint checking help to lower the effects of noise.

The V-I curve for a solar array has an intrinsically negative slope. Therefore, if a value to the contrary is seen, then it can be discarded as being corrupted by noise. So, for each value received, the algorithm checks to make sure that $\frac{\Delta V}{\Delta I} < 0$. Secondly, since this system is voltage-driven, namely by V_{ref}, it is generally expected that voltage changes will be above a certain threshold. In addition, the 16-bit ADC used in the implementation provides a quantization that is approximately 76.295μV, a value that is most likely lower than the noise level. Therefore, a minimum value is defined and the algorithm ensures that $\Delta V \geq V_{\min}$. In the event that the array goes to the right-hand side of the curve, where the current decreases rapidly, it is possible for the power output to collapse to zero. Since it is unlikely that this is desired, a minimum current threshold is declared such that $I > I_{\min}$. Finally, due to the fact that the system cannot operate below a certain voltage, a check is performed on V_{ref} to enforce this minimum value. The algorithm will keep $V_{ref} > V_{\min}$ at all times. This minimum value will also be used as the initial value of the reference voltage.

3.3 Improving Dithering to Improve MPPT Performance

The main focus of the P&O algorithm lies within the dither used to perturb the control state. By adding a dither to the signal, the Signal to Noise Ratio (SNR) is increased. According to the authors in [18], "dither is a large suitably fluctuating component added

to randomize the quantization process leading to a definable set of transition probabilities for the quantizer." In addition, it provides a predicable signal upon which MPPT can be performed. In this implementation, the dither signal is added to the reference voltage whenever it is detected that the load is demanding maximum power. Figure 13 shows the effect of dither.

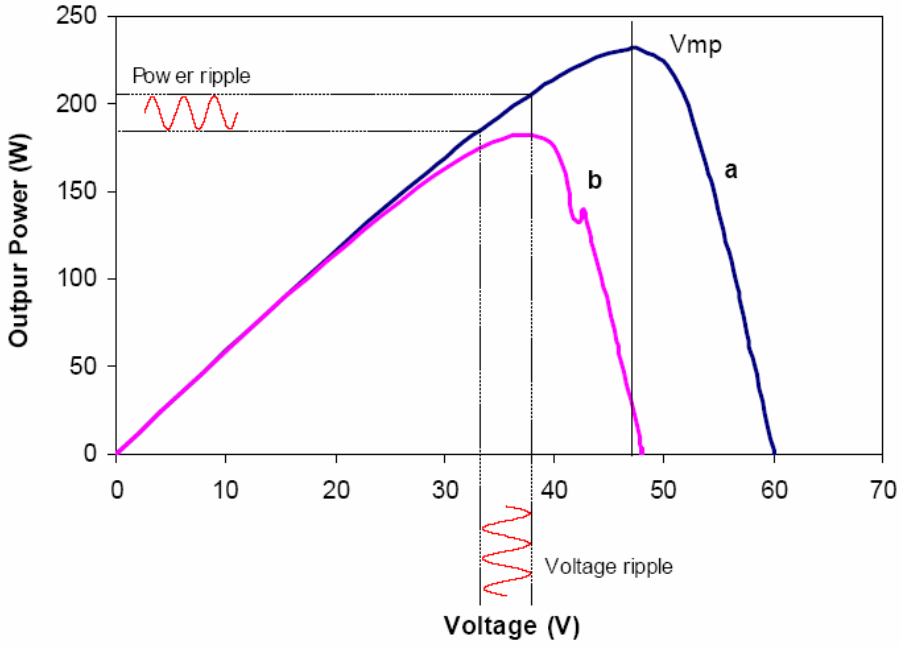

Figure 13: An illustration of the dither operation [10].

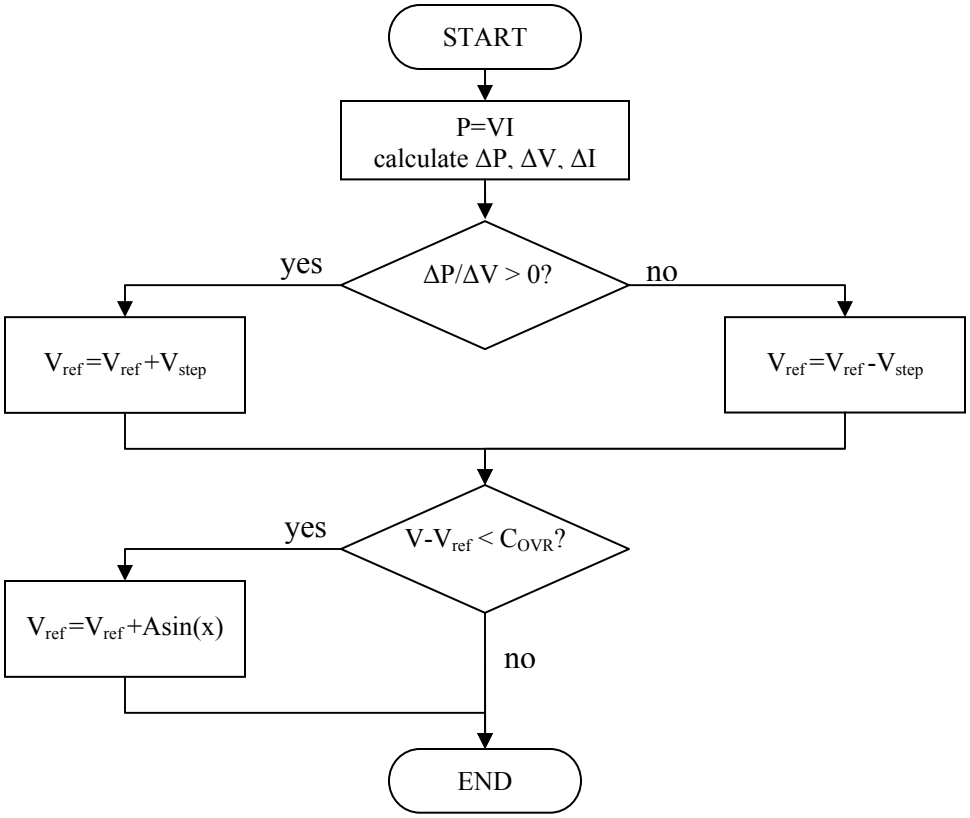

Figure 14: MPPT algorithm flowchart with dither.

Because of the approach used by the MPPT algorithm, the dither is only needed when the system is attempting to track maximum power. There are times when the load requires only a portion of the available power, and MPPT is not necessary. In this case, the present output voltage of the array will exceed V_{ref} by a pre-determined threshold C_{OVR}. When this condition, called Output Voltage Regulation (OVR) is met, the dither signal is no longer added to the reference voltage until MPPT is invoked again.

However, there exists an ideal value for dither in a given system in order to reach a specific state. Unfortunately, the addition of dither introduces inevitably a fluctuation in

the power point with time. Although the dither signal may be completely necessary for the MPPT algorithm to work, it is still a deviation from the maximum power output, and therefore is a power loss. Chapter 4 of this thesis addresses some methods by which the ideal dither amplitude can be reduced so that this power can be regained.

In some rare instances, a PV array can exhibit multiple maxima on its power curve. A modified algorithm in [10] describes a method of dithering where the amplitude is periodically increased. This can serve to prevent the algorithm from sticking on false local maxima. In addition, this allows the overall dither amplitude to be decreased as low as noise allows without being concerned about the effects of these false peaks.

3.4 Summary

This chapter discussed details about the IncCond and the P&O MPPT algorithms with special emphasis on the latter. Since the P&O algorithm relies on dithering to improve its MPP, it discusses some general approaches by which dithered can be controlled to improve the performance and efficiency of the algorithm. These methods will be tested in chapter 4 of this thesis.

CHAPTER FOUR: MPPT IMPLEMENTATION

This chapter describes the system architecture of the proposed MPPT controller and its FPGA implementation. Section 4.1 introduces the system architecture while section 4.2 describes the blocks of the architecture. Section 4.3 presents the simulation results of the VHLD models of the architectural blocks while section 4.4 explains the synthesis results by mapping the architectural blocks onto an Altera FLEX 10K FPGA chip. Section 4.5 demonstrates the steps taken towards realizing a 4-channel implementation of the FPGA controller. Section 4.6 summarizes the chapter.

4.1 System Architecture

The architecture of the MPPT system consists of a pipelined structure that facilitates efficient data flow from a starting point, represented by the ADCs, to a terminal point represented by the DACs. This pipelined architecture, shown in Figure 15, allows its architectural blocks to be processing in parallel, yielding much higher utilization.

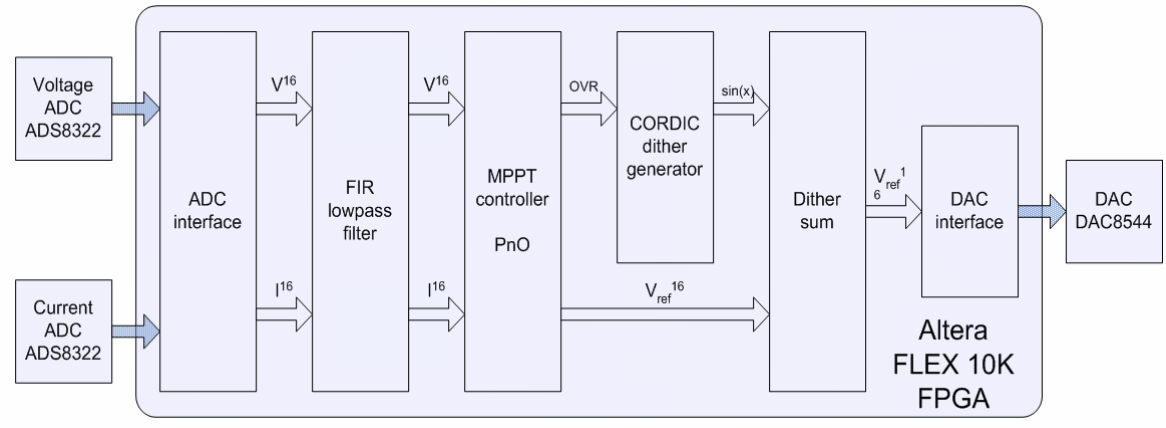

Figure 15: Pipelined system architecture.

The P&O algorithm runs on an Altera FPGA which embedded on an Altera UP2 development board. This board contains a Flex 10K70 FPGA with approximately 70,000 equivalent gates. The original oscillator of this board was replaced with another oscillator operating at 10 MHz. A daughterboard containing the necessary ADCs, which are single-input ADS8322, and DACs was designed to interface with this development board and the main DC-DC converter. A photograph of the development board and daughterboard is shown in Figure 16. The DACs are Texas Instruments DAC8544 with 4 separate outputs, capable of converting 16-bits within a maximum settling time of 10μs. The ADCs selected for this architecture are 16-bit Successive Approximation Register (SAR) devices from Texas Instruments designed to operate at a maximum of 500,000 samples per second. Beside the capability of handling parallel data, these components were selected to strike a balance of cost, speed, and ease of implementation. A list of components and approximate costs is shown in Table 2.

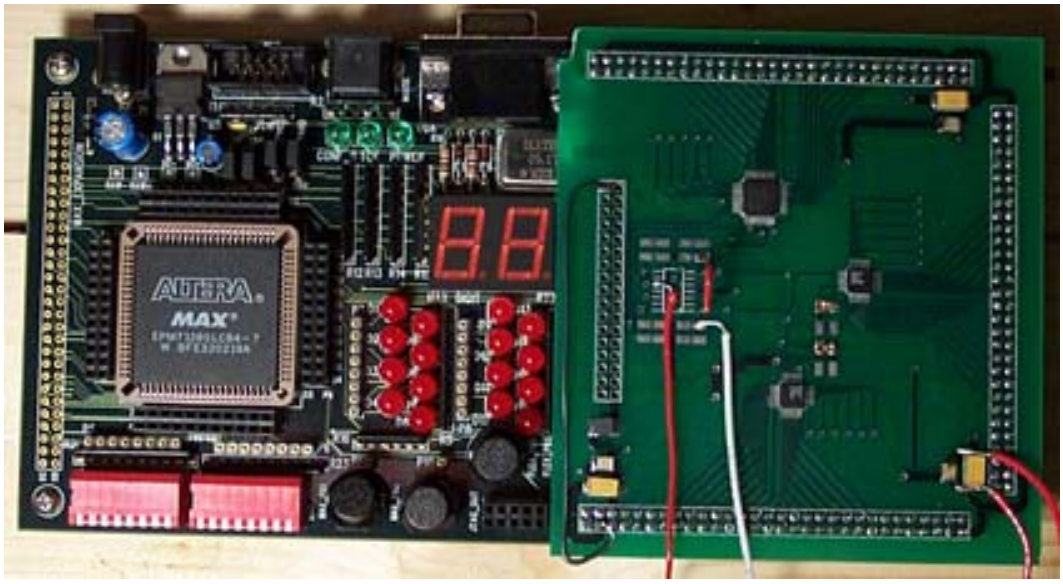

Figure 16: Photograph of FPGA test hardware.

Table 2: Components used for this design and associated costs.

Component	Approximate Price ($)
Development Board	
Altera EP1C3 FPGA	16.40
10.000 MHz oscillator	1.88
Daughterboard	
Texas Instruments DAC8544	16.24
Texas Instruments ADS8322 (2)	23.66
Passive Components (Resistors, Capacitors, etc.)	5.00
PCB Fabrication	25.00
TOTAL	88.18

4.2 System Blocks

This section describes briefly the functionality and specifications of each block in the overall system architecture.

4.2.1 ADC Interface

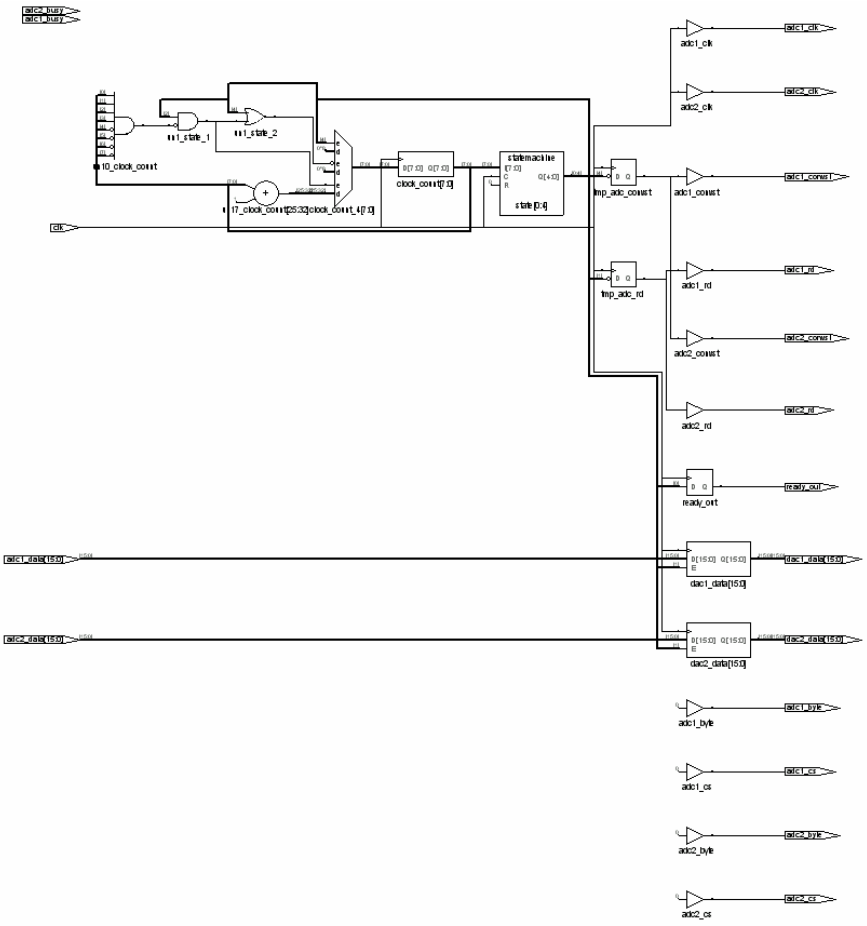

Figure 17: ADC interface block.

Because the ADC used is an SAR device, it requires one clock cycle per bit in addition to other cycles to convert and read data out of the device. Therefore, a minimum of 20 cycles are required per sample, so the 10 MHz oscillator yields the maximum sampling rate of 500 kilosamples per second. Once the data is received, a ready signal is generated for the next block (digital filter or averaging) to begin processing. The VHDL code of this block is shown in Appendix A.1.

4.2.3 Average with Downsampling

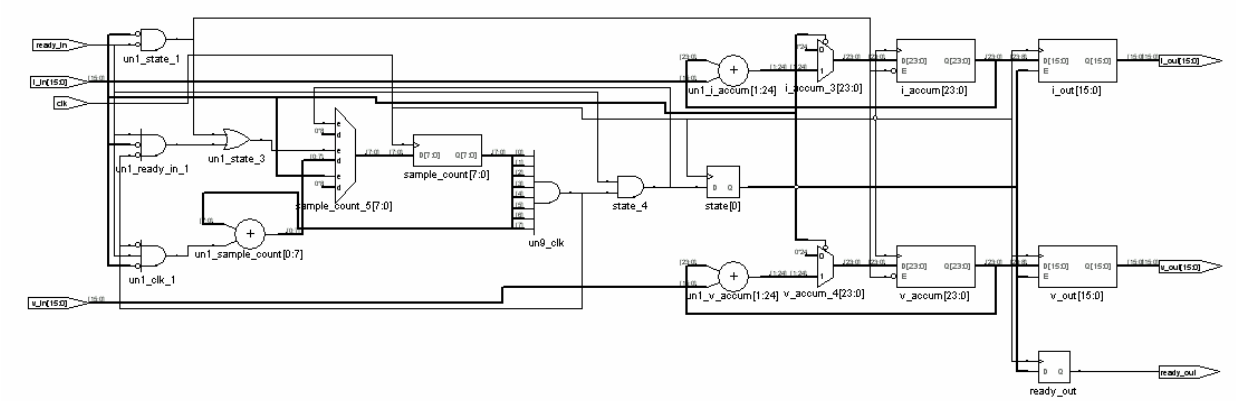

Figure 18: Downsampling block.

The averaging block accumulates 256 samples and computes their average. Essentially, this block functions as a digital filter with a rectangular window. This filter was used in [10] along with a 3rd order IIR. The benefit of using averaging is the easy with which one it can be calculated without requiring any coefficients. In addition, it can be implemented with minimal hardware in the event that the downsampling rate equals the number of samples in the average.

4.2.4 FIR Digital Filter with Decimation

An FIR filter consists of iterative multiply-and-accumulate (MAC) operations as shown in Figure 19.

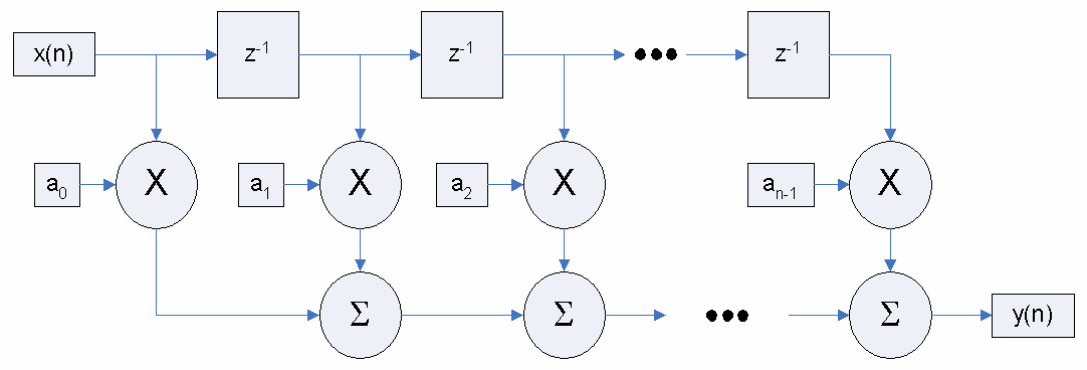

Figure 19: FIR Filter

Figure 20 shows the gate netlist of a 256-constant FIR filter. Similarly to the downsampling, the FIR filter in this system uses 256 samples. The benefit of using an FIR over an IIR is that the FIR has linear phase and is inherently stable, despite the fact that it requires a significantly higher order to achieve the same cutoff slope [26]. This filter is used to test the benefits of applying different windows to the incoming samples. The VHDL code of this block is shown in Appendix A.2.

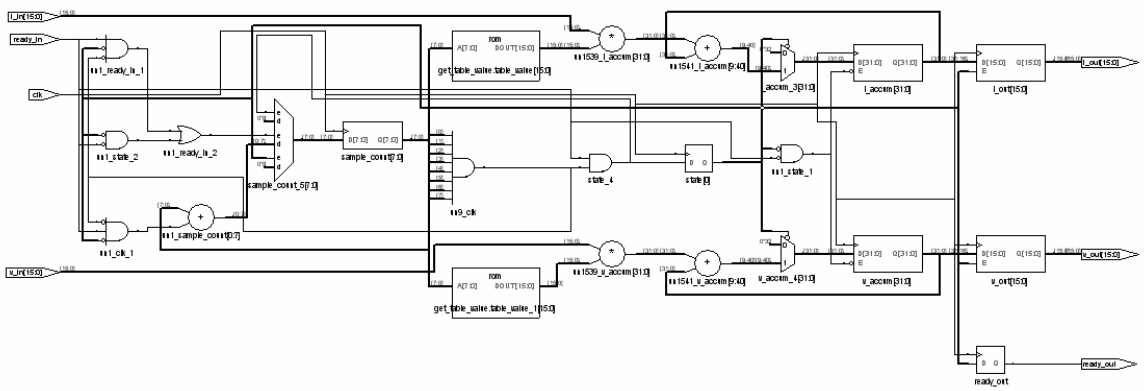

Figure 20: Gate netlist of FIR filter.

4.2.5 MPPT Controller

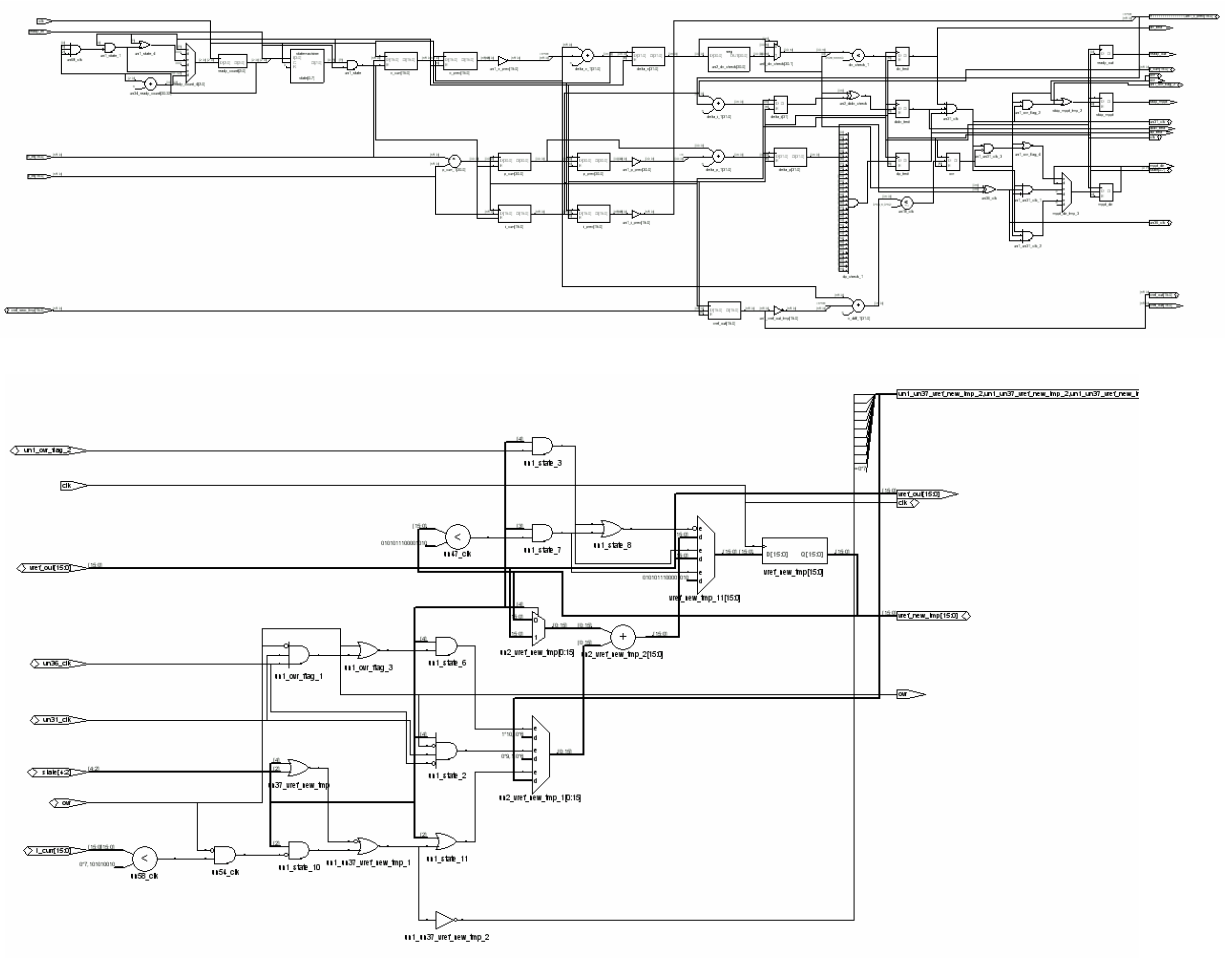

Figure 21: The MPPT controller block.

The P&O algorithm resides on the MPPT controller. This controller receives values for voltage and current, calculates all the necessary differentials and performs updates to V_{ref} after the necessary error checking has been performed. Also, it stores the power, voltage, and current values from the previous iteration. In addition, the MPPT algorithm also detects whether the system is operating in OVR or MPPT. OVR detection is enabled when the difference between the system voltage and V_{ref} exceeds a certain threshold. The

OVR flag is also used to disable dithering since it is only needed for MPPT. The VHDL code of this controller is shown in Appendix A.5.

4.2.6 CORDIC Sinusoidal Dither Generator

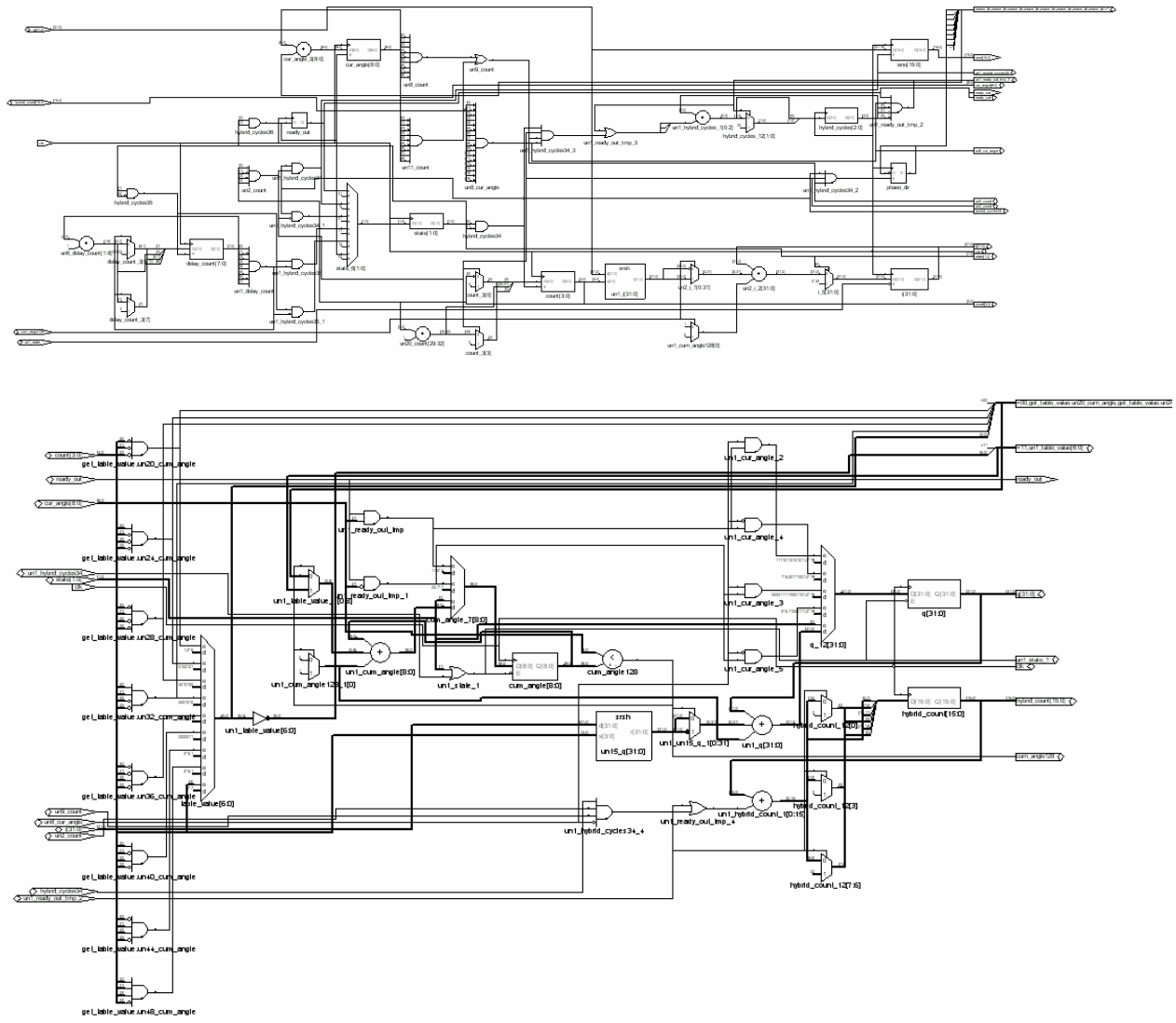

Figure 22: CORDIC sinusoidal generator.

In order to implement a sinusoidal generator in FPGA, the COordinate Rotation DIgital Computer (CORDIC) algorithm was selected. This algorithm uses a small lookup table

and a series of iterative calculations to determine values for sine and cosine in parallel. The accuracy of the algorithm is somewhat arbitrary, as more iterations and more precise coefficients can provide better results at the cost of speed and area. For this application, this proved to be a useful tradeoff, yielding very accurate calculations in only eight clock cycles. Its VHDL code is shown in Appendix A.6.

4.2.7 Reference Voltage Summation Block

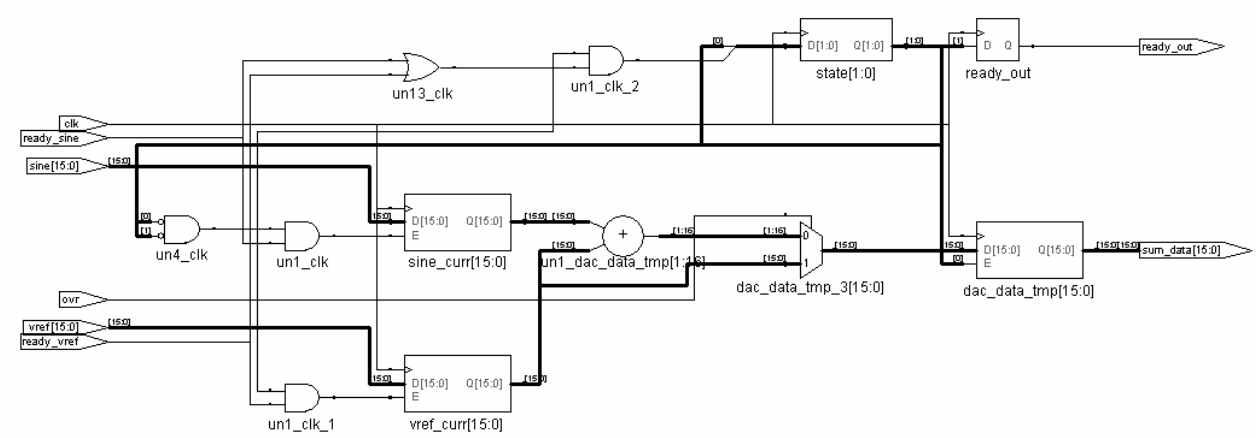

Figure 23: Reference summation block.

This block receives the value of the sinusoidal dither and V_{ref}, and adds them together. If the system is in OVR, then V_{ref} is not modified. The VHDL code of this block is shown in Appendix A.8.

4.2.8 DAC Interface

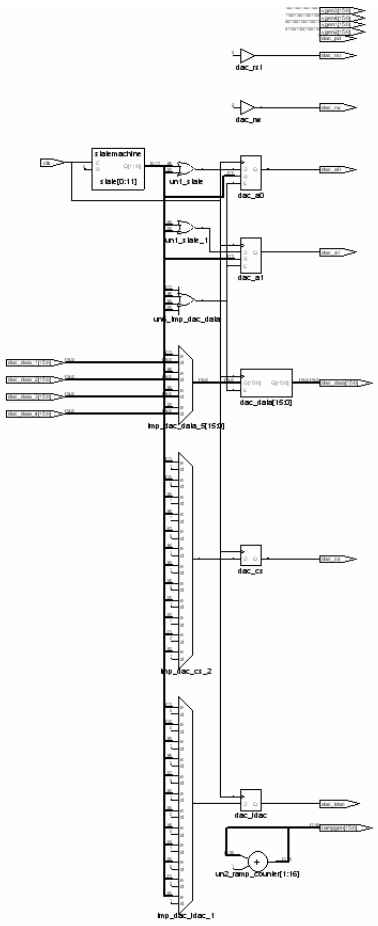

Figure 24: DAC controller.

The DAC interface receives up to four 16-bit inputs, and continually updates each channel on the quad DAC as data is ready. It also has references of 1V, 2V, 3V, and 4V along with a ramp generator for testing. The VHDL code of this controller is shown in Appendix A.9.

4.3 Simulation Results

To verify the system architecture, simulations were performed using Altera's Quartus II Web Edition, version 4.1 SP2. Each block was simulated for proper functionality before being tested in-circuit. Figure 25 shows the timing waveforms for the ADC controller, which controls both the current ADC and voltage ADC. Since the ADCs are Successive Approximation Register (SAR), a block is required for each bit sampled plus overhead. Therefore, 20 clock cycles are required to receive a value from the ADC. The clock is represented by clk, adc1_clk, and adc2_clk, while the signals adcX_busy, adcX_byte, adcX_convst, adcX_cs, and adcX_rd represent the busy flag, 8/16-bit data width flag, the conversion start flag, the chip select flag, and the read flag. Each ADC receives the same control signals. The sampled data is shown as dacX_data.

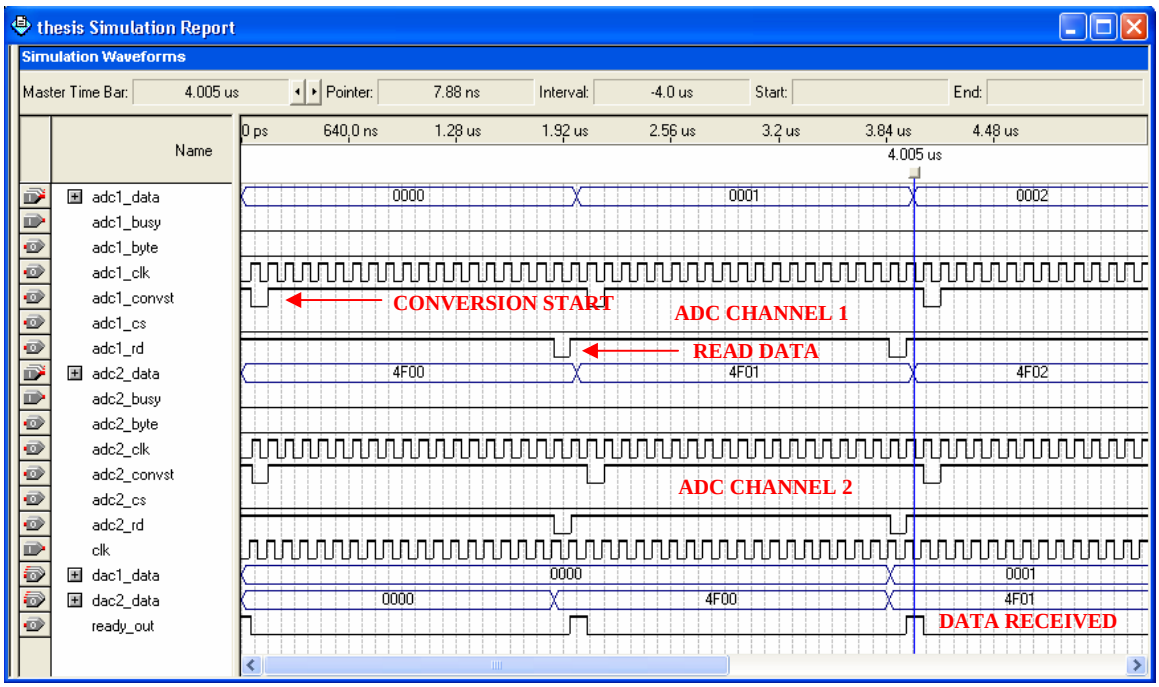

Figure 25: Simulation results of ADC interface.

Figure 26 illustrates the simulation of the MPPT controller, the unit responsible for making the decisions regarding tracking. For each clock cycle that ready_in is high, the voltage and current are read, and the differentials are processed in order to make decisions about whether or not to track, and in what direction. The signals didv_test, dp_test, dv_test, mppt_dir, skip_mppt, and ovr are internal flags based upon each calculation and provide information about the decisions made by the algorithm. The modified V_{ref} is sent out on vref_out.

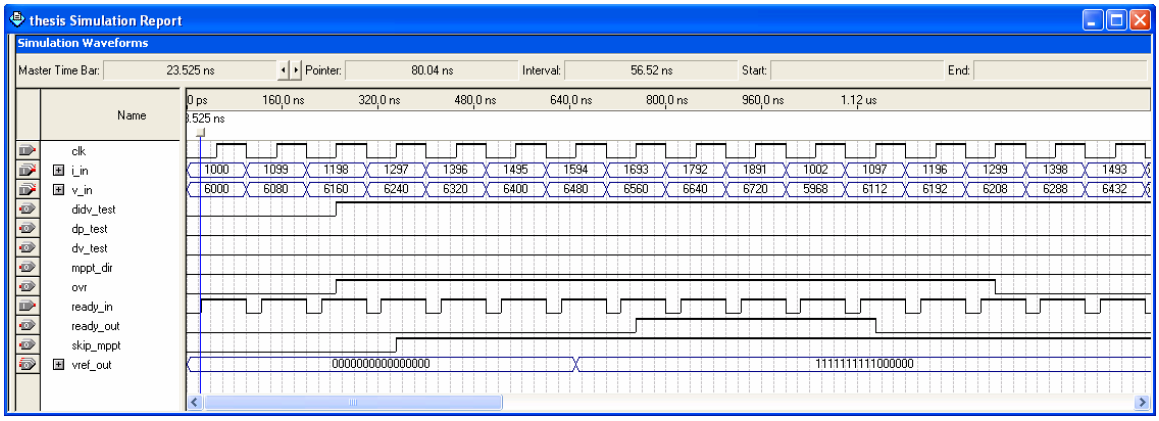

Figure 26: Simulation results of MPPT controller.

Figure 27 shows the simulation of the sinusoidal dither block, which performs by using the CORDIC algorithm. A specific number of clock cycles is required to generate a value of sin(x), at which point the algorithm raises the ready_out flag to signify that the data is ready. The algorithm also contains a delay constant which determines the output frequency of the dither.

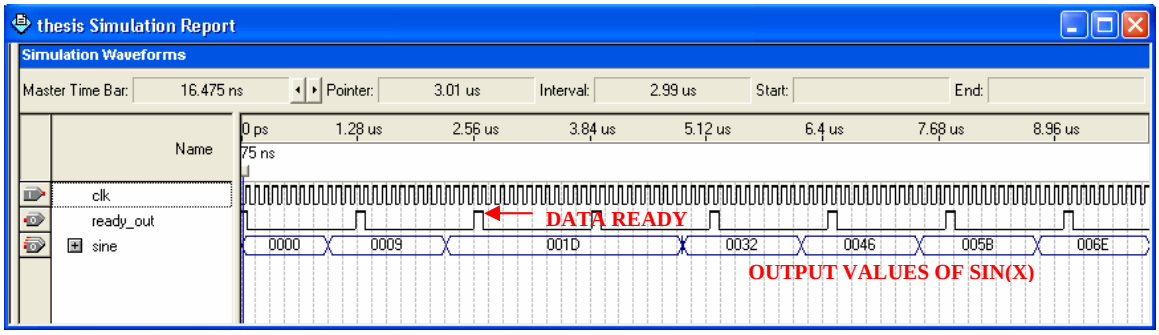

Figure 27: Simulation results of CORDIC sine generator.

Figure 28 contains the simulation data for the quad-output DAC controller. The internal voltage references and ramp generator are labeled as vgen1-4 and rampgen, respectively. The inputs to the block are dac_data_1 through dac_data_4. These inputs are sent to the DAC on dac_data in a sequential fashion, and the targeted DAC output is selected using address lines dac_a0 and dac_a1. Three clock cycles are required for a single output, therefore all four updates require 12 clock cycles. The remaining control signals dac_cs, dac_ldac, dac_pd, dac_rst, dac_rw represent the chip select, load DAC command, power-down, device reset, and read/write select flags, respectively.

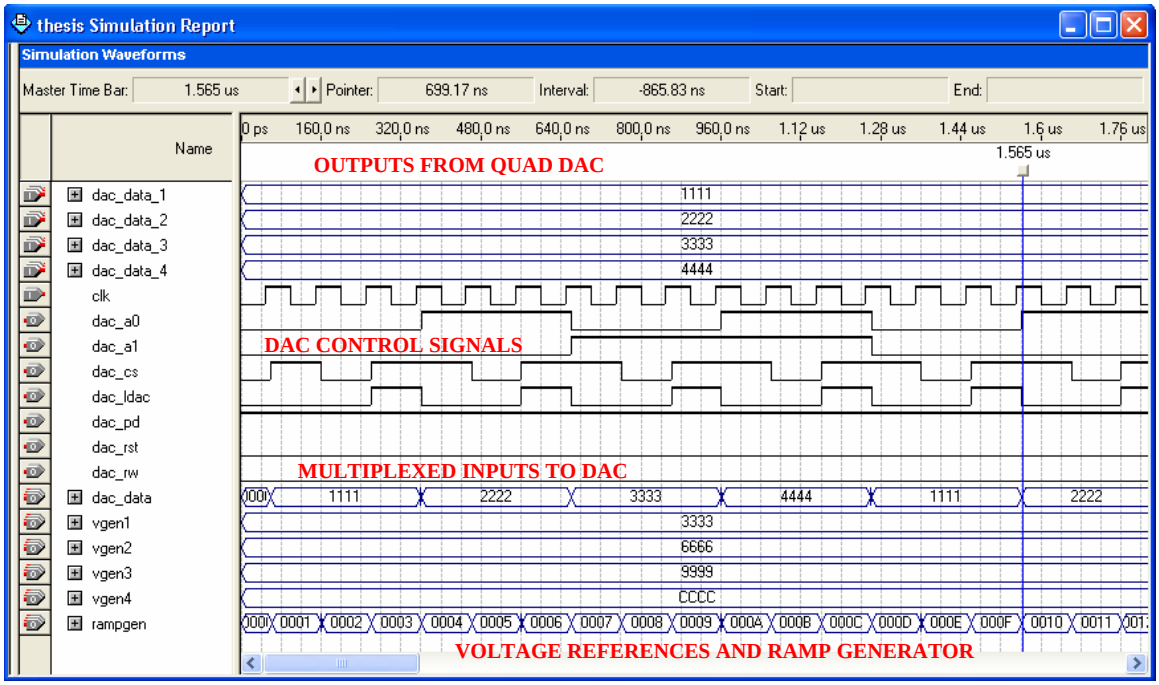

Figure 28: Simulation results of DAC interface.

Figure 29 illustrates the functional operation of the dither summation block. Essentially, this block performs gating of the dither in the event that the system is in OVR, indicated by the ovr flag. When valid data is ready from the dither generator, ready_sine should be high, and the value placed on sine. Likewise for V_{ref}, ready_vref indicates that data is ready on vref. The summation is shown on sum_data, and the ready_out flag goes high when the addition is complete.

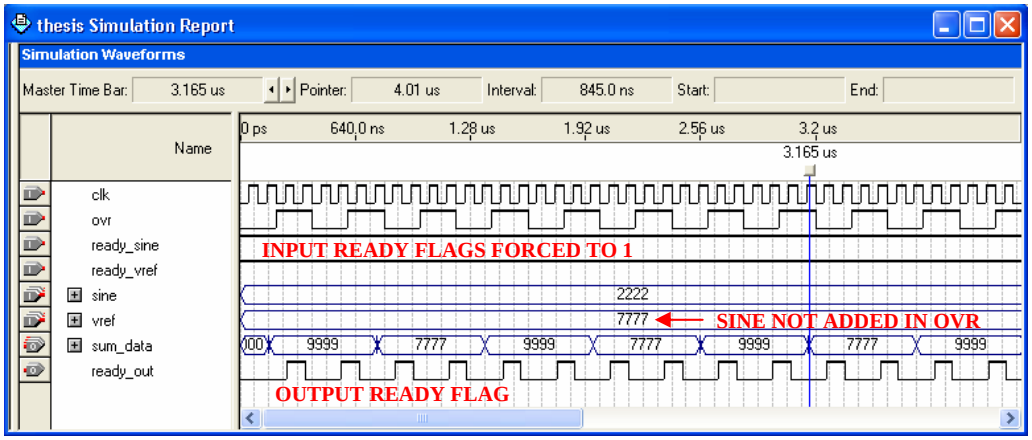

Figure 29: Simulation results of dither summation block.

The downsample block and FIR block share the same control waveforms, shown in Figure 30 and Figure 31. On each rising clock edge, if ready_in is high then i_in and v_in will be read, and added to an accumulator. After 256 samples have been collected, the average or filtered result will be calculated and appear on i_out and v_out, respectively. This can be seen best in figure 31.

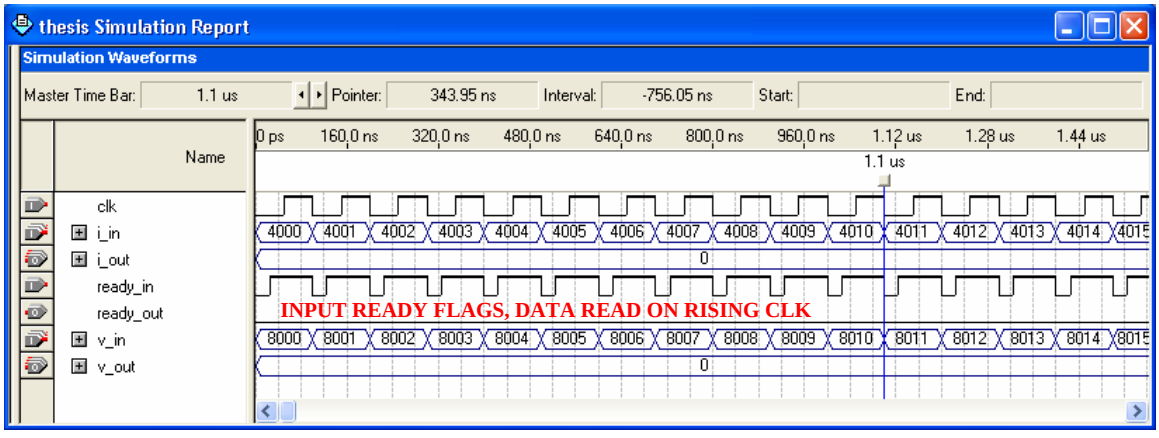

Figure 30: Zoomed- in simulation results of downsample/FIR block.

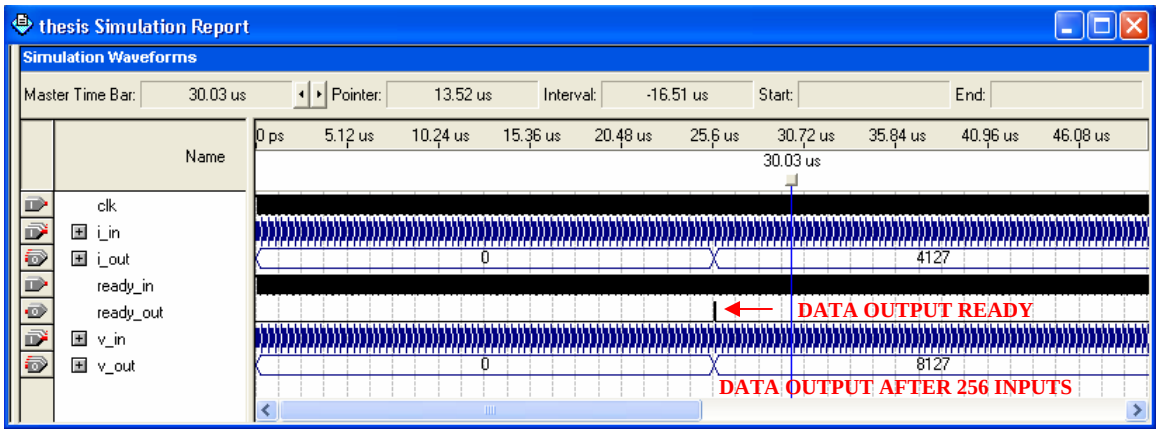

Figure 31: Zoomed-out simulation results of downsample/FIR block.

4.4 System Architecture Synthesis

After VHDL modeling, Altera's Quartus II Web Edition, version 4.1 SP2, was used to synthesize the different blocks in the system architecture. Table 3 shows the number of VHDL lines of code written to specify the functionality of each block, the number of LEs needed to implement each block in the FLEX chip, and the ratio of the implementation LEs to the available LEs in the actual device. The device utilization is also given as a reference. The utilization figures are based on the Altera FLEX 10K70 device, which contains 3,744 LEs.

Table 3: VHDL specification and synthesis of each block in the system architecture.

Module Name	Lines of VHDL	Logic Elements	Device Utilization (%)
ADC Controller	123	52	1.39
Downsampler	76	141	3.77
MPPT Controller	403	825	22.04
CORDIC	140 + 37	681	18.19
Dither Sum	82	70	1.87
DAC Controller	163	100	2.67
TOTAL	**1,024**	**1,869**	**49.92**

As shown in this table, only half of the logic capacity of the FLEX 10K is used. Although this device is a low capacity chip, it is largely sufficient to implement an entire MPPT controller. This result makes the case for using high capacity chips for implementing complex controllers such as the ones used for multi-channel based on intelligent computational approaches such as neural networks. In fact, this architecture can easily be expanded to multiple channels based on a simpler MPPT control algorithm. It has been shown that the total LE count used in the FPGA is low enough that multiple copies could easily fit inside a slightly larger device. Inter-controller communication fabric can remain entirely within the FPGA, decreasing costs and simplifying the design of the whole system. Figure 32 shows a conceptual architecture for four-channel MPPT control. The shaded area represents the FPGA inside of which are four separate MPPT algorithms running concurrently. Instead of being connected by an external bus, there exists a communications hub within the FPGA to manage the data between the individual units. Subsequently, each channel has its own ADCs and DAC to interface with the rest of the system.

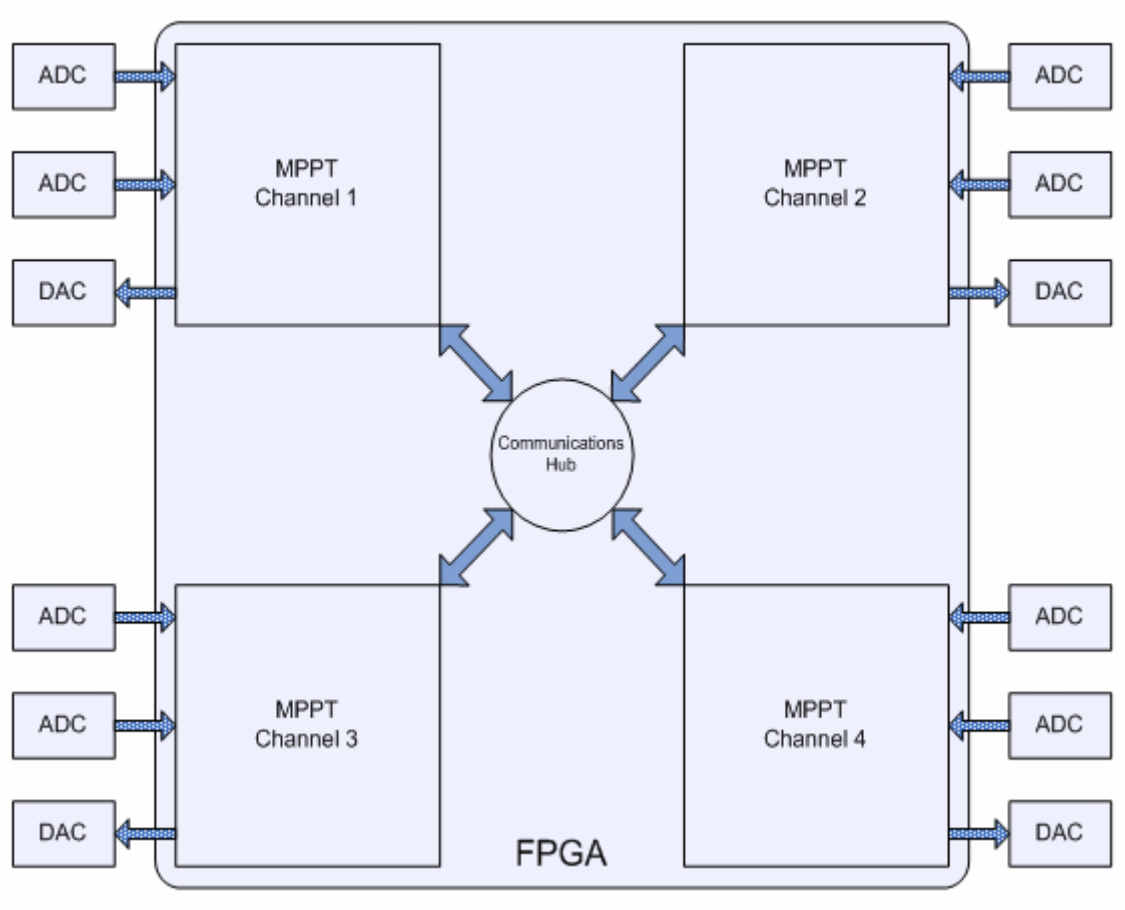

Figure 32: A four Parallel MPPT Channel architecture.

Table 4 shows the maximum delay and the resulting clock frequency of the synthesized implementation of each block in the system architecture.

Table 4: Block synthesized delays and frequencies.

Module Name	Maximum Delay (ns)	Clock Frequency (MHz)
ADC Controller	12.8	78.13
Downsampler	16.1	62.11
MPPT Controller	46.9	21.32
CORDIC	43.7	22.88
Dither Sum	13.6	73.53
DAC Controller	14.0	71.43

Default parameters of the synthesis tool were used to synthesize the blocks of the system architecture. No optimization has been performed while synthesizing these blocks. As the table shows, the slowest block is the MPPT controller with can run at a frequency of 21.32 MHz. This means that the remaining blocks in the system architecture can slow down in order to run at the same speed as the MPPT controller. Ignoring latency considerations outside the FPGA chip, the MPPT controller can produce a throughput of 21,321,961 power tracking points per second.

4.5 Four-Channel Realization

In order to obtain experimental verification of the four-channel implementation, efforts were put into designing and fabricating a new board for testing. Instead of using the FLEX 10K series, this board was designed to take an Altera Cyclone EP1C6 or EP1C12, containing approximately 6,000 and 12,000 LEs, respectively. Although these devices have a newer architecture, it seems likely that the LE usage will remain approximately the same. Therefore, we will need four times as many LEs to accommodate all the channels. If optimizations cannot make it possible to fit within 6,000 LEs, then the EP1C12 provides a simple migration path along with the ability to embed more complicated functionality with the added hardware. Figure 33 illustrates the first page of the board's schematic which mainly contains the FPGA and voltage regulation ICs. Figure 34 shows the second schematic page, where the ADCs and DACs appear, along with other miscellaneous components.

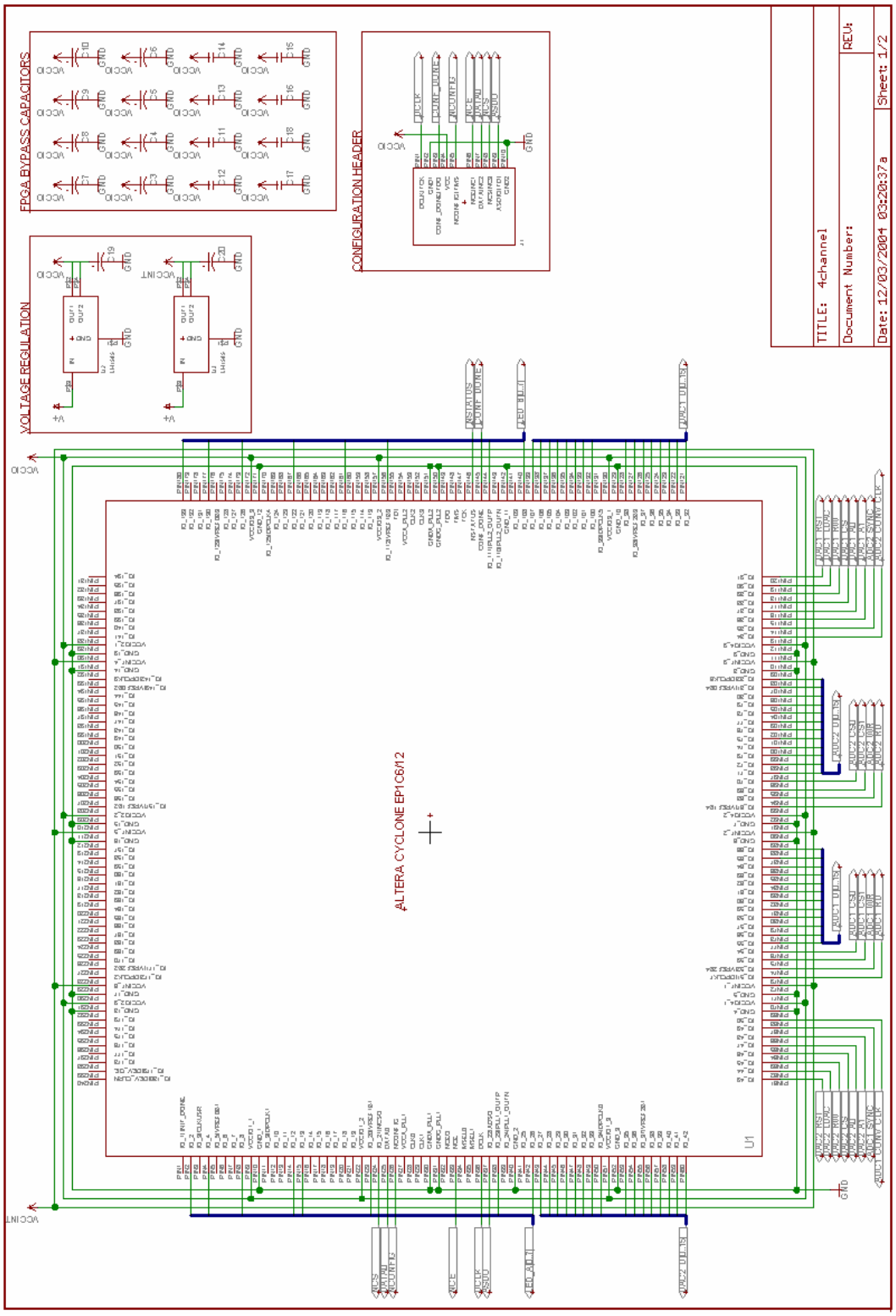

Figure 33: First schematic page of four-channel implementation.

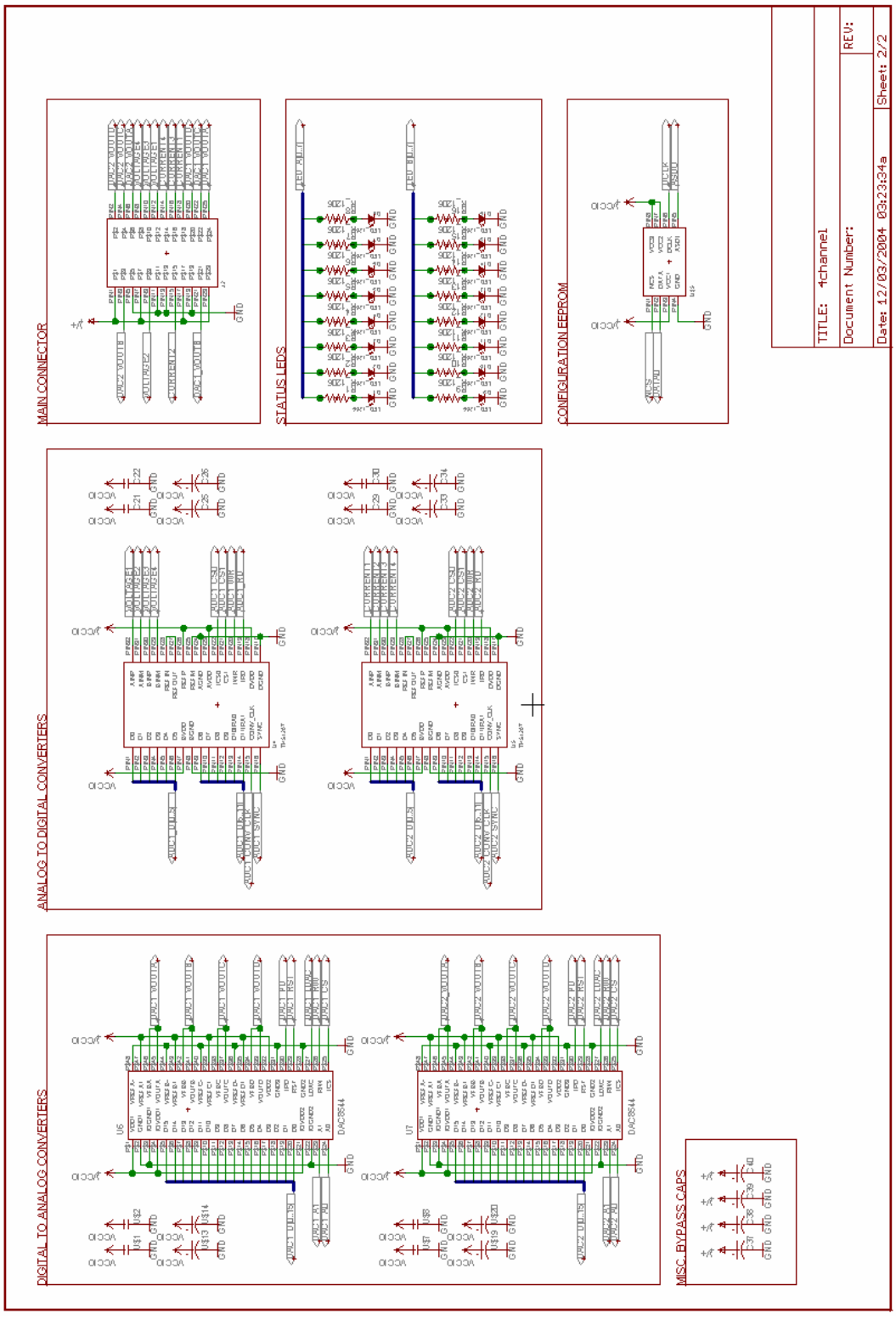

Figure 34: Second schematic page of four-channel implementation.

In order to help reduce noise picked up by the board, a four-layer design was used. This also greatly simplifies the distribution of power throughout the board and is a tremendous aid in keeping traces short. Figures 35 and 36 show the top and bottom copper layers of the board, respectively. 16 LEDs are included for debugging purposes and a 24-pin connector provides communication between the four power conversion boards. A serial EEPROM is also provided onboard for storing the FPGA configuration in order to program the device at power-on.

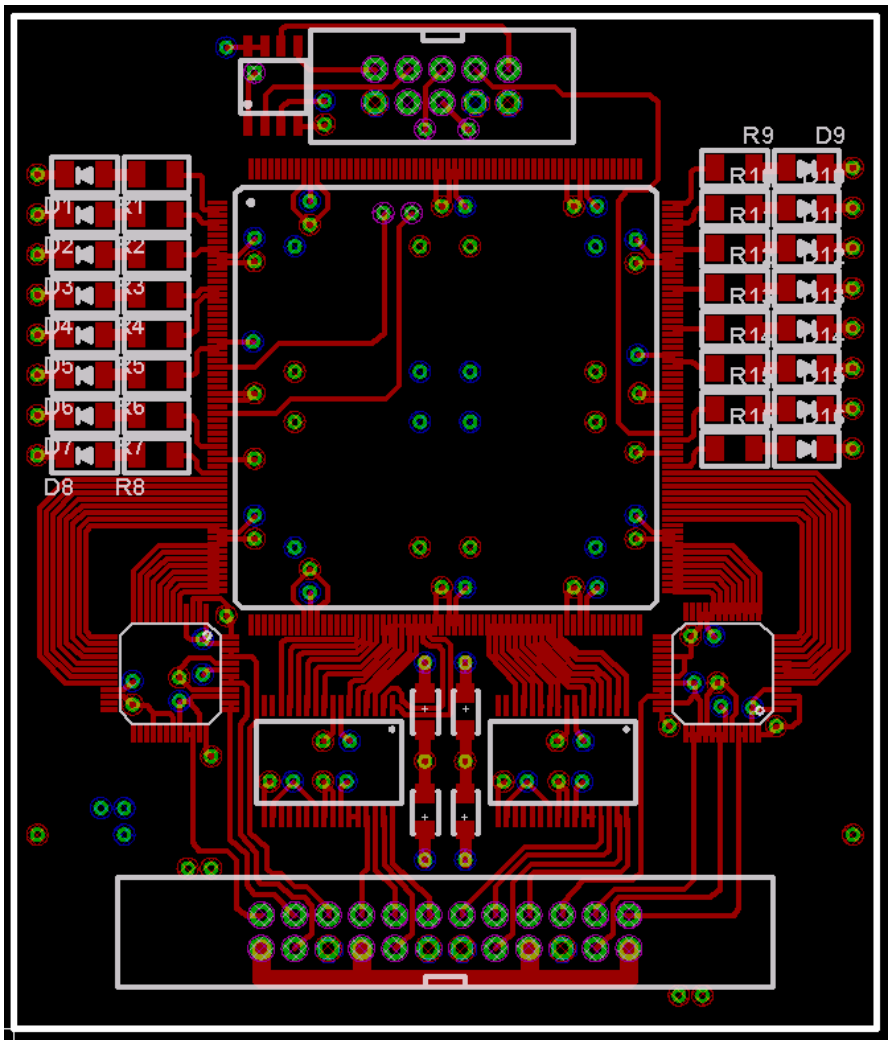

Figure 35: Top layer of preliminary four-channel FPGA board.

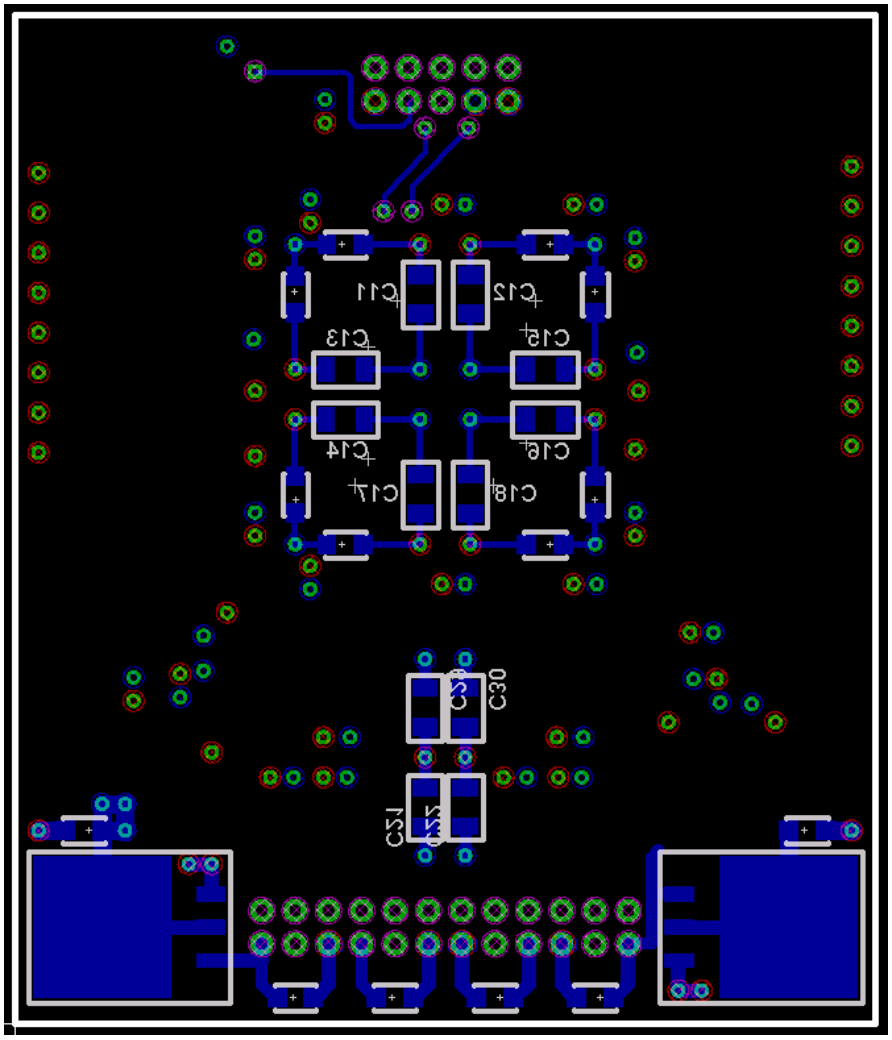

Figure 36: Bottom layer of preliminary four-channel FPGA board.

4.6 Summary

This chapter describes the architectural components of the system architecture of the proposed MPPT controller. Next, it presents the results obtained from the synthesis of the controller into an FPGA implementation. Cursory examination of synthesis results shows that this implementation can be readily expanded to handle MPPT control for multi-

channels without any incurring any area cost. In addition, a preliminary prototype of the multi-channel hardware is presented, signifying a solid foundation for future work and expansion.

CHAPTER FIVE: PROTOTYPE TESTING

This chapter presents additional findings regarding the testing of the physical prototype used to implement the MPPT control system. Section 5.1 explains the experimentation and results of the initial design. Section 5.2 discusses the results of the digital filter tests, while Section 5.3 explains the results of non-sinusoidal dithering. Section 5.4 explores the consequences of reducing ADC resolution and Section 5.5 summarizes these findings.

Array simulators are used instead of physical solar cell arrays since the latter require extensive configuration for accurate testing in general and indoors in particular. Figure 37 shows the experimental setup used to test in-circuit the designed MPPT controller. Agilent E4350B, rated at 480W (60V, 8A), is a solar array simulator in which a test I-V curve can be programmed and stored in an internal table. In order to view its outputs, the array simulator is connected to a monitoring PC through an IEEE 488.1 General Purpose Interface Bus (GPIB). The MPPT controller reads the current and voltage from the DC-DC converter, computes V_{ref}, and feeds it back to the converter. The test load is an Agilent N3300A DC Electronic Load.

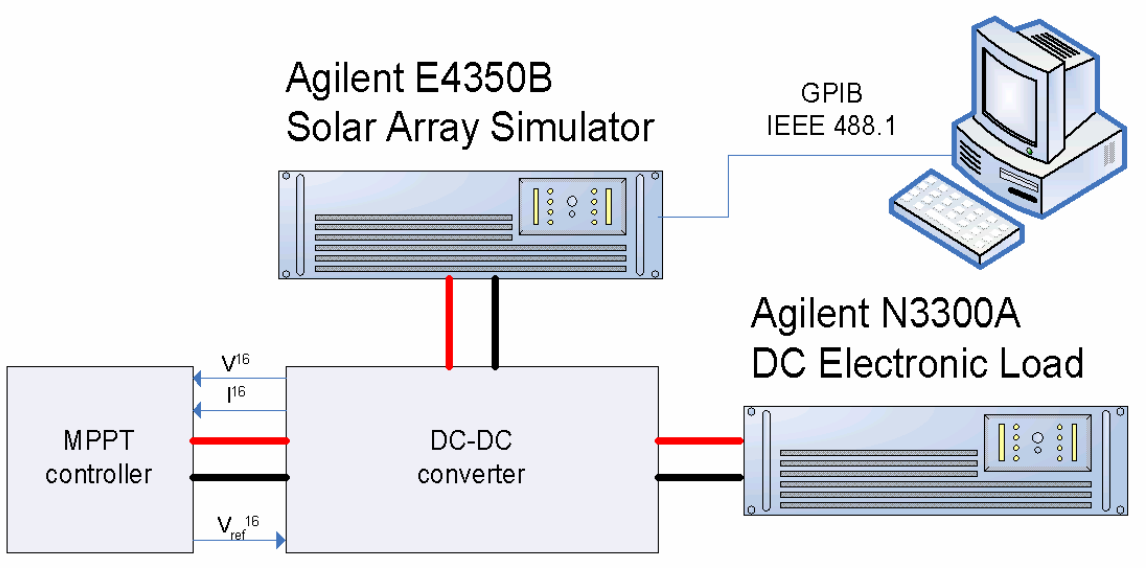

Figure 37: Diagram of experimental setup.

5.1 In-Circuit Verification

Initial tests of the FPGA implementation were successful. Tracking the MPPT worked well both at steady state and during transients. Figure 38 shows a transient between OVR and MPPT at 2 Hz. The climb to MPPT can be observed by the ramp during the initial dithered section. The trace labeled B represents V_{ref}, while the traces labeled A and D represent the output voltage and current of the solar array. The trace labeled C the converter's output voltage.

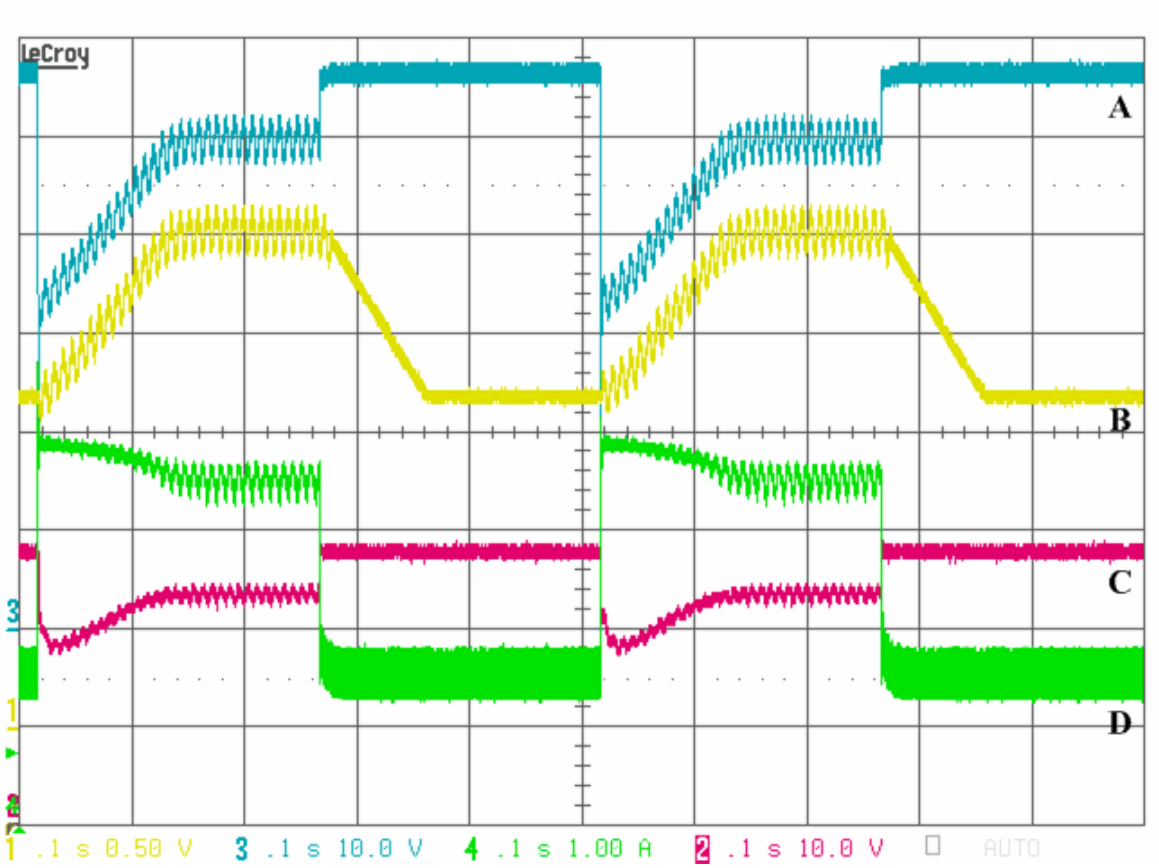

Figure 38: Oscilloscope capture of MPPT transient operation.

Figure 39 shows the characteristic power signal present when MPPT tracking is successful. The voltage signal, represented by trace A, and current signal, represented by trace B, were multiplied within the oscilloscope to produce the power trace represented by trace C. The voltage and current traces represent the input voltage and current from the solar array while the power trace represents the calculated power of the two signals combined. Close examination of the traces shows that two peaks can be observed during each cycle of the voltage and current traces. This is due to the fact that the maximum power point is crossed each half cycle. The distortion in the power trace is due to the distortion of the current trace.

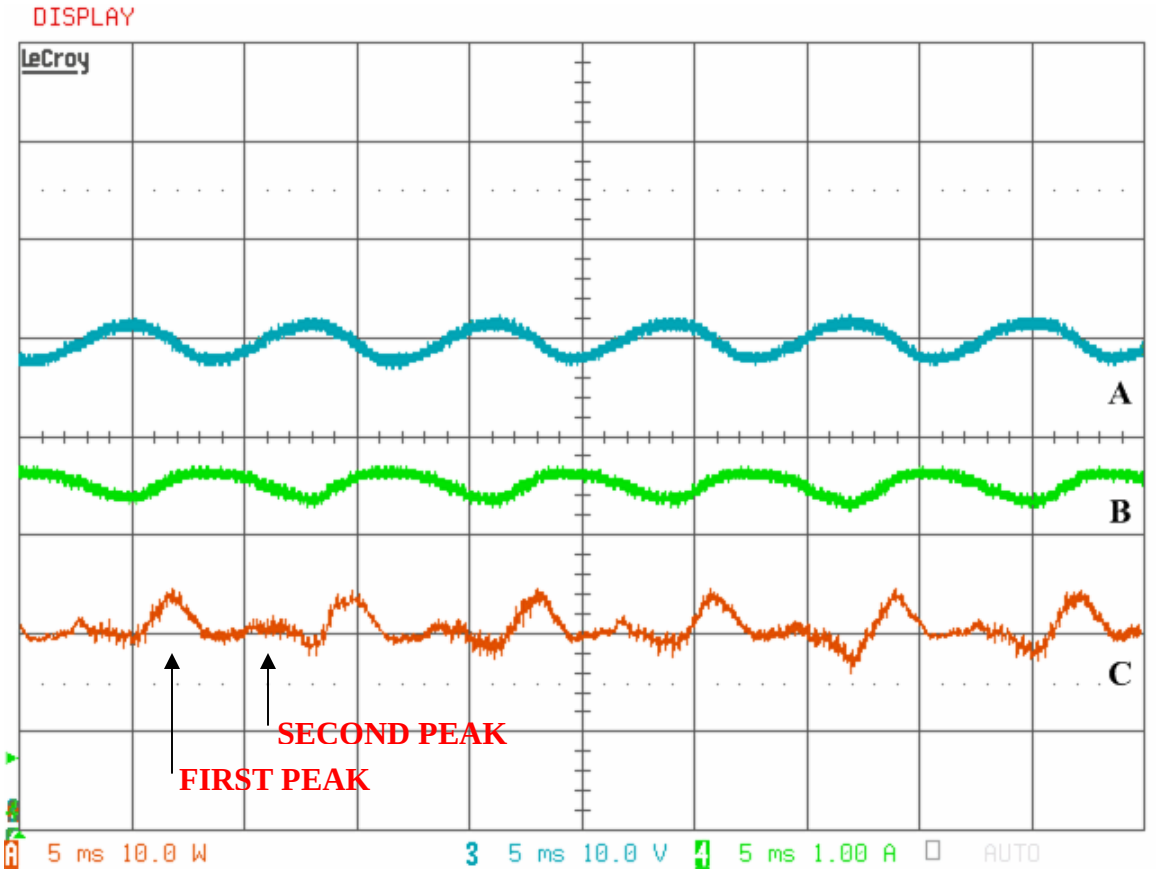

Figure 39: Oscilloscope Capture illustrating double peak of input power.

Trace A and B represent the input voltage and current from the solar array while trace C represents the output current from the converter. During MPPT, any changes in the load should not cause any changes in the input voltage and current, as the controller should be holding the array at its maximum power point. Figure 40 illustrates this condition, as the load, functioning as a constant current sink, is undergoing a transient at 100Hz. The traces are virtually unchanged, and the overlaid dither signal is still noticeable.

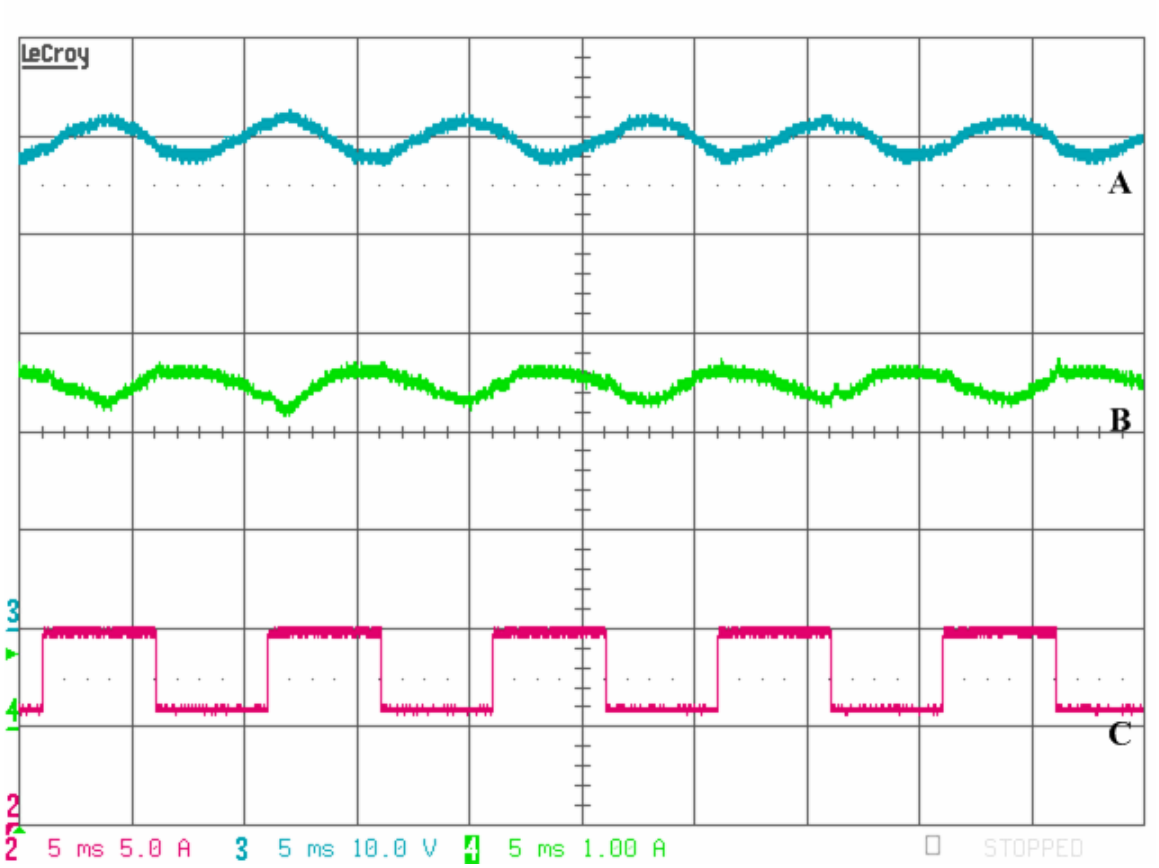

Figure 40: Oscilloscope capture showing changing load current (100Hz).

5.2 Digital Filter Performance

Figure 41 illustrates the efficiency increase for two FIR filters against the baseline which had no filtering. The y-axis represents the average output power while tracking MPPT and the x-axis represents the peak-to-peak amplitude of dither in milliVolts. The results are averages of 10 back-to-back samples.

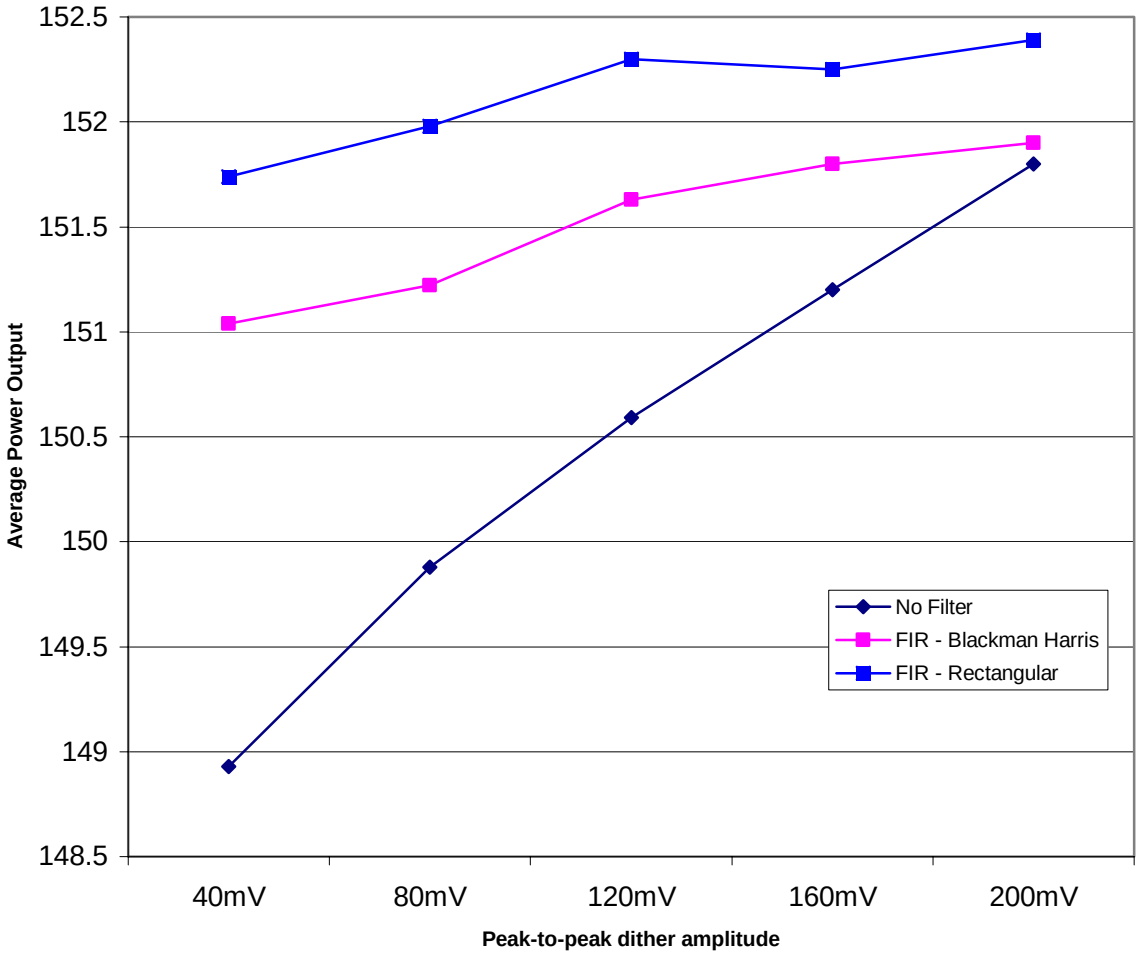

Figure 41: The average power output versus dither amplitude for different filters.

This chart demonstrates a trend which highlights the performance of the FIR filters as a means of increasing efficiency. However, this does not demonstrate the performance increase across multiple sets of samples, and ultimately how effective this filtering actually is. Ideally, it is possible to obtain even higher efficiencies at the lower amplitudes provided the noise can be removed effectively enough to allow climbing to the maximum power point.

5.3 Modified Dither

Figure 42 shows the system operating with a triangular dither at 100mV peak-to-peak. Trace B represents V_{ref} while trace A and C represent the output voltage and current of the solar array. The overall flatness of the dithered region indicates that the tracking is consistent once the peak is reached.

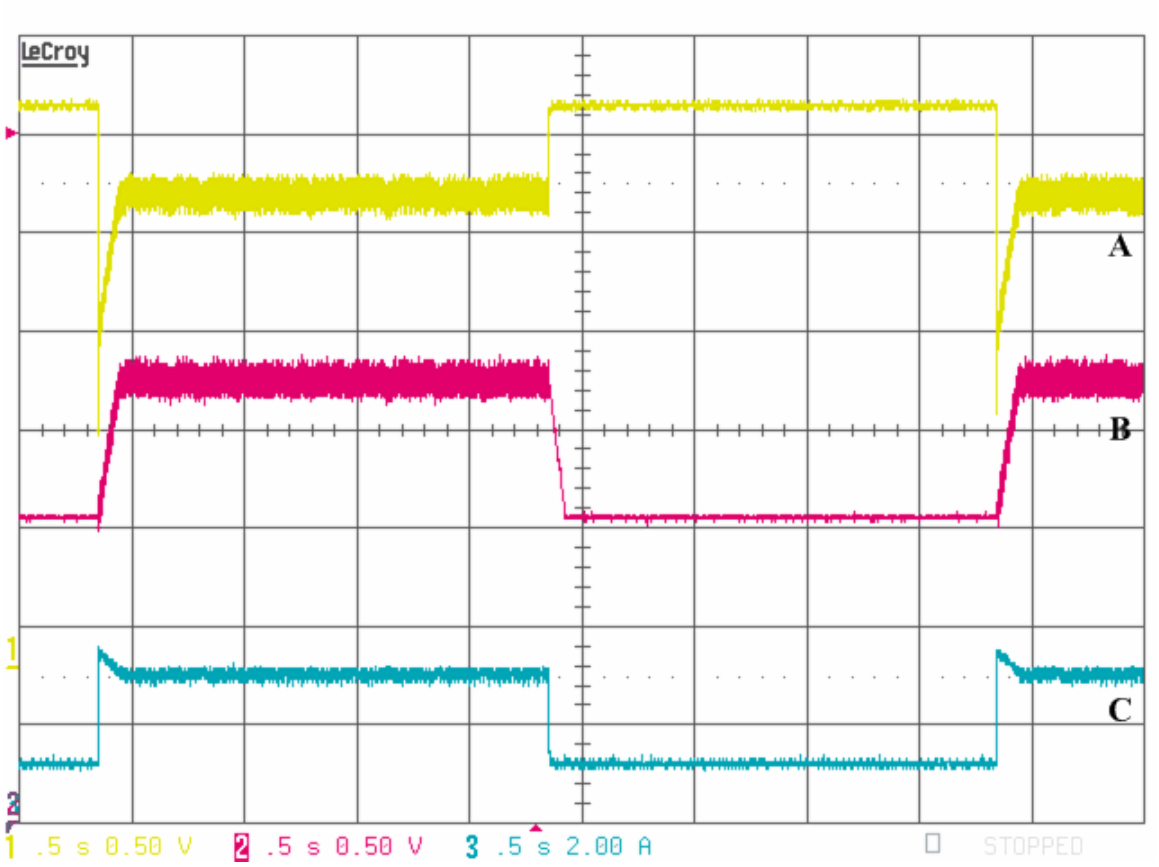

Figure 42: Input Voltage, Input Current, and V_{ref} using triangular dither.

5.4 Reduction of ADC Resolution

In order to test the resolution of the ADC required to perform effective MPPT, the resolution was reduced by clearing the lower bits of the sampled current and voltage. The curve shown in Figure 43 demonstrates the point at which the tracking begins in order to show the effects of this reduced resolution. Based on this data, a 10-bit ADC could easily be used, allowing the costs of the implementation to be reduced further.

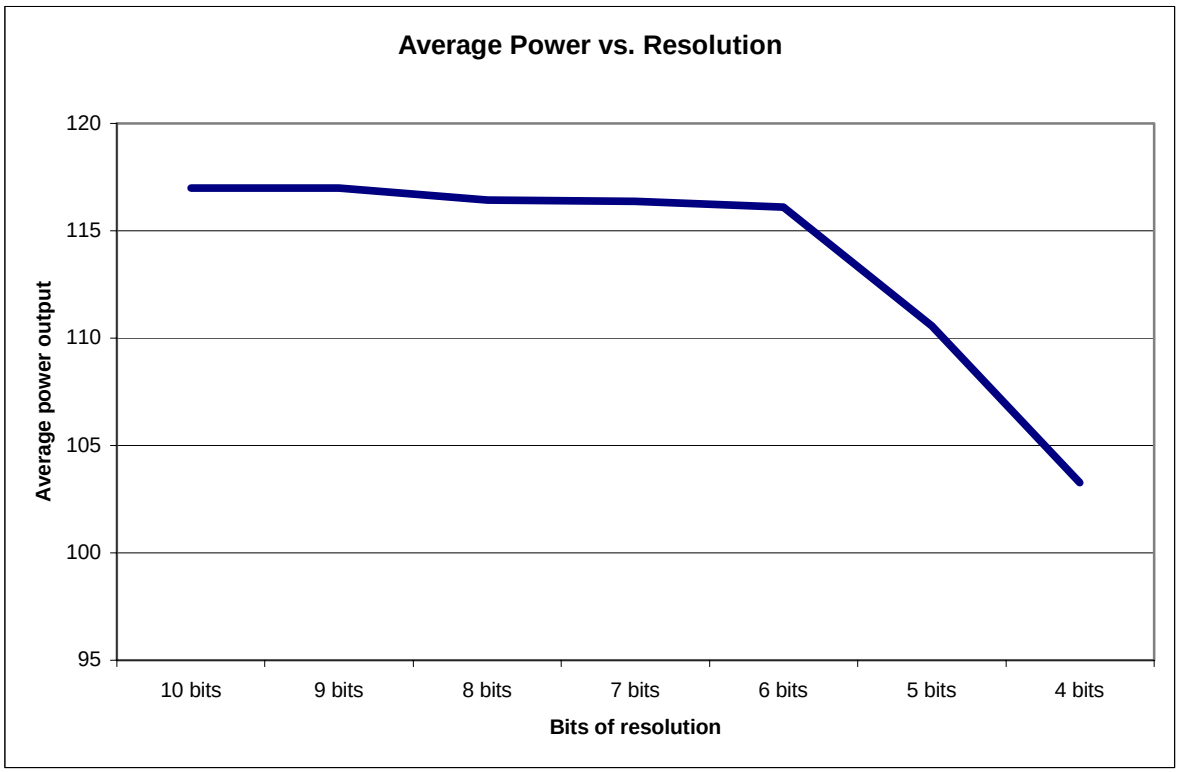

Figure 43: Average power vs. ADC resolution.

5.5 Summary

This chapter presents the test results of the FPGA-based MPP control prototype. In addition, it explores several modifications to a previously proposed design of an MPPT

control system for the benefit of future implementations. Finally, it examines the performance requirements of external components with a goal of creating a more economical implementation.

CHAPTER 6: CONCLUSIONS

This thesis proposes an FPGA-based implementation of an MPPT control system suitable for space applications. At the core of this system, the P&O algorithm is used to track the maximum power point. The algorithm runs on an Altera FLEX 10K FPGA chip. Additional functional blocks, such as the ADC interface, FIR filter, dither generator, and DAC interface, needed to support the MPPT control systems are integrated within the same FPGA device thus streamlining the part composition of the physical prototype of used to build the MPPT control system. Integrating FPGA in an MPPT control system provides numerous advantages. To meet performance requirements, FPGAs are desirable since their performance can easily surpass the performance of microcontrollers and DSPs. Thanks to their high logic capacity, FPGAs can be adapted to control MPPT for multi-channel systems in parallel without imposing complex communication between the individual controls of each channel. In addition, given their reprogrammability, FPGAs can be used to conduct in-circuit experimentation, testing and optimization of various parameters that affect the performance of the MPPT control system. For instance, they can be used to determine an optimal resolution for the ADCs and DACs used in the MPPT control system. By knowing this resolution, only the right components are selected without adding unnecessary costs to the cost of the system prototype. In addition, they can be used to design a suitable filtering scheme to attenuate the effect of noise on the performance of the MPPT control system.

While this thesis shows how FPGAs can be used to build cost-effective and highly adaptable implementations of MPPT control systems, it nevertheless evokes new directions for future research in FPGA implementations of MPPT control systems:

(i) *How much of the functionality of the ADCs and DACs can be integrated within the FPGA*? Design approaches on how to use FPGAs to convert an analog signal to a digital value or vice-versa has been proposed before [27, 28]. Such an integration can lower further the costs of the implementation and facilitates system testing and verification.

(ii) *How can the resources available on an FPGA chip be used effectively to support a multi-channel MPPT control system*? Figure 32 suggest a possible architecture to implement multi-channel control within the same FPGA. However, other architectures with efficient communication schemes can also be mapped onto an FPGA chip.

(iii) *How can an FPGA multi-system be built to control a large scale MPPT architecture*? Single chip multi-channel control architectures can be expanded further to handle MPPT control at a large scale. Such systems can provide performance and flexibility that are higher than those provided by single chip systems.

(iv) *How can the processing power of FPGAs be harnessed to implement intelligent approaches for MPPT control that can learn and anticipate adequate power outputs that may be needed in the near future*? Control approaches based on intelligent computational models such as neural

networks [29] and fuzzy logic [30] have been proposed to overcome uncertainty, variances, and disturbances in load demand and weather conditions. These models require substantial computation cycles in order to produce meaningful results. Contrary to DSPs and microcontrollers, only highly adaptable architectures such as FPGAs are sufficiently flexible and powerful to support these computations.

Addressing these issues can increase the efficiency of MPPT control systems, and expand their scope to other energy applications.

APPENDIX:
SOURCE CODE LISTING

A.1 ADC Controller

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

ENTITY adctest IS
   PORT
   (
      -- input clock
      clk                        : IN  BIT;
      -- first adc inputs
      adc1_busy                  : IN  BIT;
      adc1_data                  : IN  UNSIGNED(15 DOWNTO 0);
      -- second adc inputs
      adc2_busy                  : IN  BIT;
      adc2_data                  : IN  UNSIGNED(15 DOWNTO 0);
      -- first adc outputs
      adc1_clk                : OUT BIT;
      adc1_convst                : OUT BIT;
      adc1_byte                  : OUT BIT;
      adc1_cs                    : OUT BIT;
      adc1_rd                    : OUT BIT;
      -- second adc outputs
      adc2_clk                : OUT BIT;
      adc2_convst                : OUT BIT;
      adc2_byte                  : OUT BIT;
      adc2_cs                    : OUT BIT;
      adc2_rd                    : OUT BIT;
      -- dac outputs
      dac1_data                  : OUT UNSIGNED(15 DOWNTO 0);
      dac2_data                  : OUT UNSIGNED(15 DOWNTO 0);
      -- data ready output
      ready_out                  : OUT BIT
   );
END adctest;

ARCHITECTURE a OF adctest IS
   TYPE STATE_TYPE IS (s0, s1, s2, s3, s4);
   SIGNAL state: STATE_TYPE;
   SIGNAL clock_count: NATURAL RANGE 0 TO 255;
   SIGNAL tmp_adc_rd: BIT;
   SIGNAL tmp_adc_convst: BIT;
   SIGNAL tmp_adc1_data: UNSIGNED(15 DOWNTO 0);
   SIGNAL tmp_adc2_data: UNSIGNED(15 DOWNTO 0);
   SIGNAL delay_count: NATURAL RANGE 0 TO 7;
   SIGNAL ready_out_tmp: BIT;
BEGIN

   adc1_clk <= clk;
   adc2_clk <= clk;

   PROCESS (clk)
   BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         CASE state IS
            -- STATE 0:
            WHEN s0 =>
               state <= s1;
               clock_count <= 0;
               tmp_adc_convst <= '0';
               tmp_adc_rd <= '1';
               tmp_adc1_data <= tmp_adc1_data;
               tmp_adc2_data <= tmp_adc2_data;
               ready_out_tmp <= '0';
            -- STATE 1:
            WHEN s1 =>
               state <= s2;
```

```
               clock_count <= 0;
               tmp_adc_convst <= '1';
               tmp_adc_rd <='1';
               tmp_adc1_data <= tmp_adc1_data;
               tmp_adc2_data <= tmp_adc2_data;
               ready_out_tmp <= '0';
            -- STATE 2:
            WHEN s2 =>
               IF (clock_count=15) THEN
                  state <= s3;
                  clock_count <= 0;
               ELSE
                  state <= s2;
                  clock_count <= clock_count + 1;
               END IF;

               tmp_adc_convst <= '1';
               tmp_adc_rd <='1';
               tmp_adc1_data <= tmp_adc1_data;
               tmp_adc2_data <= tmp_adc2_data;
               ready_out_tmp <= '0';
            -- STATE 3:
            WHEN s3 =>
               state <= s4;
               clock_count <= 0;
               tmp_adc_rd <= '0';
               tmp_adc_convst <= '1';
               tmp_adc1_data <= adc1_data;
               tmp_adc2_data <= adc2_data;
               ready_out_tmp <= '0';
            -- STATE 4:
            WHEN s4 =>
               state <= s0;
               clock_count <= 0;
               tmp_adc_rd <= '1';
               tmp_adc_convst <= '1';
               tmp_adc1_data <= tmp_adc1_data;
               tmp_adc2_data <= tmp_adc2_data;
               ready_out_tmp <= '1';
         END CASE;
      END IF;
   END PROCESS;
   ready_out <= ready_out_tmp;

   -- first adc parameters
   adc1_byte <= '0';
   adc1_cs <= '0';
   adc1_convst <= tmp_adc_convst;
   adc1_rd <= tmp_adc_rd;
   dac1_data <= tmp_adc1_data;

   -- second adc parameters
   adc2_byte <= '0';
   adc2_cs <= '0';
   adc2_convst <= tmp_adc_convst;
   adc2_rd <= tmp_adc_rd;
   dac2_data <= tmp_adc2_data;
END a;
```

A.2 FIR Lowpass Filter with Decimation

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;
LIBRARY work;
USE work.fir_lut16.ALL;

ENTITY fir_decimator IS
   PORT
   (
      v_in                    : IN  UNSIGNED(15 DOWNTO 0);
      i_in                    : IN  UNSIGNED(15 DOWNTO 0);
      ready_in                : IN  BIT;
      clk                        : IN  BIT;

      v_out                   : OUT UNSIGNED(15 DOWNTO 0);
      i_out                   : OUT UNSIGNED(15 DOWNTO 0);
      ready_out                  : OUT BIT
   );
END fir_decimator;

ARCHITECTURE a OF fir_decimator IS
   TYPE STATE_TYPE IS (s0, s1);
   SIGNAL state: STATE_TYPE;

   SIGNAL v_accum: UNSIGNED(39 DOWNTO 0); -- 16+16+8=40
   SIGNAL i_accum: UNSIGNED(39 DOWNTO 0);
   SIGNAL v_out_tmp: UNSIGNED(15 DOWNTO 0);
   SIGNAL i_out_tmp: UNSIGNED(15 DOWNTO 0);
   SIGNAL ready_out_tmp: BIT;
   SIGNAL sample_count: NATURAL RANGE 0 TO 255;
BEGIN
   PROCESS (clk)
   BEGIN
      IF (clk='1' AND clk'EVENT) THEN
         CASE state IS
            -- STATE 0:
            WHEN s0 =>
               IF (ready_in='1') THEN
                  v_accum <= v_accum + v_in * get_table_value(sample_count);
                  i_accum <= i_accum + i_in * get_table_value(sample_count);

                  IF (sample_count=255) THEN
                     sample_count <= 0;
                     state <= s1;
                  ELSE
                     sample_count <= sample_count + 1;
                     state <= s0;
                  END IF;

               ELSE
                  v_accum <= v_accum;
                  i_accum <= i_accum;
                  sample_count <= sample_count;
                  state <= s0;
               END IF;
               v_out_tmp <= v_out_tmp;
               i_out_tmp <= i_out_tmp;
               ready_out_tmp <= '0';

            -- STATE 1:
            WHEN s1 =>
               v_out_tmp <= v_accum(31 DOWNTO 16);
               i_out_tmp <= i_accum(31 DOWNTO 16);

               v_accum <= X"0000000000";
               i_accum <= X"0000000000";
```

```
                sample_count <= 0;
                ready_out_tmp <= '1';
                state <= s0;
            END CASE;

        END IF;
    END PROCESS;

    ready_out <= ready_out_tmp;
    v_out <= v_out_tmp;
    i_out <= i_out_tmp;
END a;
```

A.3 FIR Coefficients for Rectangular Window

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

PACKAGE fir_lut16 IS
   SUBTYPE table_value_type IS UNSIGNED(15 DOWNTO 0);
   SUBTYPE table_index_type IS NATURAL RANGE 0 TO 255;
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type;
END;

PACKAGE BODY fir_lut16 IS
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type IS
      VARIABLE table_value: table_value_type;
   BEGIN
      CASE table_index IS
         WHEN 0 =>
            table_value := X"00F9";
         WHEN 1 =>
            table_value := X"00FA";
         WHEN 2 =>
            table_value := X"00FA";
         WHEN 3 =>
            table_value := X"00FA";
         WHEN 4 =>
            table_value := X"00FA";
         WHEN 5 =>
            table_value := X"00FA";
         WHEN 6 =>
            table_value := X"00FA";
         WHEN 7 =>
            table_value := X"00FA";
         WHEN 8 =>
            table_value := X"00FB";
         WHEN 9 =>
            table_value := X"00FB";
         WHEN 10 =>
            table_value := X"00FB";
         WHEN 11 =>
            table_value := X"00FB";
         WHEN 12 =>
            table_value := X"00FB";
         WHEN 13 =>
            table_value := X"00FB";
         WHEN 14 =>
            table_value := X"00FB";
         WHEN 15 =>
            table_value := X"00FC";
         WHEN 16 =>
            table_value := X"00FC";
         WHEN 17 =>
            table_value := X"00FC";
         WHEN 18 =>
            table_value := X"00FC";
         WHEN 19 =>
            table_value := X"00FC";
         WHEN 20 =>
            table_value := X"00FC";
         WHEN 21 =>
            table_value := X"00FC";
         WHEN 22 =>
            table_value := X"00FD";
         WHEN 23 =>
            table_value := X"00FD";
         WHEN 24 =>
            table_value := X"00FD";
         WHEN 25 =>
```

```
   table_value := X"00FD";
WHEN 26 =>
   table_value := X"00FD";
WHEN 27 =>
   table_value := X"00FD";
WHEN 28 =>
   table_value := X"00FD";
WHEN 29 =>
   table_value := X"00FD";
WHEN 30 =>
   table_value := X"00FE";
WHEN 31 =>
   table_value := X"00FE";
WHEN 32 =>
   table_value := X"00FE";
WHEN 33 =>
   table_value := X"00FE";
WHEN 34 =>
   table_value := X"00FE";
WHEN 35 =>
   table_value := X"00FE";
WHEN 36 =>
   table_value := X"00FE";
WHEN 37 =>
   table_value := X"00FE";
WHEN 38 =>
   table_value := X"00FE";
WHEN 39 =>
   table_value := X"00FF";
WHEN 40 =>
   table_value := X"00FF";
WHEN 41 =>
   table_value := X"00FF";
WHEN 42 =>
   table_value := X"00FF";
WHEN 43 =>
   table_value := X"00FF";
WHEN 44 =>
   table_value := X"00FF";
WHEN 45 =>
   table_value := X"00FF";
WHEN 46 =>
   table_value := X"00FF";
WHEN 47 =>
   table_value := X"00FF";
WHEN 48 =>
   table_value := X"00FF";
WHEN 49 =>
   table_value := X"0100";
WHEN 50 =>
   table_value := X"0100";
WHEN 51 =>
   table_value := X"0100";
WHEN 52 =>
   table_value := X"0100";
WHEN 53 =>
   table_value := X"0100";
WHEN 54 =>
   table_value := X"0100";
WHEN 55 =>
   table_value := X"0100";
WHEN 56 =>
   table_value := X"0100";
WHEN 57 =>
   table_value := X"0100";
WHEN 58 =>
   table_value := X"0100";
WHEN 59 =>
   table_value := X"0100";
WHEN 60 =>
   table_value := X"0101";
```

```
WHEN 61 =>
   table_value := X"0101";
WHEN 62 =>
   table_value := X"0101";
WHEN 63 =>
   table_value := X"0101";
WHEN 64 =>
   table_value := X"0101";
WHEN 65 =>
   table_value := X"0101";
WHEN 66 =>
   table_value := X"0101";
WHEN 67 =>
   table_value := X"0101";
WHEN 68 =>
   table_value := X"0101";
WHEN 69 =>
   table_value := X"0101";
WHEN 70 =>
   table_value := X"0101";
WHEN 71 =>
   table_value := X"0101";
WHEN 72 =>
   table_value := X"0101";
WHEN 73 =>
   table_value := X"0102";
WHEN 74 =>
   table_value := X"0102";
WHEN 75 =>
   table_value := X"0102";
WHEN 76 =>
   table_value := X"0102";
WHEN 77 =>
   table_value := X"0102";
WHEN 78 =>
   table_value := X"0102";
WHEN 79 =>
   table_value := X"0102";
WHEN 80 =>
   table_value := X"0102";
WHEN 81 =>
   table_value := X"0102";
WHEN 82 =>
   table_value := X"0102";
WHEN 83 =>
   table_value := X"0102";
WHEN 84 =>
   table_value := X"0102";
WHEN 85 =>
   table_value := X"0102";
WHEN 86 =>
   table_value := X"0102";
WHEN 87 =>
   table_value := X"0102";
WHEN 88 =>
   table_value := X"0102";
WHEN 89 =>
   table_value := X"0102";
WHEN 90 =>
   table_value := X"0102";
WHEN 91 =>
   table_value := X"0103";
WHEN 92 =>
   table_value := X"0103";
WHEN 93 =>
   table_value := X"0103";
WHEN 94 =>
   table_value := X"0103";
WHEN 95 =>
   table_value := X"0103";
WHEN 96 =>
```

```
   table_value := X"0103";
WHEN 97 =>
   table_value := X"0103";
WHEN 98 =>
   table_value := X"0103";
WHEN 99 =>
   table_value := X"0103";
WHEN 100 =>
   table_value := X"0103";
WHEN 101 =>
   table_value := X"0103";
WHEN 102 =>
   table_value := X"0103";
WHEN 103 =>
   table_value := X"0103";
WHEN 104 =>
   table_value := X"0103";
WHEN 105 =>
   table_value := X"0103";
WHEN 106 =>
   table_value := X"0103";
WHEN 107 =>
   table_value := X"0103";
WHEN 108 =>
   table_value := X"0103";
WHEN 109 =>
   table_value := X"0103";
WHEN 110 =>
   table_value := X"0103";
WHEN 111 =>
   table_value := X"0103";
WHEN 112 =>
   table_value := X"0103";
WHEN 113 =>
   table_value := X"0103";
WHEN 114 =>
   table_value := X"0103";
WHEN 115 =>
   table_value := X"0103";
WHEN 116 =>
   table_value := X"0103";
WHEN 117 =>
   table_value := X"0103";
WHEN 118 =>
   table_value := X"0103";
WHEN 119 =>
   table_value := X"0103";
WHEN 120 =>
   table_value := X"0103";
WHEN 121 =>
   table_value := X"0103";
WHEN 122 =>
   table_value := X"0103";
WHEN 123 =>
   table_value := X"0103";
WHEN 124 =>
   table_value := X"0103";
WHEN 125 =>
   table_value := X"0103";
WHEN 126 =>
   table_value := X"0103";
WHEN 127 =>
   table_value := X"0103";
WHEN 128 =>
   table_value := X"0103";
WHEN 129 =>
   table_value := X"0103";
WHEN 130 =>
   table_value := X"0103";
WHEN 131 =>
   table_value := X"0103";
```

```
WHEN 132 =>
   table_value := X"0103";
WHEN 133 =>
   table_value := X"0103";
WHEN 134 =>
   table_value := X"0103";
WHEN 135 =>
   table_value := X"0103";
WHEN 136 =>
   table_value := X"0103";
WHEN 137 =>
   table_value := X"0103";
WHEN 138 =>
   table_value := X"0103";
WHEN 139 =>
   table_value := X"0103";
WHEN 140 =>
   table_value := X"0103";
WHEN 141 =>
   table_value := X"0103";
WHEN 142 =>
   table_value := X"0103";
WHEN 143 =>
   table_value := X"0103";
WHEN 144 =>
   table_value := X"0103";
WHEN 145 =>
   table_value := X"0103";
WHEN 146 =>
   table_value := X"0103";
WHEN 147 =>
   table_value := X"0103";
WHEN 148 =>
   table_value := X"0103";
WHEN 149 =>
   table_value := X"0103";
WHEN 150 =>
   table_value := X"0103";
WHEN 151 =>
   table_value := X"0103";
WHEN 152 =>
   table_value := X"0103";
WHEN 153 =>
   table_value := X"0103";
WHEN 154 =>
   table_value := X"0103";
WHEN 155 =>
   table_value := X"0103";
WHEN 156 =>
   table_value := X"0103";
WHEN 157 =>
   table_value := X"0103";
WHEN 158 =>
   table_value := X"0103";
WHEN 159 =>
   table_value := X"0103";
WHEN 160 =>
   table_value := X"0103";
WHEN 161 =>
   table_value := X"0103";
WHEN 162 =>
   table_value := X"0103";
WHEN 163 =>
   table_value := X"0103";
WHEN 164 =>
   table_value := X"0103";
WHEN 165 =>
   table_value := X"0102";
WHEN 166 =>
   table_value := X"0102";
WHEN 167 =>
```

```
   table_value := X"0102";
WHEN 168 =>
   table_value := X"0102";
WHEN 169 =>
   table_value := X"0102";
WHEN 170 =>
   table_value := X"0102";
WHEN 171 =>
   table_value := X"0102";
WHEN 172 =>
   table_value := X"0102";
WHEN 173 =>
   table_value := X"0102";
WHEN 174 =>
   table_value := X"0102";
WHEN 175 =>
   table_value := X"0102";
WHEN 176 =>
   table_value := X"0102";
WHEN 177 =>
   table_value := X"0102";
WHEN 178 =>
   table_value := X"0102";
WHEN 179 =>
   table_value := X"0102";
WHEN 180 =>
   table_value := X"0102";
WHEN 181 =>
   table_value := X"0102";
WHEN 182 =>
   table_value := X"0102";
WHEN 183 =>
   table_value := X"0101";
WHEN 184 =>
   table_value := X"0101";
WHEN 185 =>
   table_value := X"0101";
WHEN 186 =>
   table_value := X"0101";
WHEN 187 =>
   table_value := X"0101";
WHEN 188 =>
   table_value := X"0101";
WHEN 189 =>
   table_value := X"0101";
WHEN 190 =>
   table_value := X"0101";
WHEN 191 =>
   table_value := X"0101";
WHEN 192 =>
   table_value := X"0101";
WHEN 193 =>
   table_value := X"0101";
WHEN 194 =>
   table_value := X"0101";
WHEN 195 =>
   table_value := X"0101";
WHEN 196 =>
   table_value := X"0100";
WHEN 197 =>
   table_value := X"0100";
WHEN 198 =>
   table_value := X"0100";
WHEN 199 =>
   table_value := X"0100";
WHEN 200 =>
   table_value := X"0100";
WHEN 201 =>
   table_value := X"0100";
WHEN 202 =>
   table_value := X"0100";
```

```
WHEN 203 =>
   table_value := X"0100";
WHEN 204 =>
   table_value := X"0100";
WHEN 205 =>
   table_value := X"0100";
WHEN 206 =>
   table_value := X"0100";
WHEN 207 =>
   table_value := X"00FF";
WHEN 208 =>
   table_value := X"00FF";
WHEN 209 =>
   table_value := X"00FF";
WHEN 210 =>
   table_value := X"00FF";
WHEN 211 =>
   table_value := X"00FF";
WHEN 212 =>
   table_value := X"00FF";
WHEN 213 =>
   table_value := X"00FF";
WHEN 214 =>
   table_value := X"00FF";
WHEN 215 =>
   table_value := X"00FF";
WHEN 216 =>
   table_value := X"00FF";
WHEN 217 =>
   table_value := X"00FE";
WHEN 218 =>
   table_value := X"00FE";
WHEN 219 =>
   table_value := X"00FE";
WHEN 220 =>
   table_value := X"00FE";
WHEN 221 =>
   table_value := X"00FE";
WHEN 222 =>
   table_value := X"00FE";
WHEN 223 =>
   table_value := X"00FE";
WHEN 224 =>
   table_value := X"00FE";
WHEN 225 =>
   table_value := X"00FE";
WHEN 226 =>
   table_value := X"00FD";
WHEN 227 =>
   table_value := X"00FD";
WHEN 228 =>
   table_value := X"00FD";
WHEN 229 =>
   table_value := X"00FD";
WHEN 230 =>
   table_value := X"00FD";
WHEN 231 =>
   table_value := X"00FD";
WHEN 232 =>
   table_value := X"00FD";
WHEN 233 =>
   table_value := X"00FD";
WHEN 234 =>
   table_value := X"00FC";
WHEN 235 =>
   table_value := X"00FC";
WHEN 236 =>
   table_value := X"00FC";
WHEN 237 =>
   table_value := X"00FC";
WHEN 238 =>
```

```
            table_value := X"00FC";
         WHEN 239 =>
            table_value := X"00FC";
         WHEN 240 =>
            table_value := X"00FC";
         WHEN 241 =>
            table_value := X"00FB";
         WHEN 242 =>
            table_value := X"00FB";
         WHEN 243 =>
            table_value := X"00FB";
         WHEN 244 =>
            table_value := X"00FB";
         WHEN 245 =>
            table_value := X"00FB";
         WHEN 246 =>
            table_value := X"00FB";
         WHEN 247 =>
            table_value := X"00FB";
         WHEN 248 =>
            table_value := X"00FA";
         WHEN 249 =>
            table_value := X"00FA";
         WHEN 250 =>
            table_value := X"00FA";
         WHEN 251 =>
            table_value := X"00FA";
         WHEN 252 =>
            table_value := X"00FA";
         WHEN 253 =>
            table_value := X"00FA";
         WHEN 254 =>
            table_value := X"00FA";
         WHEN 255 =>
            table_value := X"00F9";
      END CASE;
      RETURN table_value;
   END;
END;
```

A.4 FIR Coefficients for Blackman-Harris Window

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

PACKAGE fir_lut16 IS
   SUBTYPE table_value_type IS UNSIGNED(15 DOWNTO 0);
   SUBTYPE table_index_type IS NATURAL RANGE 0 TO 255;
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type;
END;

PACKAGE BODY fir_lut16 IS
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type IS
      VARIABLE table_value: table_value_type;
   BEGIN
      CASE table_index IS
         WHEN 0 =>
            table_value := X"0000";
         WHEN 1 =>
            table_value := X"0000";
         WHEN 2 =>
            table_value := X"0000";
         WHEN 3 =>
            table_value := X"0000";
         WHEN 4 =>
            table_value := X"0000";
         WHEN 5 =>
            table_value := X"0000";
         WHEN 6 =>
            table_value := X"0000";
         WHEN 7 =>
            table_value := X"0000";
         WHEN 8 =>
            table_value := X"0000";
         WHEN 9 =>
            table_value := X"0001";
         WHEN 10 =>
            table_value := X"0001";
         WHEN 11 =>
            table_value := X"0001";
         WHEN 12 =>
            table_value := X"0001";
         WHEN 13 =>
            table_value := X"0001";
         WHEN 14 =>
            table_value := X"0002";
         WHEN 15 =>
            table_value := X"0002";
         WHEN 16 =>
            table_value := X"0002";
         WHEN 17 =>
            table_value := X"0003";
         WHEN 18 =>
            table_value := X"0003";
         WHEN 19 =>
            table_value := X"0003";
         WHEN 20 =>
            table_value := X"0004";
         WHEN 21 =>
            table_value := X"0004";
         WHEN 22 =>
            table_value := X"0005";
         WHEN 23 =>
            table_value := X"0006";
         WHEN 24 =>
            table_value := X"0006";
         WHEN 25 =>
```

```
   table_value := X"0007";
WHEN 26 =>
   table_value := X"0008";
WHEN 27 =>
   table_value := X"0009";
WHEN 28 =>
   table_value := X"000A";
WHEN 29 =>
   table_value := X"000B";
WHEN 30 =>
   table_value := X"000D";
WHEN 31 =>
   table_value := X"000E";
WHEN 32 =>
   table_value := X"000F";
WHEN 33 =>
   table_value := X"0011";
WHEN 34 =>
   table_value := X"0013";
WHEN 35 =>
   table_value := X"0015";
WHEN 36 =>
   table_value := X"0017";
WHEN 37 =>
   table_value := X"0019";
WHEN 38 =>
   table_value := X"001B";
WHEN 39 =>
   table_value := X"001D";
WHEN 40 =>
   table_value := X"0020";
WHEN 41 =>
   table_value := X"0023";
WHEN 42 =>
   table_value := X"0026";
WHEN 43 =>
   table_value := X"0029";
WHEN 44 =>
   table_value := X"002C";
WHEN 45 =>
   table_value := X"0030";
WHEN 46 =>
   table_value := X"0033";
WHEN 47 =>
   table_value := X"0037";
WHEN 48 =>
   table_value := X"003B";
WHEN 49 =>
   table_value := X"0040";
WHEN 50 =>
   table_value := X"0044";
WHEN 51 =>
   table_value := X"0049";
WHEN 52 =>
   table_value := X"004E";
WHEN 53 =>
   table_value := X"0053";
WHEN 54 =>
   table_value := X"0059";
WHEN 55 =>
   table_value := X"005E";
WHEN 56 =>
   table_value := X"0064";
WHEN 57 =>
   table_value := X"006A";
WHEN 58 =>
   table_value := X"0071";
WHEN 59 =>
   table_value := X"0078";
WHEN 60 =>
   table_value := X"007F";
```

```
WHEN 61 =>
   table_value := X"0086";
WHEN 62 =>
   table_value := X"008D";
WHEN 63 =>
   table_value := X"0095";
WHEN 64 =>
   table_value := X"009D";
WHEN 65 =>
   table_value := X"00A5";
WHEN 66 =>
   table_value := X"00AD";
WHEN 67 =>
   table_value := X"00B6";
WHEN 68 =>
   table_value := X"00BF";
WHEN 69 =>
   table_value := X"00C8";
WHEN 70 =>
   table_value := X"00D1";
WHEN 71 =>
   table_value := X"00DB";
WHEN 72 =>
   table_value := X"00E5";
WHEN 73 =>
   table_value := X"00EF";
WHEN 74 =>
   table_value := X"00F9";
WHEN 75 =>
   table_value := X"0104";
WHEN 76 =>
   table_value := X"010E";
WHEN 77 =>
   table_value := X"0119";
WHEN 78 =>
   table_value := X"0124";
WHEN 79 =>
   table_value := X"012F";
WHEN 80 =>
   table_value := X"013A";
WHEN 81 =>
   table_value := X"0146";
WHEN 82 =>
   table_value := X"0151";
WHEN 83 =>
   table_value := X"015D";
WHEN 84 =>
   table_value := X"0169";
WHEN 85 =>
   table_value := X"0174";
WHEN 86 =>
   table_value := X"0180";
WHEN 87 =>
   table_value := X"018C";
WHEN 88 =>
   table_value := X"0198";
WHEN 89 =>
   table_value := X"01A4";
WHEN 90 =>
   table_value := X"01B0";
WHEN 91 =>
   table_value := X"01BC";
WHEN 92 =>
   table_value := X"01C8";
WHEN 93 =>
   table_value := X"01D4";
WHEN 94 =>
   table_value := X"01E0";
WHEN 95 =>
   table_value := X"01EB";
WHEN 96 =>
```

```
   table_value := X"01F7";
WHEN 97 =>
   table_value := X"0203";
WHEN 98 =>
   table_value := X"020E";
WHEN 99 =>
   table_value := X"0219";
WHEN 100 =>
   table_value := X"0224";
WHEN 101 =>
   table_value := X"022F";
WHEN 102 =>
   table_value := X"0239";
WHEN 103 =>
   table_value := X"0244";
WHEN 104 =>
   table_value := X"024E";
WHEN 105 =>
   table_value := X"0258";
WHEN 106 =>
   table_value := X"0261";
WHEN 107 =>
   table_value := X"026B";
WHEN 108 =>
   table_value := X"0274";
WHEN 109 =>
   table_value := X"027C";
WHEN 110 =>
   table_value := X"0284";
WHEN 111 =>
   table_value := X"028C";
WHEN 112 =>
   table_value := X"0294";
WHEN 113 =>
   table_value := X"029B";
WHEN 114 =>
   table_value := X"02A1";
WHEN 115 =>
   table_value := X"02A8";
WHEN 116 =>
   table_value := X"02AE";
WHEN 117 =>
   table_value := X"02B3";
WHEN 118 =>
   table_value := X"02B8";
WHEN 119 =>
   table_value := X"02BC";
WHEN 120 =>
   table_value := X"02C0";
WHEN 121 =>
   table_value := X"02C4";
WHEN 122 =>
   table_value := X"02C7";
WHEN 123 =>
   table_value := X"02C9";
WHEN 124 =>
   table_value := X"02CB";
WHEN 125 =>
   table_value := X"02CD";
WHEN 126 =>
   table_value := X"02CE";
WHEN 127 =>
   table_value := X"02CE";
WHEN 128 =>
   table_value := X"02CE";
WHEN 129 =>
   table_value := X"02CE";
WHEN 130 =>
   table_value := X"02CD";
WHEN 131 =>
   table_value := X"02CB";
```

```
WHEN 132 =>
   table_value := X"02C9";
WHEN 133 =>
   table_value := X"02C7";
WHEN 134 =>
   table_value := X"02C4";
WHEN 135 =>
   table_value := X"02C0";
WHEN 136 =>
   table_value := X"02BC";
WHEN 137 =>
   table_value := X"02B8";
WHEN 138 =>
   table_value := X"02B3";
WHEN 139 =>
   table_value := X"02AE";
WHEN 140 =>
   table_value := X"02A8";
WHEN 141 =>
   table_value := X"02A1";
WHEN 142 =>
   table_value := X"029B";
WHEN 143 =>
   table_value := X"0294";
WHEN 144 =>
   table_value := X"028C";
WHEN 145 =>
   table_value := X"0284";
WHEN 146 =>
   table_value := X"027C";
WHEN 147 =>
   table_value := X"0274";
WHEN 148 =>
   table_value := X"026B";
WHEN 149 =>
   table_value := X"0261";
WHEN 150 =>
   table_value := X"0258";
WHEN 151 =>
   table_value := X"024E";
WHEN 152 =>
   table_value := X"0244";
WHEN 153 =>
   table_value := X"0239";
WHEN 154 =>
   table_value := X"022F";
WHEN 155 =>
   table_value := X"0224";
WHEN 156 =>
   table_value := X"0219";
WHEN 157 =>
   table_value := X"020E";
WHEN 158 =>
   table_value := X"0203";
WHEN 159 =>
   table_value := X"01F7";
WHEN 160 =>
   table_value := X"01EB";
WHEN 161 =>
   table_value := X"01E0";
WHEN 162 =>
   table_value := X"01D4";
WHEN 163 =>
   table_value := X"01C8";
WHEN 164 =>
   table_value := X"01BC";
WHEN 165 =>
   table_value := X"01B0";
WHEN 166 =>
   table_value := X"01A4";
WHEN 167 =>
```

```
   table_value := X"0198";
WHEN 168 =>
   table_value := X"018C";
WHEN 169 =>
   table_value := X"0180";
WHEN 170 =>
   table_value := X"0174";
WHEN 171 =>
   table_value := X"0169";
WHEN 172 =>
   table_value := X"015D";
WHEN 173 =>
   table_value := X"0151";
WHEN 174 =>
   table_value := X"0146";
WHEN 175 =>
   table_value := X"013A";
WHEN 176 =>
   table_value := X"012F";
WHEN 177 =>
   table_value := X"0124";
WHEN 178 =>
   table_value := X"0119";
WHEN 179 =>
   table_value := X"010E";
WHEN 180 =>
   table_value := X"0104";
WHEN 181 =>
   table_value := X"00F9";
WHEN 182 =>
   table_value := X"00EF";
WHEN 183 =>
   table_value := X"00E5";
WHEN 184 =>
   table_value := X"00DB";
WHEN 185 =>
   table_value := X"00D1";
WHEN 186 =>
   table_value := X"00C8";
WHEN 187 =>
   table_value := X"00BF";
WHEN 188 =>
   table_value := X"00B6";
WHEN 189 =>
   table_value := X"00AD";
WHEN 190 =>
   table_value := X"00A5";
WHEN 191 =>
   table_value := X"009D";
WHEN 192 =>
   table_value := X"0095";
WHEN 193 =>
   table_value := X"008D";
WHEN 194 =>
   table_value := X"0086";
WHEN 195 =>
   table_value := X"007F";
WHEN 196 =>
   table_value := X"0078";
WHEN 197 =>
   table_value := X"0071";
WHEN 198 =>
   table_value := X"006A";
WHEN 199 =>
   table_value := X"0064";
WHEN 200 =>
   table_value := X"005E";
WHEN 201 =>
   table_value := X"0059";
WHEN 202 =>
   table_value := X"0053";
```

```
WHEN 203 =>
   table_value := X"004E";
WHEN 204 =>
   table_value := X"0049";
WHEN 205 =>
   table_value := X"0044";
WHEN 206 =>
   table_value := X"0040";
WHEN 207 =>
   table_value := X"003B";
WHEN 208 =>
   table_value := X"0037";
WHEN 209 =>
   table_value := X"0033";
WHEN 210 =>
   table_value := X"0030";
WHEN 211 =>
   table_value := X"002C";
WHEN 212 =>
   table_value := X"0029";
WHEN 213 =>
   table_value := X"0026";
WHEN 214 =>
   table_value := X"0023";
WHEN 215 =>
   table_value := X"0020";
WHEN 216 =>
   table_value := X"001D";
WHEN 217 =>
   table_value := X"001B";
WHEN 218 =>
   table_value := X"0019";
WHEN 219 =>
   table_value := X"0017";
WHEN 220 =>
   table_value := X"0015";
WHEN 221 =>
   table_value := X"0013";
WHEN 222 =>
   table_value := X"0011";
WHEN 223 =>
   table_value := X"000F";
WHEN 224 =>
   table_value := X"000E";
WHEN 225 =>
   table_value := X"000D";
WHEN 226 =>
   table_value := X"000B";
WHEN 227 =>
   table_value := X"000A";
WHEN 228 =>
   table_value := X"0009";
WHEN 229 =>
   table_value := X"0008";
WHEN 230 =>
   table_value := X"0007";
WHEN 231 =>
   table_value := X"0006";
WHEN 232 =>
   table_value := X"0006";
WHEN 233 =>
   table_value := X"0005";
WHEN 234 =>
   table_value := X"0004";
WHEN 235 =>
   table_value := X"0004";
WHEN 236 =>
   table_value := X"0003";
WHEN 237 =>
   table_value := X"0003";
WHEN 238 =>
```

```
            table_value := X"0003";
         WHEN 239 =>
            table_value := X"0002";
         WHEN 240 =>
            table_value := X"0002";
         WHEN 241 =>
            table_value := X"0002";
         WHEN 242 =>
            table_value := X"0001";
         WHEN 243 =>
            table_value := X"0001";
         WHEN 244 =>
            table_value := X"0001";
         WHEN 245 =>
            table_value := X"0001";
         WHEN 246 =>
            table_value := X"0001";
         WHEN 247 =>
            table_value := X"0000";
         WHEN 248 =>
            table_value := X"0000";
         WHEN 249 =>
            table_value := X"0000";
         WHEN 250 =>
            table_value := X"0000";
         WHEN 251 =>
            table_value := X"0000";
         WHEN 252 =>
            table_value := X"0000";
         WHEN 253 =>
            table_value := X"0000";
         WHEN 254 =>
            table_value := X"0000";
         WHEN 255 =>
            table_value := X"0000";
      END CASE;
      RETURN table_value;
   END;
END;
```

A.5 MPPT Algorithm with Error Checking

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

ENTITY mppt IS
   PORT
   (
      v_in                    : IN  UNSIGNED(15 DOWNTO 0);
      i_in                    : IN  UNSIGNED(15 DOWNTO 0);
      ready_in                : IN  BIT;
      clk                        : IN  BIT;

      vref_out                : OUT UNSIGNED(15 DOWNTO 0);
      ready_out                  : OUT BIT;

      ovr                        : OUT BIT;
      skip_mppt                  : OUT BIT;
      dv_test                    : OUT BIT;
      didv_test                  : OUT BIT;
      dp_test                    : OUT BIT;
      mppt_dir                : OUT BIT
   );
END mppt;

ARCHITECTURE a OF mppt IS
   TYPE STATE_TYPE IS (s0, s1, s2, s3, s4, s5, s6, s7);
   SIGNAL state: STATE_TYPE;

   SIGNAL v_prev: UNSIGNED(15 DOWNTO 0);
   SIGNAL i_prev: UNSIGNED(15 DOWNTO 0);
   SIGNAL p_prev: UNSIGNED(31 DOWNTO 0);
   SIGNAL vref_out_tmp: UNSIGNED(15 DOWNTO 0);

   SIGNAL ovr_flag: BIT;
   SIGNAL ready_out_tmp: BIT;
   SIGNAL skip_mppt_tmp: BIT;
   SIGNAL dv_test_tmp: BIT;
   SIGNAL didv_test_tmp: BIT;
   SIGNAL dp_test_tmp: BIT;
   SIGNAL mppt_dir_tmp: BIT;

   SIGNAL v_curr: UNSIGNED(15 DOWNTO 0);
   SIGNAL i_curr: UNSIGNED(15 DOWNTO 0);
   SIGNAL p_curr: UNSIGNED(31 DOWNTO 0);

   SIGNAL delta_v: INTEGER;
   SIGNAL delta_i: INTEGER;
   SIGNAL delta_p: INTEGER;
   SIGNAL vref_new_tmp: UNSIGNED(15 DOWNTO 0);
   SIGNAL ready_count: INTEGER RANGE 0 TO 7;

   CONSTANT VREF_MIN: INTEGER := 22282;
   CONSTANT I_MIN: INTEGER := 338;
   CONSTANT V_STEP: INTEGER := 64;
   CONSTANT V_STEP_X2: INTEGER := 128;
   CONSTANT MIN_DV: INTEGER := 30;
   CONSTANT OVR_THRESHOLD: INTEGER := 4096;

BEGIN
   PROCESS (clk)
      VARIABLE dv_check: BOOLEAN;
      VARIABLE didv_check: BOOLEAN;
      VARIABLE dp_check: BOOLEAN;
      VARIABLE v_diff: INTEGER;
   BEGIN
      IF (clk='1' AND clk'EVENT) THEN
```

```
CASE state IS
   -- STATE 0
   WHEN s0 =>

      -- ALTERATIONS: assign our values to signals
      IF (ready_in='1') THEN
         v_curr <= v_in;
         i_curr <= i_in;
         p_curr <= v_in * i_in;
         state <= s1;
      ELSE
         v_curr <= v_curr;
         i_curr <= i_curr;
         p_curr <= p_curr;
         state <= s0;
      END IF;

      -- FLAGS
      ready_out_tmp <= '0';
      ovr_flag <= ovr_flag;

      -- CONSERVATIONS
      delta_v <= delta_v;              -- FROM STATE 1
      delta_i <= delta_i;
      delta_p <= delta_p;

      dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
      didv_test_tmp <= didv_test_tmp;
      dp_test_tmp <= dp_test_tmp;

      mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
      skip_mppt_tmp <= skip_mppt_tmp;
      vref_new_tmp <= vref_new_tmp;    -- FROM STATE 3, 4, 5

      vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
      v_prev <= v_prev;
      i_prev <= i_prev;
      p_prev <= p_prev;

      ready_count <= 0;             -- FROM STATE 7


   -- STATE 1
   WHEN s1 =>
      -- ALTERATIONS: calculate our differentials
      delta_v <= TO_INTEGER(v_curr) - TO_INTEGER(v_prev);
      delta_i <= TO_INTEGER(i_curr) - TO_INTEGER(i_prev);
      delta_p <= TO_INTEGER(p_curr) - TO_INTEGER(p_prev);
      state <= s2;

      -- FLAGS
      ready_out_tmp <= '0';
      ovr_flag <= ovr_flag;

      -- CONSERVATIONS
      v_curr <= v_curr;             -- FROM STATE 0
      i_curr <= i_curr;
      p_curr <= p_curr;

      dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
      didv_test_tmp <= didv_test_tmp;
      dp_test_tmp <= dp_test_tmp;

      mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
      skip_mppt_tmp <= skip_mppt_tmp;
      vref_new_tmp <= vref_new_tmp;    -- FROM STATE 3, 4, 5

      vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
      v_prev <= v_prev;
      i_prev <= i_prev;
      p_prev <= p_prev;
```

```
   ready_count <= 0;             -- FROM STATE 7


-- STATE 2
WHEN s2 =>
   --ALTERATIONS: check requirements in parameter changes
   dv_check := (ABS(delta_v) < MIN_DV);
   didv_check := ((delta_v < 0) XNOR (delta_i < 0));
   dp_check := (delta_p = 0);
   IF (dv_check) THEN dv_test_tmp <= '1';
   ELSE dv_test_tmp <= '0'; END IF;

   IF (didv_check) THEN didv_test_tmp <= '1';
   ELSE didv_test_tmp <= '0'; END IF;

   IF (dp_check) THEN dp_test_tmp <= '1';
   ELSE dp_test_tmp <= '0'; END IF;

   v_diff:= TO_INTEGER(v_curr) - TO_INTEGER(vref_out_tmp);

   IF (v_diff < OVR_THRESHOLD) THEN
      ovr_flag <= '0';
   ELSE
      ovr_flag <= '1';
   END IF;
   state <= s3;

   -- FLAGS
   ready_out_tmp <= '0';

   -- CONSERVATIONS
   v_curr <= v_curr;             -- FROM STATE 0
   i_curr <= i_curr;
   p_curr <= p_curr;

   delta_v <= delta_v;              -- FROM STATE 1
   delta_i <= delta_i;
   delta_p <= delta_p;

   mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
   skip_mppt_tmp <= skip_mppt_tmp;
   vref_new_tmp <= vref_new_tmp;    -- FROM STATE 3, 4, 5

   vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
   v_prev <= v_prev;
   i_prev <= i_prev;
   p_prev <= p_prev;

   ready_count <= 0;             -- FROM STATE 7


-- STATE 3
WHEN s3 =>
-- ALTERATIONS: check to see if we should do mppt
   IF (ovr_flag='1') THEN
      vref_new_tmp <= vref_out_tmp - V_STEP;
      mppt_dir_tmp <= mppt_dir_tmp;
      skip_mppt_tmp <= '1';
   ELSE
      IF ((dv_test_tmp='0') AND (didv_test_tmp='0') AND
         (dp_test_tmp='0')) THEN
         IF ((delta_v < 0) XOR (delta_p <0)) THEN
            vref_new_tmp <= vref_out_tmp - V_STEP;
            mppt_dir_tmp <= '0';
         ELSE
            vref_new_tmp <= vref_out_tmp + V_STEP;
            mppt_dir_tmp <= '1';
         END IF;
         skip_mppt_tmp <= '0';
      ELSE
```

```
            mppt_dir_tmp <= mppt_dir_tmp;
            vref_new_tmp <= vref_out_tmp;
            skip_mppt_tmp <= '1';
         END IF;
      END IF;
      state <= s4;

      -- FLAGS
      ready_out_tmp <= '0';
      ovr_flag <= ovr_flag;

      -- CONSERVATIONS
      v_curr <= v_curr;             -- FROM STATE 0
      i_curr <= i_curr;
      p_curr <= p_curr;

      delta_v <= delta_v;              -- FROM STATE 1
      delta_i <= delta_i;
      delta_p <= delta_p;

      dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
      didv_test_tmp <= didv_test_tmp;
      dp_test_tmp <= dp_test_tmp;

      vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
      v_prev <= v_prev;
      i_prev <= i_prev;
      p_prev <= p_prev;

      ready_count <= 0;             -- FROM STATE 7


   -- STATE 4
   WHEN s4 =>
      -- ALTERATIONS: perform bounds checking on new vref
      IF (vref_new_tmp < VREF_MIN) THEN
         vref_new_tmp <= TO_UNSIGNED(VREF_MIN, 16);
      ELSE
         vref_new_tmp <= vref_new_tmp;
      END IF;

      state <= s5;

      -- FLAGS
      ready_out_tmp <= '0';
      ovr_flag <= ovr_flag;

      -- CONSERVATIONS
      v_curr <= v_curr;             -- FROM STATE 0
      i_curr <= i_curr;
      p_curr <= p_curr;

      delta_v <= delta_v;              -- FROM STATE 1
      delta_i <= delta_i;
      delta_p <= delta_p;

      dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
      didv_test_tmp <= didv_test_tmp;
      dp_test_tmp <= dp_test_tmp;

      mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
      skip_mppt_tmp <= skip_mppt_tmp;

      vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
      v_prev <= v_prev;
      i_prev <= i_prev;
      p_prev <= p_prev;

      ready_count <= 0;             -- FROM STATE 7
```

```
-- STATE 5
WHEN s5 =>
   -- ALTERATIONS: perform low current checking
   IF ((i_curr < I_MIN) AND (ovr_flag='0')) THEN
      vref_new_tmp <= vref_new_tmp - V_STEP_X2;
   ELSE
      vref_new_tmp <= vref_new_tmp;
   END IF;

   state <= s6;

   -- FLAGS
   ready_out_tmp <= '0';
   ovr_flag <= ovr_flag;

   -- CONSERVATIONS
   v_curr <= v_curr;             -- FROM STATE 0
   i_curr <= i_curr;
   p_curr <= p_curr;

   delta_v <= delta_v;              -- FROM STATE 1
   delta_i <= delta_i;
   delta_p <= delta_p;

   dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
   didv_test_tmp <= didv_test_tmp;
   dp_test_tmp <= dp_test_tmp;

   mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
   skip_mppt_tmp <= skip_mppt_tmp;

   vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
   v_prev <= v_prev;
   i_prev <= i_prev;
   p_prev <= p_prev;

   ready_count <= 0;             -- FROM STATE 7


-- STATE 6
WHEN s6 =>

   -- ALTERATIONS: update vref, save values for next iteration
   vref_out_tmp <= vref_new_tmp;
   v_prev <= v_curr;
   i_prev <= i_curr;
   p_prev <= p_curr;
   state <= s7;

   -- FLAGS
   ready_out_tmp <= '0';
   ovr_flag <= ovr_flag;

   -- CONSERVATIONS
   v_curr <= v_curr;             -- FROM STATE 0
   i_curr <= i_curr;
   p_curr <= p_curr;

   delta_v <= delta_v;              -- FROM STATE 1
   delta_i <= delta_i;
   delta_p <= delta_p;

   dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
   didv_test_tmp <= didv_test_tmp;
   dp_test_tmp <= dp_test_tmp;

   mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
   skip_mppt_tmp <= skip_mppt_tmp;
   vref_new_tmp <= vref_new_tmp;    -- FROM STATE 3, 4, 5

   ready_count <= 0;             -- FROM STATE 7
```

```
            -- STATE 7
            WHEN s7 =>

               -- ALTERATIONS: send ready
               IF (ready_count=3) THEN
                  ready_count <= 0;
                  state <= s0;
               ELSE
                  ready_count <= ready_count + 1;
                  state <= s7;
               END IF;

               -- FLAGS
               ready_out_tmp <= '1';
               ovr_flag <= ovr_flag;

               -- CONSERVATIONS
               v_curr <= v_curr;             -- FROM STATE 0
               i_curr <= i_curr;
               p_curr <= p_curr;

               delta_v <= delta_v;              -- FROM STATE 1
               delta_i <= delta_i;
               delta_p <= delta_p;

               dv_test_tmp <= dv_test_tmp;         -- FROM STATE 2
               didv_test_tmp <= didv_test_tmp;
               dp_test_tmp <= dp_test_tmp;

               mppt_dir_tmp <= mppt_dir_tmp;    -- FROM STATE 3
               skip_mppt_tmp <= skip_mppt_tmp;
               vref_new_tmp <= vref_new_tmp;    -- FROM STATE 3, 4, 5

               vref_out_tmp <= vref_out_tmp;    -- FROM STATE 6
               v_prev <= v_prev;
               i_prev <= i_prev;
               p_prev <= p_prev;


         END CASE;
      END IF;
   END PROCESS;

   vref_out <= vref_out_tmp;
   ready_out <= ready_out_tmp;
   dv_test <= dv_test_tmp;
   didv_test <= didv_test_tmp;
   dp_test <= dp_test_tmp;
   mppt_dir <= mppt_dir_tmp;
   skip_mppt <= skip_mppt_tmp;
   ovr <= ovr_flag;
END a;
```

A.6 CORDIC Sine Generator

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;
LIBRARY work;
USE work.cordic_lut_vhdl.ALL;

ENTITY cordic IS
   PORT
   (
      clk                        : IN  BIT;
      sine                     : OUT SIGNED(15 DOWNTO 0);
      ready_out                  : OUT BIT
   );
END cordic;

ARCHITECTURE a OF cordic IS
   TYPE STATE_TYPE IS (s0, s1, s2, s3);
   SIGNAL state: STATE_TYPE;
   SIGNAL i,q: SIGNED(31 DOWNTO 0);
   SIGNAL cum_angle: SIGNED(8 DOWNTO 0);
   SIGNAL cur_angle: SIGNED(8 DOWNTO 0);
   SIGNAL count: NATURAL RANGE 0 TO 15;
   SIGNAL delay_count: UNSIGNED(7 DOWNTO 0);
   SIGNAL phase_dir: BIT;
   SIGNAL sine_out: SIGNED(15 DOWNTO 0);
   SIGNAL ready_out_tmp: BIT;

   -- 200 mV
   CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"031C0000";
   CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FCE40000";

   -- 180 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"02CD0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FD330000";

   -- 160 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"027D0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FD830000";

   -- 140 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"022D0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FDD30000";

   -- 120 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"01DE0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FE220000";

   -- 100 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"018E0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FE720000";

   -- 80 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"013F0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FEC10000";

   -- 60 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"00EF0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FF110000";

   -- 40 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"009F0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FF610000";

   -- 20 mV
-- CONSTANT MAG_P: SIGNED(31 DOWNTO 0):=X"004F0000";
-- CONSTANT MAG_N: SIGNED(31 DOWNTO 0):=X"FFB10000";
```

```
   CONSTANT SIGNED_P127: SIGNED(8 DOWNTO 0):="001111111";
   CONSTANT SIGNED_N128: SIGNED(8 DOWNTO 0):="110000000";
   CONSTANT FREQDIV: UNSIGNED(7 DOWNTO 0):=X"BD"; -- 189
BEGIN
   PROCESS (clk)
   BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         CASE state IS
            -- STATE 0:
            WHEN s0 =>
               state <= s1;
               i <= X"00000000";
               cur_angle<=cur_angle;
               ready_out_tmp <= '0';
               IF (cur_angle(7)='1') THEN
                  cum_angle <= SIGNED_N128;  -- (-128)
                  q <= MAG_N;
               ELSE
                  cum_angle <= SIGNED_P127;  -- (+127)
                  q <= MAG_P;
               END IF;

            -- STATE 1:
            WHEN s1 =>
               IF (count=8) THEN
                  state <= s2;
                  count <= 0;
                  sine_out <= q(31 DOWNTO 16);
                  IF (cur_angle=SIGNED_P127) OR (cur_angle=SIGNED_N128) THEN
                     phase_dir <= NOT(phase_dir);
                  ELSE
                     phase_dir <= phase_dir;
                  END IF;
               ELSE
                  state <= s1;
                  IF (cum_angle > cur_angle) THEN
                     q <= q - SHIFT_RIGHT(i,count);
                     i <= i + SHIFT_RIGHT(q,count);
                     cum_angle <= cum_angle - get_table_value(count);
                  ELSE
                     q <= q + SHIFT_RIGHT(i,count);
                     i <= i - SHIFT_RIGHT(q,count);
                     cum_angle <= cum_angle + get_table_value(count);
                  END IF;
                  count <= count+1;
               END IF;
               cur_angle<=cur_angle;
               ready_out_tmp <= '0';
            -- STATE 2:
            WHEN s2 =>
               IF (delay_count=FREQDIV) THEN
                  state <= s3;
                  delay_count <= "00000000";
               ELSE
                  state <= s2;
                  delay_count <= delay_count+1;
               END IF;
               ready_out_tmp <= '0';
               cur_angle<=cur_angle;
            -- STATE 3:
            WHEN s3 =>
               state <= s0;
               ready_out_tmp <= '1';
               IF (phase_dir='0') THEN
                  cur_angle<=cur_angle+1;
               ELSE
                  cur_angle<=cur_angle-1;
               END IF;
         END CASE;
      END IF;
```

```
      sine <= sine_out;
      ready_out <= ready_out_tmp;
   END PROCESS;
END a;
```

A.7 CORDIC Lookup Table

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

PACKAGE cordic_lut_vhdl IS
   SUBTYPE table_value_type IS SIGNED(7 DOWNTO 0);
   SUBTYPE table_index_type IS NATURAL RANGE 0 TO 15;
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type;
END;

PACKAGE BODY cordic_lut_vhdl IS
   FUNCTION get_table_value (table_index: table_index_type) RETURN table_value_type IS
      VARIABLE table_value: table_value_type;
   BEGIN
      CASE table_index IS
         -- positive values
         WHEN 0 =>
            table_value := "01000000";    -- (+64)
         WHEN 1 =>
            table_value := "00100101";    -- (+37)
         WHEN 2 =>
            table_value := "00010100";    -- (+20)
         WHEN 3 =>
            table_value := "00001010";    -- (+10)
         WHEN 4 =>
            table_value := "00000101";    -- (+5)
         WHEN 5 =>
            table_value := "00000011";    -- (+3)
         WHEN 6 =>
            table_value := "00000001";    -- (+1)
         WHEN 7 =>
            table_value := "00000001";    -- (+1)
         WHEN others =>
            table_value := "00000000";    -- (0)
      END CASE;
      RETURN table_value;
   END;
END;
```

A.8 Dither and Reference Voltage Summation Block

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

ENTITY dither_sum IS
   PORT
   (
      clk                        : IN  BIT;
      sine                    : IN  SIGNED(15 DOWNTO 0);
      ready_sine                 : IN  BIT;
      vref                    : IN  UNSIGNED(15 DOWNTO 0);
      ready_vref                 : IN  BIT;
      ovr                        : IN  BIT;
      sum_data                : OUT UNSIGNED(15 DOWNTO 0);
      ready_out                  : OUT BIT
   );
END dither_sum;

ARCHITECTURE a OF dither_sum IS
   TYPE STATE_TYPE IS (s0, s1, s2);
   SIGNAL state: STATE_TYPE;

   SIGNAL ready_out_tmp: BIT;
   SIGNAL dac_data_tmp: SIGNED(15 DOWNTO 0);
   SIGNAL vref_curr: UNSIGNED(15 DOWNTO 0);
   SIGNAL sine_curr: SIGNED(15 DOWNTO 0);

   SIGNAL ready_count: INTEGER RANGE 0 TO 7;
BEGIN
   PROCESS(clk)
   BEGIN
      IF (clk='1' AND clk'EVENT) THEN
         CASE state IS
            -- STATE 0
            WHEN s0 =>
               IF(ready_sine='1') THEN
                  sine_curr <= sine;
               ELSE
                  sine_curr <= sine_curr;
               END IF;

               IF(ready_vref='1') THEN
                  vref_curr <= vref;
               ELSE
                  vref_curr <= vref_curr;
               END IF;

               IF(ready_sine='1' OR ready_vref='1') THEN
                  state <= s1;
               ELSE
                  state <= s0;
               END IF;

               dac_data_tmp <= dac_data_tmp;
               ready_out_tmp <= '0';

            -- STATE 1
            WHEN s1 =>
               IF(ovr='0') THEN
                  dac_data_tmp <= TO_INTEGER(vref_curr) + sine_curr;
               ELSE
                  dac_data_tmp <= TO_SIGNED(TO_INTEGER(vref_curr),16);
               END IF;
               ready_out_tmp <= '0';
               vref_curr <= vref_curr;
               sine_curr <= sine_curr;
```

```
               state <= s2;

            -- STATE 2
            WHEN s2 =>
               state <= s0;
               ready_out_tmp <= '1';
               vref_curr <= vref_curr;
               sine_curr <= sine_curr;
               dac_data_tmp <= dac_data_tmp;

         END CASE;
      END IF;
   END PROCESS;

   sum_data <= TO_UNSIGNED(TO_INTEGER(dac_data_tmp),16);
   ready_out <= ready_out_tmp;
END a;
```

A.9 DAC Controller

```
LIBRARY IEEE;
USE IEEE.numeric_bit.ALL;

ENTITY dac_control IS
   PORT
   (
      clk                        : IN  BIT;

      dac_data_1                 : IN  UNSIGNED(15 DOWNTO 0);
      dac_data_2                 : IN  UNSIGNED(15 DOWNTO 0);
      dac_data_3                 : IN  UNSIGNED(15 DOWNTO 0);
      dac_data_4                 : IN  UNSIGNED(15 DOWNTO 0);

      dac_data                : OUT UNSIGNED(15 DOWNTO 0);
      dac_cs                     : OUT BIT;
      dac_ldac                : OUT BIT;
      dac_pd                     : OUT BIT;
      dac_rst                    : OUT BIT;
      dac_rw                     : OUT BIT;
      dac_a0                     : OUT BIT;
      dac_a1                     : OUT BIT;

      vgen1                   : OUT UNSIGNED(15 DOWNTO 0);
      vgen2                   : OUT UNSIGNED(15 DOWNTO 0);
      vgen3                   : OUT UNSIGNED(15 DOWNTO 0);
      vgen4                   : OUT UNSIGNED(15 DOWNTO 0);
      rampgen                    : OUT UNSIGNED(15 DOWNTO 0)
   );
END dac_control;

ARCHITECTURE a OF dac_control IS
   TYPE STATE_TYPE IS (a0, a1, a2, b0, b1, b2, c0, c1, c2, d0, d1, d2);
   SIGNAL state: STATE_TYPE;
   SIGNAL tmp_dac_ldac: BIT;
   SIGNAL tmp_dac_cs: BIT;
   SIGNAL tmp_a0: BIT;
   SIGNAL tmp_a1: BIT;
   SIGNAL tmp_dac_data: UNSIGNED(15 DOWNTO 0);
   SIGNAL ramp_counter: UNSIGNED(15 DOWNTO 0);
BEGIN
   PROCESS (clk)
   BEGIN
      IF (clk'EVENT AND clk = '1') THEN
         CASE state IS
            -- STATE A0:
            WHEN a0 =>
               tmp_a0 <= '0';
               tmp_a1 <= '0';
               state<=a1;
               tmp_dac_data <= dac_data_1;
               tmp_dac_cs <= '1';
               tmp_dac_ldac <= '0';
            -- STATE A1:
            WHEN a1 =>
               state <= a2;
               tmp_dac_cs <= '0';
               tmp_dac_data <= tmp_dac_data;
               tmp_dac_ldac <= '0';
               tmp_a0 <= tmp_a0;
               tmp_a1 <= tmp_a1;
            -- STATE A2:
            WHEN a2 =>
               state <= b0;
               tmp_dac_cs <= '1';
               tmp_dac_data <= tmp_dac_data;
```

```
   tmp_dac_ldac <= '1';
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;

-- STATE B0:
WHEN b0 =>
   tmp_a0 <= '1';
   tmp_a1 <= '0';
   state <= b1;
   tmp_dac_data <= dac_data_2;
   tmp_dac_cs <= '1';
   tmp_dac_ldac <= '0';
-- STATE B1:
WHEN b1 =>
   state <= b2;
   tmp_dac_cs <= '0';
   tmp_dac_data <= tmp_dac_data;
   tmp_dac_ldac <= '0';
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;
-- STATE B2:
WHEN b2 =>
   state <= c0;
   tmp_dac_cs <= '1';
   tmp_dac_data <= tmp_dac_data;
   tmp_dac_ldac <= '1';
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;

-- STATE C0:
WHEN c0 =>
   tmp_a0 <= '0';
   tmp_a1 <= '1';
   state<=c1;
   tmp_dac_data <= dac_data_3;
   tmp_dac_cs <= '1';
   tmp_dac_ldac <= '0';
-- STATE C1:
WHEN c1 =>
   state <= c2;
   tmp_dac_cs <= '0';
   tmp_dac_data <= tmp_dac_data;
   tmp_dac_ldac <= '0';
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;
-- STATE C2:
WHEN c2 =>
   state <= d0;
   tmp_dac_cs <= '1';
   tmp_dac_data <= tmp_dac_data;
   tmp_dac_ldac <= '1';
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;

-- STATE D0:
WHEN d0 =>
   state <= d1;
   tmp_a0 <= '1';
   tmp_a1 <= '1';
   tmp_dac_data <= dac_data_4;
   tmp_dac_cs <= '1';
   tmp_dac_ldac <= '0';
-- STATE D1:
WHEN d1 =>
   state <= d2;
   tmp_a0 <= tmp_a0;
   tmp_a1 <= tmp_a1;
   tmp_dac_cs <= '0';
   tmp_dac_data <= tmp_dac_data;
   tmp_dac_ldac <= '0';
-- STATE D2:
```

```
            WHEN d2 =>
               state <= a0;
               tmp_a0 <= tmp_a0;
               tmp_a1 <= tmp_a1;
               tmp_dac_cs <= '1';
               tmp_dac_data <= tmp_dac_data;
               tmp_dac_ldac <= '1';
         END CASE;
      END IF;

      ramp_counter <= ramp_counter + 1;
   END PROCESS;
   dac_data <= tmp_dac_data;
   dac_cs <= tmp_dac_cs;
   dac_ldac <= tmp_dac_ldac;
   dac_pd <= '1';
   dac_rst <= '0';
   dac_rw <= '0';
   dac_a0 <= tmp_a0;
   dac_a1 <= tmp_a1;

   rampgen <= ramp_counter(15 DOWNTO 0);
   vgen1 <= X"3333";
   vgen2 <= X"6666";
   vgen3 <= X"9999";
   vgen4 <= X"CCCC";
END a;
```

REFERENCES

[1] M. Shahidehpour and F. Schwarts, "Don't let the sun go down on PV," in *IEEE Power and Energy Magazine*, vol. 2, 2004, pp. 40-48.

[2] M. A. Green, "Photovoltaics: coming of age," Photovoltaic Specialists Conference, 1990.

[3] T. Surek, "Progress in U.S. photovoltaics: Looking back 30 years and looking ahead 20," World Conference on Photovoltaic Energy Conversion, Osaka, Japan, 2003.

[4] B. L. Tan and K. J. Tseng, "Intelligent and reliable power supply system for small satellites," International Telecommunications Energy Conference, 2003.

[5] H. W. Bradhorst, M. J. O'Neill, and M. Eskenazi, "Photovoltaic options for increased satellite power at lower cost," World Conference on Photovoltaic Energy Conversion, 2003.

[6] E. V. Slodovnik, S. Liu, and R. A. Dougal, "Power controller design for maximum power tracking in solar installations," *IEEE Transactions on Power Electronics*, vol. 19, pp. 1295-1304, 2004.

[7] K. H. Hussein, I. Muta, T. Hoshino, and M. Osakada, "Maximum photovoltaic power tracking: an algorithm for rapidly changing atmospheric conditions," Generation, Transmition, and Distribution, 1995.

[8] H. Sugimoto and H. Dong, "A new scheme for maximum photovoltaic power tracking control," Power Conversion Conferece - Nagaoka, 1997.

[9] S. Gasner, K. Sharmit, P. Stella, C. Craig, and S. Murnaw, "The Stardust solar array," World Photovoltaic Energy Conversion, 2003.

[10] W. Wu, N. Pongratananukul, W. Qiu, K. Rustom, T. Kasparis, and I. Batarseh, "DSP-based multiple peak power tracking for expandable power system," Applied Power Electronics Conference and Exposition, 2003.

[11] C. Hua, J. Lin, and C. Shen, "Implementation of a DSP-Controlled Photovoltaic System with Peak Power Tracking," *IEEE Transactions on Industrial Electronics*, vol. 45, pp. 99-107, 1998.

[12] D. P. Hohm and M. E. Ropp, "Comparative Study of Maximum Power Point Tracking Algorithms Using an Experimental, Programmable, Maximum Power Point Tracking Test Bed," pp. 1699-1702, 2000.

[13] Altera Corporation, "FLEX10K Embedded Programmable Device Logic Family Data Sheet," 2003.

[14] Xilinx Corporation, "Virtex-4 Product Backgrounder," vol. 2004, 2004.

[15] C. H. Roth, *Digital Systems Design Using VHDL*. Boston: PWS Publishing Company, 1998.

[16] K. Skahill, *VHDL for Programmable Logic*. Menlo Park, CA: Addison-Wesley Publishing Company, 1996.

[17] S. Krauter and F. Ochs, "The integrated solar home system," World Conference on Photovoltaic Energy Conversion, Osaka, Japan, 2003.

[18] A. K. Sinha and P. R. Chadha, "Investigation on the effect of various dithers in a noisy communication channel," TENCON, 1989.

[19] K. Siri, V. A. Caliskan, C. Q. Lee, and G. C. Agarwal, "Peak power tracking in parallel connected converters," IEEE Proceedings on Circuits, Devices and Systems, 1992.

[20] H. J. Beukes and J. H. R. Enslin, "Analysis of a new compound converter as MPPT, battery regulator and bus regulator for satellite power systems," IEEE Power Electronics Specialists Conference, 1993.

[21] E. Koutroulis, K. Kalaitzakis, and N. C. Voulgaris, "Development of a microcontroller-based, photovoltaic maximum power point tracking control system," *IEEE Transactions on Power Electronics*, vol. 16, pp. 46-54, 2001.

[22] K. Siri and K. A. Conner, "Sequentially Controlled Distributed Solar-Array Power System with Maximum Power Tracking," *IEEE Transactions on Automatic Control*, 2004.

[23] W. Swiegers and J. H. R. Enslin, "An integrated maximum power point tracker for photovoltaic panels," IEEE International Symposium on Industrial Electronics, 1998.

[24] S. Walters, "Computer-Aided Prototyping for ASIC-Based Systems," *IEEE Design and Test of Computers*, vol. 4, pp. 4-10, 1991.

[25] D. M. Lewis, D. R. Galloway, M. V. Ierssel, J.Rose, and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System," International Symposium on FPGAs, 1997.

[26] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice Hall, 1996.

[27] J. Logue, "Virtex(tm) Synthesizable Delta-Sigma DAC," 1999.

[28] J. Logue, "Virtex Analog to Digital Converter," 1999.

[29] F. Giraud and Z. M. Salameh, "Analysis of the effects of a passing cloud on a grid-interactive photovoltaic system with battery storage using neural networks," *IEEE Transactions on Energy Conversion*, vol. 14, pp. 1572-1577, 1999.

[30] T.-F. Wu, C.-H. Chang, and Y.-K. Chen, "A fuzzy logic controlled single-stage converter for PV powered lighting system applications," IEEE Industry Applications Conference, 1999.