A modelica based rapid prototyping scheme for power electronic circuits

Shashank Krishnamurthy
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Recommended Citation
https://lib.dr.iastate.edu/rtd/19464

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A modelica based rapid prototyping scheme for power electronic circuits

by

Shashank Krishnamurthy

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical Engineering

Program of Study Committee:
Venkataramana Ajjarapu, Major Professor
Randall Geiger
Gurpur Prabhu

Iowa State University
Ames, Iowa
2003

Copyright © Shashank Krishnamurthy, 2003. All rights reserved.
This is to certify that the Master's thesis of

Shashank Krishnamurthy

Has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy
DEDICATED TO MY PARENTS
TABLE OF CONTENTS

CHAPTER 1. RAPID PROTOTYPING 1

1.1 Introduction 1
1.2 Rapid prototyping in the manufacturing industry 1
  1.2.1 Automotive industry example 2
1.3 Rapid prototyping in the software industry 3
1.4 Power electronic systems 4
1.5 Work scope and thesis outline 6

CHAPTER 2. RAPID PROTOTYPING OF POWER ELECTRONIC SYSTEMS 8

2.1 Introduction 8
2.2 Modeling and simulation of power electronic circuits 9
2.3 Controllers for power electronic applications 10
2.4 Rapid prototyping schemes for power electronic systems and motivation behind current work 11
2.5 Proposed rapid prototyping scheme 17
2.6 Conclusions 19

CHAPTER 3. DSPLIB—MODELICA LIBRARY OF DSP PERIPHERALS 20

3.1 Introduction 20
3.2 Dymola based simulation of modelica based physical systems 20
3.3 DSP library (DSPLib) in modelica 23
3.4 DSP unit sub library 24
  3.4.1 DSP timer 24
  3.4.2 Capture block 26
  3.4.3 Triggered timer 28
  3.4.4 Level detector 30
3.5 DSP interfaces sub library 32
### CHAPTER 6. RAPID PROTOTYPING OF DC-DC CONVERTER

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>60</td>
</tr>
<tr>
<td>6.2</td>
<td>Modeling and simulation of step down DC-DC converter</td>
<td>60</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Simulation diagram for DC-DC buck chopper</td>
<td>61</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Simulation results</td>
<td>62</td>
</tr>
<tr>
<td>6.3</td>
<td>Implementation of buck chopper</td>
<td>64</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Experimental setup and results</td>
<td>66</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Real-time control of duty ratio</td>
<td>68</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusions</td>
<td>69</td>
</tr>
</tbody>
</table>

### CHAPTER 7. RAPID PROTOTYPING OF SQUARE WAVE DC-AC INVERTER

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>70</td>
</tr>
<tr>
<td>7.2</td>
<td>Rapid prototyping of single-phase DC-AC inverter</td>
<td>71</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Simulation diagram for single-phase square wave inverter</td>
<td>71</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Simulation results</td>
<td>72</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Implementation of single phase square wave inverter</td>
<td>73</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Experimental setup and results</td>
<td>76</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Real-time control of frequency</td>
<td>77</td>
</tr>
<tr>
<td>7.3</td>
<td>Rapid prototyping of three-phase square wave inverter</td>
<td>78</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Simulation diagram for three-phase square wave inverter</td>
<td>78</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Simulation results</td>
<td>80</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Implementation of three phase square wave inverter</td>
<td>80</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Experimental setup and results</td>
<td>83</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Real-time control of frequency</td>
<td>84</td>
</tr>
<tr>
<td>7.4</td>
<td>Conclusions</td>
<td>85</td>
</tr>
</tbody>
</table>
CHAPTER 8. RAPID PROTOTYPING OF AC-DC CONVERTER

8.1 Introduction

8.2 Rapid prototyping of single-phase controlled rectifier

8.2.1 Simulation diagram for single-phase controlled rectifier

8.2.2 Simulation results

8.2.3 Implementation of single-phase controlled rectifier

8.2.4 Experimental setup and results

8.2.5 Real-time control of firing angle

8.3 Rapid prototyping of three-phase controlled rectifier

8.3.1 Simulation diagram for three-phase controlled rectifier

8.3.2 Simulation results

8.3.3 Implementation of three-phase controlled rectifier

8.3.4 Experimental setup and results

8.3.5 Real-time control of firing angle

8.4 Conclusions

CHAPTER 9. EXTENSION OF RAPID PROTOTYPING SCHEME FOR CLOSED LOOP CONTROL

9.1 Introduction

9.2 Framework for closed loop control

9.3 Modification in scheme to incorporate closed loop control

9.4 Closed loop control of DC-DC chopper

9.4.1 Closed loop control scheme

9.4.2 Experimental results

9.5 Conclusions

CHAPTER 10. CONCLUSIONS AND POTENTIAL FOR FUTURE WORK

10.1 Conclusions

10.2 Potential for future work
APPENDIX A. DC-DC CONVERTER LAB EXPERIMENT BASED ON RAPID PROTOTYPING SETUP 113

APPENDIX B. RAPID PROTOTYPING OF POWER ELECTRONIC CONVERTERS USING VISSIM RAPID PROTOTYPER SOFTWARE 125

APPENDIX C. PUBLICATIONS ARISING FROM THESIS 130

BIBLIOGRAPHY 131

ACKNOWLEDGEMENT 135
CHAPTER 1. RAPID PROTOTYPING

1.1 Introduction

Prototyping is an integral part of the complete design process of a product. Its purpose is to expose design errors and facilitate their elimination early in the design stage and preferably before the majority of resources required for production is committed. Introducing new products at faster rates is becoming crucial for remaining successful in a competitive economy; decreasing product development cycle times and increasing product complexity require new ways to realize innovative concepts.

The emerging rapid prototyping technologies are having a dramatic impact on the engineering design & validation process. When properly integrated, these technologies enable aggressive prototyping throughout the design process and decrease the likelihood of costly design changes late in the product development cycle. According to IEEE [1] rapid prototyping is defined as “a type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.”

1.2 Rapid prototyping in the manufacturing industry

Rapid prototyping schemes are widely used in the manufacturing industry [2]. In the manufacturing arena, productivity is achieved by guiding a product from concept to market quickly and inexpensively. Rapid prototyping technology aids this process. It automates the fabrication of a prototype from a three-dimensional (3-D) CAD drawing. This physical
model conveys more complete information about the product earlier in the development cycle.

The turnaround time for a typical rapid prototype part can take a few days. Conventional prototyping may take weeks or even months, depending on the method used. Fabrication processes fall into three categories: subtractive, additive, and compressive. In a subtractive process, a block of material is carved out to produce the desired shape. An additive process builds an object by joining particles or layers of raw material. A compressive process forces a semi-solid or liquid material into the desired shape, in which it is then induced to harden or solidify. Most conventional prototyping processes fall into the subtractive category. These would include machining processes such as milling, turning, and grinding. Machining methods are difficult to use on parts with very small internal cavities or complex geometries. Compressive processes, also conventional, include casting and molding. The new rapid prototyping technologies are additive processes. They can be categorized by material: photopolymer, thermoplastic, and adhesives. Rapid prototyping systems are capable of creating parts with small internal cavities and complex geometries. Also, the integration of rapid prototyping and compressive processes has resulted in the quicker generation of patterns from which molds are made.

1.2.1 Automotive industry example

Chrysler Corporation [2] has been actively involved with rapid prototyping technology since 1990. In the area of design verification, engineers on an engine upgrade program used Stereolithography (SLA) to design prototypes for distributor cap and body. Initial designs had a problem due to which the parts did not fit. The distributor cap was
redesigned and another part was built in 24 hours. If conventional methods had been used, the problem would not have been seen until the prototypes came back weeks later.

For airflow testing, Chrysler used SLA to find the optimum size for an ambient air duct. Three models were made in one day as opposed to four weeks using conventional prototyping. The models were tested, and the best choice was selected.

Chrysler has also used rapid prototyping to quickly create precise master patterns for secondary tooling applications such as room-temperature vulcanizing (RTV) molding, sand casting, resin transfer molding, vacuum forming, and squeeze molding. Parts produced include center consoles, interior trim panels, and instrument panel components.

1.3 Rapid prototyping in the software industry

In the software industry rapid prototyping pertains to the design of software prototypes. These prototypes are used to identify requirements, provide a platform for interaction between users and developers and focus on the requirements of the human interface needed. Prototyping has shown to reduce the code required to implement functions by up to 40%[3]. Some of the benefits of prototyping are:

1. Misunderstandings between software users and developers are exposed.
2. Missing services may be detected.
3. A working system is available early in the process.
4. The prototype may serve as a basis for deciding a system specification.
5. The system can support user interface validation and system testing.
1.4 Power electronic systems

Power electronic systems (PES) are widely used in a wide array of industrial applications. Computers, digital products, most modern industrial systems, automobiles, home appliances, motor controllers, and any other application that uses electrical energy involves some form of energy processing using a PES. With the advent of better semiconductor devices and faster processors, designers are now producing energy-efficient, more reliable, and very sophisticated power electronic systems. Typically the design process for a modern PES involves the following steps:

1. Design and simulation of trigger signal generation software algorithms for various power electronic controllers
2. Implementation of software controller algorithm on a target digital signal processor (DSP)
3. Validation and debugging of controller design
4. Integration of various software modules with required hardware and subsequent testing and validation

Based on a survey conducted [4] the design effort for a typical 50 week 1 engineer project is shown in figure 1.1. From the survey it was learnt that 25% of the development time is spent in developing efficient, scaled fixed-point algorithms including trapping all overflow and scaling problems, and writing efficient code to implement these algorithms. About 20% of the effort is focused on learning about DSP peripherals such as PWMs, encoders, analog and digital inputs and outputs, CAN bus, and so on, and programming these peripherals effectively to implement the control application. In many projects, engineers tend to re-use some of their earlier work such as segments of code or designs. It was estimated
that engineers spend about 15% of the project time repeating or attempting to repeat earlier code segments or designs. Another 15% of the project time is spent in debugging and validating the controller. During debugging it is tested whether the controller performs as designed at a functional level, and fixing any encountered problem during that process. Validation means testing the controller under actual end conditions – as close to reality as possible – and verifying that the controller works as designed under actual target conditions. Typically, the final stage in the project is integrating the controller with other handwritten software – user code – or external hardware, and validating the system performance. This usually takes about 25% of the total development time. This estimate however did not cover final product testing.

![Figure 1.1 Design effort distribution for a 50-week 1 engineer project](image-url)
The use of rapid prototyping tools would greatly reduce the time required for the overall design process and enables the designer to spend more time designing efficient controllers. Rapid prototyping tools integrate the modeling/simulation and implementation platforms. This enables the user to conceptualize the circuit using high level modeling tools and implement them using DSP based controllers and power electronic building blocks. The rapid prototyping tool must utilize a flexible modeling platform and a cost effective target for implementation.

1.5 Work scope and thesis outline

The objective of this thesis is to develop a rapid prototyping scheme for power electronic circuits that can be used for research as well as education. An object oriented open source modeling language Modelica has been used to describe models for DSP peripherals that enable the simulation of digitally controlled power electronic converters. To implement the converter on a cost-effective platform it is necessary for the parameters of these models to be converted into fixed-point DSP source code. This has been achieved by developing a code generator FixedCodeGen that produces code for the fixed-point TMS320F243 DSP from Texas Instruments. A real-time link is included in the generated fixed-point code that allows the user to vary parameters on the fly and fine-tune the trigger signal generator. The scheme has been utilized to configure classical power electronic converters such as DC-DC converters, DC-AC inverters and AC-DC converters. A framework for extension of the scheme for closed loop control has also been proposed and preliminary work done in this direction is also presented.
The nature of power electronic systems and currently available rapid prototyping schemes are discussed in chapter 2. The motivation behind the current work is also explained in detail. Chapter 3 gives a brief introduction into the Modelica modeling language and then covers in detail the components of the DSP library that have been developed. Chapter 4 describes the hardware and software tools used for the prototyping setup. It also discusses the real-time link and the interface circuit used in the rapid prototyping setup. Chapter 5 describes the fixed-point code generator FixedCodeGen that converts the Modelica code into DSP source code. Chapter 6 describes the rapid prototyping of a classical DC-DC and chapters 7 and 8 deal with the rapid prototyping of DC-AC and AC-DC converters respectively. Chapter 9 discusses the modifications needed to extend the scheme to closed loop controlled systems. The thesis conclusions are given in chapter 10 along with the suggestions/thoughts for future work.
CHAPTER 2: RAPID PROTOTYPING OF POWER ELECTRONIC SYSTEMS

2.1 Introduction

Power Electronics has been defined [5] as a field that involves the study of electronic circuits intended to control the flow of electrical energy. Power electronic circuits may handle power flow at levels much higher than individual device ratings. A basic power electronic system is shown in Fig. 2.1. The development of the thyristor in 1958 is widely believed to have heralded the power electronics revolution. Since then many devices and converter schemes have been designed for efficient control of energy flow.

![Diagram of a basic power electronic system](image)

Figure 2.1 A basic power electronic system
A power electronic system essentially consists of two parts:

1. A power processing unit
2. A triggering and control unit

The power processing units consists of devices and passive components that process the flow of electrical energy. The triggering and control unit switches the devices based on the logic of the control scheme that is been implemented to provide the user with the desired output. Power electronic systems are hybrid systems containing both analog and digital components. With every switching action the topology of the circuit changes and hence it is an event driven system. The special nature of power electronic systems calls for stringent requirements on the modeling and simulation platform needed, furthermore the emphasis on keeping costs low has prompted the widespread use of micro-controllers or fixed-point DSP’s as controllers for such circuits.

2.2 Modeling and simulation of power electronic circuits

Computer simulation of power electronic circuits enables the user to analyze the behavior of complex systems without having to build them. In addition they also enable him to safely study faulty or abnormal conditions without having to actually create those conditions physically in real equipment. However the accuracy of the simulation depends on the accuracy with which the components have been modeled and the identification of parasitic elements. As all influences and effects ideally are not known the simulation would essentially provide the user with a result that depends on his knowledge of the system.

Simulation has been proven to be an effective medium for power electronic education and research [6,7,8]. The presence of digital and analog components in such circuits with
high frequencies of switching and varying time constants calls for a versatile and powerful simulation platform. The simulation needs to be run for a long enough time to observe the effect of large time constants and also with a sufficiently small time step to accurately represent the effect of small time constants. Initial conditions play an important role in power electronic circuits and the simulation time depends on the accuracy of the initial estimates provided by the user. Several simulators such as Saber [9], Pspice [10], CASPOC [11], SIMPLORER [12], Dymola [13,14] etc. are been currently used for studying power electronic circuits.

2.3 Controllers for power electronic systems

The emergence of powerful and cheap digital signal processors (DSP) that have combined the peripherals of a micro-controller and the capability of fast complex signal processing has led to a revolution in the field of real-time controllers. Today’s DSP’s are capable of speeds in excess of 10 MHz. Their special architecture and high performance make it possible to implement a wide variety of complex control and measurement algorithms at high sampling rate for a reasonable cost. Power electronics systems are typically a complex combination of linear, nonlinear, and switching elements. High-frequency converters add another dimension of complexity because of their fast dynamics. Real-time power electronics systems, therefore, demand the use of high-speed data-acquisition and control. DSPs meet the processing requirements imposed by such systems. Furthermore the introduction of software tools has led to the establishment of real-time technologies (like hardware-in-the-loop simulation and rapid prototyping) as a standard for industrial research and development.
Digital Signal Processors can be broadly classified into Fixed-Point and Floating Point categories. Fixed-point devices are used in applications that have stringent cost or energy requirements. These devices are popular as they are cheaper than floating-point devices and most power electronic applications can be implemented using them. Low cost fixed-point devices are generally 20% of the lowest cost floating point devices [15].

2.4 Rapid prototyping schemes for power electronic systems and motivation behind current work

Today most of the DSP software algorithms used in consumer applications of PES are written in assembly language to optimize performance and decrease code footprint. The building and testing of complex software controllers is a difficult task in which the designer spends a major part of the time hand coding and debugging the controller algorithm in the language of the target processor. The increasing complexity and size of DSP applications and the need to decrease product development time has encouraged the use of high-level languages (HLL) such as C for programming DSP’s. The architecture of fixed-point DSP’s and their limited instruction set complicates the use of languages such as C for programming devices. This problem is aggravated by limited precision, associated scaling issues and constrained memory requirements as compared to floating-point processors. However most fixed-point DSP’s have been optimized for control applications and integrate sufficient memory and peripheral support on a single chip. Fixed-point devices are also far cheaper than floating-point devices and hence are preferred for power electronic applications. They provide the user with an affordable alternative platform to micro-controllers for realizing
control algorithms. Recent attempts at using high-level languages for fixed-point DSP’s have focused on the following techniques:

1. Use of C language compilers – New DSP’s such as the C2000 series from Texas Instruments are compiler friendly making the use of HLL more practical for them.

2. Graphically driven code generators - In this approach graphical blocks are used to realize the desired application. Once this has been constructed a code generator is run that generates source code in C or assembly. Examples of such platforms are Hyperception's RIDE [16] and Visual Solutions’ Vissim [17]

To utilize these in a Rapid Prototyping setup it is necessary to integrate these techniques with the modeling and simulation platforms for a smooth and quick transition from design to prototype implementation. A typical rapid prototyping scheme consisting of a simulation environment, code generation tools and hardware is shown in Fig. 2.2.

The user would utilize the simulation platform to build a model of the power electronic system. Using the simulation results he would fine tune the controller and get the output he desires. Then utilizing a code generator he would convert the model information into code that could be implemented on hardware. The controller can be realized using a DSP as shown in the figure or by a micro-controller. The controller would have a link to a computer that allows the user to vary the control parameters in real-time. Doing so allows for a further fine-tuning of parameters. The physical circuit for the power electronic system could be built using power electronic building blocks that allow the user to test various configurations using the same hardware. A real-time link between the PC and the controller is essential for changing the control parameters without having to stop the controller. This enables the user to quickly validate the design and refine it.
Figure 2.2 Typical rapid prototyping scheme

In order to support the development of an electronic system, a rapid prototyping system has to fulfill the following requirements:

- integration of different modeling techniques like state/event or control system design
- independence of modeling tools
- separation of modeling and implementation
- support of overall simulation and emulation

Hardware in the loop (HIL) simulation is also an essential part of any good rapid prototyping tool. In this methodology, a physical system to be tested is connected to a simulation through a real-time handshake. Hence the hardware is integrated with the simulation in a closed loop as a real component. With the presence of a real-time handshake between the hardware and the simulation, many test scenarios can be evaluated and controller designs can be debugged and validated. Hence the tool consists of a PC-based integrated developmental environment, real-time communications interface, and a target processor.

A number of rapid prototyping schemes for power electronics are available today. Most of these schemes are based on Matlab/Simulink’s Real Time Workshop [18,19,20,21,22]. Real Time Workshop allows the user to generate code for a given processor and maintain a real-time link. It uses the Simulink platform for building the model of the physical system. A popular commercially available tool using this platform is provided by dSPACE [23] and it utilizes a floating-point power pc processor as the target processor. Most power electronic systems however use fixed-point DSP’s or micro-controllers as they are relatively much cheaper and have enough functionality to satisfy the demands of the application. A generic real-time simulation approach using dual DSPs for power electronic systems has also been attempted [24]. However, this approach suffers from the drawback that multiple DSPs were used, thereby increasing the cost and complexity of the system. Furthermore due to a dual- DSP structure the data transfer hierarchy is limited. Vissim’s Rapid Prototyper [17] software is a commercial tool that uses Texas Instruments’ (TI) fixed-
point DSP as its target. Models for DSP peripherals are provided that can be coupled to other models that describe physical systems. The number of models describing power electronic systems in this setup however is small and the user has to create most of the models that he/she needs to use for simulating the system. The use of floating point blocks in this system to increase functionality causes the generated code to be large, effectively slowing down the processor. To effectively utilize a fixed-point processor its models need to be integrated with a modeling platform that has models from various physical systems including power electronic systems. In Vissim as well as Simulink the user has to model the system using an equation based approach. For a large system the user would hence have to spend a major effort to generate the equations needed to simulate the system. Furthermore in the Simulink and Vissim environments the blocks essentially have causal connections, which could cause a problem when it is not known apriori what the inputs and outputs of the system are. Causality creates complications in the effective definition of a component. In a complex system it is not easy to estimate which variables are inputs to the system and which are outputs.

For example lets take the case of a resistor (Fig 2.3), which is a very basic element present in electric circuits. The current I that flows through the resistor obeys Ohms Law and as shown in the figure depends on voltages V1, V2 and resistance R. In a complex system any one of the three variables could be the output and the other two the input to the element and would depend on the instantaneous state of the system. Causal representation of the resistor would create difficulties as it assumes that certain variables are inputs and the others output at all times. Hence there is a need for a modeling language that gives the user the flexibility to define components in a non-causal manner and let the causality be defined by
the interconnections. This would enable the user to concentrate on modeling and leave the
compiler to generate the required set of equations for solving the system. The modeling
platform must also encourage the reuse of previous work and help manage complexity as the
systems become larger.

Figure 2.3  Effect of causality on modeling of variables in a component

(Adapted from presentation by Hans Vangheluwe [25])

Object-oriented languages promote reuse, hierarchical modeling and data hiding
during modeling. Object-oriented languages like C++, UML and Java are being used widely
for different applications in various fields. Such languages however have not been designed
specifically for modeling physical systems. The Modelica object-oriented language has been
specifically designed to enable the modeling of physical systems. It has been designed as part of an international effort to provide a unified platform for the effective modeling of physical systems. To enable implementation on a fixed-point processor either a compiler or a graphical code generator should be used to convert the system model into executable code.

2.5 Proposed rapid prototyping scheme

To enable the user to simulate DSP based power electronic systems (PES) the simulation platform should have models that describe its peripherals or algorithms. These models must be built in a simulation platform that has extensive models of other domains and has the flexibility for adding user defined models. The Modelica language has been shown to be an ideal language for modeling physical systems. It is an object-oriented open source language that supports acausal modeling of components. Presently the standard Modelica library consists of models from the mechanical, electrical, thermodynamic and hydraulic domain. To utilize this environment for simulating DSP based systems a library known as DSPLib has been developed as part of the thesis work. The models in this library have been subsequently utilized by a code generator to generate appropriate fixed-point code keeping into consideration resource constraints, numerical overflow, interrupt handling, real-time links and fixed-point scaling. A real-time link in the generated code enables the user to fine tune the trigger signal generation scheme or controller parameters of the prototype using actual hardware in real-time. These ideas are realized in this contribution by the development of a fixed code generator FixedCodeGen that generated fixed-point DSP C code (for TI C2000 series DSP's) for these peripherals from the Modelica system description. The use of DSPLib and FixedCodeGen allows the user to implement his/her power electronic circuit
from its simulation in a single step. Furthermore as the target controller is a fixed-point processor it presents the user with a cost effective target for implementation.

The rapid prototyping scheme (Fig 2.4) has been used to realize classical power electronic converters such as DC-DC choppers, DC-AC square wave inverters and line synchronized AC-DC converters. These converters were implemented using power electronic building blocks provided by LabVolt and a fixed point DSP (TMS320F243) donated by Texas Instruments. The code for the DSP was automatically generated by FixedCodeGen.
from a Dymola based simulation of the converter built using Modelica based blocks including DSP components from DSPLib.

2.6 Conclusions

The chapter discusses the nature of power electronic systems and the motivation behind the rapid prototyping scheme developed. Various simulation tools and controllers currently being used for power electronic systems are also discussed. Currently existing rapid prototyping schemes are examined and the basis for the chosen rapid prototyping scheme as part of the thesis is explained.

In the next chapter, Modelica language centered modeling of physical systems- open source platform, is discussed. In this chapter, details of the library of DSP components-DSPLib that has been built as part of this thesis work are given.
CHAPTER 3. DSPLIB – MODELICA LIBRARY FOR DSP PERIPHERALS

3.1 Introduction

Modelica is an object-oriented modeling language for modeling and visualizing complex physical systems developed by the Modelica Association [26,27,28]. It supports acausal modeling of components, reuse of models and inheritance of properties. Systems can be expressed in DAE’s, ODE’s, bond graphs, finite state automata or as state charts. This wide functionality provides the user with a very powerful tool to model complex physical systems. This multi-domain capability gives the user the possibility of combining mechanical, thermodynamic, electrical, hydraulic components within the same system. Modelica models are open source and hence users have the flexibility in modifying and improving existing models or create their own models as per their specific needs. The language has been designed to allow tools to generate efficient simulation code and facilitate exchange of models and libraries.

3.2 Dymola based simulation of modelica based physical systems

A number of models built based on the Modelica language are part of the Modelica Standard Library and are available freely. This standard library contains components from the electrical, mechanical, thermal and hydraulic domains. Additional models describing power system components, petri-nets and fuzzy logic are also available. An extensive library for power electronic components, converters and measurement templates has been developed
in Modelica at Iowa State University [29,30]. This library is currently being used in an undergraduate senior level course on power electronics and drives offered by the Electrical and Computer Engineering Department [31].

The user can use these models to describe the system they wish to simulate. As the entire library is open source the user can check whether the models are suitable for their specific needs. In the event that the user does not find a specific model, he/she could create their own using standard interfaces and templates. The object-oriented nature of the language enables the user to extend the model definition to the complexity level that is desired.

Using the models the user creates the system that they would like to simulate and stores it in a Modelica *.mo file. This is the Modelica source code that can be used by any simulation engine. The Modelica format is currently recognized by ACSL [32], Dymola [33], Simulink and MathModelica [34] and can be simulated in any of these environments. At Iowa State University Dymola from Dynasim [35] has been used for simulating Modelica based systems.

The process of simulating a Modelica language centered system using Dymola is shown in Fig. 3.1. First the Modelica source code is parsed and converted into an internal representation, usually an abstract syntax tree. This representation is analyzed, type checking is done, classes are inherited and expanded, modifications and instantiations are performed, connect statements are converted to equations, etc. The result of this analysis and translation process is a flat set of equations, constants, variables and function definitions. After flattening, all of the equations are topologically sorted according to the data flow dependences between the equations. In the case of general differential algebraic equations (DAEs), this is not just sorting, but also manipulation of the equations to convert the
coefficient matrix into block lower triangular form, a so-called BLT transformation. Then an optimizer module containing algebraic simplification algorithms, symbolic index reduction methods, etc., eliminates most equations, only keeping a minimal set that eventually will be solved numerically. Then independent equations in explicit form are converted to assignment statements. Finally, C code (dsmodel.c) is generated, and linked with a numeric equation solver that solves the remaining equation system. The approximations to initial values are interactively specified by the user.

Figure 3.1 Simulation of modelica model using dymola
3.3 DSP library (DSPLib) in modelica

The DSP library ‘DSPLib’ has been developed using the Modelica package concept. It contains models that describe standard DSP peripherals like DSP timers, capture blocks and output ports. These basic peripherals are used extensively in power electronic circuits for external synchronization, output feedback and trigger signal generation. The models in DSPLib cover the basic functionality of these peripherals and help the user incorporate them into their simulation. The components of DSPLib also play an important role in the generation of fixed-point code that is compatible with TI’s C2000 family of processors. The fixed-point code generator ‘FixedCodeGen’ generates DSP code based on the presence of these models in the simulation diagram. The models developed have been subdivided into three sub-libraries (Fig. 3.2):

1. DSP Unit:
2. DSP Interfaces:
3. DSP Elec:

![DSP library graphical user interface](image-url)
The DSPUnit library contains basic peripherals like timers, triggered timers, capture blocks and a level detector. The DSPInterfaces library contains models for ring counters and triggered ring counters. These components are extensively used for generating trigger signals for power electronic converters. The DSPelec contains a specialized AC2DC block that can be used for configuring a line synchronized rectifier converter.

3.4 DSP unit sub library

3.4.1 DSP timer

![DSPTimer graphical symbol](image)

Figure 3.3 DSPTimer graphical symbol

The DSPTimer (Fig. 3.3) block is essentially a square wave generator and is modeled as a 16-bit up counting asymmetric DSP timer. The user specifies the frequency and duty ratio of the square wave that he/she desires, which is used to generate the output. Using these values the time period and on time period (width) is calculated. Internally the timer model normalizes the current time into multiples of the desired period \( t \) and a remainder \( sqtime \). The remainder \( sqtime \) is used to decide whether the switch is on or off. If the remainder is greater than the on time the output is low implying that the desired on-time has been
exceeded and the device needs to be turned off. This process is described in Algorithm 3.1 and a sample variation of period, width (ontime), sqtime, t and v (timer output) with time are shown in Figures 3.4 and 3.5. The DSPtimer block has been used for generating trigger signals for the DC-DC converter and is illustrated in Chapter 6.

Step 1: Using input values of Frequency and Duty ratio calculate the period and the on-time for the timer.

\[
\text{period} = \frac{1}{\text{frequency}} \\
\text{on time} = \left(\frac{\text{duty}}{100}\right) \times \text{period}
\]

Step 2: Convert simulation time into multiples of period (t) and a remainder sqtime.

\[
\text{t} = \text{div}(\text{time}, \text{period}) \\
\text{sqtime} = (\text{time} - \text{t} \times \text{period})
\]

Step 3: Compare remainder with on time. If greater then Timer output (v) is low else high.

\[
\text{v} = \text{if sqtime} \geq \text{on time} \text{ then low else high}
\]

**Algorithm 3.1: Timer operation**

Figure 3.4 Variation of DSPtimer variables 'sqtime', 'period' and 'width' with time
The timer model currently does not take into account rise and fall time of signals. These can be incorporated if desired by extending the model definition. DSP timers are used extensively for trigger signal generation in power electronic circuits.

3.4.2 Capture block

The DSPcapture block senses the rising or falling edge of a given signal and generates positive pulses at each of these events and is used for synchronizing signals with external clocks. The block (Fig. 3.6) has two outputs that represent the rising and falling
edges respectively and is modeled around the capture pin present on the DSP’s in the C2000 family. Using flags the user can select either edge or both for detection. The current value of the input signal is compared with its value 0.1msec before to ascertain whether there has been a change. If the input signal changes at a rate faster than this the model would produce erroneous results. This can be corrected by appropriately reducing the value of the delay. This delayed has been modeled around the clock frequency of the target DSP and can be changed appropriately. However setting this value exactly with the clock causes the simulation to run very slowly and hence an intermediate value needs to be chosen. The capture block equation flow is shown in Algorithm 3.2. The variation of these variables (v1, v2 and v3) with time is shown in figure 3.7. The DSP capture block has been utilized for line synchronization in the AC-DC converters illustrated in chapter 8.

Step 1: Store current value of Capture block input signal as valk. Store the input value 0.1msec before current time as valkminus1.

valk = v1 (input signal)
valkminus1 = delay(v1, 0.0001)

Step 2: If flags for up trigger and down trigger are set check if appropriate transition has occurred using valk and valkminus1.

v2 = if valk > valkminus1 then high else low (UP Trigger)
v3 = if valkminus1 > valk then high else low (DN Trigger)

<table>
<thead>
<tr>
<th>Step 1: Store current value of Capture block input signal as valk. Store the input value 0.1msec before current time as valkminus1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>valk = v1 (input signal)</td>
</tr>
<tr>
<td>valkminus1 = delay(v1, 0.0001)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2: If flags for up trigger and down trigger are set check if appropriate transition has occurred using valk and valkminus1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2 = if valk &gt; valkminus1 then high else low (UP Trigger)</td>
</tr>
<tr>
<td>v3 = if valkminus1 &gt; valk then high else low (DN Trigger)</td>
</tr>
</tbody>
</table>

Algorithm 3.2: Capture block operation
3.4.3 Triggered timer

The triggered timer (Fig. 3.8) is essentially the same as the normal timer model. The main difference is that it can be started using an external trigger pulse. The timer gets reset with the arrival of every trigger pulse and hence can be used for synchronized trigger signal generation. The operation of the triggered timer is illustrated in Algorithm 3.3. The timer also
generates positive pulses at every period (IntP) and (IntC) as shown in Fig. 3.9. The timer gets reset (P1) with the arrival of every trigger pulse (T) as shown in Fig 3.10.

Algorithm 3.3: Triggered timer operation

Step 1: Calculate period and on time from input values

\[
\text{period} = \frac{1}{\text{frequency}}
\]
\[
\text{on time} = ((\text{duty/100}) \times \text{period})
\]

Step 2: When trigger input is high reset timer count.

\[
\text{when} \ (T \geq 1) \\
\text{trigtime} = \text{time} \\
\text{continue} = 1
\]

Step 3: Normalize current time into a multiple (t) of timer period and a remainder (suptime)

\[
\text{timertime} = (\text{acttime} - \text{trigtime}); \\
\text{t} = \text{div}(\text{timertime}, \text{period}); \\
\text{suptime} = (\text{timertime} - \text{t} \times \text{period});
\]

Step 4: If remainder (suptime) is greater than on-time then Timer output is low else high

\[
\text{P1} = \text{if} \ (\text{suptime} \geq \text{width}) \text{ then low else high;}
\]

Step 5: If suptime is equal to period or on time set appropriate flag

\[
\text{IntP} = \text{if} \ \text{suptime} \geq \text{period} \text{ then high else low} \\
\text{IntC} = \text{if} \ \text{suptime} = \text{on time} \text{ then high else low;}
\]

Figure 3.9. Trigger pulses generated at compare and period match
3.4.4 Level detector

The level detector block behaves like a zero crossing detector and is used in conjunction with the triggered multi function timer block to model trigger signal schemes for AC to DC converters. The block has two parameters 'Vup' and 'Vdn', which are the output values when the signal is greater than zero and when it is less than zero respectively. The
operation of this block is illustrated in Algorithm 3.4. The variation of the output voltage ‘Vout’ for a given sinusoidal input ‘SigIn’ is shown in Fig. 3.12. The level detect block is used in the AC2DC bloc

Step 1: Store the input signal is variable SigIn.
\\[ \text{SigIn} = \text{voltage of input signal} \]

Step 2: Compare the input signal with Vup and Vdn and appropriately assign output value.
\\[ \text{Vout} = \text{if } \text{SigIn} > 0 \text{ then Vup else Vdn} \]

Algorithm 3.4: Level detect block operation

Figure 3.12. Variation of leveldetect block variables ‘Vout’ and SigIn’ with time
3.5 DSP interfaces sub library

3.5.1 Multi-Function timer

The multi-function timer (Fig. 3.13) behaves in a manner similar to a ring counter. The multi-function timer targets output port C of the DSP and is 8 bits wide. The various parameters that the model requires are:

1. Number of bits on (Port width that the user desires)
2. Frequency
3. Shift
4. On For (which decides the number of units of time for which a bit will be consecutively on.)

The operation of this model is best illustrated in Algorithm 3.5 and further explained with an example. Assume that bits IOPC0 to IOPC5 are set high, frequency is set at 60Hz, shift set to 1 and On For to 3. The model calculates the port width that the user desires. In this case the number of bits the user has enabled is 6. The model then scales the internal frequency of the timer by multiplying this value with the user-defined frequency. Hence for the values specified above the internal timer frequency is set at 360 Hz. The timer then
calculates the ‘period’ (1 time unit) based on this value. The ‘shift’ value specifies the sequence in which bits are turned on as each unit of time elapses. Hence in our case after IOPC0 is high for the first unit, bit IOPC1 would be turned on for the next unit of time. The ‘On For’ parameter decides the number of units of time for which a bit will be consecutively on. In our case bit IOPC0 will be on for three units and therefore would be on when bits IOPC1 and IOPC2 are high. The values that are set in the example generate the necessary trigger signals for a 180° square wave inverter. Changing the ‘On For’ parameter to 2 would change the trigger signal generation into one needed for a 120° square wave inverter.

| Step 1: Calculate period and number of pins of port that are needed |
| period = 1/frequency |
| portson = From user input, calculate port width been used |

| Step 2: Calculate time for which each pin is on and time to shift from one pin to another |
| sector = width/portson |
| timeshift = sector*shift |
| ontime = sector*onfor |

| Step 3: Normalize current time into a multiple of period (t) and remainder (sqtime). |
| t = div(time, period) |
| sqtime = (time - t*period) |

| Step 4: Check remainder to decide which pins of port need to be on. |
| v0 = if sqtime is within pin on time then high else low |

**Algorithm 3.5: Multi-function timer operation**

Internally the model converts the frequency into time period (period) and normalizes the current time into multiples of the desired period (t) and a remainder (sqtime). It calculates the start time and the stop time for each individual pin and using these values and the dynamic value of ‘sqtime’ it decides whether the pin should be high or low. The algorithm that achieves this for pin IOPC0 of the port is described above (Algorithm 3.5). The
waveforms for the various variables ‘sqtime’, ‘period’ and ‘ontime’ are shown in Fig. 3.14. The value of ‘sqtime’ varies from 0 to period. Each pin is on for the ‘ontime’ specified.

Figure 3.14 Variation of multi-function timer block variables with time

3.5.2 Triggered multi-function timer

This model is very similar to the Multi-Function Timer model. The only difference is that an external trigger (Trig.v) is required to start its operation. Internally the model converts
the frequency into time period (period) and normalizes the current time into multiples of the desired period and a remainder (sgtime). It then calculates the start time and the stop time for each individual pin. Using these values and the dynamic value of ‘sgtime’ it decides whether the pin (CO.v) should be high or low (Fig 3.16). The timer gets reset with the arrival of every trigger pulse (Fig 3.17). The operation of the triggered multi function timer is illustrated in Algorithm 3.6. With the arrival of each trigger pulse (Trig) the value of ‘sgtime’ is reset to zero and the generation of output pulses is reset simultaneously.

---

Step 1: Calculate period and number of pins of port that are needed

\[
\text{period} = \frac{1}{\text{frequency}} \\
\text{portson} = \text{From user input calculate port width been used}
\]

Step 2: When trigger input is high reset timer count.

\[
\text{when (Trig.v >= 1) }
\]

\[
\text{trigtime} = \text{time}
\]

Step 3: Calculate time for which each pin is on and time to shift from one pin to another

\[
\text{sector} = \text{width/portson} \\
\text{timeshift} = \text{sector*shift} \\
\text{ontime} = \text{sector*onfor}
\]

Step 4: Normalize current time into a multiple of period (t) and remainder (sgtime).

\[
\text{timertime} = (\text{acttime - trigtime}); \\
\text{t} = \text{div(timertime, period)}; \\
\text{sqtime} = (\text{timertime} - \text{t*period});
\]

Step 5: Check remainder to decide which pins of port need to be on.

\[
\text{C0 = if sqtime is within pin on time then high else low}
\]

Algorithm 3.6: Triggered multi-function timer operation
Figure 3.16 Variation of triggered multi-function timer variables 'sqtime' and 'period' with time

Figure 3.17 Variation of triggered multi-function timer block variables 'Trig' and 'C0' with time
3.6 DSP elec sub library

![AC2DC](image)

**Figure 3.18 Graphical symbol for AC2DC block**

The DSPElec sub-library currently consists of one block, which is the AC2DC block to be used for generation of trigger signals for single-phase and three-phase AC to DC converters. Internally the AC2DC block consists of three individual blocks: capture block, triggered timer block and the triggered multi-function timer block. The blocks were combined to form a single block to ease the generation of fixed-point code.

The various user-defined parameters for this block are:

1. Frequency
2. Firing Angle ‘Alpha’ in percentage
3. Pulse Width
4. Port Width

Using the above values the block internally calculates the firing angle in terms of time with respect to the period of operation. The capture block is used to synchronize the block with the external signal. On the rising edge of the input signal the triggered timer starts counting towards the period of operation. The timer stops counting when it reaches the value corresponding to the firing angle and generates a compare match interrupt. The triggered
timer uses this command to start generating output signals at the various pins at the requisite intervals. The internal connections of the various blocks are shown in Fig 3.19. The algorithm outlining the operation of the block is shown in Algorithm 3.7.

```
Step 1: Calculate period and number of pins of port that are needed
    period = 1/frequency
    firing angle = alpha*period/100

Step 2: When capture input is high generate synchronize signal.
    when (Sigk > Sigkminus1)
        Uptrigger = High and start triggered timer
    Else Uptrigger = low

Step 3: When uptigger is high start triggered timer. Compare value with firing angle for stop
    Current time is counted from Uptrigger and stored in uptime
        Firing angle match = if (uptime = firing angle) then 5 else 0

Step 4: Normalize current time into a multiple of period (t) and remainder (sqtime).
    timertime = (acttime - trigtime);  
    t = div(timertime, period);  
    sqtime = (timertime - t*period);

Step 5: Check remainder to decide which pins of port need to be on.
    CO = if sqtime is within pin on time then high else low
```

Algorithm 3.7: AC2DC block operation
3.7 Conclusions

In this chapter the components of the library of DSP peripherals DSPlib written in the Modelica language are described. The use of Modelica enables the easy reuse of these components and their extension. The models built allow the user to simulate various classical power electronic circuits. Simulation of these models has been done using Dymola, and the results are reported.

In the next chapter the details of hardware and software tools for rapid prototyping of power electronic circuits is described. The real-time link used between the controller and the DSP is discussed and the interface circuit built for protection and isolation of signal level electronics from the power-switching module is explained.
CHAPTER 4. HARDWARE SOFTWARE TOOLS FOR IMPLEMENTING POWER ELECTRONIC CONTROLLERS

4.1 Introduction

Micro-controllers have been traditionally used to provide solutions for digital control applications. However they lack both the performance and the architecture to perform real-time math intensive advanced control algorithms at a desired bandwidth. The arrival of powerful and cheap digital signal processors (DSP) that have combined the peripherals of a micro-controller and the capability of fast complex signal processing has led to a revolution in the field of real-time controllers. Their special architecture and high performance make it possible to implement a wide variety of nontrivial control and measurement algorithms at high sampling rate and reasonable cost.

A typical developmental platform for designing applications would consist of a DSP board with sufficient memory (for the intended application), IO support and a communication link to a PC based environment for designing software code. The work for this thesis has been done on TMS320F243 DSP based developmental platform donated generously by Texas Instruments. The platform consists of an EVM board with a JTAG link to a host PC that has the Code Composer Studio integrated development environment (IDE) for writing application code. A real-time link between the PC based Code Composer software and the DSP has been established for variation of control parameters on the fly.
4.2  TMS 320 overview

The TMS320 family from Texas Instruments consists of fixed-point, floating-point and multiprocessor digital signal processors (DSP). The following characteristics make the TMS320 family the right choice for a wide range of processing applications [36]:

- Very flexible instruction set
- Inherent operational flexibility
- High-speed performance
- Innovative parallel architecture
- Cost effectiveness

Texas Instruments introduced the TMS32010 in 1982 and it was the first fixed-point DSP in the TMS320 family. Today, the TMS320 family [36](Fig. 4.1) consists of the 'C1x, 'C2x, 'C24x, 'C5x, 'C54x, and 'C6x fixed-point DSPs; 'C3x and 'C4x floating-point DSPs; and 'C8x multiprocessor generations of DSPs.

The TMS320F243 is part of the C24x family of fixed-point DSPs, and is a member of the C2000 platform. The C24x series of DSP controllers combines real-time processing capability with controller peripherals to create an ideal solution for control system applications. Processors within a particular generation of the TMS320 family have the same CPU structure but different on-chip memory and peripheral configurations. TMS320 processors integrate memory and peripherals thus reducing system costs and saving circuit board space.
4.3 C24x overview

Control application designers have recognized the opportunity to redesign existing digital motion control systems to use advanced algorithms that yield better performance and reduce system component count. The use of DSPs enable the [36]:

- Design of robust controllers for a new generation of inexpensive motors, such as AC induction, DC permanent magnet, and switched-reluctance motors
- Full variable-speed control of brushless motor types that have lower manufacturing cost and higher reliability
- Energy savings through variable-speed control, saving up to 25% of the energy used by fixed-speed controllers
- Increased fuel economy, improved performance, and elimination of hydraulic fluid in automotive electronic power steering (EPS) systems
- More efficient and quieter operation due to less generation of torque ripple, resulting in less loss of power, lower vibration, and longer life
- Elimination or reduction of memory lookup tables through real-time polynomial calculation, thereby reducing system cost
- Use of advanced algorithms that can reduce the number of sensors required in a system
- Control of power switching inverters, along with control algorithm processing
- Single-processor control of multi-motor systems

The TMS320C24x DSP controllers have been designed to meet the needs of control-based applications. A high performance DSP core (Fig. 4.2 has been integrated with the on-chip peripherals of a micro-controller into a single-chip solution to yields a device that is an affordable alternative to traditional micro-controller units (MCUs) and expensive multi-chip designs. At 20 million instructions per second (MIPS), the 'C24x DSP controllers offer significant performance over traditional 16-bit micro-controllers and microprocessors.

The presence of a powerful control optimized core enables the user to utilize advanced control algorithms in their applications. Math intensive techniques such as adaptive control, Kalman filtering, and state control can be utilized. The high-speed central processing unit (CPU) allows the digital designer to process algorithms in real time rather than approximate results with look-up tables. It is source- and object-code compatible with the other members of the 'C24x generation, source code compatible with the 'C2x generation, and upwardly source code compatible with the 'C5x generation of DSPs from
Texas Instruments. The functional block diagram [38] of the C243 processor is shown in figure 4.3:

![Diagram of the C243 processor](Figure 4.3 TMS320F243 DSP core [37])

This various peripherals present on the DSP include:

- Timers
- Serial communications ports (SCI, SPI)
- Analog-to-digital converters (ADC)
- Event manager
- System protection, such as watchdog timers
- CAN bus
- Capture Pins for external triggering
- JTAG bus for communication
Figure 4.3  Functional block diagram of C243
4.4 Code composer studio overview

Code Composer Studio (CCS) [38,39] is a fully integrated development environment that supports Texas Instruments TMS320C6000, TMS320C5000 and TMS320C2000 DSP platforms. Its main features are [39]:

- Integrated development environment with editor, debugger, project manager and code profiler.
- C/C++ Compiler, Assembler, and Linker (Code Generation Tools)
- Instruction set simulator
- Real-time debugging
- Real-time analysis and data visualization tools

Code Composer Studio (CCS) integrates all host and target tools in a unified environment to simplify DSP system configuration and application design. This easy to use development environment allows DSP designers full access to all phases of the code development process. CCS has an open architecture that allows TI and third parties to extend the IDE's functionality by seamlessly plugging-in additional specialized tools. CCS addresses each phase of the code development cycle including design, code and build, debug, analyze and optimize.
4.5 Code development using code composer

The design of any software module using code composer studio (CCS) can be divided into four distinct phases:

1. Setup Target
2. Code and Build
3. Debug software code
4. Profile software code

4.5.1 Target setup

Code Composer Studio's setup program provides a graphical interface for configuring multiple boards and multiple CPUs on the development system. Its features are [39]:

- Wizard for setup of multiple processors
- On-line help displays proper setup information
- Run-time object library includes I/O modules, pre-emptive scheduler and APIs
- Real-time application analysis and debug capability
- Configuration tool with graphical visualization capabilities

4.5.2 Code and build

CCS provides the user with a fully integrated development environment that provides access to all its features and tool sets from a single easy-to-use interface. The source code editor is tuned for writing C and DSP assembly code offers developers a number of ways to be productive. Color syntax highlighting for C, C++, assembly, and scripts allows users to
easily spot errors in keywords or missing comment delimiters. Floating toolbars support
advanced operations, such as finding the next matching brace and indenting text. The Editor
is fully integrated with other facilities in Code Composer Studio and allows developers to
easily edit code and see both source and disassembly at the same time.

4.5.3 Debugging software code

Code Composer Studio's integrated debugger has DSP-specific capabilities and
advanced breakpoints to simplify development. Conditional or hardware breakpoints are
based on full C-expressions, local variables or CPU register symbols. A General Extension
language (GEL) script file can also be automatically executed when a particular break point
is reached. Global breakpoints are also available for multiprocessor systems. Developers can
debug code quickly by selectively stepping into, over, or out of C-functions or assembly sub-
routines. The quick watch window facility lets developers see the value of a variable in a C
or assembly source window instantaneously.

4.5.4 Interactive profiling

Code Composer Studio's interactive profiler makes it easy to measure code performance
and ensure the efficient use of the DSP target's resources during a debugging and
development session. Profile points accumulate hits and collect statistics on the number of
instruction cycles executed or other events that have elapsed since the previous profile point
was hit. This allows high-usage areas of code to be targeted during optimization, helping
developers produce finely tuned code.
4.6 Real-Time link

A real time handshake between the PC and the DSP is an essential part of any rapid prototyping setup. A real-time link enables the user to change control parameters present in the application without having to stop the processor. This gives the user the flexibility to modify and refine the application and also test it under varying operating conditions.

Texas Instruments provides generic real-time monitor routines called RT_MON that can be used for maintaining a real-time link between its DSP's and the Code Composer development environment. RT_MON can be invoked in a user application by invoking the real-time mode while running the program. To incorporate the real-time mode a function call needs to be made in the user program to the monitor routine. Using the monitor routines enables a real-time refresh of all DSP memory registers in CodeComposer. We can select the real time refresh rate, which defines the time after which CodeComposer would refresh its contents with that of the DSP. The monitor routine utilizes the unused cycle of the processor to refresh the register variable values on the PC. Hence the monitor routine would not perform well if the utilization of the processor were high. To circumvent this problem Texas Instrument introduced a Rude Real-Time mode during which the monitor routine stops the execution of the program to refresh the register variable. The user hence has a tradeoff between his program performance versus his real-time needs. The monitor routine RT_MON also occupies memory space and requires that all interrupt service routines (ISR) to return to the main program for its proper execution. In power electronic systems that are interrupt driven it is possible that the program remains in ISR's throughout and hence the use of RT_MON would create problems.
As an alternative to RT_MON a real-time link between CodeComposer IDE and the DSP was developed that would allow the user to vary control parameters. Control parameters that the user needed to vary are specified in specific memory addresses in the data section of the DSP memory. During the program execution the memory space of the DSP is accessible from the CodeComposer IDE and changes can be made in any address. Hence the program is written so that it accesses the control parameter from the memory address at specific instances. If the user changes the value stored in the register it would reflect on the program performance after it accesses the new value. This stripped down communication allows for full control over the program performance without any problems associated with high-utilization of DSP or infinite ISR loops. This form of real-time control has been used for varying parameters like firing-angle for AC-DC converters, duty-ratio for DC-DC converters and frequency for DC-AC converters on the fly without any loss in performance. As a real-time monitor routine (RT_MON) is not used the resulting program size is correspondingly smaller.
4.7 DSP interface circuit

For the purpose of protection it is necessary to isolate the output signals of the DSP and the switching control unit of the power electronics module. This isolation prevents inadvertent damage of the DSP board during operation. The interface is realized with the help of an opto-isolator based circuit. To obtain the desired logic an inverter is placed after the opto-isolator. The circuit diagram for the DSP-Gate Drive interface circuit and the layout of the PCB for it is shown in figures 4.4 and 4.5 respectively.

Figure 4.4 DSP gate drive interface circuit schematic
4.7 Conclusions

In this chapter the TMS320F243 DSP and the code composer integrated development environment been used in the rapid prototyping scheme is described. The C243 DSP combines real-time processing capability with controller peripherals to create an ideal solution for control system applications. The real-time link between the DSP and the PC based code composer used in the rapid prototyping scheme and the interface circuit used for protection is also discussed.

In the next chapter the automatic fixed-point code generator FixedCodeGen is discussed.
CHAPTER 5. FIXEDCODEGEN- AUTOMATIC FIXED-POINT CODE GENERATOR

5.1 Introduction

The physical system built using Modelica blocks can be simulated using the Dymola simulation tool. As shown in chapter 3 the Dymola simulator converts the Modelica based system description into a floating-point C code called dsmodel.c. The floating point C code generated is according to the dsblock specification. However as this code is floating point in nature it cannot be used for implementation in a fixed point DSP. The code needs to be converted into a form recognizable by Texas Instruments compiler tools for generation of an executable for their fixed-point processor. To achieve this an application called FixedCodeGen has been developed. FixedCodeGen generates fixed-point DSP source code compatible with the TMS320F243 processor from a Modelica system containing specific blocks from the DSP library DSPLib. The DSP source code generated is upward compatible with the entire Texas Instruments’ C2000 family of processors.

5.2 FixedCodeGen overview

The FixedCodeGen application has been written in C using Microsoft Visual C++ developmental tool. The program reads the floating point C code description (dsmodel.c) in Dymola of the physical system generated during the translation process. Alternatively the Modelica file (*.mo) or the Modelica flat file (*.mof) could also have been used for the code
generation process. The dsmodel.c file name is constant irrespective of the name of the Modelica system and hence its use simplifies the code conversion. The program identifies specific blocks from the DSP library DSPLib that are present in the code and reads the parameters of those blocks. The data for each block is read using a separate function and hence the program can be easily extended for new blocks. The program currently generates DSP code for the DSP Timer block the Multi Function Timer block and the AC2DC block. These blocks can be configured for generating trigger signals for various power electronic converters, which is the focus of this thesis.

### 5.3 FixedCodeGen program flow

The main functions of the FixedCodeGen program are:

1. Read the input file dsmodel.c and identify the various DSP blocks present
2. Identifies any resource conflicts (in case of multiple use if a DSP peripheral at the same time) that are present. If any resource conflicts are present the application stops and an error message is generated.
3. In the absence of resource conflicts the applications reads the input parameter for the various DSP blocks.
4. Based on these input parameters, it generates appropriate values for DSP registers.
5. Initializes DSP status registers and other registers needed by the program whose values have been calculated.
6. Assigns register to a variable needed for Real-Time control. The variable is generally a control parameter such as firing angle, duty ratio or frequency. By assigning a fixed
address the user can vary its contents in real-time for varying the output of the power electronic converter.

7. Write an infinite while loop for continuous execution.
8. Writes Interrupt Service Routine if required and generates file rtvector.asm.
9. Generate a log file genlog.txt documenting the programs execution.

![Flowchart of FixedCodeGen program flow]

**Figure 5.1 FixedCodeGen program flow**

The FixedCodeGen program has been developed in a modular fashion so that it can be easily extended to incorporate new blocks and target different processors. Each block
from DSPLib has a separate set of functions that read values, generate appropriate values for registers and write DSP code for the DSP registers. Furthermore appropriate data scaling is also done to prevent numeric overflow of registers.

5.4 Files generated by fixedcodegen

The various files that the program generates are:

1. C243xyz.c
2. rtvector.asm
3. Genlog.txt

C243xyz.c is the fixed point DSP source code and can be used to generate a DSP executable code. The entire code generated is in the C language and hence can be easily read and understood. Furthermore even though the source code generated is for the TMS320F243 it is upward source compatible with the entire C2000 family of processors from Texas Instruments. Figure 5.2 contains the source code for a DC-DC converter that has been realized using the DSPtimer block. The code generated is essentially in four sections as shown below. In the example shown no DSP interrupts are used and hence there are no ISR's. ISR's if present are defined using standard 'C' function declarations.

1. Header section containing real-time variables (program specific), include files and initialization section to DSP registers, which are independent of the program itself. This set of instructions remains fixed for each program and essentially consists of code that clears registers such as those for interrupts, watchdog timers and timer counters.
Furthermore the FixedCodeGen utility also puts a date and file stamp that allows the user to keep track of the generated code. The pointer to the real-time variable varies depending on the DSP block chosen and is the only sub-section that varies from program to program. In this particular example the real-time variable chosen is for duty-ratio variation of the timer output and is assigned to memory address 0xA001.

2. Application specific register assignment section: This section contains initialization of DSP register values derived from the blocks that are present in the Modelica system. Individual functions generate scaled values for DSP registers that are assigned in this section of the output source code. Hence if multiple DSP blocks utilize the same register the specific function would specify the value to be addressed. If an impossible combination were needed then the program would generate an error.

3. Infinite while loop section: This section contains an infinite loop that enables the user to run the program continuously on the DSP. The program can be stopped using the ‘Stop DSP’ command, which has been built into CodeComposer. The stop command is activated using the reset interrupt that is mapped using the vectors file. The while loop section is also used for the automatic refresh of real-time variables.

4. Interrupt service routine (ISR) section: This section is generated depending on the blocks that are present in the Modelica system. A separate function for each ISR is generated and appropriate clearing of the interrupt service register is done at the head of each function. Currently timer period interrupts and capture pin interrupts are recognized and appropriate ISR’s are generated.
/* Section 1: Initialization Section*/
/* This file was created on: Wed Nov 27 18:45:20 2002 */
/* C243 code generated from modelica model DSPLib.Examples.DSPBuck.mo */
#include "F243_c.h"

unsigned int (*duty) (unsigned int *) 0xA001;

void main(void)
{
    SCSR = 0x0001;
    WDCR = 0x00E8;
    WSGR = 0x0040;
    EVIFRA = 0xFFFF;
    .

/* End */

/* Section 2: Application specific section */
OCRA = 0x3040;
T1PR = 20000;
*duty = 12000;
T1CMPR = *duty;
T1CON = 0x1042;
.
.
/* End */

/* Section 3: Infinite while loop section */

while(1){
    T1CMPR = *duty;
}
/* End */

}
The program can also be extended to generate code for fixed-point processors of other manufacturers like Motorola and Analog Devices. This can be done by changing the functions that generate DSP code based on the values generated from the DSP blocks. Hence the same simulation diagram can be used to generate fixed-point code for different processors without much significant modification. This inbuilt functionality helps the user to extend the rapid prototyping scheme easily.

Rtvector.asm is an assembly source file that maps the ISR function name to a specific memory address in the vector table of the DSP. Each interrupt has a specific priority and is assigned a specific memory address in the program space. In rtvector.asm each ISR that is present in the main source code is mapped to its priority level and memory address.

Genlog.txt is a log file that the application generates and contains detailed listing on the progress of code generation and values of input and output parameters.

5.5 Conclusions

In this chapter the fixedcodegen application that converts the Modelica system information into DSP source code compatible with the C2000 family of TI processors is discussed. The application ensures that there are no resource conflicts and that all variables are scaled properly. The application also generates a log file that enables the user to check the input and output parameters.

The next chapter deals with the rapid prototyping of a step-down DC-DC converter.
CHAPTER 6. RAPID PROTOTYPING OF DC-DC CONVERTER

6.1 Introduction

The DC-DC converter is a power electronic circuit used to transfer energy between two DC circuits operating at different voltage and current levels [40]. It can be considered to be the equivalent to an AC transformer with a continuously variable turns ratio. There are number of topologies for DC-DC converters such as:

1. Step-Down Converter (Buck Chopper)
2. Step-Up Converter (Boost Chopper)
3. Step-Up/Down Converter (Buck-Boost Chopper)
4. Cuk Converter
5. Bi-Directional and Multi-level converters

DC-DC converters are widely used in switched mode power supplies and dc motor drive applications. They can also be used for regenerative braking of drives to return energy back to the supplies resulting in energy savings. This feature is widely used in traction motor drives used in transportation systems. The chapter concentrates on the rapid prototyping of the step down or buck chopper configuration.

6.2 Modeling and simulation of the step-down DC-DC converter

In the step down DC-DC converter the average output voltage is a function of the average input voltage and the duty ratio of the switching element and is given by relation 6.1.

\[
V_{\text{output}} = D \times V_{\text{input}} \quad [6.1]
\]
This relation holds true only for a continuous conduction mode. For discontinuous mode of operation the relation depends on the values of circuit parameters like switching frequency and inductance also [40,41]. Hence in a buck chopper the output voltage can be controlled by varying the duty-ratio of the switching element. This is done using two methods:

1. Keeping on-time constant and varying the off time of the switch. This changes the frequency of operation of the converter.

2. Keeping switching frequency constant and varying the on time.

The second method is preferred as it results in a fixed clock that can be used to drive the gate circuitry of the switch. A square wave signal of desired duty-ratio needs to be generated using the timers of the controller for realizing the trigger signal scheme for the converter.

6.2.1 Simulation diagram for DC-DC buck chopper.

The DSPTimer block has been used for providing trigger signals for the DC-DC buck chopper circuit. The DSP timer block represents an up-counting 16 bit asymmetric timer. The block allows the user to set the frequency and duty ratio of the square wave signal that is fed to the gate of the controllable switch. The load used is a passive RL circuit with values that are available in the lab. The simulation diagram is shown in Fig. 6.1 and the example is part of the Simulation sub-library of DSPLib.
The main circuit parameters for the simulation are:

a. Switching frequency: 1kHz

b. Duty ratio: 50%

c. Load Inductance: 0.8H

d. Load Resistance: 300Ω

e. Source Voltage: 50V

6.2.2 Simulation results

The circuit was simulated using the Dymola simulator for a period of 0.5s. The waveforms for the control signal given to the switch (DSPTimer.v) are shown in figure 6.2. The output of the block is a square wave of duty ratio 50% at a frequency of 1kHz.
For a duty ratio of 50% the ideal average output voltage assuming no losses must be 25V. As shown in Fig. 6.3 the average output voltage obtained is 24.75V. The drop in voltage is from the ideal value is due to the resistance of the source and switch. The average value for the resistor voltage was measured using the Averager block from the measurements sub-library of the Power Electronics library developed at Iowa State University [27]. The simulation diagram represents the power electronic system shown in figure 2.2 and 2.4.
6.3 Implementation of buck chopper

The buck chopper is implemented using IGBT modules provided by LabVolt Inc [42] and controlled by a TMS320F243 DSP. The code for the DSP is auto generated from the simulated Modelica system using the application FixedCodeGen. Hence after simulating the system the application is run on the dsmodel.c file generated during simulation. Using the C243xyz.c and rtvector.asm file generated a code composer project is setup which allows the user to create an executable code on the DSP. The program flow of the C243xyz.c file for the DC-DC converter is shown in Fig 6.5. The executable is downloaded onto the program space of the DSP and run to generate the desired output. The timer output is obtained from the T1CMP pin (Fig. 6.4) on the I/O connector on the C243 board. The output is given to the switching control unit of IGBT 1 through the interface circuit.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC, +5 Volts</td>
<td>2</td>
<td>VCC, +5 Volts</td>
</tr>
<tr>
<td>3</td>
<td>PWM1/CMP1/IOPA6</td>
<td>4</td>
<td>PWM2/CMP2/IOPA7</td>
</tr>
<tr>
<td>5</td>
<td>PWM3/CMP3/IOPB0</td>
<td>6</td>
<td>PWM4/CMP4/IOPB1</td>
</tr>
<tr>
<td>7</td>
<td>PWM5/CMP5/IOPB2</td>
<td>8</td>
<td>PWM6/CMP6/IOPB3</td>
</tr>
<tr>
<td>9</td>
<td>PWM2/CMP2/IOPA7</td>
<td>10</td>
<td>PWM4/CMP4/IOPB4</td>
</tr>
<tr>
<td>11</td>
<td>CMP6/IOPB3</td>
<td>12</td>
<td>T1/CMP/IOPB4</td>
</tr>
<tr>
<td>13</td>
<td>T2/CMP/IOPB5</td>
<td>14</td>
<td>IOPD5</td>
</tr>
<tr>
<td>15</td>
<td>TDIR/IOPB6</td>
<td>16</td>
<td>TCLKIN/IOPB7</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>XF/IOPC0</td>
<td>20</td>
<td>BIO/IOPC1</td>
</tr>
<tr>
<td>21</td>
<td>CAP1/QEP0/IOPA3</td>
<td>22</td>
<td>CAP2/QEP1/IOPA4</td>
</tr>
<tr>
<td>23</td>
<td>CAP3/IOPA5</td>
<td>24</td>
<td>IOPD4</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>26</td>
<td>PDPINT</td>
</tr>
<tr>
<td>27</td>
<td>SCITXD/IOPA0</td>
<td>28</td>
<td>SCIRXD/IOPC3</td>
</tr>
<tr>
<td>29</td>
<td>SPI/SIMO/IOPC2</td>
<td>30</td>
<td>SPI/SIMO/IOPC3</td>
</tr>
<tr>
<td>31</td>
<td>SPI/CLK/IOPC4</td>
<td>32</td>
<td>SPI/STE/IOPC5</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>34</td>
<td>GND</td>
</tr>
</tbody>
</table>

Figure 6.4 Pinout for DC-DC step down converter
Initialize System Registers

Setup Timer 1 to output square wave

Initialize duty ratio to 50%, frequency 1kHz

Assign memory address OxA001 for duty ratio and initialize it

Check OxA001 after each cycle for duty ratio value

Run timer

Real-Time

Figure 6.5 Program flow for DC-DC chopper
6.3.1 Experimental setup and results

The circuit used for implementing the Buck chopper is shown in Fig. 6.6. The IGBT power electronic module from LabVolts consists of 6 devices with over-current protection and gate drive circuits. One of the devices of the module was used as the switching element for the buck chopper circuit. The timer output is given as a control signal to the gate of the IGBT through the interface circuit described in Chapter 4. The waveforms obtained on running the DSP with the executable for the trigger signal, resistor voltage and current is shown in Fig. 6.7. The results obtained are comparable with those obtained from the simulation. The drop in voltages from the simulated and theoretical values is due to the resistive loss across the components. A lab experiment based on the setup is given in Appendix A.
Figure 6.7 Output voltage and voltage across resistor (Duty ratio = 50%)

The output voltage across the resistor was measured using a Terco differential probe on a 1:100 setting. The mean output voltage obtained was 24.4V, which is comparable to the theoretical value of 25V. The waveforms were observed on a Tektronix TDS300 series oscilloscope. The scale for the resistor voltage waveform is 100mv/div and for the output voltage it is 500mv/div.
6.3.2 Real-Time control of duty ratio

Real-time control of the duty ratio can be done through Code Composer while the DSP is running. As mentioned earlier the duty ratio value is obtained from memory address 0xA001.

Using Code Composer’s View Memory option the contents of the address can be changed as shown in Fig 6.8. The new value of duty ratio would be used in the next cycle of the timer operation and the control signal would change accordingly. The output of the circuit would however settle down depending on its time constant. For a duty ratio value equal to the period register value the resulting waveform would have a duty of 100%. The period register value been used can be read of the input file C243xyz.c. The output obtained on changing the duty ratio to 20% is shown in Fig. 6.9.

![Figure 6.8 Code composer window for editing memory address 0XA001](image-url)
6.4 Conclusions

The rapid prototyping of a step-down DC-DC converter is discussed in this chapter. The steps involved in realizing the converter from simulation to implementation are explained. Results obtained from the real-time control of duty ratio of the chopper are also shown. The use of this setup for performing lab experiments is explained in Appendix A.

The next chapter concentrates on the rapid prototyping of the single phase and three phase square wave inverters.
CHAPTER 7. RAPID PROTOTYPING OF SQUARE WAVE DC-AC INVERTER

7.1 Introduction

The DC-AC inverter is a power electronic circuit used to change energy from direct current to alternating current. The role of an inverter is to convert a DC input voltage into an AC output voltage of desired magnitude and frequency. The output voltage waveforms of ideal inverters should be sinusoidal, however the waveforms of practical inverters are non-sinusoidal due to the presence of harmonics. For high-power applications low harmonics content is desired whereas for low and medium power applications square or quasi-square wave voltages may be acceptable. There are a number of different schemes for implementing inverters such as:

1. Square-Wave Inverters
2. Pulse Width Modulated (PWM) Inverters
3. Space Vector Modulated (SVM) Inverters

The chapter focuses on the rapid prototyping of square wave single-phase and three-phase square wave inverters. Their simulation and concurrent implementation with real-time control on frequency using models from DSPLib and FixedCodeGen is discussed in this chapter.
7.2 Rapid prototyping of single-phase square wave DC-AC inverter

The single-phase square wave inverter is realized using four switching devices each with anti-parallel diodes. To get the desired output opposite pairs of switches are turned on at the desired frequency. The implementation of the inverter from simulation to hardware realizations is discussed in the following sections. The inverter circuits represent the power electronic system shown in the modeling as well as implementation sections of figures 2.2 and 2.4.

7.2.1 Simulation diagram for single-phase square wave inverter

The multi function timer block from the DSP interfaces library has been utilized for providing the necessary trigger signals for the single-phase inverter. For the square wave switching scheme two square waves phase shifted by 180° is given to the switches [41]. The switches conduct in pairs to generate a square wave output on the load side. The simulation diagram for the inverter is shown in Fig. 7.1. The Multi-function timer block generates a square wave output of 50% duty ratio at pins C0 and C1 that are phase shifted by 180° with respect to each other. The main circuit parameters for the simulation are:

a. DSP Multi function parameters
   1. Frequency – 60 Hz
   2. Onfor – 1
   3. Shift –1
   4. IOPC0 and IOPC1 = 1, rest 0.

b. Load Inductance: 0.8H

c. Load Resistance: 300Ω

d. Source Voltage: 50
7.2.2 Simulation results

The circuit was simulated using the Dymola simulator for a period of 0.5s. The waveforms for the control signal given to the switches are shown in figure 7.2. The output load voltage and voltage across the resistor obtained from simulation is shown in figure 7.3.
7.2.3 Implementation of single-phase square wave inverter

The single-phase inverter is implemented using IGBT modules provided by LabVolt Inc [40] and controlled by a TMS320F243 DSP. The code for the DSP is auto generated from the simulated Modelica system using the application FixedCodeGen. Hence after simulating the system the application is run on the dsmodel.c file generated during simulation. Using the
C243xyz.c and rtvector.asm file generated a Code Composer project is setup which allows the user to create an executable code on the DSP. The program flow of the C243xyz.c file for the DC-AC converter is shown in Fig 7.5. The executable is downloaded onto the program space of the DSP and run to generate the desired output. The timer output is obtained from IOPC0 and IOPC1 pins (Fig. 7.4) on the I/O connector on the C243 board. The output is given to the switching control unit of the IGBT module through the interface circuit.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC, +5 Volts</td>
<td>2</td>
<td>VCC, +5 Volts</td>
</tr>
<tr>
<td>3</td>
<td>PWM1/CMP1/IOPA6</td>
<td>4</td>
<td>PWM2/CMP2/IOPA7</td>
</tr>
<tr>
<td>5</td>
<td>PWM3/CMP3/IOPB0</td>
<td>6</td>
<td>PWM4/CMP4/IOPB1</td>
</tr>
<tr>
<td>7</td>
<td>PWM5/CMP5/IOPB2</td>
<td>8</td>
<td>PWM6/CMP6/IOPB3</td>
</tr>
<tr>
<td>9</td>
<td>PWM2/CMP2/IOPA7</td>
<td>10</td>
<td>PWM4/CMP4/IOPB4</td>
</tr>
<tr>
<td>11</td>
<td>CMP6/IOPB3</td>
<td>12</td>
<td>T1/CMP/IOPB4</td>
</tr>
<tr>
<td>13</td>
<td>T2/CMP/IOPB5</td>
<td>14</td>
<td>IOPD5</td>
</tr>
<tr>
<td>15</td>
<td>TDIR/IOPB6</td>
<td>16</td>
<td>TCLIK/IOPB7</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>XF/IOPC0</td>
<td>20</td>
<td>BIO/IOPC1</td>
</tr>
<tr>
<td>21</td>
<td>CAP1/QEPO/IOPA3</td>
<td>22</td>
<td>CAP2/QEP1/IOPA4</td>
</tr>
<tr>
<td>23</td>
<td>CAP3/IOPA5</td>
<td>24</td>
<td>IOPD4</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>26</td>
<td>PDPINT</td>
</tr>
<tr>
<td>27</td>
<td>SCITXD/IOPA0</td>
<td>28</td>
<td>SCIRXD/IOPC3</td>
</tr>
<tr>
<td>29</td>
<td>SPI/SIMO/IOPC2</td>
<td>30</td>
<td>SPI/SOM/IOPC3</td>
</tr>
<tr>
<td>31</td>
<td>SPI/CLK/IOPC4</td>
<td>32</td>
<td>SPI/STE/IOPC5</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>34</td>
<td>GND</td>
</tr>
</tbody>
</table>

Figure 7.4 Pin out for single phase square wave inverter
Initialize System Registers

Setup Timer 1 to output square wave

Initialize duty ratio to 50%, frequency twice user defined

Assign memory address 0xA000 for frequency and initialize it

Real-Time

Check 0xA000 for new frequency value

Run timer

Output square wave at IOPC0 and IOPC1

Figure 7.5 Software flow for single-phase inverter
7.2.4 Experimental setup and results

The circuit used for implementing the inverter is shown in Fig. 7.6. The IGBT power electronic module from LabVolts consists of 6 devices with over-current protection and gate drive circuits. The output from pins IOPC0 and IOPC1 is given as a control signal to the gates of the IGBT through the interface circuit described in Chapter 4. The waveforms obtained on running the DSP with the executable for the trigger signal, resistor voltage and output voltage is shown in Fig. 7.7. The results obtained are comparable with those obtained from the simulation. The scale for the resistor voltage waveform and output voltage waveform is 500mv/div.

![Circuit diagram for single phase square wave inverter](image)

**Figure 7.6 Circuit diagram for single phase square wave inverter**
7.2.5 Real-Time control of frequency

Real-time control of the frequency can be done through Code Composer while the DSP is running. The frequency value can be changed by editing the contents of memory address 0xA002 using Code Composer’s View Memory option. The new value of frequency would be used in the next cycle of the timer operation and the output would change accordingly. The output obtained on changing the frequency to 10Hz is shown in Fig. 7.8.
7.3 Rapid prototyping of three-phase square wave inverter

7.3.1 Simulation diagram for three-phase square wave inverter

The multi function timer block from the DSP interfaces library has been utilized for providing the necessary trigger signals for the single-phase inverter. For the square wave switching scheme six square waves phase shifted by 60° is given to the switches [41]. The switches conduct in pairs to generate a square wave output on the load side. The simulation diagram for the inverter is shown in Fig. 7.9.
Figure 7.9 Simulation diagram for three-phase square wave inverter

The Multi-function timer block generates a square wave output of 50% duty ratio at pins C0 and C1 that are phase shifted by 180° with respect to each other. The main circuit parameters for the simulation are:

a. DSP Multi function parameters
   1. Frequency – 60 Hz
   2. Onfor – 3
   3. Shift –1
   4. IOPC0 to IOPC5 = 1, rest 0.

b. Load Inductance: 0.8H
c. Load Resistance: 300Ω

d. Source Voltage: 50V

7.3.2 Simulation results

The circuit was simulated using the Dymola simulator for a period of 0.5s. The output phase voltage and voltage across the resistor obtained from simulation is shown in figure 7.10.

![Figure 7.10 Output voltage and voltage across resistor for three-phase inverter](image)

7.3.3 Implementation of three-phase square wave inverter

The single-phase inverter is implemented using IGBT modules provided by LabVolt Inc [40] and controlled by a TMS320F243 DSP. The code for the DSP is auto generated from the simulated Modelica system using the application FixedCodeGen. Using the C243xyz.c and rtvector.asm file generated a Code Composer project is setup which allows the user to create an executable code on the DSP. The program flow of the C243xyz.c file for the three-phase DC-AC converter is shown in Fig 7.12. The executable is downloaded onto the program space of the DSP and run to generate the desired output. The timer output is obtained from
pins IOPC0 to IOPC5 (Fig. 7.11) on the I/O connector on the C243 board. The output is
given to the switching control unit of the IGBT module through the interface circuit.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC, +5 Volts</td>
<td>2</td>
<td>VCC, +5 Volts</td>
</tr>
<tr>
<td>3</td>
<td>PWM1/CMP1/IOPA6</td>
<td>4</td>
<td>PWM2/CMP2/IOPA7</td>
</tr>
<tr>
<td>5</td>
<td>PWM3/CMP3/IOPB0</td>
<td>6</td>
<td>PWM4/CMP4/IOPB1</td>
</tr>
<tr>
<td>7</td>
<td>PWM5/CMP5/IOPB2</td>
<td>8</td>
<td>PWM6/CMP6/IOPB3</td>
</tr>
<tr>
<td>9</td>
<td>PWM2/CMP2/IOPA7</td>
<td>10</td>
<td>PWM4/CMP4/IOPB4</td>
</tr>
<tr>
<td>11</td>
<td>CMP6/IOPB3</td>
<td>12</td>
<td>T1CMP/IOPB4</td>
</tr>
<tr>
<td>13</td>
<td>T2CMP/IOPB5</td>
<td>14</td>
<td>IOPD5</td>
</tr>
<tr>
<td>15</td>
<td>TDIR/IOPB6</td>
<td>16</td>
<td>TCLKIN/IOPB7</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>XF/IOPC0</td>
<td>20</td>
<td>BIO/IOPC1</td>
</tr>
<tr>
<td>21</td>
<td>CAP1/QEP0/IOPA3</td>
<td>22</td>
<td>CAP2/QEP1/IOPA4</td>
</tr>
<tr>
<td>23</td>
<td>CAP3/IOPA5</td>
<td>24</td>
<td>IOPD4</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>26</td>
<td>PDPINT</td>
</tr>
<tr>
<td>27</td>
<td>SCITXD/IOPA0</td>
<td>28</td>
<td>SCIRXD/IOPC3</td>
</tr>
<tr>
<td>29</td>
<td>SPI/SIMO/IOPC2</td>
<td>30</td>
<td>SPI/SOM1/IOPC3</td>
</tr>
<tr>
<td>31</td>
<td>SPI/CLK/IOPC4</td>
<td>32</td>
<td>SPI/STE/IOPC5</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>34</td>
<td>GND</td>
</tr>
</tbody>
</table>

Figure 7.11 Pin out for three-phase square wave inverter
Initialize System Registers

Setup Timer 1 to output square wave

Initialize duty ratio to 50%, frequency six user defined

Assign memory address 0xA000 for frequency and initialize it

Check 0xA000 for new frequency value

Real-Time

Run timer

Output square waves at IOPC0 to IOPC5

Figure 7.12 Software flow for three-phase inverter
7.3.4 Experimental setup and results

The circuit used for implementing the inverter is shown in Fig. 7.13. The IGBT power electronic module from LabVolts consists of 6 devices with over-current protection and gate drive circuits. The output from pins IOPC0 and IOPC1 is given as a control signal to the gates of the IGBT through the interface circuit described in Chapter 4. The waveforms obtained on running the DSP with the executable for the trigger signal, resistor voltage and output voltage is shown in Fig. 7.14. The results obtained are comparable with those obtained from the simulation. The scale for the resistor voltage waveform and the output voltage waveform is 200mv/div.
7.3.5 Real-Time control of frequency

Real-time control of the frequency can be done through Code Composer while the DSP is running. The frequency value can be changed by editing the contents of memory address 0xA002 using Code Composer’s View Memory option. The new value of frequency would be used in the next cycle of the timer operation and the output would change accordingly. The output obtained on changing the frequency to 10Hz is shown in Fig. 7.15.

Figure 7.14 Resistor voltage and output voltage for a frequency of 60 Hz
7.4 Conclusions

The rapid prototyping of single-phase and three-phase square wave inverters is discussed in detail in this chapter. The trigger signals necessary for the inverters, the simulation diagrams and the steps for realizing the inverter using Lab Volts modules is discussed. Results obtained from the real-time control of frequency of operation of the inverter are also shown.

In the next chapter rapid prototyping of the single phase and three phase line synchronized controlled rectifiers is discussed.
CHAPTER 8. RAPID PROTOTYPING OF AC-DC CONVERTER

8.1 Introduction

The AC-DC converter is a power electronic circuit used to change energy from alternating current to direct current. The output voltage of controlled rectifier circuits is varied by controlling the firing angle or delay of the device. Phase controlled thyristors are turned on by applying a gate pulse and turned off by natural or forced commutation. Phase controlled rectifiers can be divided into single phase and three phase types based on the input supply.

The thesis focuses on the rapid prototyping of both single-phase and three-phase rectifiers. Their simulation and concurrent implementation with real-time control on frequency using modelica based models from DSPLib and FixedCodeGen is discussed in this chapter.

8.2 Rapid prototyping of a single-phase controlled rectifier

8.2.1 Simulation diagram for single-phase controlled rectifier

The triggered multi function timer block from the DSP interfaces library has been utilized for providing the necessary trigger signals for the single-phase rectifier. A level detector is used to convert the input sinusoidal signal into a square wave signal. This is used by the capture block embedded inside the triggered multi-function block for line
synchronization. Trigger signals for switch pairs are phase shifted by 180° for both positive and negative half conduction. The simulation diagram for the rectifier is shown in Fig. 8.1.

The Triggered multi-function timer block generates trigger pulses at pins C0 and C1 that are phase shifted by 180° with respect to each other. The main circuit parameters for the simulation are:

a. DSP Multi function parameters

1. Frequency = 60 Hz

2. Alpha = 4.16
3. IOPC0 and IOPC1 = 1, rest 0.

b. Load Inductance: 0.8H
c. Load Resistance: 100Ω
d. Source Voltage: 120

8.2.2 Simulation results

The circuit was simulated using the Dymola simulator for a period of 0.5s. The output load voltage and voltage across the resistor obtained from simulation is shown in figure 8.2. The average output voltage obtained was measured using an averager block and was calculated to be 103V

![Figure 8.2 Output voltage for firing angle of 15°](image)

8.2.3 Implementation of single-phase controlled rectifier

The single-phase rectifier is implemented using thyristor modules provided by LabVolt Inc [40] and controlled by a TMS320F243 DSP. The code for the DSP is auto generated from the simulated Modelica system using the application FixedCodeGen. Hence after simulating the system the application is run on the dsmodel.c file generated during
simulation. Using the C243xyz.c and rtvector.asm file generated a Code Composer project is setup which allows the user to create an executable code on the DSP. The program flow of the C243xyz.c file for the converter is shown in Fig 8.5. The executable is downloaded onto the program space of the DSP and run to generate the desired output. The triggered multifunction timer output is obtained from IOPC0 and IOPC1 pins (Fig. 8.4) on the I/O connector on the C243 board. The output is given to the switching control unit of the thyristor module through the interface circuit described in chapter 4.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC, +5 Volts</td>
<td>2</td>
<td>VCC, +5 Volts</td>
</tr>
<tr>
<td>3</td>
<td>PWM1/CMP1/IOPA6</td>
<td>4</td>
<td>PWM2/CMP2/IOPA7</td>
</tr>
<tr>
<td>5</td>
<td>PWM3/CMP3/IOPB0</td>
<td>6</td>
<td>PWM4/CMP4/IOPB1</td>
</tr>
<tr>
<td>7</td>
<td>PWM5/CMP5/IOPB2</td>
<td>8</td>
<td>PWM6/CMP6/IOPB3</td>
</tr>
<tr>
<td>9</td>
<td>PWM2/CMP2/IOPA7</td>
<td>10</td>
<td>PWM4/CMP4/IOPB4</td>
</tr>
<tr>
<td>11</td>
<td>CMP6/IOPB3</td>
<td>12</td>
<td>T1CMP/IOPB4</td>
</tr>
<tr>
<td>13</td>
<td>T2CMP/IOPB5</td>
<td>14</td>
<td>IOPD5</td>
</tr>
<tr>
<td>15</td>
<td>TDIR/IOPB6</td>
<td>16</td>
<td>TCLKIN/IOPB7</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>XF/IOPC0</td>
<td>20</td>
<td>BIO/IOPC1</td>
</tr>
<tr>
<td>21</td>
<td>CAP1/QEP0/IOPA3</td>
<td>22</td>
<td>CAP2/QEP1/IOPA4</td>
</tr>
<tr>
<td>23</td>
<td>CAP3/IOPA5</td>
<td>24</td>
<td>IOPD4</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>26</td>
<td>PDPINT</td>
</tr>
<tr>
<td>27</td>
<td>SCITXD/IOPA0</td>
<td>28</td>
<td>SClRXD/IOPC3</td>
</tr>
<tr>
<td>29</td>
<td>SPISMO/IOPC2</td>
<td>30</td>
<td>SPI SOM1/IOPC3</td>
</tr>
<tr>
<td>31</td>
<td>SPICLK/IOPC4</td>
<td>32</td>
<td>SPISTE/IOPC5</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>34</td>
<td>GND</td>
</tr>
</tbody>
</table>

Figure 8.4 Pin out for single phase rectifier
Figure 8.5 Software flow for single-phase rectifier
8.2.4 *Experimental setup and results*

The circuit used for implementing the rectifier is shown in Fig. 8.6. The thyristor module from LabVolts consists of 6 devices with protection and gate drive circuits. The output from pins IOPC0 and IOPC1 is given as a control signal through the interface circuit described in Chapter 4. The waveforms obtained on running the DSP with the executable for the trigger signal, resistor voltage and output voltage is shown in Fig. 8.7. The results obtained are comparable with those obtained from the simulation. The average output voltage was measured to be 100V; the drop in voltages from the simulated and theoretical values is due to the resistive loss across the components. The scale for the input and output voltage waveforms is 1V/div.

![Experimental setup for single phase rectifier](image)

*Figure 8.6 Experimental setup for single phase rectifier*
8.2.5 Real-Time control of firing angle

Real-time control of the firing angle can be done through Code Composer while the DSP is running. The firing angle value can be changed by editing the contents of memory address 0xA000 using Code Composer’s View Memory option. The new value would be used in the next cycle of the timer operation and the output would change accordingly. The output obtained on firing angle to 30° is shown in Fig. 8.8.
Figure 8.8 Input voltage and output voltage for a firing angle of 30°

8.3 Rapid prototyping of three-phase controlled rectifier

8.3.1 Simulation diagram for three-phase controlled rectifier

The triggered multi function timer block from the DSP interfaces library has been utilized for providing the necessary trigger signals for the three-phase rectifier. A level detector is used to convert the input sinusoidal signal into a square wave signal. This is used by the capture block embedded inside the triggered multi-function block for line synchronization. The synchronizing signal been used is the phase voltage and hence to get
the desired firing angle a value of $30^\circ$ is added. The simulation diagram for the rectifier is shown in Fig. 8.9.

![Simulation diagram for three-phase wave rectifier](image)

**Figure 8.9 Simulation diagram for three-phase wave rectifier**

The Multi-function timer block generates a square wave output of 50% duty ratio at pins C0 and C1 that are phase shifted by $180^\circ$ with respect to each other. The main circuit parameters for the simulation are:

- **a. DSP Multi function parameters**
  1. Frequency = 60 Hz
  2. Alpha = 3
  3. IOPC0 to IOPC5 = 1, rest 0.

- **b. Load Inductance: 0.5H**
c. Load Resistance: 100Ω

d. Source Voltage: 60V

8.3.2 Simulation results

The circuit was simulated using the Dymola simulator for a period of 0.5s. The output phase voltage and voltage across the resistor obtained from simulation is shown in figure 8.10. The average output voltage was measured to be 142V.

![Figure 8.10 Output voltage and input line voltage for 0° firing angle](image)

8.3.3 Implementation of three-phase controlled rectifier

The three-phase rectifier is implemented using thyristor modules provided by LabVolt Inc [40] and controlled by a TMS320F243 DSP. The code for the DSP is auto generated from the simulated Modelica system using the application FixedCodeGen. Using the C243xyz.c and rtvector.asm file generated a Code Composer project is setup which allows the user to create an executable code on the DSP. The program flow of the C243xyz.c file for the three-phase rectifier is shown in Fig 8.11. The executable is downloaded onto the program space of the DSP and run to generate the desired output. The timer output is obtained from pins IOPC0 to
IOPC5 (Fig. 8.12) on the I/O connector on the C243 board. The output is given to the switching control unit of the thyristor module through the interface circuit.

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal</th>
<th>Pin #</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC, +5 Volts</td>
<td>2</td>
<td>VCC, +5 Volts</td>
</tr>
<tr>
<td>3</td>
<td>PWM1/CMP1/IOPA6</td>
<td>4</td>
<td>PWM2/CMP2/IOPA7</td>
</tr>
<tr>
<td>5</td>
<td>PWM3/CMP3/IOPB0</td>
<td>6</td>
<td>PWM4/CMP4/IOPB1</td>
</tr>
<tr>
<td>7</td>
<td>PWM5/CMP5/IOPB2</td>
<td>8</td>
<td>PWM6/CMP6/IOPB3</td>
</tr>
<tr>
<td>9</td>
<td>PWM2/CMP2/IOPA7</td>
<td>10</td>
<td>PWM4/CMP4/IOPB4</td>
</tr>
<tr>
<td>11</td>
<td>CMP6/IOPB3</td>
<td>12</td>
<td>T1/CMP/IOPB4</td>
</tr>
<tr>
<td>13</td>
<td>T1/CMP/IOPB5</td>
<td>14</td>
<td>IOPD5</td>
</tr>
<tr>
<td>15</td>
<td>TDIR/IOPB6</td>
<td>16</td>
<td>TCLKIN/IOPB7</td>
</tr>
<tr>
<td>17</td>
<td>GND</td>
<td>18</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>XF/IOPC0</td>
<td>20</td>
<td>BIO/IOPC1</td>
</tr>
<tr>
<td>21</td>
<td>CAP1/QEP0/IOPA3</td>
<td>22</td>
<td>CAP2/QEP1/IOPA4</td>
</tr>
<tr>
<td>23</td>
<td>CAP3/IOPA3</td>
<td>24</td>
<td>IOPD4</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
<td>26</td>
<td>PDPINT</td>
</tr>
<tr>
<td>27</td>
<td>SCI/TXD/IOPA0</td>
<td>28</td>
<td>SCI/RXD/IOPC3</td>
</tr>
<tr>
<td>29</td>
<td>SPI/SOM/IOPC2</td>
<td>30</td>
<td>SPI/SOM/IOPC3</td>
</tr>
<tr>
<td>31</td>
<td>SPI/CLK/IOPC4</td>
<td>32</td>
<td>SPI/STE/IOPC5</td>
</tr>
<tr>
<td>33</td>
<td>GND</td>
<td>34</td>
<td>GND</td>
</tr>
</tbody>
</table>

Figure 8.11 Pin out for three-phase rectifier
Figure 8.12 Software flow for three-phase rectifier
Figure 8.13 Schematic for three-phase rectifier implementation

8.3.4 Experimental setup and results

The circuit used for implementing the rectifier is shown in Fig. 8.13. The thyristor module from LabVolts consists of 6 devices with protection and gate drive circuits. The output from pins IOPC0 and IOPC5 is given as a control signal to the gates of the thyristor through the interface circuit described in Chapter 4. The waveforms obtained on running the DSP with the executable for the trigger signal, resistor voltage and output voltage is shown in Fig. 8.14. The scale for the input and output voltage waveforms is 500mV/div. The average output voltage was measured to be 140V. The results obtained are comparable with those obtained from the simulation. The drop in voltages from the simulated and theoretical values is due to the resistive loss across the components.
8.3.5 *Real-Time control of firing angle*

Real-time control of firing angle can be done through Code Composer while the DSP is running. The firing angle value can be changed by editing the contents of memory address 0xA000 using Code Composer’s *View Memory* option. The new value would be used in the next cycle of the timer operation and the output would change accordingly. The output obtained on changing firing angle to 30° is shown in Fig. 8.9.
8.4 Conclusions

The chapter discusses the rapid prototyping of single-phase and three-phase line synchronized rectifiers. The trigger signals necessary for the rectifiers, the simulation diagrams and the steps for realizing them using Lab Volts modules is discussed. Results obtained from the real-time control of firing angle are also shown.

The next chapter discusses the modifications needed to be made in the rapid prototyping scheme to incorporate closes loop control strategies.
CHAPTER 9. EXTENSION OF RAPID PROTOTYPING SCHEME FOR CLOSED LOOP CONTROL

9.1 Introduction

Digital signal processors provide the user with a platform to realize real-time math intensive control algorithms. Fixed-point DSP's such as those in the C2000 family from Texas Instruments contain sufficient memory and peripheral support to realize most control schemes. They also present the user with a cost-effective alternative in comparison to floating-point processors. The rapid prototyping scheme discussed has been used to configure classical converters and control them in an open loop. The chapter concentrates on closed loop control and suggests a framework for its implementation using Modelica based libraries and the FixedCodeGen automatic code generator.

9.2 Framework for closed loop control

The rapid prototyping scheme discussed in the thesis can be extended for incorporating closed loop control techniques. The framework for a generic closed loop scheme is shown in figure 9.1. Feedback from the plant would pass through a signal conditioning circuit and an A/D converter to generate a digital signal that would be fed to the DSP. Alternatively the DSP's inbuilt A/D converter could also be used if it is suitable for the desired application. The feedback signal would be compared with the reference signal and
the error would be calculated. The error signal would be used by the controller to generate
the desired control signal.

The controller is shown as a single-input single-output system but could also be a
multi-input multi-output system. This would necessitate the multiplexing of the A/D
converter channels to obtain the inputs needed by the controller. The DSP would also need to
provide control signals to trigger the data converter and read its output after the end of
conversion.

![Diagram](image.png)

**Figure 9.1 Framework for closed loop control implementation**

The digital signal obtained from the data converter needs to be processed and bit-
shifted to generate a signal compatible with the stored reference signal. The controller in its
general form can be represented as a transfer function and the coefficients of the various
terms would need to be represented accurately in fixed-point. The control signal generated
would then pass through a limiter and the interface circuit before it is applied to the plant. The limiter would impose maximum and minimum conditions on the control signal and the interface circuit would serve as protection between the DSP and the physical plant. A real-time link between the host-PC based development environment such as Code Composer and the DSP would enable the user to vary the parameters of the controller on the fly. This allows the user to fine tune his controller and achieve the desired performance without having to stop the program.

9.3 Modifications in scheme to incorporate closed loop control

To effectively represent the control scheme new models would need to be incorporated into DSPLib. These models need to be as general as possible to accommodate large number of control philosophies. Based on the framework shown in figure 9.1 models for A/D converters, controllers and limiters would be required.

The A/D converter model must be flexible enough to incorporate variable bit width and sampling frequency. Ports on the C243 processor are 8 bits wide and hence its desirable to model 8 bit ADC's. However if higher resolution were required the inbuilt 10-bit ADC present in the C243 DSP would need to be modeled. The ADC input line will need to be multiplexed for realizing multi input multi output controllers. The ADC will also need to receive control signals from the C243 DSP for starting conversion and transmitting the converted data signal into the port.

The controller can be represented as a transfer function with variable coefficients. The various coefficients need to be assigned values keeping into consideration fixed point scaling and overflow constraints. Deadband and PI controllers have been implemented on TI
DSP's and additional models for Clarke and Parks transform have been developed in the Digital Motion Control Library [43]. These models can be used as a reference for developing Modelica based models.

FixedCodeGen has been designed in a modular fashion [section 5.3] to ease the addition of new blocks and facilitate code generation. Separate functions read, decode and generate values for individual DSP registers. Models representing A/D converters and the controller would need to be read from the Modelica based system and their various parameters would need to be stored. Based on these values fixed-point code with appropriate constraints on scaling and overflow would need to be generated.

The extension of the scheme to closed loop systems would allow the users to rapidly test and develop sophisticated control schemes for converters and electric drives. The scheme would serve as an excellent test bed to realize applications for industries as well as research projects. The use of modular hardware tools enables the scheme to be implemented rapidly and tested.

### 9.4 Closed loop control of DC-DC buck chopper

The Code Composer environment has been used to develop a closed loop control for the DC-DC step down converter. The example illustrates the extensions needed in the rapid prototyping scheme to incorporate closed loop control. A simple scheme involving a proportional controller and an 8-bit A/D is used to generate the desired output voltage.
9.4.1 Closed loop control scheme

The basic structure of the fixed-point code used to realize the closed loop control scheme is shown in figure 9.2. The schematic for the closed loop controller is shown in figure 9.3.

```c
#include "F243_c.h"
/* Real time variables and pointers */
unsigned int (*Vref)= (unsigned int *) 0xA002;

void main(void)
{
    /* Start of initialization */
    SCSR = 0x0001;
    /* End of initialization */
    /* Initialize Controller Values */
    *Vref = 30;
    while(1){
        T1CMPR = *dutycon;
    }
    /* Interrupt Service Routine*/
    interrupt void timer2_isr(void)
    {
        /* Input signal from ADC */
        vactual = (*adout)*4;
        /* Apply Controller */
        (*dutycon) = (*dutycon) + volerror*alpha;
        /* Apply Limiter */
        /* Control ADC */
    }
}
```

Figure 9.2 Fixed-point code structure for DC-DC closed loop control
The voltage across the resistor is used as the feedback signal for the controller. The resistor voltage is applied to a voltage divider circuit and then through a low pass circuit. The low pass circuit eliminated most of the harmonics and the average stepped down DC voltage is sent to an 8-bit A/D converter. A control signal is sent to the A/D converter when conversion is complete and the digitized signal is transmitted to the DSP through Port A. The DSP compares the feedback signal with a preset reference and generates an error signal. The error signal is amplified through a proportional gain and added onto the previous control signal to generate the desired duty ratio output. The duty ratio value is clamped to between 90% and 5%. The fixed-point code that achieves this has been realized using the Code Composer integrated development environment. The code has been developed in the same structure shown in section 5.4.

Figure 9.3 Schematic for DC-DC closed loop control
Figure 9.2 illustrates the basic flow of the fixed-point code and the specific sections relating to the various blocks in the closed loop control scheme such as the controller, limiter and A/D converter. To incorporate these models into the automatic code generation scheme functions would need to be written to generate the specific code associates with each block.

9.4.2 Experimental results

The closed loop control scheme was realized using LabVolts modules [42]. The IGBT module was utilized as the switching device and a passive RL load was used. The controller was set to obtain a fixed average output voltage. The average output voltage obtained on fixing the reference and varying the source voltage is shown in figure 9.4. The reference value was set to 30V and the input voltage was varied from 52V to 75V. At higher input voltages due to the decreased duty ratio there is a larger ripple voltage. We can see that the average output voltage is nearly 30V (1 division corresponds to 10V) irrespective of the input voltage. The stepped variation of the input voltage was in the sequence 50V-75V-60V-40V-75V-50V.
Figure 9.4 Output voltage across resistor for input voltage variation

The average output voltage obtained on varying the desired reference for a fixed input is shown in figure 9.5. In this case the input voltage was fixed at 50V and the reference was varied from 25V to 45V (1 division corresponds to 10V). The order of the set reference value was 30V-45V-35V-25V-45V-30V.
9.5 Conclusions

In this chapter the extension of the rapid prototyping scheme for closed loop control is discussed. The various steps to be carried out are discussed and are illustrated using a DC-DC converter example. The control strategy has been implemented using the Code Composer IDE but illustrates the modifications needed to be made in FixedCodeGen and DSPLib.

The next chapter discusses the thesis conclusions and suggestions/thoughts for future work.
CHAPTER 10. CONCLUSIONS AND POTENTIAL FOR FUTURE WORK

10.1 Conclusions

A rapid prototyping scheme for power electronic circuits integrating the Modelica modeling platform with a C243 fixed-point processor is presented in this thesis. It enables the user to conceptualize the circuit using high level modeling tools and implement them using DSP based controllers and power electronic building blocks. This scheme provides the user with a flexible platform for rapidly implementing various converter topologies.

The use of the Modelica language provides the user with a powerful tool to model power electronic systems. Modelica models are open source and hence users have the flexibility in designing their own models as per their specific needs. The platform has been used to develop a library of standard peripherals called DSPLib that allows the user to simulate digitally controlled power electronic systems. Due to the open sourced nature of the modeling platform the functionality of the existing models can be easily extended to incorporate new algorithms and peripherals.

Various classical topologies such as DC-DC buck chopper, DC-AC square-wave inverters and AC-DC phase controlled converters have been implemented using this approach. The C243 DSP from Texas Instruments' is a popular cost effective target for implementing controllers in power electronic systems. However the scheme can target any processor in the C2000 family of TI as they are upward source compatible. Due to the open
sourced nature of the modeling platform the functionality of the existing models can be easily extended to incorporate new algorithms and peripherals.

The fixed-point code generation application FixedCodeGen can also be extended to target DSP's from other manufacturers like Motorola and Analog Devices. This enables the rapid prototyping scheme to be very flexible in nature and provides the user with a powerful tool to realize power electronic circuits. The embedded real-time link allows the user to vary control parameters on the fly and fine tune the trigger signal generation or controller.

The rapid prototyping scheme can also be used for educational purposes. Students can utilize the tool for modeling, simulation and concurrent real-time implementation of power electronic circuits. A possible lab experiment for a DC-DC buck chopper shown in Appendix A demonstrates how this tool can be used in a laboratory environment for educational purposes. The framework for extension of the scheme to closed loop control has also been discussed and preliminary work done in this direction is presented.

10.2 Potential for future work

In this thesis a rapid prototyping scheme for power electronic system has been presented. The effectiveness of the scheme has been demonstrated by the implementation of classical power electronic circuits. Open-loop control of the output has been achieved in all circuits. Complex power electronic systems with closed loop control can be built using this scheme. This would involve the development of models such as PI controllers, A/D converters and error detectors in Modelica and the extension of FixedCodeGen to incorporate these models into the fixed-point code generation scheme.
The concept of real time control demonstrated here can be extended to control power electronic systems from remote locations through the use of web based technologies. With the help of appropriate hardware support users would be able to remotely design, simulate, test and validate various converter topologies and controller schemes.
APPENDIX A. DC-DC CONVERTER LAB EXPERIMENT BASED ON PROPOSED RAPID PROTOTYPING SCHEME

A.1 Objective:
- Understand the working of the Buck Chopper. (DC to DC Converter)
- Study the waveforms obtained by an IGBT buck chopper feeding a R&L (passive) load.

A.2 Instruments and components required:

<table>
<thead>
<tr>
<th>Instrument Details</th>
<th>Model No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply (120 Vdc, 0-120Vdc)</td>
<td>8821</td>
</tr>
<tr>
<td>DC Metering Module (200V, 5A)</td>
<td>8412</td>
</tr>
<tr>
<td>IGBT Chopper/Inverter</td>
<td>8837</td>
</tr>
<tr>
<td>Oscilloscope</td>
<td>Tektronix 2221A</td>
</tr>
<tr>
<td>TMS320F243 EVM Kit</td>
<td></td>
</tr>
<tr>
<td>EVM Power Supply</td>
<td></td>
</tr>
<tr>
<td>JTAG Emulator and cable</td>
<td></td>
</tr>
<tr>
<td>Interface Circuit</td>
<td></td>
</tr>
<tr>
<td>Power Supply (24 V)</td>
<td>30004</td>
</tr>
<tr>
<td>Inductor</td>
<td>8321</td>
</tr>
<tr>
<td>Resistance</td>
<td>8311</td>
</tr>
<tr>
<td>2 Differential probe</td>
<td>Terco MV 1971</td>
</tr>
</tbody>
</table>
A.3 Introduction:

Figure A.1 Buck converter connected to a RLE Load

Figure A.1 shows a buck chopper driving a RLE load. This converter operates from a fixed dc voltage $V_s$ and controls the average output voltage $V_o$ across the load. Modifying the duty ratio 'k' of the control signal by switching the IGBT on and off, one can control the average output dc voltage ($V_o$) as desired.

Figure A.2 shows the current and voltage waveforms of the RL load for a given duty ratio. The waveform shown corresponds to continuous conduction case.

Figure A.2 Inductor current and output voltage waveforms
The equation relating to the output voltage $V_o$ to the input dc voltage is as follows:

$$V_o = k \cdot V_s \quad \text{Eq. 1}$$

where

$V_s =$ Input voltage;

$V_o =$ output voltage and $k =$ Duty ratio = (Ton/T)

### A.4 Modeling and simulation of the Buck Chopper

The buck chopper circuit to be simulated is shown in figure 3. The various models used in the circuit are:

1. Constant DC voltage source; From Electrical/Sources sub-library
2. Three pin switch; From Power Electronics Library/Switches sub-library
3. Diode; From Power Electronics Library/Switches sub-library
4. Inductor, Resistor and ground; From Electrical/Basic sub-library
5. Averager; From Power Electronics Library/Measurement blocks sub-library
6. DSPTimer; From DSPLibrary/DSPunit sub-library

Graphically model the buck converter

Graphically model the buck chopper as shown in figure 3. For connecting two components place the cursor on the pin of one and draw a line to the second. The wire symbol in the menu is not to be used and is used only for defining symbols for individual components.

Double click on the various components and set the following parameters:

Voltage Source: Component Name “Vin” Value 120V

Resistor: Component Name “Rd” Value 300 ohms

Inductor: Component Name “Ld” Value 1.6H

DSPTimer: Component Name “DSPTimer1”, Frequency 1000Hz and duty ratio 50

Now save the entire circuit as Dcbuck.mo in your home directory.

Simulation

From the File menu translate the buck chopper schematic. Now using the Dymola main window set the final time to be 0.2s and the output interval to be 0.0001s and then simulate the circuit.

Perform the simulation for different duty ratios by changing the value in the DSPTimer block in the schematic. For a list of duty ratios to be simulated check the questions at the end of the lab notes.
Observing results

Once the simulation is complete, open the Plot Window. Click the Variables command. A window called Plot Variables would pop up. The Plot Variable window would display all the variables that were calculated during the simulation.

Select the three variables Vin.v, Rd.v, Ld.i, and plot them. To view the waveform in detail, you can press the left mouse button and create a rectangular window over the portion of the waveform. Once the window is selected just release the left mouse button.

A.5 Experimental realization of the buck chopper

The buck chopper setup uses the Lab-Volt power supply module, IGBT chopper/inverter power module and the TMS320F243 EVM board for trigger signals. For the experimental setup (Fig A.4) the various modules that are used with the chopper/inverter are:

1. Power Supply (120 Vdc, 0-120Vdc)
   Purpose: DC supply from terminals 7 and N of the 3-phase power supply module (8821) is given to the IGBT Chopper/Inverter module. This is the input voltage \( V_s \) to the buck chopper.

2. IGBT Chopper/Inverter Power Module (8837)
   Purpose: The Lab-Volt IGBT Chopper/inverter module is a multipurpose switch module that can be used to setup various topologies of chopper and inverter circuits.

3. TMS320F243 EVM Board
   Purpose: The TMS320F243 is a powerful DSP optimized for motion control applications. The DSP is used to generate trigger signals for the required converter topology. With a
real-time link existing between the DSP and the PC on the fly variation of parameters is possible.

Figure A.4  Experimental setup for IGBT buck chopper driving a RL load

The buck chopper circuit will be realized using lab volts modules. The control signal necessary for the switching element will be provided by the C243 fixed-point DSP. The software code for generating the switching signal will be generated from the simulation of the converter performed using Dymola/Modelica.

Ask the TA to provide you with the FixedCodeGen application and the base files for the Code Composer development environment. Save these files in the same folder as the simulation file for Modelica/Dymola.
Now double-click on the FixedCodeGen application. The application will read through the dsmodel.c file created during the simulation and generate a fixed-point code file C243xyz.c and a vectors file rtvector.asm. The C243xyz.c file is the source code for realizing the trigger signal generation scheme for the buck chopper and is derived from the DSPtimer block present in the Modelica simulation diagram.

A.6 Setup for IGBT buck chopper connected to a passive RL Load:

![Circuit diagram of the IGBT buck chopper feeding a passive RL load](image)

Figure A.5 Circuit diagram of the IGBT buck chopper feeding a passive RL load

Figure A.4 shows how the various modules have to be inter-connected. Detailed connections to each module are shown in Fig. A.5

**Caution:** High voltages are present in this laboratory exercise! Don’t make or modify any banana jack connections when the power is on.
Make sure all the power supplies are on the O (OFF) position. Set the main switch of the Enclosure/Power Supply (LabVolt 8821) unit to the O (OFF) position.

a. Connect the module as shown in schematic for RL load (and according to step by step connections described below confirming to Figs. A.5)

b. Connect positive side of DC voltage source (7) to IGBT Chopper input (1) and connect the negative side of DC voltage source (N) to IGBT chopper input (2). See Fig. A.5

c. In the IGBT module (8837), set the braking switch to OFF position. S1 and S2 to OFF position (down). Connect “Low Power Input 24 V” of IGBT power module unit to 24 V (LabVolt 30004.) power supply.

d. Connect IGBT output (3) to current (+) terminal of ammeter (2.5 A) (8412).

e. Connect (-) terminal of ammeter to inductor module 8321 (Fig. A.5) Now connect the resistor module 8311 to the inductor as shown in Fig. A.5. Now connect the resistor to terminal 2 of the IGBT chopper/inverter module. This finishes the load connections.

f. For measurement purposes connect a voltmeter across the load as shown in Fig. A.5. Keep in mind the polarity while connecting it to an oscilloscope.

g. To view the output voltage waveform connect a differential probe across the load. Connect the red lead of the differential probe (DP) to terminal 3 of the IGBT Chopper / Inverter module and blue lead to terminal 2 of the IGBT module.

h. To view the current waveform, connect a differential probe across the resistor. Ensure that the blue leads of both differential probes are connected to terminal 2 of the IGBT Chopper/Inverter module.
i. Connect T1CMP pin of the DSP (Pin 13 on I/O port on board) to switching
control input 1 on LabVolt IGBT power module via the output conditioning
circuit (Gate Drive Interface). Connections for the interface board are shown in
figure A.6.

![Diagram](image)

**Figure A.5 Gate drive interface board connection**

j. Connect the 5V and GND pins on the Interface board to the Agilent 3646A power
supply using the wires provided.

k. From the C243 DSP board use the provided connector to connect the T1CMP pin
and GND pin on I/O port to C0 and GND pin on connector J2 on interface board.

l. Connect the LVO and GND pin of connector J1 to the IGBT switching module
pin1 and ground as shown in figure

m. To observe the output waveforms on the oscilloscope output, i.e. connect BNB
wire from Voltage DP to Channel 1 and Channel 2 to BNB cable from current
DP.
n. Connect the 5V power supply to the DSP board and switch on the power supply to the board.

o. Start Code Composer on the PC. Open C243xyz.pjt project from the files provided by the TA.

p. View the fixed-point code file C243xyz.c, which is the source code for the trigger signal generation. The ratio of T1CMPR to T1PR sets the duty ratio of the output signal.

q. From the toolbar build the project to generate an executable code.

Study of Buck Chopper feeding RL load

a. As a passive load use 300 ohms resistance and 1.6 H inductance.

b. Connect the voltmeter and the load in parallel.

c. Connect current DP across the resistive load. Connect BNB wire from DP to oscilloscope channel 2

d. Turn on the power supply units 30004, 3646A and 8821.

e. Set the DC power supply (8821) to 120 V.

f. Load the program C243xyz.out present in the folder onto the DSP run the program.

g. View memory space 0xA001 using the View memory option on the toolbar.

h. Vary the duty ratio setting in steps by editing the contents of memory register 0xA001.
Input Voltage is 120 V

<table>
<thead>
<tr>
<th>Output Voltage</th>
<th>Output Current</th>
<th>0xA001</th>
<th>Duty Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volts</td>
<td>Amperes</td>
<td></td>
<td>%</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td>80</td>
</tr>
</tbody>
</table>

Table 1. Buck chopper circuit experimental data

i. Plot the waveforms observed on the oscilloscope for output voltage of 60V on a graph paper, separately.

j. Stop the program from the Code Composer environment

k. Turn off all the power supplies.
A.7 Questions:

1. Calculate the minimum inductance for continuous conduction of the chopper circuit.

2. Briefly describe the effect of switching frequency on the size of the inductor and capacitor for a chosen ripple voltage and ripple current.

3. What is the effect of switching control frequency of an IGBT buck chopper to the output voltage ripple and current ripple for a given inductor and capacitor?

4. Prepare a table of input voltage, average output voltage, and average output currents for duty ratios of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 and 0.8 from the Dymola simulation.

5. From the simulation results plot the output currents for each of these duty ratio values and submit them in the report.

6. For a duty ratio value of 0.5 measure the maximum and minimum values of output current. Plot the waveforms for diode current, diode voltage, switch current and switch voltage for this value. (Simulation)

7. Also calculate the ratio of $\Delta V_o / V_o$ for a duty ratio of 0.5. $V_o$ is the output voltage across the resistor from the simulation and from the experiment.

8. Calculate the values of $I_{\text{min}}$ and $I_{\text{max}}$ and the value of the current ripple through the inductor. Compare the theoretical value obtained with that from the experimental and simulation result for a duty ratio of 50%.

9. Compare the results obtained from simulation with those obtained from the experiment and explain any variations.
APPENDIX B. RAPID PROTOTYPING OF POWER ELECTRONIC CONVERTERS USING VISSIM RAPID PROTOTYPER SOFTWARE

B.1 Introduction

The Vissim rapid prototyper software is an embedded control system developer software for TI's C2000 family of fixed-point processors. The software supports code generation for the C24x and the more recent C28x series of processors. It provides the user with a graphical environment to develop embedded control systems and use the information to generate DSP specific code in an efficient manner. A real time interface is built using a JTAG link and allows the user to run hardware in the loop simulations and validate their design. The use of the software for generating trigger signals for power electronic converters is illustrated using an example. The drawback in utilizing the scheme for simulation of such systems is also discussed.

B.2 The rapid prototyping process using vissim

The first step in the rapid prototyping process is to develop a model of the system including the controller (algorithm) that is targeted for the DSP. The user constructs his model by selecting pre-built blocks from a block library and simply wires (graphically connects) the blocks into a diagram. The Vissim library consists of fixed and floating-point blocks that enable the user to develop the model of their system using an equation based approach. This process entails the user to generate the equations for his system and then
utilize limited blocks to realize them. Once the system model is completed, a simulation is run to validate the behavior of the algorithm. The results of the simulation can be viewed using output display blocks. The next step is to validate the behavior of the simulated algorithm on the target DSP. The user then selects the block(s) representing the controller and generates ANSI C code using Vissim. Vissim rapid prototyper automatically compiles-links-downloads the algorithm onto the target DSP. The Vissim GUI is retained while the algorithm executes on the DSP to enable the user to validate the performance of algorithm on the target. The user could visualize plot responses of variables or view the effects of changing gain values. Once the algorithm has been validated on the target DSP, the ANSI C code can be generated directly to a Code Composer Studio project - allowing the code to be integrated with other user developed code.

B.3 Buck chopper using vissim rapid prototyper software

The use of the vissim software for prototyping power electronic systems is illustrated by realizing a step down DC-DC converter. In the buck chopper the average output voltage is a function of the average input voltage and the duty ratio of the switching element and is given by the relation

$$V_{\text{output}} = D \times V_{\text{input}}$$

The control signal fed to the switching device is a square wave of variable duty ratio that is generated using the timer present on the DSP. Vissim enables the user to develop the controller signal using the square wave block present in its library. The block can be configured to generate a square wave of desired frequency and duty ratio. However the buck chopper circuit itself would need to be modeled either using a state space approach or using
the limited blocks present in the Vissim library. The blocks allow the user to model the system using an equation-based approach, which is a cumbersome task for modeling the event driven circuit. Hence the vissim tool can be used only for simulating the trigger signal generation scheme and not the entire buck chopper circuit. The platform also currently does not have the models needed to simulate a power electronic system such as switches, loads etc.

B.1 Buck chopper trigger signal generation using vissim

The Vissim model of the buck chopper trigger signal generation scheme is shown in figure B.1. The square wave block is shown in detail in figure B.2 and consists of an integrator block, which generates a triangular wave whose value is compared to the on time. Depending on the duty ratio value set the required square wave is generated as an output of the merge block and fed to the output port of the DSP. The entire trigger signal generation scheme is then compiled into DSP code and downloaded onto its RAM space.
The Labvolt IGBT module is used as the switching element for the buck chopper circuit. The output from the DSP is fed to the gate drive circuit using an interface circuit for protection and isolation. The buck chopper circuit has been realized using LabVolt modules and is shown in figure B.3.
The system is simulated from the Vissim menu. The generated trigger signal is fed to the IGBT gate. The input voltage to the IGBT buck chopper is set at 40V. In the software the duty ratio is initialized to 50%. The output voltage is 20V, which is consistent with the theoretical value. The real-time variation of duty ratio is possible by varying the on time specified in the square wave block. However the output current and other component currents cannot be seen in simulation, as the buck chopper circuit has not been modeled.

B.4 Conclusions

The vissim rapid prototyper software provides the user with the ability to realize trigger signal generation schemes and embedded control strategies for fixed-point DSP’s. However due to the limited library of components the user is limited to simulating only the controller operation. The simulation of the entire power electronic system cannot be achieved due to the lack of models describing switches and passive loads.

The rapid prototyping scheme presented in the thesis integrates an object-oriented open-source modeling platform Modelica with a cost-effective fixed-point DSP. The presence of a large number of models describing physical systems and power electronic systems in specific enables the user to easily model and simulate various converter topologies. The development of the DSPLib library of DSP peripherals and the automatic code generator FixedCodeGen integrates the simulation platform to the implementation platform. This allows the user to model/simulate and implement power electronic systems in real-time.
APPENDIX C. PUBLICATIONS ARISING FROM THESIS


BIBLIOGRAPHY


<http://dspvillage.ti.com/docs/catalog/resources/techdocs.jhtml?navSection=user_guides&familyId=110>

[38] Texas Instruments, Code Composer Studio white paper Publication No: SPRA520,


[43] Texas Instruments, Digital Motion Control (DMC) Library, Accessed June 5th 2003,
<http://focus.ti.com/docs/tool/toolfolder.jhtml?PartNumber=SPRC080>
ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. V. Ajjarapu for his support and guidance throughout this research project. I would also like to thank Dr. V.V. Sastry for his constant support and encouragement throughout my research and for providing me with valuable insights regarding my work. I would like to express my gratitude towards my committee members Dr. G.M. Prabhu and Dr. R.L. Geiger for their comments and suggestions regarding my work.

Special thanks are due to Jason Boyd for helping me with the PCB layout and fabrication for the hardware in the thesis. I would also like to thank Steven Kovarik and other members of the Computer Support Group for their help in installing various softwares needed for this work. I am also deeply indebted to my friends and colleagues for their constant support and valuable advise throughout my work. In addition I am grateful to all the professors and graduate students in the electric power program of Iowa State University.

Finally I would like to thank my parents and brothers for their blessings and encouragement that has helped me overcome all the hurdles that I have faced in my life.