Fast Prototyping and Deployment of Control Algorithms for Power Conversion Applications

Bijesh Poyil and Martin Murnane
Analog Devices, Inc.

Abstract

Model driven development has been adopted by the industry as a solution for fast prototyping and to reduce time to market. However, a significant amount of time and effort must typically be put in the final implementation stage to match the performance of the product to the performance of the model. The full potential of model driven development is not realized in practice due to this. In this article, we discuss how we can address this gap by following some guidelines and techniques during model development. We also cover how we can generate efficient code from the models to reduce the time to market of the product.

Introduction

The increasing depth of penetration of distributed energy resources such as grid tied solar inverters has resulted in the power conversion community to look for better, efficient, and cost-effective solutions for these markets. There are many algorithms and topologies in the literature to improve the output quality and efficiency of the power conversion process. Silicon vendors are coming up with new control processors with features and hardware support to implement these algorithms efficiently. It is very expensive to build the hardware prototype of a full inverter and to experiment with its performance under various conditions. Moreover, any malfunctioning of the algorithm during experimentation could damage the entire system. There are also associated safety standards to be met for the products in these markets. So the power conversion industry has always been slow in adopting these innovations into the final product.

Model driven development has been adopted as a solution to this problem. In model driven development, a full model of the system is built and simulated before a hardware prototype is generated. This verifies the algorithm functionality and reduces the risk significantly. Moreover, current modeling tools support code generation directly from the model that simplifies and relaxes safety certification criteria. However, the industry has not embraced full model driven development mainly because 1) the performance of the end product and the model varies significantly, and 2) the generated code is not very efficient for the target control processor and requires manual modification before taking it to the product.

In this article, we discuss techniques and approaches that can make the model performance very close to the final product performance to minimize the risk of hardware changes and delays. We also discuss how the code can be generated efficiently from these models to get the product faster to market.

Model Development Guidelines

Model Driven Development

Consider a simplified diagram of a grid-tied residential solar inverter, as shown in Figure 1. The solar radiation on the solar panel generates dc proportional to the intensity of the radiation. The converter converts this dc to ac, which can be used by home appliances and also can be fed to the grid. Current and voltages from various points in the signal chain are sensed by appropriate sensors and will be fed to the control processor in the inverter. The algorithm running on the control processor analyzes these signals and controls the power modules such that the generated current and voltage are of required frequency, magnitude, and phase with the grid. In this case, the solar panel acts as the power source, and the grid and the home appliances act as the sink. In a different power conversion system, the sources and sink would be different, but most of them will fall into the structure shown in Figure 2.

The primary aim of a power conversion system/algorithm designer is to arrive at the right components and algorithms for the block's control processor and converter hardware (shown in Figure 2) and meet the desired performance for all source and load variations. So it is important to clearly know the environment the system is going to operate in while designing the system. For example, while designing a solar inverter for a system (shown in Figure 1), the designer should know the places the inverter is expected to install, variations in intensity of solar radiations, the efficiencies of the solar panel, grid conditions, etc. In a model driven development, the designer first creates the model of the converter, simulates the expected variation, and verifies that the model works as expected. Most often the modeling tools will provide models and library blocks for modeling sources and sinks. For example, Simscape Power Systems™ from Mathworks has models for grids, photovoltaic (PV) panels, and various loads. These can be used to simulate and verify various use cases of the system.

The system performance depends on all the components of the system. In some cases, the designer has the freedom to start the design from scratch and decide on all the components of the system to meet constraints on source and load. In some other cases, part of the system may already be fixed due to reasons outside the control of the designers, and their degree of freedom is limited to few components. In this article, we assume the main aim of the designer is to choose and implement the right control algorithm for an existing topology—but most of the guidelines explained can be applied to a generic case as well.

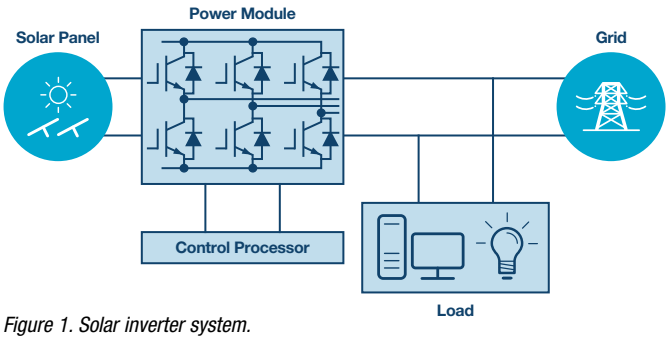

Figure 1. Solar inverter system.

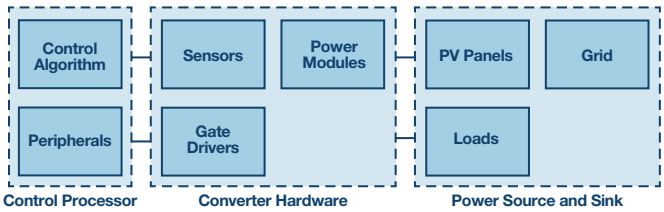

Figure 2. Power conversion components.

Structuring the Model

It is important to structure the model in a modular way with the right interfaces. A well-structured model helps to analyze and adapt the model quickly to various use cases. Modeling tools typically provide various options to group the components at appropriate levels of abstraction and for reuse. For example, Simulink has provisions to create subsystems, library models, or reference models. Consider the power conversion system shown in Figure 2. A top-level view of a Simulink model is given as an example in Figure 3. In this figure, the power converter and control processor are encapsulated into a subsystem labeled as ADIInverter. Solar panel and grid models available with Simscape Power Systems are used to model the source with provisions to configure intensity and temperature. The ADIInverter subsystem in the figure can be further partitioned hierarchically into control processor and control algorithm blocks.

All blocks other than the control algorithm running on the control processor are hardware blocks. So the accuracy of simulation reflecting all the constraints of these components is the most important criteria.

The interfaces of these blocks are analog signals and the most appropriate choice for these are continuous models. The block control algorithm is meant for running on a microcontroller and should only use discrete states and fixed steps. It would be good to keep that as a separate model with different configuration and solver settings and reference that model from the top-level model. This will also be helpful in code generation and processor in loop (PIL) testing of the algorithm, as explained later.

Solver Step Size and Data Types

The speed and accuracy of the simulation is mainly decided by the solver type and step size. A small step size will give more accurate results but will make the simulation run slower. We want to simulate the hardware components with maximum accuracy. A continuous solver with a variable step size should work in most cases. However, when the switching frequencies are high, manual adjustments for the maximum step size may be required. For example, PWM generation at a switching frequency of 100 kHz (as shown in Figure 4a) may become distorted (as shown in Figure 4b) if the step size is large. It is always a good idea to check the output of the fast switching devices to confirm that the step size is sufficient. Since the control algorithm runs on a microcontroller, it should be using a discrete model with a fixed step size. The step size used should be the greatest common divisor (GCD) of the sampling period used in the system. Most often the modeling software chooses this automatically.

The data types used also decide the accuracy of simulation. Simulation with double precision arithmetic will always be more accurate than a simulation with single precision arithmetic. For simulating the hardware blocks, it is recommended to use the highest data type supported by the modeling software. But for the control algorithm, we want to get the performance of the algorithm the same as it runs on the control processor and not more accurate. So we should be using the data type supported by the control processor. For example, if the control processor is an Analog Devices ADSP-CM41x processor, the appropriate data type is single precision floating point, as it comes with a Cortex-M4 processor with a floating-point unit (FPU). If the control processor is a fixed-point processor, such as a Cortex®-M3, the algorithm should be designed and implemented in fixed-point data types. Modeling software may support automatic conversion from a floating-point data type to a fixed point that will help to make the development faster.

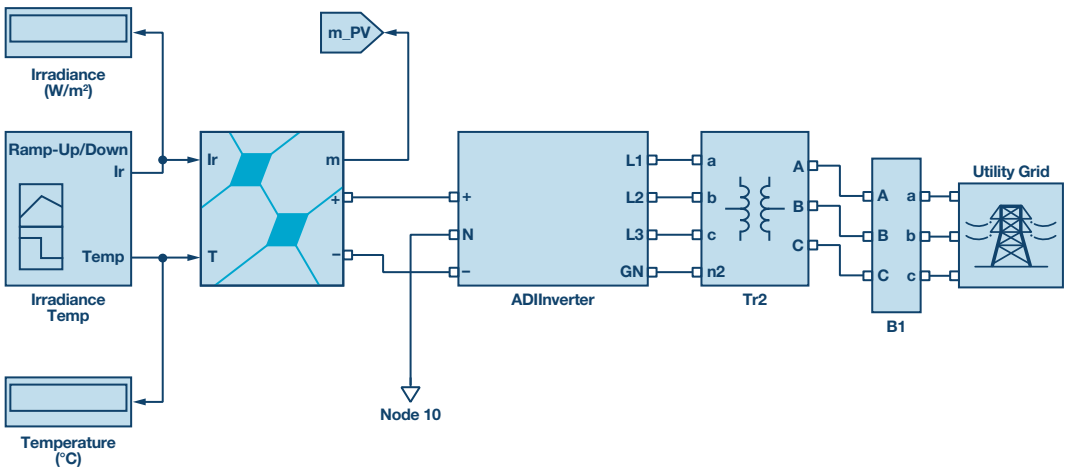

Figure 3. Example Simulink model.

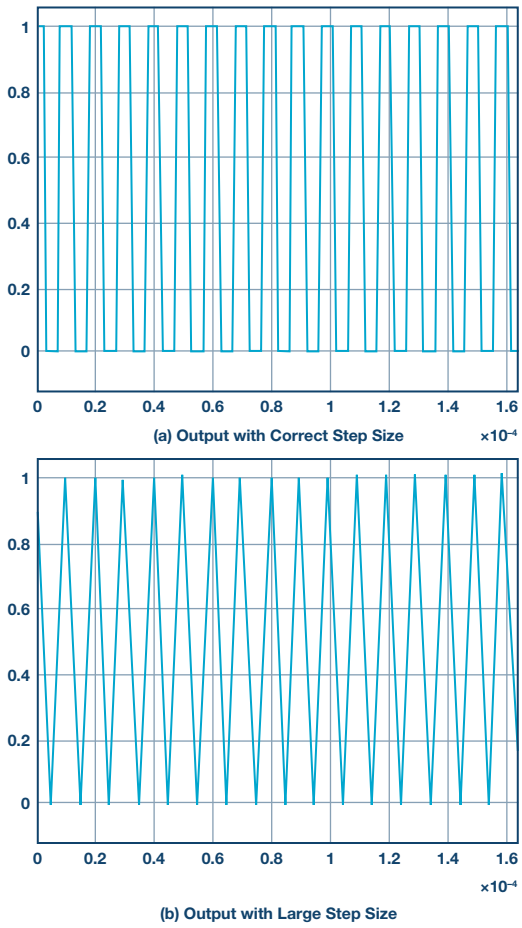

Figure 4. Simulated PWM output at 100 kHz.

Sampling Period and Precision

The current and voltage signals sensed by the sensors at various points in the power conversion signal chain are made available to the algorithm through analog-to-digital converters (ADCs) of the control processors. The sampling rate for the ADC is mainly decided by the switching frequency of power modules and how fast it should be controlled. The sampling frequency has significant impact on the control algorithm performance and dynamics. So simulation should be done by choosing the appropriate sampling rate for the system. The ADCs for control processors accept input only in predefined ranges. The output of the sensors should be normalized in a way that the range of the sensed signal fits exactly in the range of the ADC for the best performance.

The resolution and accuracy of an ADC also varies from one processor to another and this plays a significant role in algorithm performance. High accuracy ADCs help to control the output better, and help to simplify the algorithm and to reduce the control frequency for a specified control criteria. To get an accurate simulation, these characteristics should be reflected in the model. For example, Analog Devices ADSP-CM41x processors come with 16-bit ADCs with more than 13 effective number of bits (ENOB). The ADC block should be modeled such that it takes a continuous signal as the input and output discrete signals at the desired sampling frequency and accuracy. The simulation accuracy can be further improved if the ADC models support the provision to choose the sampling point that is important in some current sampling scenarios.

Code Generation

Verifying the performance of the algorithm by developing the model and running the simulation of the use cases significantly reduces the risk and improves the time to market. However, current modeling tools provide features to do much more before we go to a hardware prototype. All silicon vendors provide evaluation platforms for developing algorithms on their processors. It would give additional confidence on the performance of the algorithm if we can run and verify the algorithm performance on the evaluation hardware, but compilers for the embedded processors normally accept only C/C++ code and is typically time consuming to develop these codes manually during the modeling and verification stage. So in the past, this stage has been pushed to the later stages of development. Fortunately, most of the modeling software now supports the provision to generate codes automatically from the model. The model for the control algorithm can be configured to generate codes with predefined API. The simulation tools also provide a PIL option to run the generated code on the target directly from the modeling environment. In PIL simulation, the input and output of the control algorithm are exchanged with the evaluation board through interfaces such as UART. This option can be used to compare performance of running the algorithm on the target and running the algorithm on the host machine

Typically, the modeling and simulation software provide support for generating C code—targeting a broader range of processors. The hardware vendors will have differentiating features on the processors to speed up execution for the application that the processor is designed for. For example, the ADSP-CM41x processor comes with a math unit accelerator to speed up mathematical operation such as sine, cosine, and square root. It is important to make use of such features to get the best performance out of the processor. Modeling tools provides provision to replace part of the codes with custom codes or an entire algorithm block with a different code. For power conversion algorithms, optimized code can be generated by providing handwritten optimized routines for common algorithm blocks such as direct quadrature zero (DQZ) transforms, phase-locked loop (PLL), etc. Code generation may be configured to use these handwritten routines instead of the default generic routines. Silicon vendors could be providing model libraries to speed up execution of algorithms on their processors. These options may be exploited to generate an optimized code for the control processor.

Apart from the control algorithm code, the control processor also needs codes for configuring the peripherals such as ADC, PWM, etc. and a framework code for maintaining the timings and other functionalities of the system. The modeling tools can be used to generate the code for these as well. However, the framework codes are expected to do much more than run the control algorithm. Developing models for all these associated tasks and generating code from them may not be an efficient approach. In such cases, the framework and peripheral configuration codes may be developed separately with provision to integrate the generated control algorithm codes.

Hardware-in-the-Loop (HIL) Simulation

The simulation of the power modules and the system normally runs on a host PC. Even in PIL simulation, only the control algorithm runs on the target control processor. All other parts of the system are simulated by the modeling software on the host machine. Since this simulation takes so many resources and as much execution time, it is not possible to run these in real time in the software. The system dynamics and performance of ADCs and PWMs are not verified in such testing. HIL simulation hardware overcomes this drawback by using field programmable gate arrays (FPGAs) to simulate the converter components, sources, and sinks. It helps

to run the entire simulation in real time and to see the actual effect of ADC sampling and PWM control. The HIL hardware is typically provided by separate vendors with a provision to interface control processors. It should be noted that HIL platforms won't be able to simulate the detailed switching characteristics of the power modules. These effects should be analyzed separately to minimize the risk while taking it to the final product.

Conclusion

The modeling tools have greatly improved during recent years. In this article, we have discussed various approaches to make the model output very close to the final product output. However, it should be noted that there are some characteristics such as electromagnetic compatibility (EMC) that cannot be verified in a simulation environment. It is important to identify these characteristics and analyze and verify through alternate methods.

The steps explained in the article, except the HIL stage, have been successfully employed in designing and developing control algorithm targeting ADSP-CM41x processors for an inverter with 3-level ANPC topology.

Acknowledgment

We would like to acknowledge the contribution of all our colleagues in the AEG business unit at Analog Devices, Inc.

About the Authors

Bijesh Poyil is an engineering manager at Analog Devices India, Bangalore. He is currently part of the Industrial Systems Group (ISG) within the Automation Energy (AEG) business unit of ADI. His areas of expertise include power conversion, computer vision, and machine learning algorithms targeting embedded systems. He holds an electronics and communications engineering degree from National Institute of Technology Calicut India. He can be reached at *bijesh.poyil@analog.com*.

Martin Murnane is the system architect for the Energy Storage and Conversion Team at Analog Devices in Limerick, Ireland. Previously, Murnane was part of ADI's automotive team. Prior to joining ADI, he worked in several roles involving application development in energy recycling systems, Windows-based application software/database development, and product development using strain gage technology (BMS). He holds an electronic engineering degree and an M.B.A. from the University of Limerick. He can be reached at *martin.murnane@analog.com*.

Analog Devices, Inc.
Worldwide Headquarters

Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
U.S.A.
Tel: 781.329.4700
(800.262.5643, U.S.A. only)
Fax: 781.461.3113

Analog Devices, Inc.
Europe Headquarters

Analog Devices GmbH
Otl-Aicher-Str. 60-64
80807 München
Germany
Tel: 49.89.76903.0
Fax: 49.89.76903.157

Analog Devices, Inc.
Japan Headquarters

Analog Devices, KK
New Pier Takeshiba
South Tower Building
1-16-1 Kaigan, Minato-ku,
Tokyo, 105-6891
Japan
Tel: 813.5402.8200
Fax: 813.5402.1064

Analog Devices, Inc.
Asia Pacific Headquarters

Analog Devices
5F, Sandhill Plaza
2290 Zuchongzhi Road
Zhangjiang Hi-Tech Park
Pudong New District
Shanghai, China 201203
Tel: 86.21.2320.8000
Fax: 86.21.2320.8222