Comparison of Dataflow Architecture and Real-Time Workshop Embedded Coder in Power Electronics System Control Software Design

Jinghong Guo, Stephen H. Edwards*, and Dusan Borojevich

Center for Power Electronics Systems
The Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 USA

*Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 USA

***Abstract*— Dataflow software architecture has been proposed to design reusable and reconfigurable control software for power electronics systems. Mathwork's Real-time Workshop is a commercial software tool supports fast prototyping and automatic program building. In this paper, the proposed dataflow architecture approach will be compared with the Real-time Workshop from software infrastructure, control functionality and flexibility. Several power electronics control applications will be used to compare the performance of code generated from the two approaches.**

I. Introduction

Dataflow architecture has been proposed to address software development and maintenance issues encountered by legacy "main-program-and-subroutine" software style [1] [2] in control software design for power electronics systems. Within the dataflow architecture, the control software is composed with function bodies (ECOs) and data channels. The dataflow approach provides a way to design reusable and reconfigurable control software from a design library of standard software modules. A real-time kernel, named DARK, is also developed to support fast context switching, distributed control and real-time scheduling. The generated control software could be tailored with different scheduling methods and code optimization levels to meet application real-time requirements as well as performance, without affecting the application code.

Mathworks Simulink [3] is a widely used software package for modeling, simulating and analyzing dynamical systems. Real-time Workshop [4] generates optimized, portable and customizable code from Simulink models, which could run on many production targets. These two software packages together provide a software platform for rapid prototyping process and automatic program building.

From software construction point of view, both approaches attempt to reduce engineering effort. Dataflow architecture allows users to design software from functional self-contained library blocks at the C code level. Simulink and Real-Time Workshop save software design effort by providing a graphical modeling environment and automatic C code generation. Though dataflow approach requires a dataflow graph description, which is handwritten so far, this software architecture has the potential to incorporate a graphical design interface to further reduce the software design period and cost.

However, the two approaches differ from each other at the constructed software. A real-time kernel designed for dataflow architectural software provides abundant real-time control features to meet requirements from different kinds of applications. These real-time control features range from static single thread scheduling to preemptive multithread scheduling. The bare board embedded C code generated from Simulink and Real-Time workshop only supports single tasking or preemptive multitasking. The dataflow architectural software is easy to design for distributed control application, while Simulink and Real-Time Workshop do not have special mechanism to facilitate distributed control software design.

In this paper, the differences rooted in the software infrastructure, such as the basic software component division, component interaction mechanisms and system organization, are analyzed. Section II will give an overview of dataflow architecture infrastructure. Section III will introduce Simulink/Real-Time Workshop package and analyze the software structure of the generated embedded C code. After these qualitative comparisons between these two approaches, a 3-phase close loop inverter control is used as examples in section IV to quantitatively show the performance comparison of software generated from these two approaches. It will also be analyzed how those software style features discussed in section II and III relate to the generated software performance.

II. Dataflow Architecture

Software written using a dataflow architecture consists of a collection of independent components running in parallel that communicate via data channels; such a design can be succinctly depicted graphically. A node is a computational component, and an arrow is a buffered data channel. A control algorithm is divided into nodes first. Each concurrently executing node is a self-contained software part

This work was supported primarily by the ERC Program of the National Science Foundation under Award Number EEC-9731677.

with well-defined functionality. Data channels provide the sole mechanism by which nodes can interact and communicate with each other, ensuring lower coupling and greater reusability. Data channels can also be implemented transparently between processors to carry messages between components that are physically distributed. Choosing this component model for embedded control software alleviates many of the negative aspects of the more traditional main-program-and-subroutine organization. More importantly, however, it also opens up the possibility of developing a library of commonly recurring, standardized control software functions encapsulated in reusable data flow components. To best capitalize on the reusability features of the dataflow architecture style, the design of a library of standardized software components will be discussed. Based on the design library, a new system can be rapidly configured from an existing collection of components.

In the dataflow architecture, a control application is composed of ECOs and data channels, and the connections between ECOs and data channels are described in a dataflow graph. Consequently, designing a control application mainly involves constructing such a dataflow graph by selecting ECOs from the design library and connecting them together. Additional user-defined or application-specific ECOs are also easily supported. Annotations on the graph specify ECO startup parameters, ECO priorities, ECO execution policies, data channel property choices, and data channel buffering policies. Fig. 1 shows how a control application built up from individual ECOs and run-time system structure.

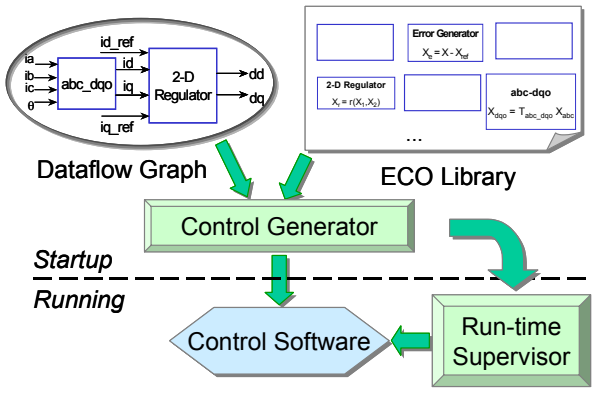

Fig. 1 Dataflow programming.

Unlike applications that are built up from simple subroutines, dataflow applications require support for their unique features, including support for concurrent execution data channel buffering, inter-process synchronization, and interrupt handling. To explore the overhead required to support dataflow applications, the control application is working with a custom designed real-time micro-kernel, Dataflow Architecture Real-time Kernel (DARK) [5], which supports lightweight process management, data channel management, system resource allocation, device driver support, and interrupt handling.

DARK supports a set of OS features, designed as compilation parameter, from which allow users to choose to satisfy their application requirements. For example, DARK supports four ECO scheduling methods: preemptive multi-thread, non-preemptive multi-thread, single thread with dynamic scheduling and single thread with static single thread. For data channel connection, DARK supports two ways: queued data channel, or mailbox data channel. The combination of ECO scheduling and data channel connection methods provides different real-time kernel features to meet specific applications requirements. For applications that all ECOs have the same sample rate (or update rate), the execution sequence can be predefined. Thus, the static scheduled single thread and mailbox features can be applied, which introduces the least runtime overhead.

III. MATHWORKS SIMULINK AND REAL-TIME WORKSHOP SOFTWARE

In recent years, Mathwork's Simulink and Real-Time Workshop software packages have been widely used in industry and academia for modeling and simulating dynamic systems and generating C code for rapid prototyping or embedded control. Simulink is a software package for modeling and simulating dynamic systems. It provides a graphical design environment that allows designers to build models as block diagrams. Real-Time Workshop generates optimized, portable and customizable ANSI C code from Simulink models to create stand-alone implementations of models that operate in real-time and non-real-time in a variety of target environments. The relationships between Mathworks' MATLAB, Simulink and Real-Time Workshop are shown in Fig. 2.

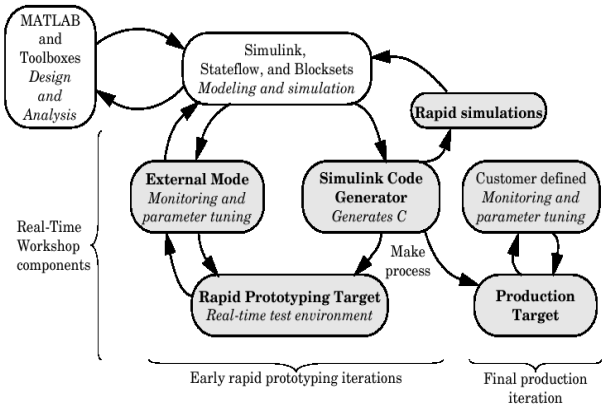

Fig. 2. Relationships between Mathwork's MATLAB, Simulink and Real-time Workshop software packages.

The overall software design procedure using Simulink and Real-Time Workshop package is first draw block diagram

based system model in Simulink, and then build target C code using Real-Time Workshop. Simulink provides standard block libraries and a graphical design environment. It supports hierarchical design, which means a block can be composed of several sub blocks. It also allows users to create their customized library to simplify their specific design procedure. The designed model can be simulated in Simulink to adjust system model structure or model parameters. After several such iterations, when the simulation results match the design specifications, the model can be translated into C code through Real-Time Workshop. Real-Time Workshop allows users to choose from several code formats for different code running targets. To generate C code that can be complied for embedded systems, Embedded Coder format is applied for the design example in this paper.

```
main()
  {
      Initialization (including installation of rt_OneStep as
        an interrupt service routine for a real-time clock)
      Initialize and start timer hardware
      Enable interrupts
      While(not Error)and (time <final time)
        Background task
      EndWhile
      Disable interrupts (Disable rt_OneStep from
        executing)
      Complete any background tasks
      Shutdown
  }
```

(a) Pseudo main program.

```
rt_OneStep()
  {
      Check for interrupt overflow or other error
      Enable "rt_OneStep"(timer)interrupt
      ModelStep—Time step combines output, logging,
        update
  }
```

(b) Pseudo ISR program.

Fig. 3 Pseudo code of Embedded Coder generated C program.

The overall software design procedure using Simulink and Real-Time Workshop package is first draw block diagram based system model in Simulink, and then build target C code using Real-Time Workshop. Simulink provides standard block libraries and a graphical design environment. It supports hierarchical design, which means a block can be composed of several sub blocks. It also allows users to create their customized library to simplify their specific design procedure. The designed model can be simulated in Simulink to adjust system model structure or model parameters. After several such iterations, when the simulation results match the design specifications, the model can be translated into C code through Real-Time Workshop. Real-Time Workshop allows users to choose from several code formats for different code running targets. To generate C code that can be complied for embedded systems, Embedded Coder format is applied for the design example in this paper.

To generate embedded C code, there are some constrains on model blocks. It requires that all blocks in the model are either discrete time block or continuous time block but can be sampled at discrete time. If multiple sample rates are used in a system, it requires that the lowest sample will be chosen as the base rate and other higher sample must be multiple time of the base rate. The purpose of these constraints is for the Embedded Coder to generate C code with some basic real-time scheduling support.

The C code generated from Embedded Coder is in legacy main-program-and-subroutine style, composed of a sample main program, an interrupt service routine (ISR) to implement the control algorithm and data structure descriptions. The pseudo code of the main program and the ISR, rt_OneStep (), is shown in Fig. 3. In the main program, after initialization, the DSP enters an infinite loop to wait for interrupts. The interrupts occur at the base sample specified in the Simulink model. And in the ISR, ModelStep is called to implement control in the current time step. The structure of ModelStep is shown in Fig. 4, where MdlOutput computes the outputs of a model, MdlUpdate updates model states, and MdlDerivatives computes derivates for model states if necessary.

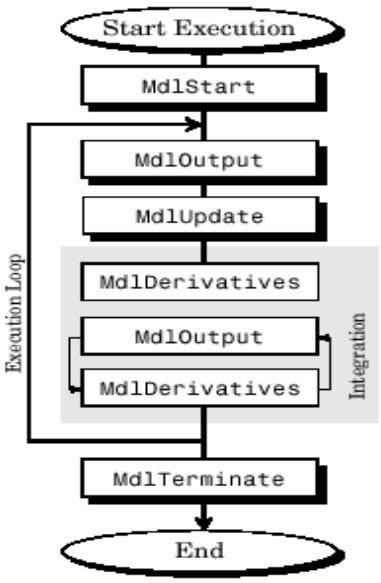

Fig. 4 ModelStep structure.

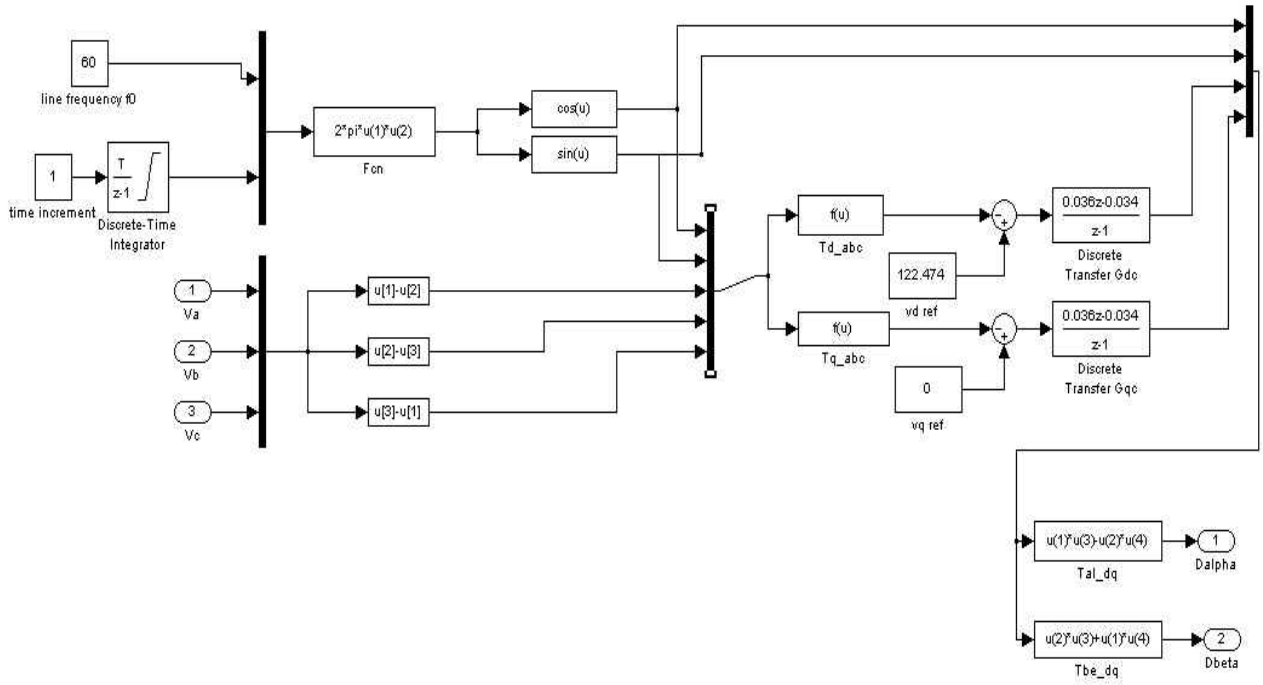

Fig. 5 Simulink Model of voltage close loop control of 3-phase inverter.

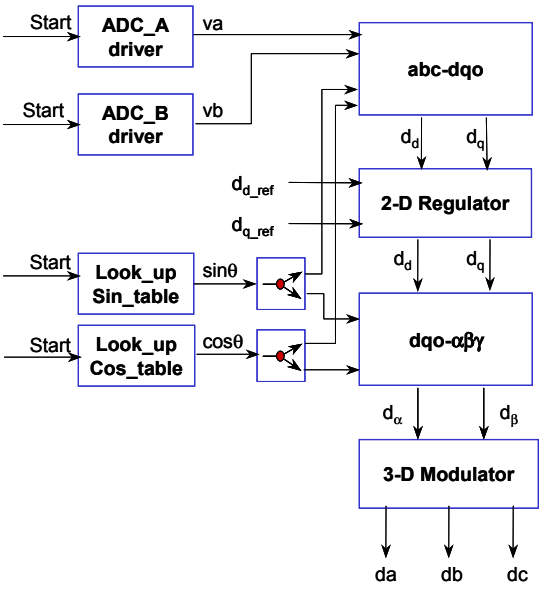

Fig. 6 Dataflow graph of voltage close loop control of 3-phase inverter.

IV. Comparison of Dataflow Approach and Simulink & Real-time Workshop Package

In this paper, a 3-phase voltage source inverter (VSI) with closed voltage loop is chosen as the design example. The specifications of the 3-phase VSI are:

Input: Vdc = 200 V;

Outputs: balanced 3-phase sinusoidal with line-to-line voltage 100V;

Switching frequency: fs = 10kHz;

Output inductance L = 300 uH at each phase;

Output capacitance C = 100 uF at each phase.

The voltage loop is design to have phase margin 35 degree and 10 dB gain margin.

The dp transformation technology is used to simplify the close loop control design and SVM technology is used to implement the modulator. The digital controller is assumed to run in an Analog SHARC DSP (ADSP 21160). Analog Device also provides a software development environment Visual DSP/Visual DSP ++, which support ANSI C.

A. Software design procedure

Fig. 5 shows the Simulink model of the close loop control of the 3-phase VSI, while Fig. 6 shows its dataflow graph. From the high end user point of view, the two software

construction approaches have similarities. For the user of Simulink and Real-Time Workshop package, the main task is using Simulink as a graphical interface to drag and pull blocks from design libraries and then chooses a desired target for the Real-Time Workshop to compile into C code. When using the dataflow approach, the designer only needs to provide a dataflow description file to describe ECOs and their connections.

B. Code structure and performance analysis

Though there is a significant similarity in Fig. 5 and Fig. 6, both composed of function blocks and data connections, the generated code are in different structures because of different block implementation methods and block connection mechanisms.

The generated code from Real-Time Workshop Embedded Coder [6] is in main-program-and-subroutine style as presented in section III. A block in a Simulink model has two tasks during one time step: compute output and update states if necessary. In the generated C code, output computation for individual blocks are combined into MdlOutput, while states updating for individual blocks are combined into MdlUpdate.

Inter-procedure calls are reduced in order to optimize the generated code for real-time execution. The block execution sequences i which pre-scheduled. nerated storage units hanges execution sequ k to the Blocks in software, whil un-time

self-contained functionality. The execution sequence of processes can be statically scheduled, or dynamically scheduled. In dynamically scheduling, Each ECO process can be activated at its own sample rate and the activation depends on its input data channels status. There is no constraint between sample rates of different ECO process. However, context switching between ECO processes and maintaining data channels introduce run-time performance overhead.

Fig. 7 shows the DSP execution cycles during one switching period for Embedded Coder generated C code and dataflow software with different real-time kernel features. The code efficiency of Embedded Coder generated C code is compared to that of dataflow C code with mailbox data channels and static single thread scheduling. From Fig. 7, it can be seen that the Embedded Coder generated C code actually takes more time on computation than any dataflow counterparts. What the Embedded Coder optimized during its code generation is mainly reduced inter-procedure calls.

Since the Embedded Coder generated C code is in main-program-and-subroutine style, it has the drawbacks inherent to its software style. First, the generated code is naturally fit in centralized control structure. Significant extra engineering effort is needed to split the generated code into distributed control system since the blocks in Simulink model are combined. Second, there is possibility that the generated code

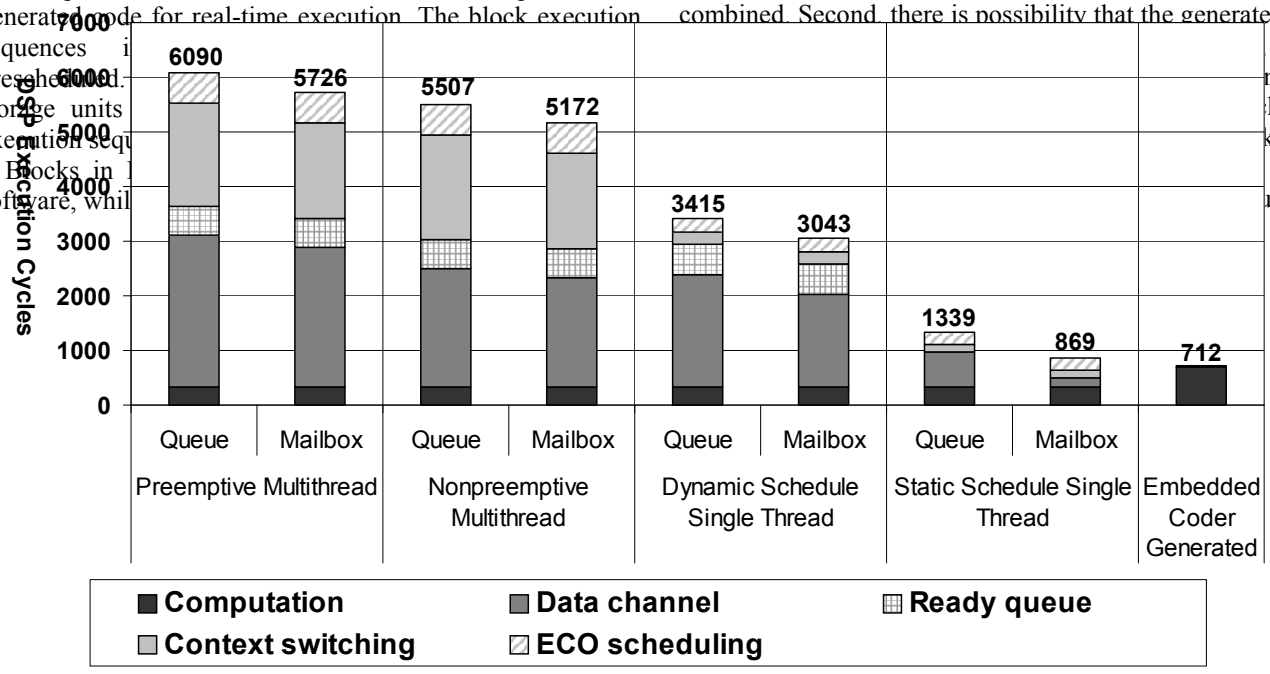

Fig. 7. Code performance comparison.

performance overhead, every ECO process is independent, which makes the software easily run in distributed control system or multi-processor system. The inter-process communications is carried through data channels, which can be designed upon network communication protocols. The ECO processes allocation mechanisms have been designed and the ECO inter-processes communication protocol is under research [7].

V. Conclusion and Future Work

In this paper, two software design approaches to construct control software for power electronics systems, dataflow architecture and MATLAB Simulink and Real-Time Workshop package, are compared at their design methodologies and code infrastructures. A 3-phase close loop control is used as an example to show the performance of software generated from these two approaches. The causes for performance differences are analyzed.

From the analysis, it can be seen that Simulink and Real-Time Workshop package takes the advantage of graphical design and modeling environment and automatic code generation to reduce software design period and cost. The generated embedded C code is featured with reduced inter-procedure calls to optimize run-time performance. On the other hand, dataflow architecture provides more flexible real-time control options, felicitates distributed control design and requires less system redesign efforts.

MATLAB provides ways to design customized target compiler, which can be used by Real-Time Workshop to generate customer C code. This opens the possibility to use Simulink as high-end graphical design interface and let Real-Time Workshop generate dataflow styled C code with optimized inter-procedure calls. The future work thus is to investigate the feasibility of combine advantages of both approaches to construct a better platform to design control software for power electronics systems, featured with intense real-time requirements and distributed control structure.

References

[1] Jinghong Guo, Stephen Edwards, and Dushan Boroyevich. "Desinging reusable, reconfigurable control software for power electronics systems." CPES 2002 Power Electronics Seminar and NSF/Industry Annual Review, April, 2002

[2] Jinghong Guo, Stephen H. Edwards, and Dushan Borojevic. "Elementary control objects: Toward a dataflow architecture for power electronics systems." IEEE . In *Proceedings of the IEEE 33rd Annual Power Electronics Specialists Conference, PESC 02*, 2002, pp. 1705-1710.

[3] MathWorks Simulink web site, http://www.mathworks.com/products/simulink/.

[4] MathWorks Real-Time Workshop web site, http://www.mathworks.com/products/rtw/.

[5] K.Singh and S. H. Edwards, "DARK: Designing A High Performance Micro-kernel for Power Electronics Controllers," *CPES Seminar,* Virginia Tech, Blacksburg, VA, 2002, pp. 362-367.

[6] MathWorks Real-Time Workshop Embedded Coder web site, http://www.mathworks.com/products/rtwembedded.

[7] Parool Mody and Stephen H. Edwards, "Distributed Communication Protocol for Power Electronics Systems," *CPES Seminar,* Virginia Tech, Blacksburg, VA, 2003, in publishing.