Specification and Validation of New Control Algorithms for Electric Drives using SpecC Language

Slim Ben Saoud, Daniel D. Gajski

Technical Report ICS-01-44
July 25, 2001

Center for Embedded Computer Systems
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

Slim Ben Saoud
Fulbright Visitor @ CECS
INSAT-Tunis-TUNISIA
sbensaou@ics.uci.edu
http://www.cecs.uci.edu/~sbensaou

Daniel D. Gajski
CECS
UCI-California-USA
gajski@ics.uci.edu
http://www.cecs.uci.edu/~gajski

Specification and Validation of New Control Algorithms for Electric Drives using SpecC Language

Slim Ben Saoud, Daniel D. Gajski

Technical Report ICS-01-44
July 25, 2001

Center for Embedded Computer Systems
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

Slim Ben Saoud
Fulbright Visitor @ CECS
INSAT-Tunis-TUNISIA
sbensaou@ics.uci.edu
http://www.cecs.uci.edu/~sbensaou

Daniel D. Gajski
CECS
UCI-California-USA
gajski@ics.uci.edu
http://www.cecs.uci.edu/~gajski

Abstract

Today, the shortest time-to-market in the electric drives industries is being a pressing requirement, consequently development time of new algorithms and new control systems and debugging them must be minimized. This requirement can be satisfied only by using a well-defined System-level design methodology and by reducing the migration time between the algorithm development language and the hardware specification language.

In this report, we propose to use the SpecC language for the development (specification and validation) of new control algorithms. This includes the specification of the control systems (algorithms and I/O interfaces) in SpecC and its validation by simulation using a SpecC specification model of the process under control.

This new approach will allow designers to implement easily the retained specification according to the SpecC methodology. Indeed, the same language (SpecC) is used for the study of new control systems and their design and implementation.

We first begin with a brief presentation of the electric drives and of the SpecC language. Then, we discuss the specification models in SpecC of the whole system including the control unit and the process under control. We illustrate this approach by an application example of a DC system. Finally, we present the main advantages of the SpecC language in the development of new control systems.

Contents

1 Introduction 1

2 Electrical Drives 2

3 SpecC Language 2

3.1 Design Consideration for System Level Design Language 2

3.2 Traditional Languages 3

3.3 SpecC Language 3

3.3.1 Structural Hierarchy 3
3.3.2 Behavioral Hierarchy 3
3.3.3 Communication 4
3.3.4 Synchronization 4
3.3.5 Exception Handling 4
3.3.6 Timing 4
3.3.7 Additional Features 5

4 Electrical Drives Specification Using SpecC 5

4.1 Overview 5

4.2 Process Specification 5

4.3 Control Device Specification 6

4.4 Case of D.C. System 7

4.4.1 DC Process 7
4.4.2 Digital Control Device 8
4.4.3 Results 8

5 SpecC Language Advantages 9

6 Conclusion 9

References 10

Appendix: SpecC code for the DC system 11

List of Figures

Figure 1: Electrical drive structure 2
Figure 2: Language Comparison 3
Figure 3: Basic structure of SpecC program 4
Figure 4: Behavioral hierarchy 4
Figure 5: Top-level specification model of electrical drive system 5
Figure 6: Detailed specification model of electric drives systems 6
Figure 7: Specification model of the control device 7
Figure 8: Specification model of a DC system 7
Figure 9: OIE sensor Specifications 7
Figure 10: Control device specification – case of DC system 8
Figure 11: Specification model results 8

Specification and Validation of New Control Algorithms for Electric Drives Using SpecC Language

Slim Ben Saoud, Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

ABSTRACT

Today, the shortest time-to-market in the electric drives industries is being a pressing requirement, consequently development time of new algorithms and new control systems and debugging them must be minimized. This requirement can be satisfied only by using a well-defined System-level design methodology and by reducing the migration time between the algorithm development language and the hardware specification language.

In this report, we propose to use the SpecC language for the development (specification and validation) of new control algorithms. This includes the specification of the control systems (algorithms and I/O interfaces) in SpecC and its validation by simulation using a SpecC specification model of the process under control.
This new approach will allow designers to implement easily the retained specification according to the SpecC methodology. Indeed, the same language (SpecC) is used for the study of new control systems and their design and implementation.

We first begin with a brief presentation of the electric drives and of the SpecC language. Then, we discuss the specification models in SpecC of the whole system including the control unit and the process under control. We illustrate this approach by an application example of a DC system. Finally, we present the main advantages of the SpecC language in the development of new control systems.

1 Introduction

Today, variable speed motor control systems have a wide range of applications from industrial robotics to domestic washing machines, each with a specific set of requirements. Therefore, Motor control is being a vast market (estimated to be $5 billion annually for motors and motor controllers [1]) and the motor control industry is being a strong aggressive sector. Each industry to remain competitive has to answer the customer and governments demands for lower cost, greater reliability, environmental concerns regarding power consumption, emitted radiation and requirements for greater accuracy achievable only by the use of sophisticated control algorithms. Developments are usually done according to two fields:

- Control algorithms: Motor control researchers are increasingly developing new sophisticated control algorithms to increase performances: i.e. Sensorless control, self-adaptive control, Neural network control, Fuzzy logic control… These developments are always characterized by a growth of complexity and needs more performance devices.

- Control device: Motor control circuit designers are increasingly developing new hardware systems with new dedicated processors in order to obtain real-time implementation of these sophisticated control algorithms [2,3]. Some ASM (Application Specific Microprocessor) for motion control applications are developed [4,5,6]. These processors include both high performance core (usually DSP core [7]) and almost all the required peripherals and memory (analog input channels, encoder interface, PWM outputs, serial communication channels, Timers, …). Today, industries are working on developing fully integrated solutions for motor control [1](ASSPs : Application Specific Standard Products) which will allow inherent benefits like lower cost, greater reliability, greater flexibility, lower power consumption and greater precision. These solutions are becoming a key market for IC manufacturers like Analog Devices, Hitachi and Texas Instruments.

The shortest time-to-market is a pressing requirement, consequently development time of new algorithms and new control device and debugging them must be minimized. This requirement can be satisfied only by using a well-defined System-level design methodology and by reducing the migration time between the algorithm development language and the hardware specification language.

In this report, we use the SpecC language for the development and validation of new control algorithms. This will allow designers to implement easily this algorithm according to the SpecC methodology [8]. Indeed, the same language (SpecC) is used for validation of the algorithm and specification of the device.

We first begin with a brief presentation of the electrical drives and of the SpecC language. Then, we present the specification model of the electric drive system in SpecC (control unit and process under control). Finally, we illustrate this approach by an application example of a DC system and we present the main advantages of the SpecC language in the development of new control systems.

2 Electrical Drives

The electrical machine control is performed following the diagram of figure 1. Such a system is composed of two main parts:

- The process to control (CMS: Converter / Motor / Sensors);
- The control unit.

The control unit receives process state information from the sensors and generates control signals to the converter switches.

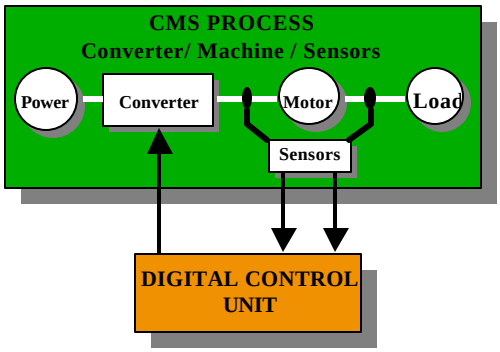


Figure 1: Electrical drive structure

As shown in figure 1, electrical drives have the following basic I/O requirements:

- currents/voltages measurements;
- position/speed measurements;
- pulse width modulation for power converter switching.

Today, modern applications mostly employ A.C. motors: PMSM (permanent magnet synchronous motors), IM (Induction motors), SyncRel (Synchronous reluctance motors). In fact, if the complete life of the drive is considered A.C. drives performance and cost are better than those of D.C. drives since their higher initial cost is quickly balanced by the reduced energy consumption and the lower or absent maintenance.
So in most of systems, two phase currents (generally measured by Hall sensors) are sufficient since the third one can be easily computed. Position signals, needed by speed/position control and field oriented control are measured either by using optical encoders (generally incremental encoders) or resolvers. Pulse width modulation (PWM) is achieved in several ways either hardware or software, using either the single microprocessor or external ASIC.

According to the previous description, all motor control systems require, besides the powerful processor core, a significant array of additional circuits for correct operation, including such functions as:

- Analog to Digital conversion for current or voltage feedback: requires both high accuracy and fast conversion rate: usually 10-12 bit analog to digital converters with a few µs conversion times are needed;
- Pulse width modulation (PWM) blocks for generation of the inverter switching commands: PWM generation represents one of the most interesting part in drive design and the chosen modulation technique affect both performance and system complexity. Simple modulations do not require complex calculation, so they can be easily implemented either by HW and SW without any external component; more complex algorithms often present high computational load, then they require external ASIC or dedicated microprocessors;
- Position/sensor interfaces for higher-performance applications: Encoder outputs are two quadrature square wave signals which frequency is up to some MHz.;
- Serial ports for host communications: Because modern drives cannot neglect communications, high speed serial channels and or specific interfaces (e.g. CAN bus) are often highly desired;
- General-purpose digital input/output ports.

3 SpecC Language [8,9,10]

3.1 Design Consideration for System Level Design Language

A system can be described at any one of several distinct levels of abstraction (logic level, architecture level, conceptual level, …). Each of them serves a particular purpose.
In particular, at the conceptual level, it is possible to describe the system's functionality without any notion of its components. Description of such level can serves as specification of the system for designers to work on. Indeed, increasingly designers need to conceptualize the system using an executable specification language, in order to verify the correctness of the system's intended functionality.
According to the Co-Design methodologies, it is desirable that the specification language be used for all models at

all stages of the design process (homogeneous methodology). Therefore, this methodology does not suffer from simulator interfacing problems or cumbersome translations between languages with different semantics. Instead, one set of tools can be used for all models and synthesis tasks are merely transformations from one program into a more detailed one using the same language. This is also important for reuse, because design models in the library can be used in the system without modification ("plug-and-play"), and a new design can be used directly as a library component.

Such specification and modeling language must be executable, modular and complete. Furthermore, these concepts should be organized orthogonally (independent from each other) so that the language can be minimal. In addition to these requirements, the language should be easy to understand and easy to learn.

3.2 Traditional Languages

Most of traditional languages lack one or more of the requirements discussed in the previous section and therefore cannot be used for system modeling without problems arising. Figure 2 lists examples of current languages and shows which requirements they support and which are missing.

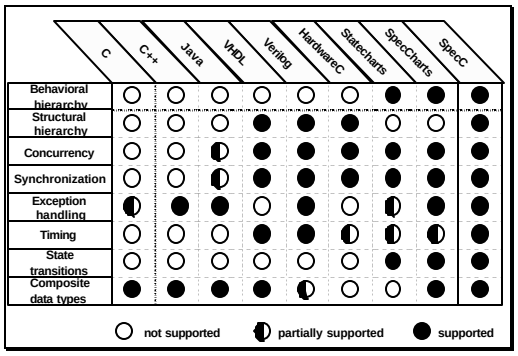


Figure 2: Language Comparison

3.3 SpecC Language

The SpecC language is built on top of the ANSI-C programming language, the defacto standard for software development. It is a true superset, such that every C program is also a SpecC program. C was selected because of its high use in software development and its large library of already existing code.

The SpecC language is based upon the program state machine (PSM) model of computation. The SpecC model clearly separates communication from computation. It consists of a hierarchical network of behaviors and channels and supports "plug-and-play" for easy IP reuse.

In addition, the SpecC language has extensions for hardware design. It supports all the concepts that have been identified as requirements for embedded systems design, such as structural and behavioral hierarchy, concurrency, explicit state transitions, communication, synchronization, exception handling, and timing. Some of these special constructs are described in the next sections.

3.3.1 Structural Hierarchy

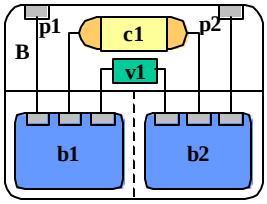
Semantically, the functionality of a system is captured as a hierarchical network of behaviors interconnected by hierarchical channels. Syntactically, a SpecC program consists of a set of ***behavior***, ***channel*** and ***interface*** declarations:

- A ***behavior*** is a class consisting of a set of ports, a set of component instantiations, a set of private variables and functions, and a public *main* function. In order to communicate, a behavior can be connected to other behaviors or channels through its ports. The functionality of a behavior is specified by its functions starting with the *main* function.
- A ***channel*** is a class that encapsulates communication. It consists of a set of variables and functions, called methods, which define a communication protocol.
- An ***interface*** represents a flexible link between behaviors and channels. It consists of declarations of communication methods, which will be defined, in a channel.

For example, the SpecC description in figure 3b specifies the system shown in figure 3-a. The example system specifies a behavior ***B*** consisting of two sub-behaviors ***b1*** and **b2**, which execute in parallel and communicate via integer ***v1*** and channel ***c1***. Thus structural hierarchy is specified by the tree of child behavior instantiations and the interconnection of their ports through variables and channels. Behaviors define functionality and the time of communication, whereas channels define how the communication is performed.

3.3.2 Behavioral Hierarchy

The composition of child behaviors in time is called behavioral hierarchy in SpecC. Child behaviors can either be executed sequentially or concurrently. Sequential execution is specified by standard imperative statements or as a finite state machine with explicit state transitions. Concurrent execution is either parallel or pipelined (Figure 4).
Syntactically, behavioral hierarchy is specified in the *main* function of a composite behavior.



(a)

```
interface I1
{
  bit[63:0] Read(void);
  void Write(bit[63:0]);
};

channel C1 implements I1;

behavior B1(in int, I1, out int);

behavior B(in int p1, out int p2)
{
  int v1;
  C1  c1;
  B1  b1(p1, c1, v1),
      b2(v1, c1, p2);

  void main(void)
  { par { b1.main();
          b2.main();
        }
    }
};
```

(b)

Figure 3: Basic structure of SpecC program

Sequential execution	*FSM execution*

behavior B_seq
{
B b1, b2, b3;
 	
void main(**void**)
{ b1.main();
b2.main();
b3.main();
}
};
	behavior B_fsm
	{
	B b1, b2, b3,
	b4, b5, b6;
	void main(**void**)
	{ **fsm** { b1:{…}
	b2:{…}
	…}
	}
	};

Concurrent execution	*Pipelined execution*

behavior B_par
{
B b1, b2, b3;
 	
void main(**void**)
{ **par**{b1.main();
b2.main();
b3.main();
} }
};
	behavior B_pipe
	{
	B b1, b2, b3;
	
	void main(**void**)
	{**pipe**{b1.main();
	b2.main();
	b3.main();
	} }
	};

Figure 4: Behavioral hierarchy

3.3.3 Communication

The clear separation of communication from computation is one of the strengths of the SpecC language.
Communication can be modeled by use of variables or channels between behaviors (Figure 3). Variables are used to represent a shared memory communication model in SpecC. However, channels are used to represent more complex communication including protocols, which invlove synchronization, timing, buffering, error correction, etc.
The specification of the channel is separated in the interface declaration and the channel definition. The interface defines a set of function prototype declarations without the actual function body. The channel encapsulates the communication media and provides a set of function implementations.

3.3.4 Synchronization

Concurrent behaviors usually need to be synchronized in order to be cooperative. In SpecC, a built-in type ***event*** serves as the basic unit of synchronization. Events can be used only as arguments to *wait* and *notify* statements.
A *wait* statement suspends the current behavior from execution until one of the specified events is notified by another behavior. The *notify* statement triggers all specified events so that all behaviors waiting on one of these events can resume their execution.

3.3.5 Exception Handling

SpecC provides support for two types of exceptions, namely abortion (or trap) and interrupt.
For abortion, the execution of the initial behavior is aborted immediately and will not be resumed. This exception is usually used to model the reset of a system.
In contrast to this, an interrupt exception will resume the execution of the initial behavior.

3.3.6 Timing

In the design of embedded systems, the notion of real time is an important issue.
SpecC differentiates between two types of timing information, ***exact timing*** and ***timing ranges***.
Exact timing is used when the timing is known, as in the execution delay of an already synthesized component.
Timing ranges are used to specify timing constraints at the specification level. SpecC supports timing information in terms of ***timing diagrams*** with minimum and maximum time constraints. Timing ranges are specified as 4-tuples *T={l1; l2; 10; 20}* with the range statement. This specifies that at least *min* but not more than *max* time units spent between labels *l1* and *l2*.

Timing ranges allows avoiding the over-specification problem often obtained with the hardware description languages such as VHDL.

3.3.7 Additional Features

In addition to the concepts explained so far in this chapter, the SpecC language supports other constructs that are necessary for system-level design. It provides explicit support for Boolean (*bool*) and bitvector (*bit[:]*) types, in addition to all types provided by ANSI-C. SpecC also provides constructs for binary import of precompiled SpecC code and support of persistent annotation for objects in the language.

It is very important that the advantage of SpecC lies in its orthogonal constructs, which implements orthogonal concepts. The SpecC language covers the complete set of system concepts with a minimal set of constructs. It is therefore easy to learn and easy to understand.

4 Electrical Drives Specification Using SpecC

In the traditional way, developers of new control algorithms validate their studies by simulation using standard language (C, C++, MATLAB, …). The control algorithm is tested using mathematical models of the process written in the same program, with the same language. Therefore, designers of the control devices have to translate this specification from the original language (standard language) to the co-design methodology language. This introduces a time/schedule delay.

In this work, we propose to use the SpecC language to specify the whole motor drive system that includes control algorithms, I/O modules and Process to control. In contrast to other language, the SpecC allows to specify the system functionality in a clear and precise manner and the obtained specification, used for simulation, will serves, without the need for tedious rewrites, as the input to the synthesis and exploration stages in the SpecC design methodology.

In this section we present the general case of electrical drives then we describe the case of a DC system as an example. This approach can be generalized to all of Power Electronics and Electrical drives system.

4.1 Overview

Figure 5 shows the top level of the electric drive specification in SpecC, consisting of process and control device sub-behaviors running in parallel. The process sub-behavior is specified using mathematical models of the electric device in order to validate the control algorithm. It receives control signals and generates information about the electric process state. Control device on the other hand, captures this information and generates control signals according to the used algorithm and to the user orders.

The highest behavior in the hierarchy (*Process_CTL*) is the "Main" behavior similar to the main()-function in each C program. This main-behavior contains the testbench including the process specification (*Process*) and the control system under Test (*CTL*).

In the following sections we describe these modules in more details.

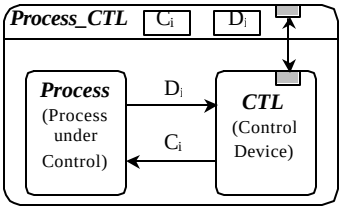


Figure 5: Top-level specification model of electrical drive system

4.2 Process Specification

The electric drive is composed of three module categories: Converter, Motor/Load, and Sensors. On the physical process these modules operate in parallel. Then in our specification we reproduce this structure by using three parallel behaviors (Figure 6). Each of these behaviors will be decomposed on child-behaviors according to the following considerations:

- In the motor/load model, we usually distinguish two modes: electric mode and mechanical mode. So, when digitized, the model is composed of two equation systems: one for the electric mode and one for the mechanical mode. Then the motor behavior is decomposed of two child-behaviors (*Electric* behavior and *Mechanic* behavior).
- On the physical process, we usually use several different sensors. Each of them is specified in a child-behavior ($sensor_1$, $sensor_2$, …).
- According to the fact that these modules don't have the same temporal constraints and rates, we propose to add to each behavior a clock (represented by another sub-behavior Clk_x) that generates its corresponding computing step for the simulation. These clocks must be defined according to the user specification.

Usually, we use the same clock for the simulation of electrical device, and different clocks for different sensors.

The final specification model of the process under control is then represented by figure 6.

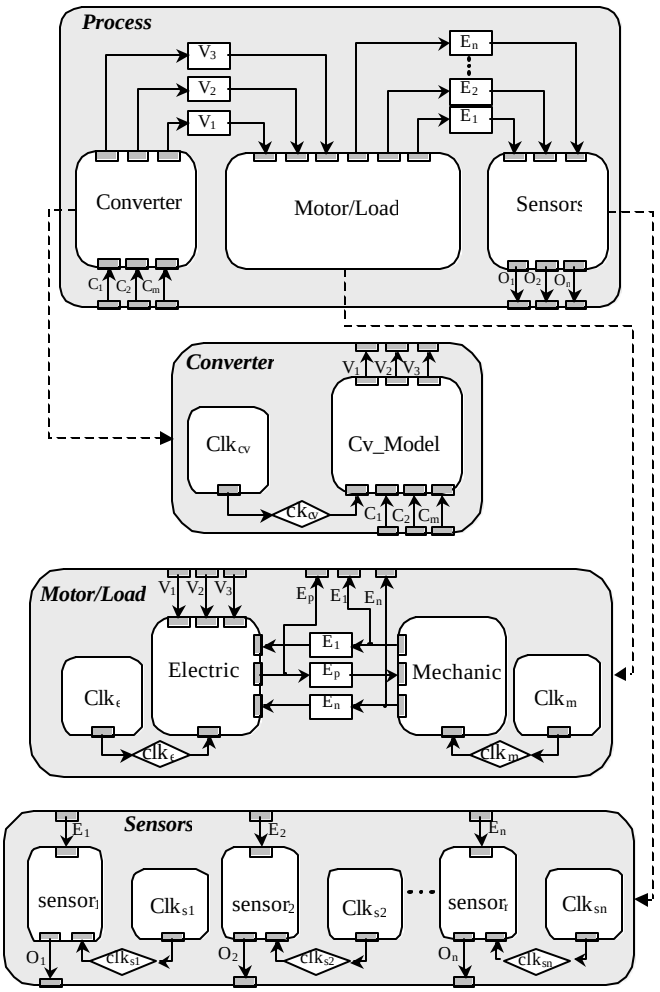


Figure 6: Detailed specification model of electric drives systems

This specification model present several advantages:

- The SpecC specification describes the process in a clear, modular and precise manner. Available parallelism and behavior dependencies are explicitly shown. This greatly eases the understanding and therefore supports quick exploration of different design alternatives at the system level in the first place.
- According to this principle, we have a modular structure that is easy for use and configure with a library of components. User has just to make his choice of modules from this library according to his application.
- Manipulation of time is very useful since we define for each module or sub-module (for the sensors) a clock behavior that specifies the computing time of corresponding behaviors.
- Using the sensor modules we obtain a good representation of the physical system and then a good validation of the control device. So, some phenomena can be studied like resolution of the converter and the encoder sensors, influence of delays and noises,…

4.3 Control Device Specification

Besides the algorithm implementation, all motor control systems require a significant array of additional circuits for correct operation, including such functions as:

- Analog to digital conversion for capture of electric magnitudes (current and voltage);
- Position sensor interfaces for capture of mechanical magnitudes (position and speed);
- Pulse width modulation (PWM) blocks for the generation of the converter switching commands;
- Serial ports for host communication;
- General-purpose digital input/output ports;
- Watchdog timer and event timers, …required for real time embedded control systems.

According to the user application some or all of these blocks are integrated in the control device. So in our specification we reserve for each of them a sub-behavior that can be decomposed of some child-behavior… These sub-behaviors will be specified inside two principle behaviors, which are the *ACQ* behavior for the information capture and the *PWM* behavior for the generation of control signals.

On the other hand, in the electric drive, we usually distinguish two control loops: an outer motion loop and an inner current loop. The motion loop handles the mechanical load and maintains rotary position and velocity. It has typically bandwidths of the order of 20 to 30 Hz with sample rates of 500Hz to 3 kHz. The current loop handles the dynamics of the motor electrical system and controls torque production. It has typically bandwidths of the order of 1 to 2 kHz with sample rates of up to 20 kHz.
Then, a behavior *CTL_Alg* including two sub-behaviors one for the motion control (*M_Alg*) and one for the current control (*C_Alg*) can specify the control algorithm.

Each of these behaviors is associated to a clock generator behavior (Clk_x).

The Figure 7 represents the specification model of the control device.

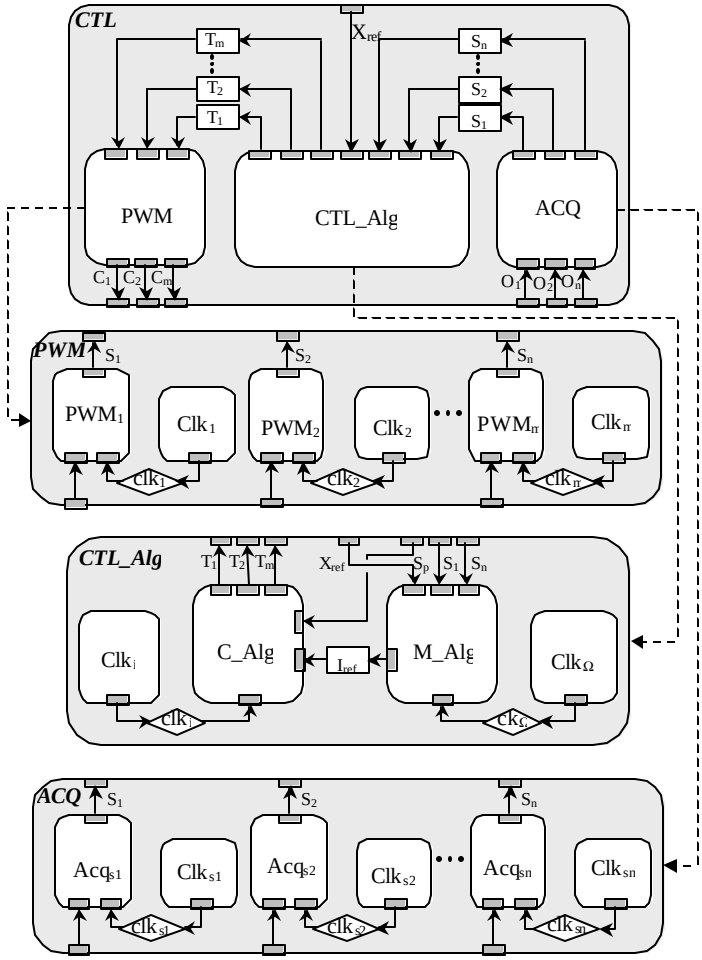


Figure 7: Specification model of the control device

4.4 Case of D.C. System

To validate this new approach used in the development (specification and validation) of new control algorithms for electric drives, and based on the SpecC language, we describe in this section, the case of a DC system. This process is composed of a DC motor fed by a four-quadrant chopper and associated to current and speed sensors.

4.4.1 DC Process

The DC system is composed of a DC motor, a four-quadrant chopper, a Hall sensor for the current capture and an Optical incremental encoder for the speed.
The SpecC model of this system is represented by figure 8, where each module has its own clock generator.

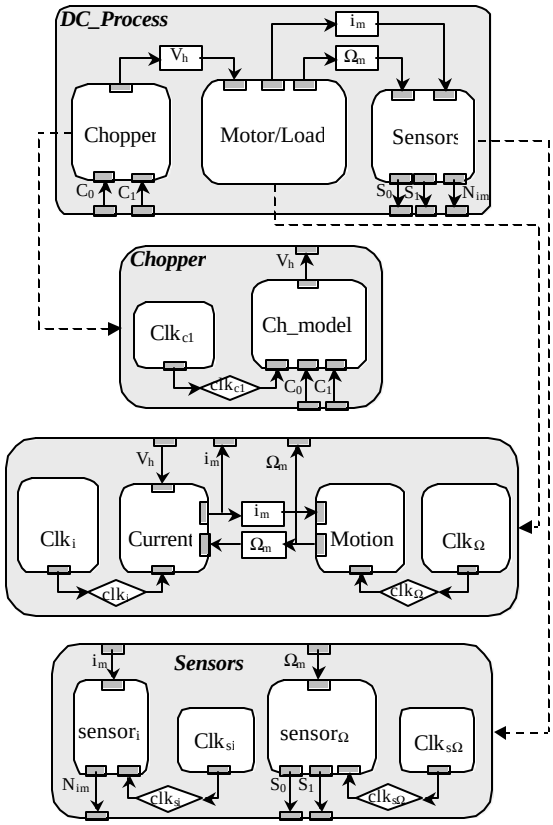


Figure 8: Specification model of a DC system

The Hall sensor output is a voltage magnitude that represents the current value. This output voltage depends on the Hall sensor characteristics and the current value. In our case this dependency is represented in the behavior *Sensor*$_i$ by the equation1: Vout=im*5/15 volts. A more sophisticated model (including influence of noises, temperature, wear,…) can be specified in this behavior in order to reproduce more precisely the Hall sensor output.
The optical incremental encoder generates two quadrature square wave signals (S0 and S1) with the same frequency (proportional to the motor frequency) as represented by figure 9.

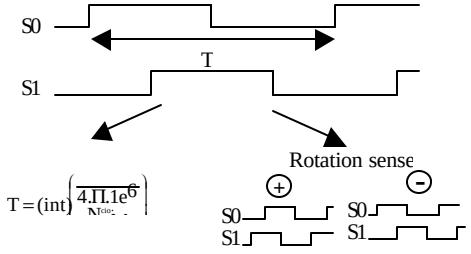


Figure 9: OIE sensor Specifications

These signals are reproduced by the behavior *Sensor*$_W$. More complex models can be added to this behavior

specification in order to reproduce more precisely the optical incremental encoder outputs.

4.4.2 Digital Control Device

This device includes three main parallel behaviors that describe respectively the PWM modules, the control algorithms and the different used sensors. Each of them is distributed in different child-behaviors according to figure 10.

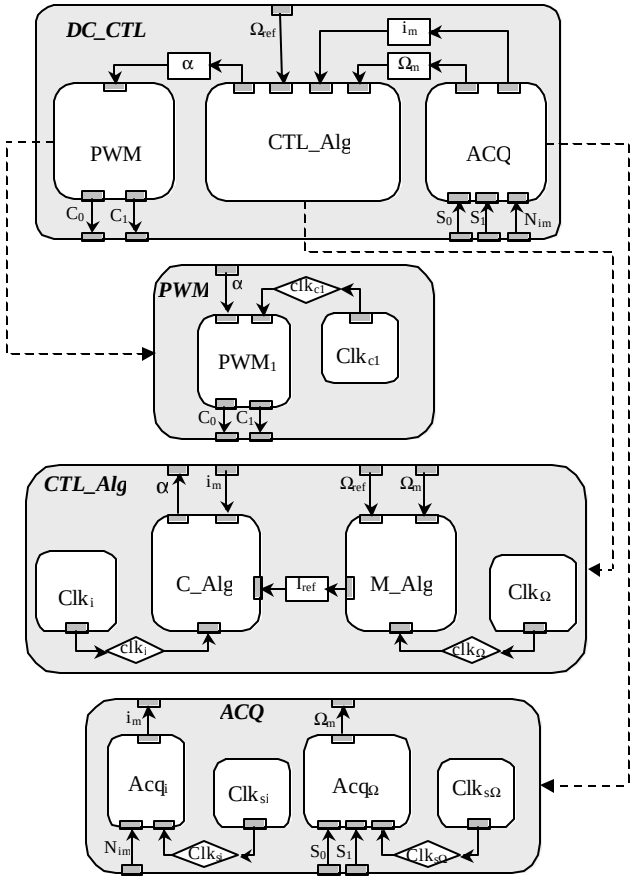


Figure 10: Control device specification – case of DC system

The Control Algorithm behavior is composed of two sub-behaviors: one for the motion control and one for the current control. The current loop period is 284µs while the motion loop period is 20ms.

The current is measured by Hall sensor and the information is obtained in a voltage form. So, the control device has to include an ADC module. The current capture requires both high accuracy and fast conversion rate: usually a 10-12 bits ADC with a few µs conversion times are needed. In order to have a precise specification of this module, the description of the ADC is done using two parameters: resolution (number of bits) and delay (time necessary for conversion). More specifications can be added to this behavior in order to represent more precisely the ADC component functioning.
The current control module uses the average value of the current (at the scale of the current control period). So, we add to the current capture behavior a module to compute this average value.

For the speed control, the rotation direction and the speed absolute value are obtained from the optical incremental encoder signals S0 and S1.

To control the four-quadrant chopper, at least, two control signals are required. These signals (C0 and C1) are generated by the PWM behavior according to the current control order. This behavior can be defined with three configurable parameters: clock, period, pulse width.

4.4.3 Results

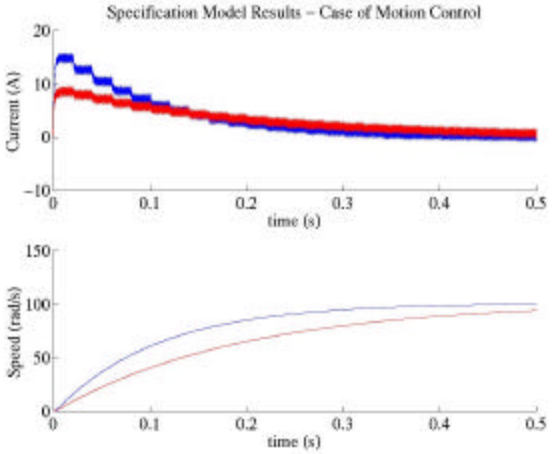


Figure 11: Specification model results

Using the specification model of the DC system, user can test the control algorithm and the influence of its different parameters (regulators parameters, control periods,…). He can also test the impact of the I/O circuits like the resolution of the CAN and it's delay and the precision of the speed capture module according to the used clock…

Figure 11 shows results obtained with two different speed control parameters.

More sophisticated models can be used in order to represent others complex phenomena like the influence of the temperature, of the wear, of the noise,… The user will be able to add these specification in very easy way and he will be then able to validate his control device under specific conditions…

The use of SpecC language in the development of new control systems present several advantages that are developed in the following section.

5 SpecC Language Advantages

During this project, we use the SpecC language for the specification and validation of new control systems. According to this work we note several main advantages of this language that can be described as follows:

- The obtained specification model is executable and validation by simulation is done easily. Indeed, results storage, restitution and manipulation for verification can be performed clearly in the *testbench* module.
- The SpecC language offers modularity in form of structural and behavioral hierarchy, allowing the hierarchical decomposition of the specified system. The electric drives systems are then described in a clear, modular and precise manner. Available parallelism, behaviors dependencies and temporal constraints are explicitly shown. This greatly eases the understanding and the modification of the specification model.
- The SpecC language supports the inclusion of precompiled design libraries into the specification description. This simplifies the handling of component libraries and also allows a speedy compilation. The modular specification model of electric drives can be obtained easily by the association of specific component libraries to the SpecC language. Therefore, user has just to make her/his choice of modules from these libraries according to his application.
- The SpecC language has extensions for hardware design. It supports all the required concepts for embedded systems design, such as structural and behavioral hierarchy, concurrency, explicit state transitions, communication, synchronization, exception handling and timing.
  Figure 2 compares some traditional language against a set of these requirements. The SpecC language, shown in the last column of this table, has been specifically designed to support all the required concepts. Moreover, SpecC precisely covers these requirements in an orthogonal manner.
- The obtained specification model will serves, without the need for tedious rewrites, as the input to the synthesis and exploration stages in the SpecC design methodology for the final control device design.

On the other hand, the SpecC language is built on top of the ANSI-C programming language, the de-facto standard for software development. It is a true superset, such that every C program is also a SpecC program. This implies two main additional advantages of the SpecC languages:

- The major works in the control of Power Electronics and Electromechanical Systems are done in C language. Then, the large amount of already existing codes and libraries can be re-used easily with SpecC language. Conversion of these C programs to SpecC programs can be done rapidly and we estimate the manually conversion time to be between few minutes and one hour according to the complexity of the program.
- The ANSI-C is well known by the control systems developers. Then, the adaptation of these developers to the SpecC language is very easy and can be done in a short time. A 4 hours tutorial seems to be sufficient for C language programmers to be able to develop specification models in SpecC.

6 Conclusion

In this report, we described the use of the SpecC language to the specification and validation of new control systems for power electronics and electric drives.

The SpecC specification of electrical drives is captured in a natural, clear and precise manner showing explicitly available parallelism and behavior hierarchy and dependencies. This greatly eases the understanding and the use of this specification model in order to validate control devices.

The modular structure used in this specification allows facilitating the validation of new control algorithms. Indeed, the SpecC can integrate a library of electrical modules including different motor models, different converters and different sensors. So the user can select among these modules those corresponding to his/her application process. He will be able, also to choice the control device modules like PWM generators, Speed acquisition module,… and configure them according his control constraints.
Only the modules corresponding to the chosen configuration will be selected in the library of modules. This allows to have optimized running programs at the simulation stage.

The SpecC language is built on top of the ANSI-C programming language. This allows the re-use of the large amount of already existing C codes and libraries and facilitates the adaptation of the control developers to the SpecC language.

The main advantage of the use of SpecC language is that the obtained specification, used for simulation, will

serves, without the need for tedious rewrites, as the input to the synthesis and exploration stages in the SpecC design methodology for the final control device design. This will reduce significantly the time-to-market by minimizing largely communication among designers and customers.

Acknowledgments

The authors would like to thank the Fulbright Scholar Program for supporting this project. We would also like to thank Andreas Gerstlauer and Rainer Doemer for their interesting comments and ideas.

References

[1] Analog Devices, Products and Datasheets, Whitepapers, "ASSPs for Motion Control Applications Use Embedded Digital Signal Processing Technology", http://www.analog.com/publications/whitepapers/products/motion2.html , 2001

[2] D. Krakauer, "Single chip DSP Motor Control Systems Catching on in Home Appliances", Appliance magazine, October 2000

[3] C. Cecati, "Microprocessors for Power Electronics and Electrical Drives Applications", http://sant.bradley.edu/ienews/99_3/drCECATI/paper.htm , IES Newsletter, vol. 46, no. 3, September 1999

[4] J.F. Moynihan, P. Kettle, A. Murray, "High Performance Control of AC servomotors using an Integrated DSP", Intelligent Motion, May 1998 Proceedings

[5] A. Murray, P. Kettle, "Towards a single chip DSP based motor control solution", Proceedings PCIM - Intelligent Motion, May 1996, Nurnberg, Germany, pp. 315-326

[6] F. Moynihan, "High-Performance Motion Control", PCIM-Europe N1/2, 1999

[7] Texas Insruments, Digital Signal Processing Solution for AC Induction Motor Application Note BPRA043, 1996

[8] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, "SpecC: Specification Language and Methodology", Kluwer Academic Publishers, 2000

[9] A. Gerstlauer, R. Dömer, Junyu Peng, D. Gajski, "System Design: A Practical Guide with SpecC", Kluwer Academic Publishers, 2001

[10] D. Gajski, J. Zhu, R Dömer, "The SpecC+ Language", University of California, Irvine, Technical Report ICS-TR-97-15, February 1997

Appendix: SpecC code for the DC system