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

Slim Ben Saoud
L.E.C.A.P.-E.P.T./ I.N.S.A.T.
B.P. 676, 1080 Tunis Cedex, TUNISIA
SlimBenSaoud@fulbrightweb.org

Daniel D. Gajski and Rainer Dömer
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

Abstract-- **In the traditional way, developers of new control algorithms validate their studies by simulation using standard language (C, C++, MATLAB, …). 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 languages, 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.

Keywords-- **Electric Drives, Control, Specification, Validation**

I. INTRODUCTION

Today, 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… These demands are achievable only by the use of sophisticated control algorithms. Therefore, developments are usually done according to two fields:

- Control algorithms research: 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 [2,3,4]… These developments are always characterized by a growth of complexity and needs more performance devices.
- Control device development: 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 [5,6]. Some ASM (Application Specific Microprocessor) for motion control applications are developed [7,8,9]. These processors include both high performance core (usually DSP core [10]) 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 paper, 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 [11]. 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 present the main advantages of the SpecC language in the development of new control systems.

II. 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.

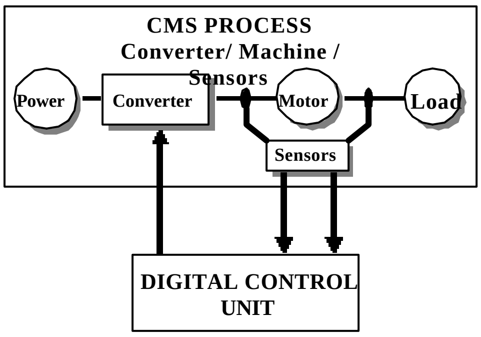


Figure 1: Electrical drive structure

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. 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 ext ernal 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/speed 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.

III. SPECC LANGUAGE

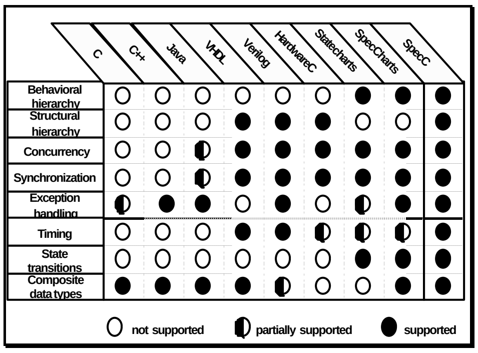
A. *Design Consideration for System Level Design Language*

According to the Co-Design methodologies [12,13,14], 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.

B. *SpecC Language*

Most of traditional languages lack one or more of the System-Level design language requirements 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 [15].



	C	C++	Java	VHDL	Verilog	HardwareC	Statecharts	SpecCharts	SpecC
Behavioral hierarchy	○	○	○	○	○	○	●	●	●
Structural hierarchy	○	○	○	●	●	●	○	○	●
Concurrency	○	○	◐	●	●	●	●	●	●
Synchronization	○	○	◐	●	●	●	●	●	●
Exception handling	◐	●	●	○	●	○	◐	●	●
Timing	○	○	○	●	●	◐	◐	◐	●
State transitions	○	○	○	○	○	○	●	●	●
Composite data types	●	●	●	●	◐	○	○	●	●

○ not supported ◐ partially supported ● supported

Figure 2: Language Comparison

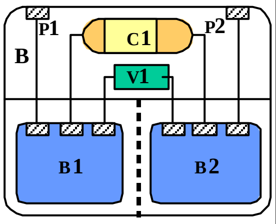
The SpecC language is built on top of the ANSI-C programming language, the defacto standard for software development. It is a true superset, such that every C program is also a SpecC program. C was selected because of its high use in software development and its large library of already existing code.

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.

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 3-b 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.



(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

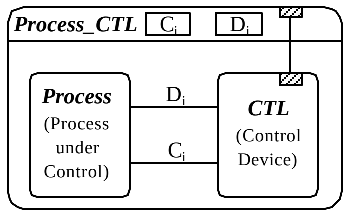
In addition, the SpecC language has extensions for hardware design. It supports all the concepts that have been identified as requirements for embedded systems design, such as structural and behavioral hierarchy, concurrency, explicit state transitions, communication, synchronization, exception handling, and timing (figure 2).

IV. ELECTRICAL DRIVES SPECIFICATION USING SPECC

In this section we present the specification model of electrical drives. This approach can be generalized to all of other industrial systems.

Figure 4 shows the top level of the electric drive specification in SpecC, consisting of process and control device sub-behaviors running in parallel. 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.



Ci: Control Signals / Di: State process information

Figure 4: Top-level specification model of electrical drive system

A. 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 5). 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 5.

B. 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.

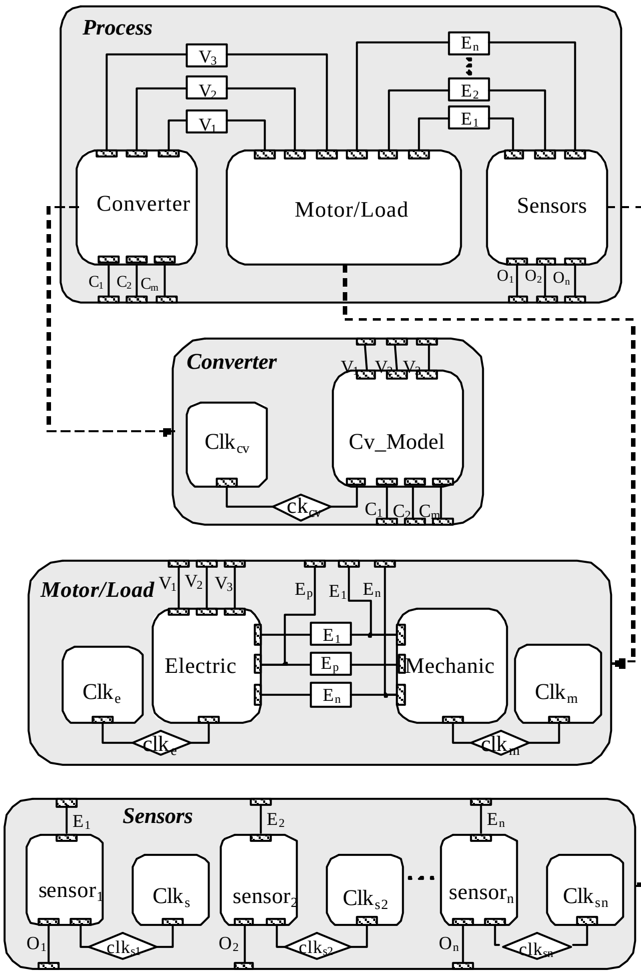


Figure 5: Detailed specification model of electric drives systems

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 6 represents the specification model of the control device.

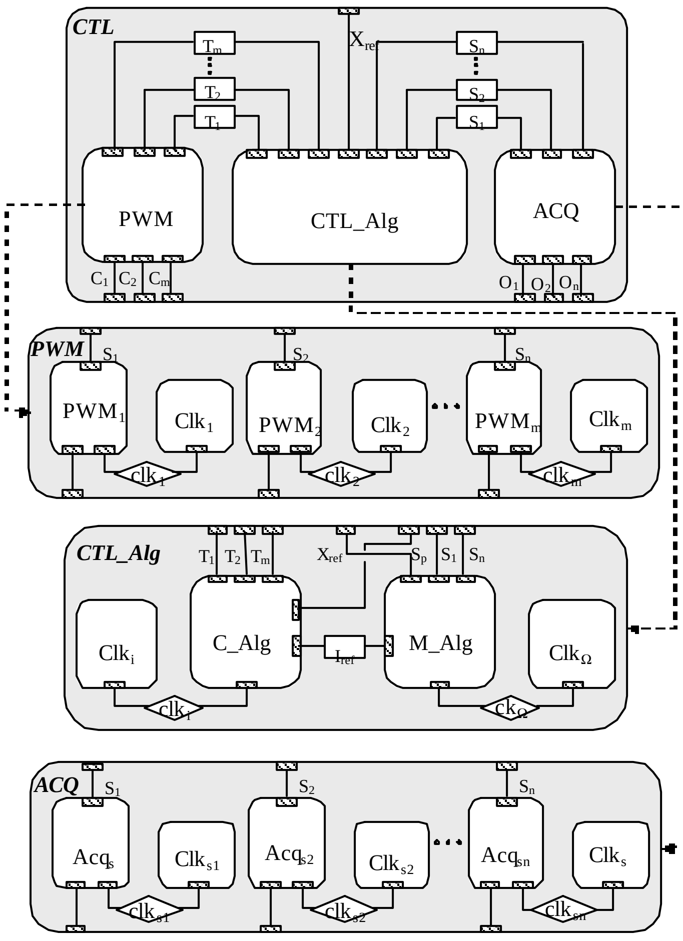

CTL
T_m
T_2
T_1
X_{ref}
S_n
S_2
S_1
PWM
CTL_Alg
ACQ
C_1 C_2 C_m
O_1 O_2 O_n
PWM
S_1 S_2 S_n
PWM_1 Clk_1 PWM_2 Clk_2 PWM_m Clk_m
clk_1 clk_2 clk_m
CTL_Alg
T_1 T_2 T_m X_{ref} S_p S_1 S_n
Clk_i C_Alg I_{ref} M_Alg Clk_Ω
clk_i ck_Ω
ACQ
S_1 S_2 S_n
Acq_s Clk_{s1} Acq_{s2} Clk_{s2} Acq_{sn} Clk_s
clk_{s1} clk_{s2} clk_{sn}

Figure 6: Specification model of the control device

To illustrate this new approach of digital control development, two applications have been developed:

- The first one concerning the speed control of a dc[1] machine. 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 used control system is based on a cascade regulation algorithm, in which the speed control strategy includes an inner current loop.
- The second one concerning the speed control of an induction machine. The process is composed of an induction motor fed by an inverter and the used control system is based on an indirect field oriented control algorithm.

For each of these applications, the obtained specification model was validated by simulation. The obtained results are similar to those obtained by C language description.
Otherwise, the number of code lines and the simulation duration are equivalent to those of C language programs.
However the SpecC language presents several advantages that are developed in the following section.

V. 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.
  Furthermore, 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 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.
  As shown in Figure 2, the SpecC language has been specifically designed to support all the required

[1] Direct-current

concepts. Moreover, SpecC precisely covers these requirements in an orthogonal manner.

So, 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.

VI. Conclusion

In this paper, we introduce a new specification language (SpecC) to the development 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 main advantage of the use of SpecC language is that the obtained specification model, 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.

In our future works, we intend to apply the SpecC methodology to the development and design of new sophisticated control systems.

Acknowledgments

The authors would like to thank the Fulbright Scholar Program for supporting this project. We would also like to thank Andreas Gerstlauer for his 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] K. Ohnishi, N. Matsui and Y. Hori, “Estimation, identification and sensorless control in motion control system”, Proc. IEEE, vol. 82, August 1994

[3] M. El-sharkawi, A. El-samahy amd M. El-sayed, “ High performance drive of dc brushless motors using neural network”, IEEE Trans. On energy conversion, vol. 9, june 1994

[4] J. Jang and C. Sun, “ Neuro-fuzzy modeling and control”, Proc.IEEE, vol. 83, March 1995

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

[6] 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

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

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

[9] F. Moynihan, “High-Performance Motion Control”, PCIM-Europe N1/2, 1999

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

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

[12] D. D. Gajski, F. Vahid, S. Narayan, J. Gong, “Specification and Design of Embedded Systems”, Prentice Hall, 1994

[13] R.K. Gupta, “Co-Synthesis of Hardware and Software for Digital Embedded Systems”, Kluwer Academic Publishers, 1995

[14] R. Niemann, “Hardware/Software Co-Design for Data Flow Dominated Embedded Systems”, Kluwer Academic Publishers, 1998

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