Seamless approach for the design of control systems for Power Electronics and Electric Drives

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 Andreas Gerstlauer
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 paper, we propose to apply the SpecC methodology to the the design of control systems for power electronics and electric drives. We first begin with an executable specification model of the control device. Then, we describe the different steps and transformations used to convert this model to a communication model, which can be then transformed to an implementation model ready for manufacturing.

Keywords: **Co-design, Embedded Systems, Control, Electric process.**

I. 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. This is achievable only by the use of sophisticated control systems [2,3,4].
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.
The goal of this work is to introduce a new seamless approach for the development of complex control systems. This approach will be discussed using an application of the DC motor control. A generalization of this study to any other control system can be done easily using the same steps discussed in the following sections.
The control device will be described in four models, which represent four different levels of abstraction in our design approach [5,6]. All these models are executable and validated by simulation.
The rest of the paper is organized as follows: We first begin with a brief presentation of the used approach. Then we describe an executable specification model of the control system and we discuss the refinement of this model into architecture model, which accurately reflects the system architecture. Based on the retained architecture model, communication protocols between the system components are defined and communication model is developed.

II. DESIGN APPROACH

Managing the complexity at higher levels of abstraction is not possible without having a very well-defined system-level design flow [7,8,9]. Therefore, in this project we propose a new seamless approach, which is a set of models and transformations on the models (Figure 1). The models written in programming language (SpecC language) are executables descriptions of the same system at different levels of abstraction in the design process. The transformations are a series of well-defined steps through which the initial specification is gradually mapped onto a detailed implementation description ready for manufacturing.
This new approach is based on 4 well-defined models, namely a specification model, an architecture model, a communication model, and finally, an implementation model. After each design step, the design model is statically analyzed to estimate certain quality metrics such as performance, cost, and power consumption. Analysis and estimation results are

reported to the user and back annotated into the model for simulation and further synthesis.

In this paper we focus on the synthesis flow which contains the steps of specification, architecture exploration and communication synthesis. Implementation can then be done easily using standard tools.

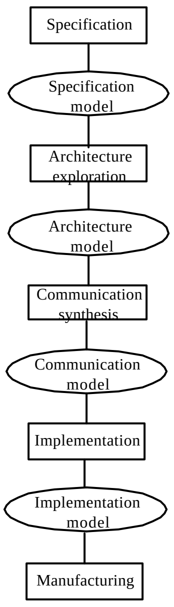

Figure 1: Design Approach

III. SPECIFICATION

The system design process starts with the specification model written by the user to specify the desired system functionality. This model is a purely functional, abstract model that is free of any implementation details.

Figure 2 shows the specification model of the DC control system in SpecC language. The used control algorithm (*CTL_Alg*) is composed of two control loops: an outer motion loop (*M_Alg*) and an inner current loop (*C_Alg*). Each of them is specified in a separate sub-behavior and associated to a clock-behavior that generated the synchronization event to activate the corresponding control loop at the predefined periodic step.

The I/O modules necessary for the control device functioning are specified in two behaviors: the *PWM* behavior represents the PWM[1] module functioning while the *ACQ* behavior represents the information acquisition modules.

The *PWM* behavior generates two complementary signals C_0 and C_1 with the same frequency as the current control module clock and according to the pulse width value α (for C_0) obtained by the current control behavior.

The current acquisition behavior (*Acq$_i$*) captures the current value (N_{im} obtained from the used ADC[2] component) and computes its average value over the current control period (i_m). While the speed acquisition behavior (*Acq$_W$*) computes the speed value (Ω_m) from the two signals S_0 and S_1 generated by the optical incremental encoder (sensor used on the process under control).

As shown on figure 2, the SpecC specification describes the control device functionality in a clear and precise manner.

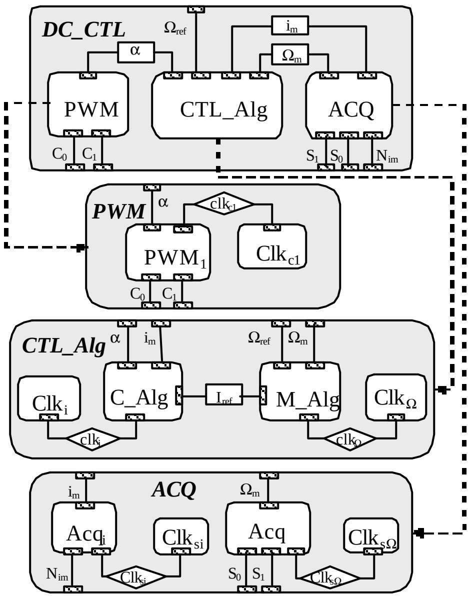

Figure 2: Specification model of the control device

IV. ARCHITECTURE EXPLORATION

Architecture exploration is the first part of the system synthesis process that develops system architecture from the specification model. The purpose of architecture exploration is to map the computational parts of the specification onto the components of system architecture. The steps involved in this process are allocation, partitioning and scheduling. Through this process, the specification model is gradually refined into the architecture model.

A. Allocation

The first task of the architectural exploration process is the allocation of a system target architecture consisting of a set of components and their connectivity. Allocation selects the number and types of processing elements (PEs), memories and busses in the architecture, and it defines the way PEs and memories are connected over the system busses. Components

[1] Pulse Width Modulation

[2] Analog to Digital Converter

and protocols are taken out of a library and can range from full-custom designs to fixed IPs[3].

After an architecture has been allocated, the first step in implementing the specification on the given architecture is to map the specification model behaviors onto the architecture's processing elements.

For the control device application, usually the I/O modules are done by hardware modules (ADC, Timers, …) while the control algorithm is implemented in a standard processor. However, sometimes, this solution is not adequate for sophisticated algorithm running in real time. Than, usually we remove a part of the control algorithm from the processor and we implemented it by hardware. This part is usually the current loop because it represents the most severe time constraints. In this work we propose to study this case of application and to demonstrate the advantage of our methodology in the design of real-time control systems.

Therefore, the retained model is composed of a hardware component for each I/O module, an ASIC for the current control module and a processor core (DSP56600 core) for both of the speed control module and the interface with the user (Figure3).

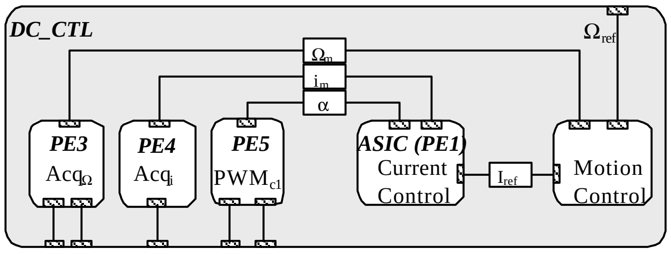

Figure 3: Architecture models after behavior partitioning

Formerly local variables used for communication between behaviors mapped to different components now become global, system-level variables (α, i_m, Ω_m and I_{ref}).

B. Variable Partitioning

After behavior partitioning, communication between behaviors mapped to different PEs is performed via global, shared variables. Global variables have to be assigned to local memory in the PEs or to a dedicated shared memory component. In the refined model after variable partitioning, global variables are replaced with abstract channels and code is inserted into the behaviors to communicate variable values over those channels.

The number of exchanged variable is very limited. So local copies of global variables are added to the correspondent PEs (Figure 4). Updated data values are communicated between these PEs through 4 different abstract channels: $C_{\Omega m}$, C_{im}, C_α and C_{Iref}.

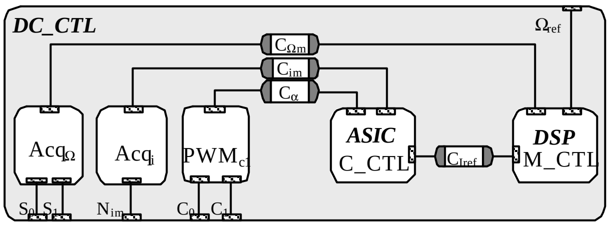

Figure 4: Architecture model after variable partitioning

According to this architecture, no synchronization is needed between the I/O modules and the control modules since the exchange is done only in one side and through memory blocks included in these I/O hardware modules.

On the other hand, synchronization between the ASIC and the DSP is done by interruption. Indeed, at each Tc period, the ASIC interrupts the DSP and begins the transfer of Iref within the channel C_{Iref}.

C. Channel Partitioning & scheduling

Channel partitioning is the process of mapping and grouping the abstract, global communication channels between components onto the busses of the target architecture. In the refined model, additional top-level channels are used to represent system busses. Then channel partitioning is reflected by hierarchically grouping and encapsulating the abstract, global channels under the top-level bus channels.

According to the previous specification, we distinguish at least two main possibilities of channel partitioning:

- As shown on figure 5, channels can be mapped onto two buses:
  - one bus between the ASIC and PE4/PE5: this bus will be managed by the ASIC;
  - one bus between the DSP and PE1/PE3: this bus will be managed by the DSP.

Using this solution buses are managed easily since each bus will have only one master that initiate each transfer. However it presents a main inconvenient, in fact the number of pin in the coprocessor will be important since it used two different busses.

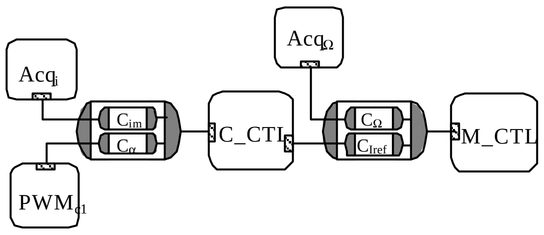

Figure 5: Solution 1 of Channel partitioning

[3] Intellectual Property

- For that reason, we propose a second possibility on which only one bus is used as a common bus to all components as shown on figure 6. However, this solution includes a major difficulty of the bus management, in fact it will have two masters. So a management protocol should be added to resolve conflicts when these two masters tries to use the bus at the same time.

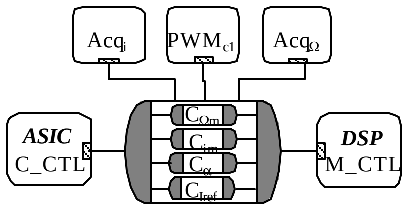

Figure 6: Solution 2 of Channel partitioning

In order to simplify the communication process, we choice to use another variables partition with only one bus and one master. So, we introduce some modification to our architecture model as shown on figure 7.

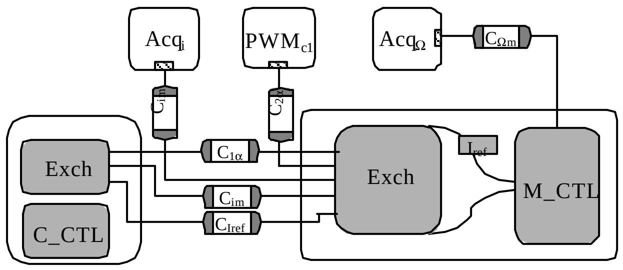

Figure 7: Modification of variable partitioning

Synchronization for the transfer is done by interruption. At the beginning of each new Tc period, the ASIC interrupts the DSP and both start the exchange process. Inside this process, The ASIC sends α and waits for i_m and i_{ref} while the DSP (the master) begins by reading the α value, then it writes this data to PE5, and reads i_m value from PE4 and finally it sends i_m and i_{ref} values to the ASIC. Acquisition of Ω_m value is done by the master at the beginning of each Tm period..
The final architecture model using one bus is represented by figure 8.

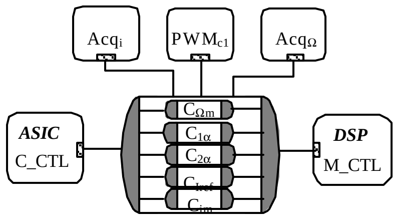

Figure 8: Architecture model after channel partitioning

V. Communication Synthesis

The purpose of communication synthesis is to refine the abstract communication in the architecture model into an actual implementation over the wires of the system busses. This requires insertion of communication protocols for the busses, synthesis of protocol transducers to translate between incompatible protocols, and inlining of protocols into hardware and software.

A. Protocol Insertion

During the protocol insertion, a description of the protocol is taken out of the protocol library in the form of a protocol channel and inserted into the corresponding virtual system bus channel (Figure 9).
The abstract communication primitives provided of the bus channel are rewritten into an implementation using the primitives provided by the protocol layer. The outer application layer of the bus channel implements the required semantics over the actual bus protocol. This includes tasks like synchronization, arbitration, bus addressing, data slicing, and so on.
All the abstract bus channels in the model are replaced with their equivalent hierarchical combinations of protocol and application layers that implements the abstract communication of each bus over the actual protocol for that bus.

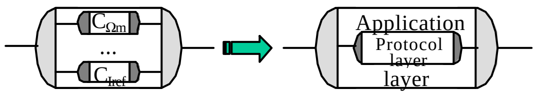

Figure 9: Protocol insertion principle

In this example, after protocol insertion, the processor is the central component and the master of the system bus. The software on the processor initiates all data transfers on the processor bus from and to the hardware components. However, these exchanges are initiated either by an external clock or by the hardware component that send an event (IT) at each T_c period to the processor by triggering its interrupt in order to execute the exchanges process (Figure 10).

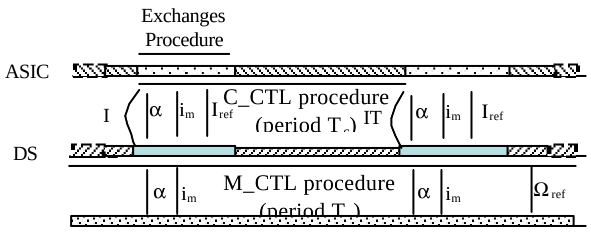

Figure 10: HW/SW Synchronization diagrams

The DSP 56600 protocol [10] is employed for ASIC and DSP while another simple memory protocol is used for memory blocks. We suppose that ASIC and DSP use the same protocol and that timing constraints are compatible between these two protocols. Otherwise we have to insert transducer.

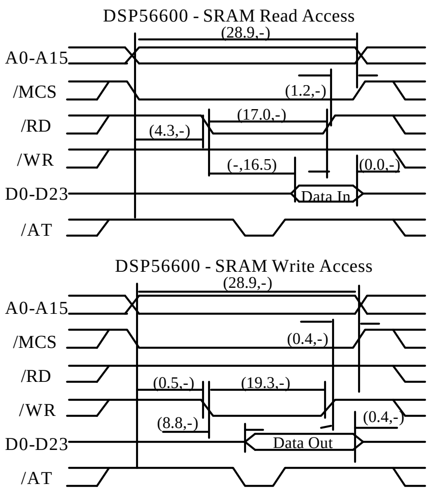

Figure 11: Protocols of the DSP56600 external bus

B. *Protocol Inlining*

Protocol inlining is the process of inlining the channel functionality into the connected components and exposing the actual wires of the busses. The communication code is moved into the components where it is implemented in software or hardware. On the hardware side, FSMDs that implement the communication and bus protocol functionality are synthesized. On the software side, bus drivers and interrupt handlers that perform the communication using the processor's I/O instructions are generated or customized.

For the ASIC, communication primitives are inlined into the exchanges sub-behavior. Therefore, exchanges SFSMD model is created and inserted into the ASIC SFSMD model (Figure 12).

The exchanges hardware module synchronizes with the DSP by raising the processor's interrupt line IRQC in its first state S1 until a transfer with the address of the custom hardware is recognized. Then the RD control signal is sampled until a falling edge has been detected that signals the beginning of a bus write cycle. Communication continues at the same manner for two write cycles.

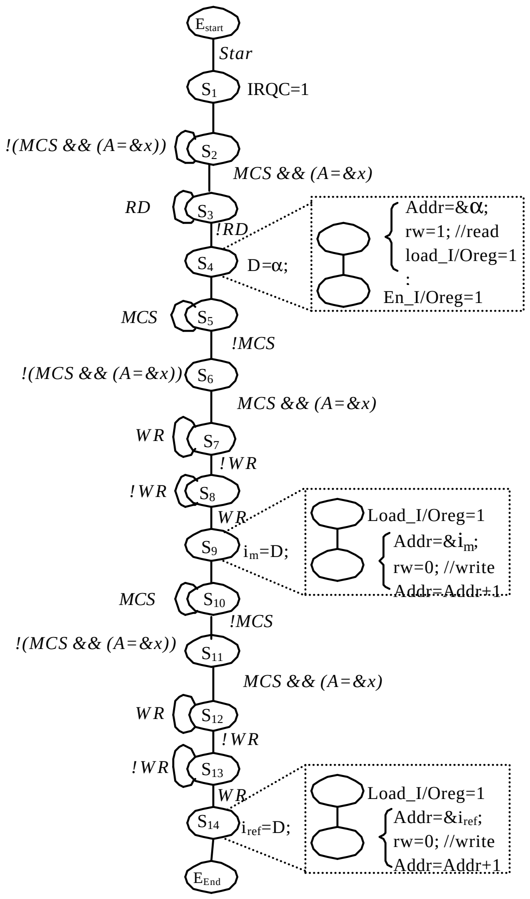

Figure 12: ASIC Communication SFSMDs

The communication model obtained after protocol inlining is shown in Figure 13.

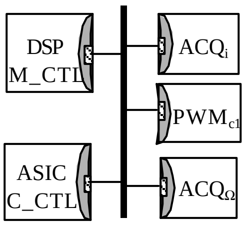

Figure 13: Communication model after protocol inlining

We note that connections are done between memories (inside the I/O models) and PE1/PE2 according to the figure 14.

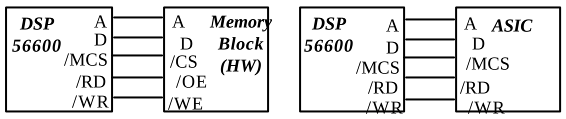

Figure 14: Components interconnections

In our application, only one side of the register control is done by the master: for example for the Rα (register that stored α), it is controlled by PE2 only in writing operation, so the /WE signal is connected to the master /WR signal while the /OE is active controlled in local by the PWM hardware.
In the FSMD of the I/O module, the register will be controlled by the FSMD-controller only on one way (partially). For example in the case of the PWM module this register is only controlled for read by the FSMD-controller while it is controlled for write for the Acq_i, by its own FSMD-controller. The master (PE2) when transferring data with the DSP does the other control side.

We note that in this application and for necessity we used at the same time high and low levels design.

The obtained communication model is validated and is ready for use directly to generate the implementation model. The leaf behaviors of the design model will be fed into different tools in order to obtain their implementation [11].

VI. CONCLUSION

In this paper we apply the SpecC system-level design methodology to the design of control systems of Power Electronics and Electric drives processes. We presented the study of a DC motor drive with a control system based on a DSP for the motion control, an ASIC for the current control and three additional hardware components for I/O processes. This study can be easily generalized to any other process control.
We have shown the various steps that gradually refines the initial specification down to an actual communication model ready for implementation and manufacturing.
The well-defined nature of the presented approach models and transformations helps focusing design efforts on central issues, provides the basis for design automation tools, and enables application of formal methods.
In future works, we will develop some libraries specific to the control system design and apply the SpecC methodology to the design of new sophisticated algorithms.

ACKNOWLEDGMENTS

The authors would like to thank the Fulbright Scholar Program for supporting this project. We would also like to thank Rainer Dömer 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”, 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”, IES Newsletter, vol. 46, no. 3, September 1999

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

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

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

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

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

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

[10] Motorola, Inc., Semiconductor Products Sector, DSP Division, DSP 56600 16-bit Digital Signal Processor Family Manual, DSP56600FM/AD, 1996

[11] R. Dömer, D. Gajski, J. Zhu, "Specification and Design of Embedded Systems", *it+ti magazine*, Oldenbourg Verlag, Munich, Germany, No. 3, June 1998.