Modeling, Synthesis, and Validation of Heterogeneous Biomedical Embedded Systems

(Invited Paper)

Gunar Schirner
Department of Electrical and Computer Engineering
College of Engineering
Northeastern University
Boston, Massachusetts 02115
Email: schirner@ece.neu.edu

***Abstract*—The increasing performance and availability of embedded systems increases their attractiveness for biomedical applications. With advances in sensor processing and classification algorithms, real-time decision support in patient monitoring becomes feasible. However, the gap between algorithm design and their embedded realization is growing.**

This paper overviews an approach for development of biomedical devices at an abstract algorithm level with automatic generation of an embedded implementation. Based on a case study of a Brain Computer Interface (BCI), this paper demonstrates capturing, modeling and synthesis of such applications.

I. Introduction

With advances in algorithm design, biomedical devices gain significantly in importance in a wide range of applications: from real-time analysis physiologic data to alert medical staff in event of anomalies to augmentation systems enhancing human interaction. For the efficient development of algorithms for these real-time implementations, a close interaction with the underlying system is required. However, algorithm development currently uses abstract tools and methodologies (Matlab, Labview) that are primarily focused on the algorithm design.

In this paper, we evaluate a methodology for system-level design of biomedical devices, which allows abstract development of algorithms and offers a the same time an automatic generation of virtual platforms for performance evaluation and back-end synthesis. The methodology aims to support rapid algorithm development with straight-forward realization in an embedded system. The methodology also bridges the gap between algorithm developer and embedded system architect. It allows the algorithm developer to naturally express the algorithm and offers a concrete specification for the embedded system architect. In result of the enabled collaboration, the system-level design approach will lead to shorter development cycles and a faster time-to-market.

The remainder of the paper is organized as follows. After briefly outlining the related work in Section II, Section III overviews the proposed methodology. Section IV describes the case study application of brain-computer interfacing. Finally, Section V concludes the paper.

II. Related Work

Combating the disparity between the moderate increase in design efficiency and the rapid increase in chip design complexity as expressed in the productivity gap [1], [2], [3] has long been the aim design automation. With the unparalleled increase in software complexity the challenges have even increased further, creating a wider gap as expressed in the system design gap [4]. In order to increase the efficiency in designing complex embedded systems designers move to Electronic System Level (ESL) utilizing higher levels of abstraction.

Instead of low-level implementation, System-Level Design [5] captures system functionality at higher levels of abstraction. This results in fewer, coarser grained modules that are expressed irrespective of their later mapping to hardware or software. System Level Design Languages (SLDLs), such as SystemC [6] and SpecC [7], have been developed to efficiently capture high-level designs, providing parallel execution, modularity, hierarchy, and separating at the same time the orthogonal aspects of computation and communication. These languages are based on common programming languages C++ and C, which eases the acceptance for software developers.

After successfully capturing a design at the high level, the next major task is the design space exploration to explore suitable platform candidates considering trade-offs between hardware and software implementations. ESL Synthesis Methodologies (see overview in [8]) are emerging that systematically refine a system specification toward a platform implementation. Examples include Dedalus [9], SCE [10], SystemCoDesigner [11], Metropolis [12], Koski [13], and PeaCE/HOPES [14]. These methodologies use the concept of virtual platforms (VPs) for a functional performance simulation of real platforms (consisting of processors, hardware accelerators, memories, and bus hierarchies).

In this paper, we harness the power of the ESL environment System-on-Chip (SCE) [10] for exploring design alternatives and generating an embedded implementation for biomedical applications focusing on body brain interface systems.

III. Approach

A. Overview

The overall design flow is highlighted in Fig. 1. At its input, the user specifies the application as a set of algorithms, basically in form of a set of parallel C behaviors (see Section III-B). In addition, system architecture decisions are entered describing the target platfrom as a composition of processing elements (such as general purpose processors, digital signal processors, custom hardware components, or IP blocks), communication hierarchy (busses, hierarchical busses connected by routers, or network-on-chip topology) and memory elements. It also contains the mapping of the behaviors of algorithm onto the processing elements on the platform, their scheduling parameters and policies, as well as the communication parameters.

In the usage scenario of biomedical applications, we anticipate that the responsibilities will be split between the algorithm designers for developing the biomedical application at an abstract level, and the embedded platform designers who will define the platform composition depending on the application demands.

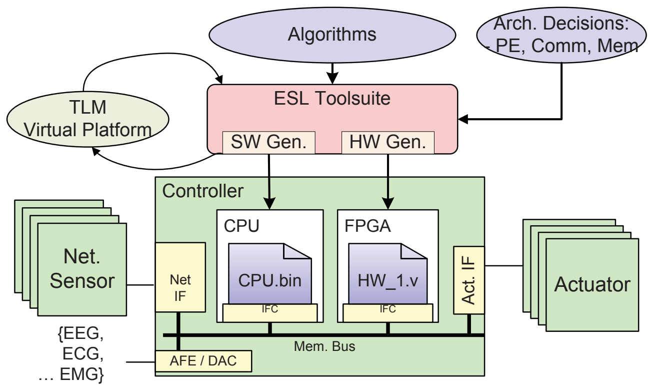

Fig. 1. Design flow overview.

The ESL Toolsuite SCE[10] then generates a Virtual Platform (VP) in form of a Transaction-Level Model (TLM). The VP (see Section III-C) offers rapid feedback about the performance influence of the architecture decisions. It employs abstract models of the processors involved, including a representation of dynamic dynamic scheduling, bus models for communication modeling as well as the memory and IO components. For the application designers, the VP will give initial results about the computational complexity and target-specific timing complexity of the captured algorithms. The embedded platform designer on the other hand, will use the application-specific VP for identifying suitable custom platforms meeting the application requirements.

Once the application development is finalized and a suitable platform (performance, power, reliability) is found, the back-end generation of the ESL tool suite will be used for generating target binaries for each processor in the system (Section III-D) as well as synthesizing custom hardware components through High-Level Synthesis (HLS) and instantiation of Intellectual Property (IP) components. The bottom half of Fig. 1 shows an anticipated target architecture of a processor assisted by custom accelerators in an FPGA.

B. Application Capture

Fig. 2 illustrates basic features of the input specification. Following the semantic definitions [7], [15], the algorithm's computation is captured in behaviors, and communication is expressed in channels or through shared variables. Each behavior contains C code capturing the computation. Behaviors are composed hierarchically defining containment as well as the behavioral hierarchy. Behavioral hierarchy expresses the execution semantics allowing sequential, parallel, pipelined and state machine execution.

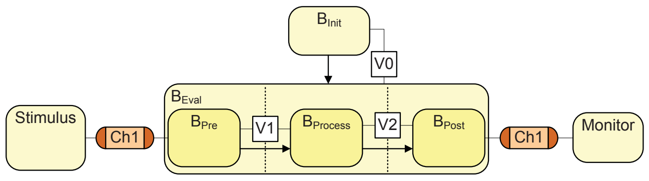

Fig. 2. Algorithm specification

In Fig. 2 the initialization B_{init} is sequentially followed by the main processing B_{Eval}. A variable $V0$ is used for communication between these behaviors. With the sequential execution, no additional synchronization is required. B_{Eval} contains three behaviors (B_{Pre}, $B_{Process}$ and B_{Post}) arranged in a pipeline fashion with pipelined variables ($V1$ and $V2$) connecting the pipeline stages. Two additional behaviors ($Stimulus$, $Monitor$) run in parallel communicating through channels. Channels offer standardized communication and synchronization with primitives for buffered and buffered communication, as well as various synchronization alternatives.

C. Virtual Platform

Given the application definition and the architecture decisions, the SCE generates virtual platforms at increasing level of detail [16]. Starting from an application model reflecting target-specific application execution, via a task model that shows dynamic scheduling effects, via a firmware model capturing basic application and communication medium specific drivers, to the complete transaction-level processor model. The latter includes a complete synchronization chain (interrupt chain), hardware interrupt scheduling and a transaction level bus interface.

Fig. 3 (source [16]) shows an example of a generated processor TLM. The processor TLM is connected via transaction-level models of the communication system to IP components and custom hardware units. Simulating both computation and communication, the TLM-based virtual platform offers early insight into the envisioned target platform: from scheduling decisions, processor load factors, to detailed communication load analysis. Being generated out of the application specification, the VP offers both functional as well as performance validation.

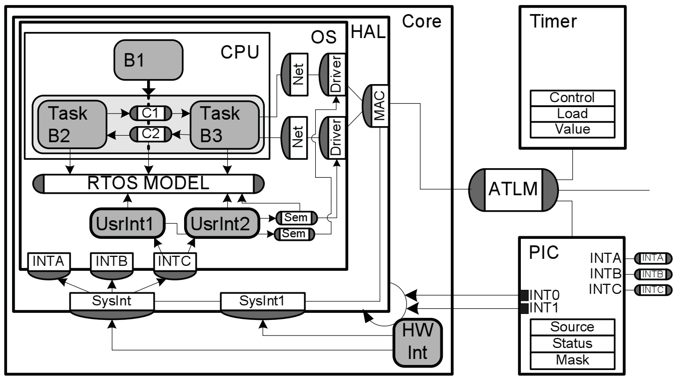

Fig. 3. Processor transaction-level model [16]

D. Software Synthesis

After the functional and performance goals have been validated using the virtual platform, the TLM serves as an input to the software synthesis [17]. Fig. 4 outlines the flow.

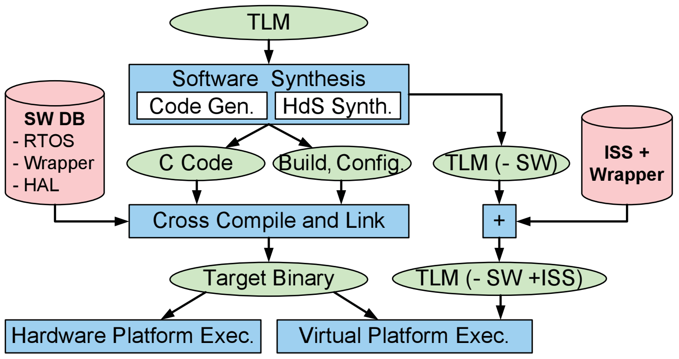

Fig. 4. Software synthesis [17]

Software synthesis deals with two major tasks the code generation and the Hardware-dependent Software synthesis. Code generation extracts the application code out of the modeled tasks in the TLM. It flattens the hierarchy and translates SLDL-specific constructs into C-code. The HdS synthesis then generates multi-tasking code targeted for the selected execution platform (either RTOS-based or for bare-metal C execution). For communication between tasks, it generates wrapper code that realizes the earlier abstract communication primitives with services provided by the underlying multi-tasking solution. For external communication, communication with components outside of the processor, HdS generates drivers and interrupt handlers (if selected) that realize selected communication scheme over the underlying communication media.

IV. Experimental Results

To validate the proposed solution, we have captured the application code of a Brain Computer Interface (BCI) application. Non-invasive BCI utilize Electroencephalography (EEG) signals to detect brain activity and infer the human intent. As one example, Steady State Visually Evoked Potential (SSVEP) is the response to an oscillating stimulus with fixed frequency. The subject will be presented with a screen with checker sequences flickering at different frequencies. After a settling period, the frequency of the checker sequence in focus of the subject is detectable from the visual cortex. Different communication (e.g. simulation of key presses) and control (e.g. wheelchair control) applications can be constructed using this and further advanced principles [18], [19], [20], [21].

Fig. 5 shows an initial model of a SSVEP-based BCI. In particular, we focused on the detection and classification mechanism. The stimulus to the model is pre-recorded EEG data in which a subject focused in random order one of 3Hz, 5Hz, 7Hz, 9Hz flickering checkerboards. Using a commercial g.Tec EEG, 16 channels of EEG were recorded. The output is controlled via a control channel in the monitor with the reference output.

In the current design, each of the 16 EEG channels passed through an FFT and then submitted to a threshold comparison. Based on its result, a local decision engine forwards the vote to centralized fusion engine. There, the highest likely focused on frequency will be selected. We currently implement linear classifiers.

Using SCE [10], we have generated virtual platforms for the application containing a ARM9EJ-S and a Blackfin BF527 [22] processor, respectively. Stimulus and monitor were mapped to custom hardware components attached to the processors memory bus. System simulation was performed using TLM Virtual Platforms as well as ISS-based virtual platforms. Different target operating systems of uC/OS-II [23] and RTEMS [24] were explored.

14,851 lines of application and driver code were generated. Including platform support and OS code the ARM9 including uC/OS utilizes a text size of 185kB and the Blackfin with RTEMS (4.11) utilizes 207kB. In this version, both operating systems were generically configured. In future, we will include application-specific OS configurations.

V. Conclusion

This paper has given a brief overview over modeling of biomedical applications in the context of a system-level design flow. It discussed algorithm capturing in a behavioral hierarchy of C behaviors that communicate through predefined channels and shared variables (with strict behavior synchronization). Our experimental results based on a Brain Computer Interface application showed modeling, VP generation and software synthesis of such applications. In future, we will focus on expanding the synthesis capabilities (e.g. automatic OS configuration) and validate VP accuracy.

Acknowledgment

The author would like to thank Rohan Kangralkar and Soumya Begur for their work on modeling the BCI application, and Deniz Erdogmus for his guidance the the application. The author would like to thank Rainer Dömer, and Andreas Gerstlauer as well as the Center for Embedded Computer Systems at the University of California Irvine. The teams

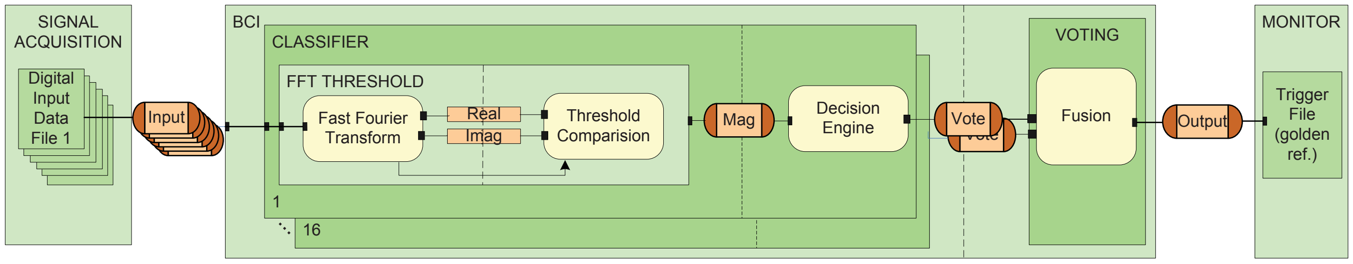

Fig. 5. Example Brain Computer Interface (BCI) application

dedicated and creative work made the advances described in this paper using the SCE possible.

[1] International Technology Roadmap for Semiconductors (ITRS), "ITRS home," http://www.itrs.net/, 2008.

[2] T. Givargis and F. Vahid, *Embedded System Design*. Wiley, 2002.

[3] G. Martin, "Overview of the mpsoc design challenge," in *Proceedings of the Design Automation Conference (DAC)*, San Francisco, CA, Jul. 2006.

[4] W. Ecker, W. Müller, and R. Dömer, *Hardware Dependent Software: Principles and Practice*. Springer, 2009.

[5] A. L. Sangiovanni-Vincentelli, "Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, March 2007. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/263.html

[6] Open SystemC Initiative (OSCI), http://www.systemc.org/, 2008.

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

[8] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, pp. 1517–1530, 2009.

[9] H. Nikolov, M. Thompson, T. Stefanov, A. D. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. F. Deprettere, "Daedalus: Toward composable multimedia MP-SoC design," in *Proc. of the ACM/IEEE Int. Design Automation Conference (DAC '08)*, June 2008, pp. 574–579.

[10] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-Chip Environment: A SpecC-based Framework for Heterogeneous MPSoC Design," vol. 2008, no. 647953, p. 13, 2008.

[11] J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, pp. 1–23, 2009.

[12] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Environment for Electronic System Design," *IEEE Computer*, vol. 36, no. 4, April 2003.

[13] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, "Uml-based multiprocessor soc design framework," *ACM Trans. Embed. Comput. Syst.*, vol. 5, pp. 281–320, May 2006. [Online]. Available: http://doi.acm.org/10.1145/1151074.1151077

[14] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y.-P. Joo, "PeaCE: A hardware-software codesign environment of multimedia embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–25, 2007.

[15] W. Müller, R. Dömer, and A. Gerstlauer, "The Formal Execution Semantics of SpecC," in *Proceedings of the International Symposium on System Synthesis*, Kyoto, Japan, Oct. 2002.

[16] G. Schirner, A. Gerstlauer, and R. Dmer, "Fast and accurate processor models for efficient mpsoc design," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 15, pp. 10:1–10:26, Feb. 2010.

[17] G. Schirner, A. Gerstlauer, and R. Dömer, "Automatic Generation of Hardware dependent Software for MPSoCs from Abstract System Specifications," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*, Seoul, Korea, Jan. 2008.

[18] D. Erdogmus, K. H. II, B. Oken, B. Roark, and M. Fried-Oken, "nitial design of an aac device with noninvasive bci, adaptive language modeling, and rapid serial visual presentation (rsvp)," in *BCI Meeting 2010*, 2010.

[19] B. Blankertz, M. Krauledat, G. Dornhege, J. Williamson, R. Murray-Smith, and K.-R. Mller, "A note on brain actuated spelling with the berlin brain-computer interface," in *Lecture Notes in Computer Science, 4557*, 2007.

[20] I. Iturrate, J. Antelis, A. Kbler, and J. Minguez, "A non-invasive brain-actuated wheelchair based on a p300 neurophysiological protocol and automated navigation," *IEEE Transactions on Robotics*, vol. 25, no. 3, p. 614627, 2009.

[21] X. Perrin, R. Chavarriaga, F. Colas, R. Siegwart, and J. Milln, "Brain-coupled interaction for semi-autonomous navigation of an assistive robot," *Robotics and Autonomous Systems*, vol. 58, no. 12, pp. 1246–1255, 2010.

[22] A. D. Inc., *ADSP-BF527: Low Power Blackfin Processor with Advanced Peripherals*, http://www.analog.com/en/processors-dsp/blackfin/adsp-bf527/processors/product.html.

[23] J. J. Labrosse, *MicroC/OS-II: The Real-Time Kernel*. CMP Books, 2002.

[24] R. S. C. Members, "Real-Time Operating System for Multiprocessor Systems home page," www.rtems.com.