Overcoming the Gap between Design at Electronic System Level (ESL) and Implementation for Networked Electronics

Tommy Baumann, Maik Hauguth, Horst Salzwedel
Technical University of Ilmenau, Dept. of Computer Science and Automation
D-98693 Ilmenau, Germany
Tommy.Baumann, Maik.Hauguth, Horst.Salzwedel@TU-Ilmenau.de

Keywords: Electronic System Level (ESL), Architecture Model, Performance Analysis, Synthesis

Abstract

Transition to model based design of systems at electronic system level (ESL) has greatly reduced the complexity by raising the level of abstraction. It resulted in improvements in quality of design, reduction of development time, and reduction of number of iterations in design. A new emerging problem is the widening of the gap between design at ESL and implementation. In this paper a method is presented to overcome this gap. For the design of a inexpensive high precision positioning system based on GPS, additional sensors, and differential signals, functional requirements are mapped into a MLDesigner [1] model containing functional model, environmental model, and use cases. Annotations on the model are automatically translated in an architectural model with channels, resources, events, and memory from a developed standardized library. After design iterations on architecture from a selected sub-model, an implementation framework for hardware and software code is generated and I/O specific parameters are added. The code can be used on FPGAs and embedded processors.

INTRODUCTION

Rapid advancements in microelectronics enables embedded systems to execute complex algorithms at speeds of super computers.

Electronic systems in aircraft, automobiles, home entertainment systems, mobile telephones, GPS/Galileo navigation systems, multi-functional systems such as Quint Network Technology (QNT) or Systems On Chip (SoC) are today networked embedded systems that exhibit complexities that can no longer be developed at reasonable cost and acceptable technical risk at implementation level or functional level. Integrated hierarchical model based design methodologies and tools have to be used to integrate the design flow from concept to implementation. Moving model based design methodologies to electronic system level (ESL) has permitted rapid development of verified executable specifications [9], model systems at system level [10], fast optimization of networked architectures at performance level [4] or size resources in complex reconfigurable electronics [2]. The move towards higher abstraction has made it possible to cope with the increased complexities. However, it has widened the gap between design and implementation. In this paper a method is presented that overcomes the gap between abstract system models for design and the realization in hard and software at RTL level. An integrated design methodology and extensions for the tool MLDesigner [5] are presented that makes design decision on function, performance, and architecture at ESL and translates this design automatically to VHDL. Figure 1

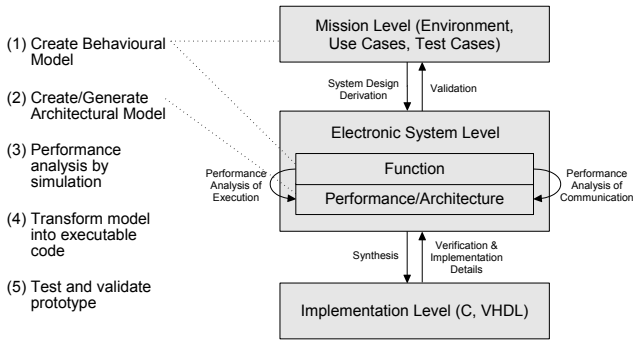

Figure 1. Mission level system design flow

shows in overview the proposed design flow from mission level with informal descriptions of the systems tasks, use- and testcases and its environment to implementation in hard- and software. The link between mission level and implementation is the electronic system level. Function and architecture is designed and validated at that level. The left side of Figure 1 shows neccessary steps for a systems design and their connection to the particular abstraction levels.

The new design methodology is demonstrated for the design of a Cheap High Precision Positioning (Chippo) system. Figure 2 shows the functional characteristics of the Chippo system. The position measurements of a GPS receiver are enhanced by measurements of accelerations and angular rates of an inertial measurement system on a chip and integrated by a Kalman navigation filter. The accuracy of the GPS receiver is enhanced by differential GPS corrections received

[1] MLDesigner is a trademark of MLDesign Technologies

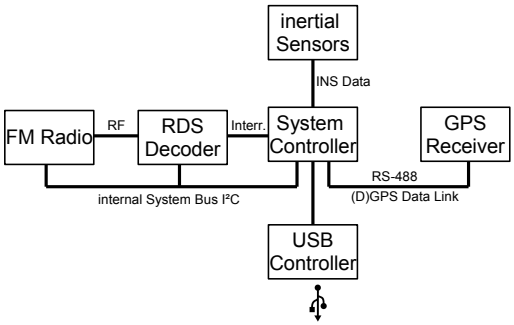

Figure 2. Concept of the GPS based positioning system Chippo

from RASANT [2]. The interface to external systems is realized by USB. For the development of this system some of the hardware components are available off the shelf, others have to be developed and integrated into the overall positioning system. We now describe the different steps of the developed design methodology.

FUNCTIONAL SPECIFICATION

An embedded system that controls processes or that processes analog and digital signals is often characterized by a considerable algebraic complexity. A mathematical model of its functions is essential for a successful design. Usually core functions of embedded systems can be described by algebraic or differential equations. However, continous processes must be discretized for an implementation in digital systems. The validated mathematical model is the basis for the system design.

The mathematical model of the sensor fusion system within Chippo is a discrete Kalman navigation filter [6]. This model was developed and validated with GNU Octave [3]. Characteristic trajectories for the missions of the system, use cases, are used for the validation.

To permit a unified integrated design, the mathematical model was described by a hierarchical block diagram of the modeling tool MLDesigner. MLDesigner models and simulations can be defined by a graphical model editor and are stored in XML, permitting standardized methods for transformations and translations of the models. Multiple execution domains permit to model functional behavior, the environment and the architecture in their respective native forms. The execution domains include discrete events (DE), synchronous and dynamic data flow, extended finite state machines (FSM) and continuous and analog domains.

A hierarchical functional model of the system and the environment was developed and validated by simulation against the Octave model for the use cases defined. Implementable algorithms are used for all mathematical descriptions. First all non-algebraic functional characteristics are defined. These include processes for control and data handling. Doing this in the appropriate form already for the abstract model assures that the model can later be implemented.

Critical for later integration of the system are the correct interfaces between components and to the environmental model. Inputs from the environmental models are typically events that cause state transitions in the model. In our design methodology these inputs are modeled as port objects. Port objects react to events and trigger execution of functions of the model. Another method to receive information from the environment is by sensors, which are polled for physical and logical values at defined times. The outputs of the environmental model are modeled as memory objects that represent the state of the environmental model at all times. The environmental model updates these memories and the sensor model of the system reads the values from these memories. Port as well as memory objects may use complex data structures for describing the status of a system. To describe a priori known implementation details like range of values, a defined structured annotation is used. These define specific characteristics for each model element. Constraints from iterations in the design process may be defined here for the abstract model.

For the Chippo project, validated functional behavior was defined by hierarchical MLDesigner modules. Figure 3 shows the top-level hierarchy of the functional and environmental model. The states of the environmental model, e.g., the number of available radio stations or kinematic states are realized by memories. The characteristics of the environmental model can be changed by parameters. Additional elements for reception and decoding of differential GPS data are added. The

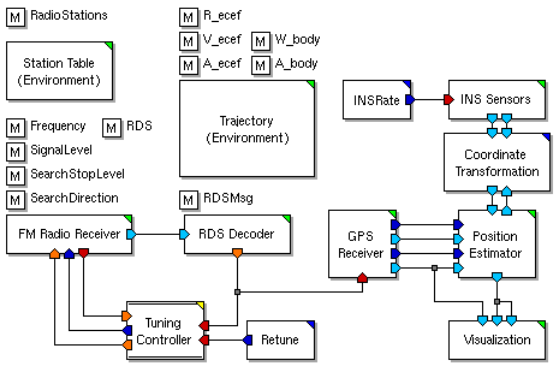

Figure 3. Top level structure of the Chippo functional model

functional model of the Chippo system therefore includes the validated functional model as well as the event based processes for operation of the system. The system is validated

[2]RASANT is a service provided by the German ARD radio stations, which broadcasts RDS GPS corrections via its FM stations

by simulation against the mathematical model of the system. Figure 4 compares the accuracy of the differential GPS and sensor enhanced Chippo functional model and the pure GPS position estimation.

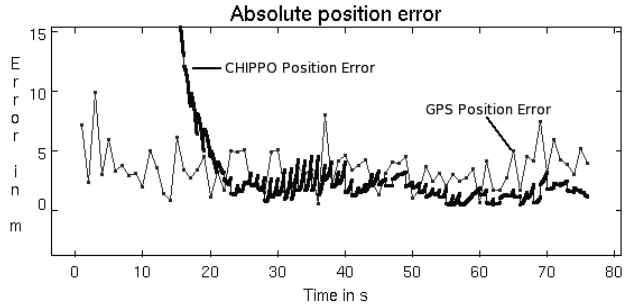

Figure 4. Comparison of the differential GPS and sensor enhanced Chippo functional model and pure GPS

ARCHITECTURAL SPECIFICATION

After system behavior is specified, accompanying execution- and communication structure must be pinpointed in the form of an architecure model. Executable components (partitions) are operating resources of their assigned functions of the functional model. Data transfer between partitions is modeled by communication components (channels). Communication structure arises from the partitions, because interacting functions within the functional model are mapped on channels within the architectural model. The architectural model permits performance analysis of execution and communication components by simulation on ESL level. Specific properties of partitions and channels are abstracted by parameterized resources. Utilization of resources is expressed in terms of quantities and queuing of events. Resources restrict the choice of assignable functions to partitions and throughput of channels. For example, quantity resources can model memory or CPU utilization of partitions and queue resources can control access of channels within the model. The architectural model is also the basis for the following synthesis.

Figure 5 shows how functions (F) are mapped on partitions (P) to create an architectural model automatically. XSLT [3] scripts have been developed that map architectural into functional and inverse functional into architectural models. Only certain parts of the behavioral model are assigned to real partitions. Parts belonging to the environmental model are assigned to a virtual partition (Env). Of course the architectural model can also be modeled directly. The lower part of Figure 5 shows the several elements of the architectural model. They are defined in a common library,

[3]XSLT Extensible stylesheet language for transformation

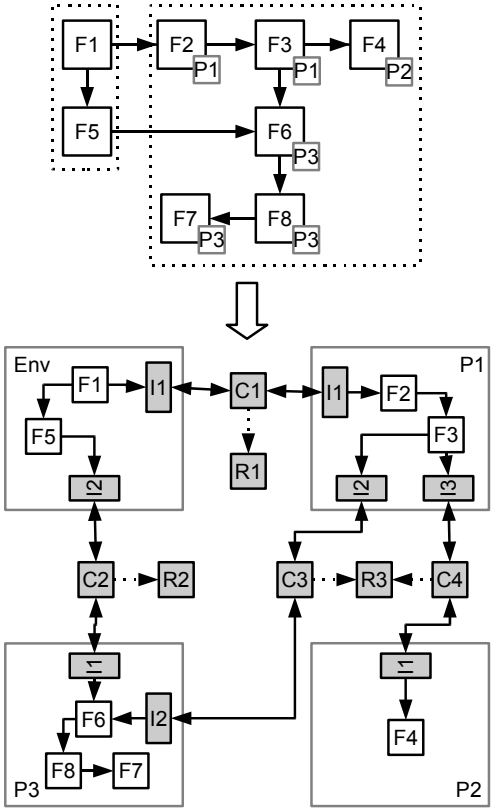

Figure 5. Generation of architectural model from behavioral model

the *Network Block Set*. It contains partition patterns (P), a generic interface (I), channels (C), and different resources (R). Each channel can be understood as a one to one connection between exactly two partitions, chanalizing the whole communication between them. Data transfer is accomplished on the abstract level with help of data structures protocol packet and data packet. Protocol packets are timeless and control interface handshake sequences. Data packets transfer data and adresses information. They are transfered completely and delayed based on channel performance. Channel resources model channel usage rate and define network topology by their linkage. This allows to connect several one to one channels to a network. Furthermore the resource link concept is a fast and easy possibility to iterate through different network topologies. The connection of partitions and channels is realized by means of generic, channel and partition independent, interfaces. Interfaces manage data that are sent to queues. They have an active or passive request behavior regarding channels. Examples of active channels are USB 2.0, FlexRay, and Wishbone[4] shared bus, and for

[4]The WISHBONE System-on-Chip (SoC) Interconnect Architecture for Portable IP Cores is a portable interface for use with semiconductor IP cores.

passive serial interface EIA232 and Wishbone dedicated bus. The generic interfaces allow to exchance such channels without the need of interface adaption. As mentioned above channels are abstracted from existing bus protocols. It is up to the engineer on which abstraction level they are designed and which attributes are relevant for performance analysis. For example a USB 2.0 channel can neglect electrical and mechanical aspects of the bus. Furthermore USB hubs can be omitted with the restriction that all devices are directly connected to the USB host controller. Distances between hubs are neglected. Relevant attributes are transfer speed, transfer mode packet size (maximum payload size per transaction) and poll interval (bus access period).
Simulation of the architectural model allows analysis of performance parameters (e.g. timing) of execution and communication structure. Iterated variations on architectural model lead to an optimized design. For simulation a discrete event (DE) based execution model is used. This permits the delay of the data flow within partitions and on connection channels. For analysis the usage rate of resources is observed. If a modeled architecture variant is not realizable, an error message is displayed.
Figure 6 shows the architectural model of the Chippo sys-

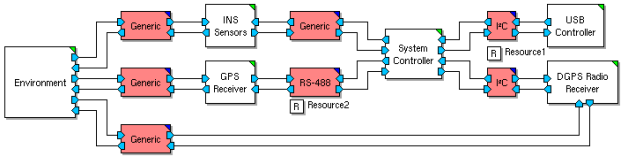

Figure 6. Chippo architectural model

tem. It consists of partitions System Controller, INS Sensors, GPS Receiver, USB Controller and DGPS Radio Receiver, furthermore, it contains the special partition Environment for the environmental model. Every partition has exactly one interface with one input and one output port for every connected channel. Thru these ports data are chanalized. To explain the principle of chanalisation Figure 7 shows the internal structure of partition System Controller. There are four interfaces connected with different functions, whereby each interface externaly communicates with one in another partition.
After mapping functions of the Chippo project to architectural partitions, functions that are neccessary for operation and interaction of their partitions were modeled. In particular, application specific communication protocols between partitions were modeled by FSMs. The extended model was validated by comparison with results of the functional and mathematical models. Measurements of architectural model performance values (like data rate and volume) lead to predictions of required communication and execution capabilities.

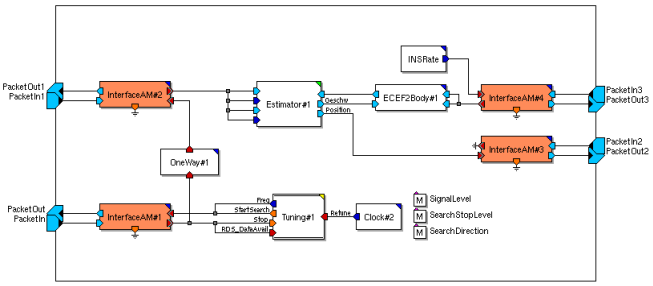

Figure 7. System Controller partition of Chippo architectural model

IMPLEMENTATION

The development of a working prototype at the implementation level requires transformation of the abstract models of function and architecture into hardware and software components. It is desirable that as little as possible has to be tuned by hand. The main task in translating the ESL functional/architectural model into a prototype is the translation of the algorithms into hard- and software languages (IP Cores) and mapping of transactions between architectural components into implementation specific communication structures. The architectural model, as we described it in the previous section, permits to generate a message passing communication system, which can be used for hardware and software. Mapping of transactions into message passing communication models is controlled by modeled elements of the architectural model and annotations. The advantage of this method is its easy realization into languages like VHDL that do not explicitly support this construct.
The ESL functional/architectural model includes all information about communication system and distribution of functional and architectural components. Components of the architectural model become IP cores or independent SW processes of the implemented prototype. Transformation of models is carried out independently for each partition. Each partition is transformed by an XSLT transformation into implementable functional code plus interfaces. Modifications of the XSLT scripts permit to generate code for different description languages or intermediate descriptions. To perform this translation, XSLT transformations have been developed that translate annotated XML descriptions of the model into implementable code. XSLT transformations for translations into VHDL code have been developed that permit implementation of partitions in FPGAs and ASICs. Dependent on the structure of the communication system and the architectural components, corresponding interfaces and architectural com-

ponents have to be generated. There are a variety of standards that define communication between architectural components at board level, e.g., VME, PCI, as well as on ICs, e.g., Wishbone.

Table 1 depicts developed XSLT transformations into VHDL code and documentation at ESL and implementation level. Dependent on structure and interfaces of architectural components synchronous or Wishbone interfaces can be generated.

The transformations also generate the documentation automatically. However, it must be noted that initially not all implementation details become known for a complete implementation. As more details of the implementation are known, they can be added to the annotation of the ESL functional/architectural model, permitting more detailed performance analysis and more detailed generation of the interface. Figure 8 depicts the flow for iterative transformations for transforming an ESL model into code for implementation of hardware and software. A method has been developed to integrate newly generated code with already implemented details. It is based on the DIFF[5] utility. Added implementation details during an iteration will not be overridden for a new code generation.

The ability to translate functional components into HW and SW languages depends on how the modules are being described. Only carefully coded standardized libraries at ESL level can be translated into implementable code that approaches hand coding. Different methods and tools for code generation are available. Descriptions like functional specifications by hierarchical FSM permit an automatic translation into efficient and high quality code. The generation of ANSI C and VHDL code from hierarchical FSMs was solved in references [1] [8]. It was shown that realization and implementation of the communication interface plays a pivotal role for integration into an implementation. Maintaining the hierarchical structure of ESL functional/architectural models permits the simplification of the code generation by following the hierarchy step by step.

[5]DIFF is Linux tool to compare the content of different files

Table 1. Developed XSLT transformations for hardware implementations and documentation

Model	Transformation	Documentation
System	Schematic	Model documentation
Module	Schematic	Model documentation
	Wishbone IP core	Wishbone datasheet
	VHDL component	
FSM	Wishbone IP core	Model documentation
	VHDL component	Wishbone datasheet
Primitive		Model documentation

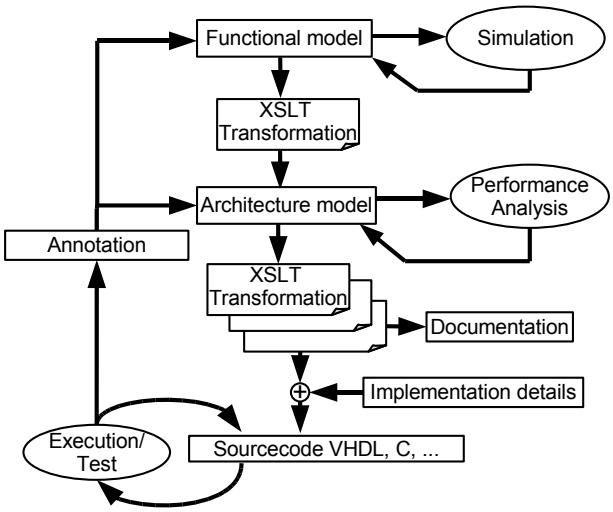

Figure 8. Iterative transformations for the generation of hardware and software code

For the Chippo example the architectural components have been translated into Wishbone compatible VHDL IP cores. The automatic generation of communication software processes is intended. ESL virtual prototypes, described by MLDesigner XML descriptions have been translated into HW/SW code by XSLT transformations. Generation of implementation details, integration and test are strongly dependent on the realization of the used architectural components. Manufacturer specific tools may be used here.

CONCLUSION

An integrated design method for overcoming the gap between design at ESL and implementation has been developed. The new design methodology was demonstrated for the design of a high performance GPS based positioning system with differential corrections and additional inertial measurements. During the functional level design step, a navigation filter was designed to meet the functional requirements of the system. In the next design step architectural elements were automatically added to this model by XSLT transformations from annotations. Sizing information for the architectural components were determined by performance simulation. In the third step the ESL model was translated by an XSLT script into HW/SW code of a prototype:

- algorithms are translated into HW and SW languages (IP Cores)
- channels and their interfaces are translated into interface IP Cores
- resources of the ESL model determine the topology of the implementation of the buses

- some of the interface IP Cores are already defined in e.g. the Wishbone Specification [7]

Implementation details were added iteratively by updating annotations on the ESL model.

ACKNOWLEDGEMENT

We thank MLDesign Technologies Inc. for the use of their MLDesigner modeling and simulation tool without which this work had not been possible.

REFERENCES

[1] Chris Lanfear; Matt Volckmann. 2005. *Bridging The Software Complexity Gap*. Venture Development Corp. Electronic Design Europe Digital Magazine.

[2] D. Bueno; C. Conger; A. Leko; I. Troxel; A. George. 2005. *RapidIO-based Space System Architectures for Synthetic Aperture Radar and Ground Moving Target Indicator*. High-Performance Embedded Computing (HPEC) Workshop, MIT Lab, Lexington, MA.

[3] Eaton, John W. 2004. *GNU Octave Documentation*. University of Wisconsin. http://www.octave.org.

[4] K.M. McNeir; M. Zens; H. Salzwedel. 2003. *System-Level Partitioning Using Mission-Level Design Tool for Electronic Valve Application*. SAE 2003 World Congress, Detroit, Michigan.

[5] MLDesign Technologies Inc. 2006. *MLDesigner Documentation*. http://www.mldesigner.com.

[6] Qi, Honghui; Moore, John B. 2002. *Direct Kalman filtering approach for GPS/INS integration*. IEEE Transactions on Aerospace and Electronic Systems.

[7] Richard Herveille. 2002. *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. opencores.org.

[8] Rath, Holger; Salzwedel, Horst. 2004. *ANSI C Code Generation for MLDesigner Finite State Machines*. Shaker, Aachen. 49. Internationales Wissenschaftliches Kolloquium IWK'2004.

[9] Rath Holger; Schorcht Gunar; Salzwedel Horst. 2006. *Validation of Executable Specification of Automotive Power Management System by Simulation (in German)*. Hanser automotive, Carl Hanser Verlag, Munich

[10] Schorcht, Gunar; Troxel, Ian; Farhangian, Keyvan; Unger, Peter; Zinn, Daniel; Mick, Colin K.; George, Alan; Salzwedel, Horst. 2003. *System-level simulation modeling with MLDesigner*. Modeling, Analysis and Simulation of Computer Telecommunications Systems 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium.