A Mission Level Design Language Based on AleC++

Bojan Anđelković, Vančo Litovski, Volker Zerbe

Abstract - Modern complex system design demands modeling on a high level of abstraction together with the system environment components. Such model enables mission level system simulation in the context of its operational conditions. A mission level design language providing mission and system level verification is presented in this paper. Also, this language enables designers to describe some of the components at implementation level to test and validate the system implementation at mission level. In this way a uniform design framework is achieved from mission/system down to implementation level.

I. INTRODUCTION

Modern System-on-Chip (SoC) designs grow in complexity and combine analog, digital and mixed-signal hardware components, as well as embedded software and non-electrical elements on one chip. Therefore, such designs require powerful modeling languages covering system architecture, mission/operational environment modeling, embedded software, register-transfer-level, analog and mixed-signal modeling and verification at different abstraction levels. Since system architects need to come to a proof of the system concept very early in the design flow, mission/operational and system level modeling and simulation are becoming very important step during the design process. At the same time, hardware designers need a design language capable to describe various analog, digital and non-electronic components. It is also necessary to enable proper design verification at system and implementation levels.

Mainstream hardware description and verification languages such as VHDL-AMS, SystemC, OpenVera, PSL, SystemVerilog do not meet all of the requirements in modern mixed-signal SoC design [1]. VHDL-AMS is not appropriate for specifying software and system level behavior. SystemC can be used for architectural tradeoffs and early application software verification, but it does not support modeling of analog and mixed-signal systems. SystemVerilog enables creation of efficient testbenches and assertions for simulation-based and formal property verification of digital systems. However, it does not provide analog and mixed-signal modeling and verification constructs.

B. Anđelković, and V. Litovski are with the Department of Electronics, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia & Montenegro, E-mail: (abojan,vanco)@elfak.ni.ac.yu

V. Zerbe is with Computer Science and Automation Faculty, Technical University of Ilmenau, Germany, E-mail: volker.zerbe@tu-ilmenau.de

This paper presents the idea of developing a uniform design language covering different levels of abstraction in modeling and simulation of mixed-signal SoC. This Mission Level Design Language is based on AleC++ [2]. In this way designers can describe mission and system level modules and, after system validation at this level, replace some of the components with implementation level, more detailed models (digital circuits, transistor level models etc.) to verify the complete system implementation in its working environment. Such design language provides a uniform design framework from mission/operational level down to implementation.

II. MISSION LEVEL DESIGN

Typical design flow of modern mixed-signal SoC is shown in Fig. 1. It starts with modeling of architecture at high-level of abstraction to decrease simulation time and to get an early feedback of the complete system behavior. The system architecture should be refined and tested for functional correctness at mission/operational level to validate the complete system in the context of its operational conditions. Mission level design and simulation integrate architectures, functions, system environments and missions into a single framework. In this approach a virtual prototype of the entire heterogeneous system including typical operational conditions is developed. In that way it is possible to verify the impact of design changes and implementation decisions on overall system performance, improving chances of first-pass system success. After validation at mission and architectural levels, hardware/software partitioning is performed and functional models of the system components are developed. At the end system modules and the complete system should be modeled and validated at implementation level.

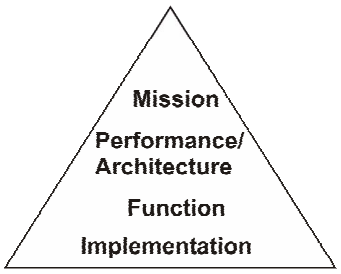

Fig. 1. Levels of the design flow for complex mixed-signal SoC.

Mission level design concept will be illustrated by an example of a navigation system. The system mission is to help in orientation on the way from point A back to A via B and C, as shown in Fig. 2.

1-4244-0117-8/06/$20.00 © 2006 IEEE

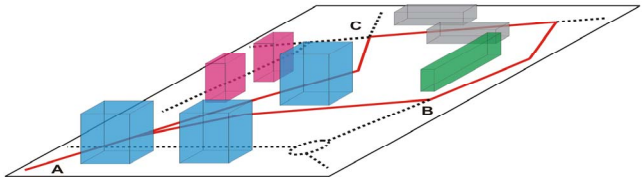

Fig. 2. Navigation system mission.

It consists of various components such as electronic compass, GPS module, gyros etc. During mission level design process it is necessary to model the whole system and validate its functionality on the desired path. Therefore, together with the system model, appropriate model of the system environment should be built.

A. Mission Level Modeling and Simulation

The tool MLDesigner is used to develop and simulate models at mission level [3]. Modeling is based on creating block diagrams using predefined primitive modules (primitives) from libraries. It is also possible to develop own primitives in a C++ like language. Multi-domain modules can be combined and simulated together. Supported domains are continuous time, discrete event, dynamic data flow, synchronous data flow, finite state machines and higher order functions. MLDesigner provides mission/operational level design tradeoff that includes modeling and simulation of dynamic Use Cases, mission environment (e.g., terrain, channel models) and operational modeling (e.g., human input devices, human output devices). It also enables system level design tradeoff (communication network design, embedded systems design) and functional level design tradeoff (algorithm design, hardware/software partitioning).

Fig. 3 shows system level model of the electronic compass [4] that is a part of the navigation system.

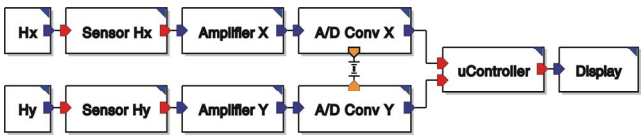

Fig. 3. Electronic compass mission level model.

The model is developed in MLDesigner together with some modules necessary to verify the compass functionality. The compass generates at the output the value of azimuth α. It is the angle between magnetic north and the heading direction. Magnetic north is the direction of "horizontal" component of the earth's magnetic field, the earth's field component perpendicular to gravity. The compass consists of the following building blocks: magnetic field sensors, amplifiers, A/D converters and microcontroller. Blocks "Hx File Data" and "Hy File Data" enable reading of input magnetic field strengths from files. The microcontroller executes software module that calculates the desired azimuth value from the signals proportional to the magnetic field strengths. This can be done by evaluating the arctan function using CORDIC (COordinate Rotating DIgital Computing) algorithm [5].

Fig. 4 shows mission level simulation results of the compass for the path given in Fig. 2.

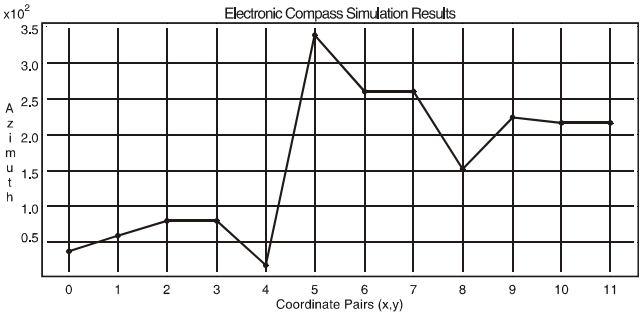

Fig. 4. Compass mission level simulation results.

III. Mission Level Design Based on AleC++

A. The AleC++ Language Features

AleC++ (Analog and Logic Electronic C++) is a proprietary object-oriented Hardware Description Language (HDL) developed for use in the simulator Alecsis [2]. It can be used for modeling of hardware/software systems from various domains at different levels of abstraction. Being a superset of C++ it can be used to describe analog, digital and mixed-signal hardware components, software modules, as well as non-electrical elements. AleC++ also provides some additional useful modeling features both for modeling of hardware components and system-level descriptions not found in other HDLs [2].

The basic element of hierarchical system description in AleC++ is module. The module behavior can be described using C++ like statements. AleC++ enables structural and behavioral modeling styles as well as the combination of the two. The module can also contain various parameters.

B. Mission Level Modeling in AleC++

Having in mind that MLDesigner models are hierarchical and based on primitives it can be concluded that these models can be described in AleC++. In order to accomplish that task, it is necessary to translate mission level design elements into the AleC++ language by developing appropriate translator [6]. In this way it is possible to use a vast library of predefined MLDesigner primitives in simulations in the Alecsis simulator. Organization of the simulator Alecsis extended by translator of MLDesigner primitives into equivalent AleC++ modules is shown in Fig. 5.

Functionality of MLDesigner primitives is defined using the Ptolemy language [3]. It is a preprocessor

language that allows the designer to use C++ code. The external interface of a primitive contains input/output port definitions and parameter definitions. For each port definition, MLDesigner generates an entry in the primitive source code. For parameters it is possible to specify name, type and default value. The Ptolemy language provides appropriate constructs for the definition of methods that describe the functionality of the primitive. These methods are executed at different stages in simulation of primitive instances such as instance creation and deletion, simulation start-up time and during simulation. The functionality of these methods is defined using C++ code and the Ptolemy language only defines the method structure.

Since AleC++ is a superset of C++, mapping of mission level primitives into AleC++ can be easily implemented. The correspondence between MLDesigner elements and appropriate AleC++ constructs is shown in Table I.

TABLE I
CORRESPONDENCE E BETWEEN MLDESIGNER AND ALEC++ ELEMENTS

MLDesigner	AleC++
Primitive, Module	Module
System	Root module
Parameters	Module parameters
Ports	Ports
Functionality in C++	C++ code, process statements, equation statements

Primitives and modules in MLDesigner correspond to AleC++ modules. A complete system model that can be executed/simulated is equivalent to *root* module in AleC++. Primitive parameters can be mapped into parameters of the equivalent AleC++ module. Since methods defined in the Ptolemy language can be executed at different stages during the simulation process, they can be mapped to different processes in AleC++ (initial, per moment, per iteration etc.). C++ code for methods can be easily included in AleC++ modules and statements for specifying equations.

To make things clearer, the process of AleC++ code generation from MLDesigner primitives will be illustrated on an example of amplifier module that is a part of the electronic compass system. In MLDesigner, amplifier module has an input and an output port, as well as, parameter *gain* specifying amplification value. The AleC++ module name is the same as MLDesigner module name. After AleC++ module declaration, declarations for all ports are written. All float type ports in MLDesigner modules correspond to ports of type *node* in AleC++. Statements in the Ptolemy language description relating to writing values to output ports are translated into equivalent AleC++ statements for defining equations. All other C/C++ code is mapped to AleC++ without change. Amplifier primitive in MLDesigner, its description in the Ptolemy language and the complete equivalent AleC++ module that translator generates for that model are given in Fig. 5.

MLDesigner Model

```
input
{
  name {input }
  type {float }
}
output
{
  name {output }
  type {float }
}
defparameter
{
  name        {gain }
  type        {float }
  default     {"1.0" }
  desc        {"Gain of the star." }
}
go
{
  output%0 << double(gain) * double(input%0);
}
```

AleC++ Model

```
module current SDFGain(node input;node output) {
 action(double gain) {
    process initial {

    /* GO Method from MLDesigner model */
    eqn SDFGain, {output}.v = gain * {input}.v;

 }//end Process block
}//end Action block
}//end Module
```

Fig. 5. MLDesigner primitive module, Ptolemy language description and equivalent AleC++ model for amplifier

When the translator generates equivalent AleC++ code for digital modules, declarations of integer ports in MLDesigner module map to ports of type *signal* in AleC++ together with appropriate port direction. The process created for digital modules has sensitivity list containing all input ports of type signal. Statements in the MLDesigner module description relating to writing values to output ports are converted into equivalent AleC++ signal assignment statements.

C. Implementation Level Modeling in AleC++

As described, after validation at mission and architectural levels, mission level models can be translated into equivalent AleC++ descriptions. Then, implementation level models described in the AleC++ language can replace some of the compass mission level modules. It enables to test various implementations at mission level in the context of system's working environment using the Alecsis simulator.

To illustrate implementation level modeling in AleC++, implementation of the amplifier module is

considered. The mission level module for the amplifier is replaced by the implementation consisting of two cascaded common-source MOSFET amplifiers and non-inverting amplifier with operational amplifier (Fig. 6). The operational amplifier is described at behavioral level while the complete non-inverting amplifier circuit is described at structural level. The MOSFET amplifier circuit is described at transistor level in AleC++ using SPICE model card for MOSFET. Some parts of AleC++ description for the amplifier are shown in Fig. 7. The amplifier is designed to have almost the same gain as in the mission level module. The designer should take care that in this case analog input signals should stimulate the compass system because transistor level models are used. Generated simulation results for sinusoidal input signals are given in Fig. 8. Because input signals are in phase the compass generates just two values for azimuth, for positive and negative values, respectively.

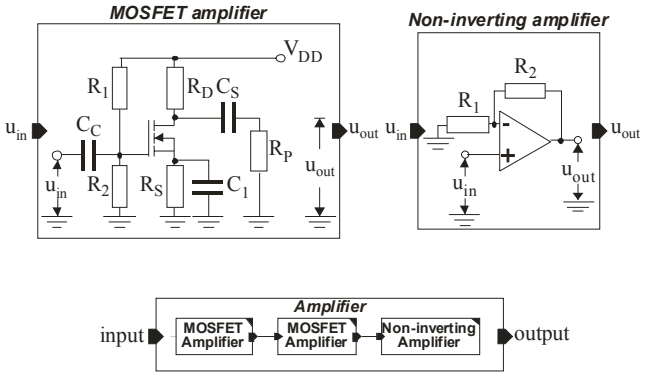

Fig. 6. Amplifier module implemented as two cascaded MOSFET amplifiers and non-inverting amplifier with opamp

```
module simpleMOS (node input,output) {
  vgen Vdd;
  resistor Rd, R1, R2, Rg, Rp, Rs;
  capacitor Cc, C1, Cs;
  mosfet m1;

  Vdd(vdd, 0) 12V;
  Rg (input1, input) 1k;
  Cc(g, input) 6n;
…
  //MOSFET transistor
  m1(d, g, s, 0) {model=my_nmos;l=2u;w=6u;};
}
module opamp (node minus; node plus; node
output) {
   vcvs vout; //voltage controlled source
   vout (output,0,plus,minus) gain=1e5;
   action () {
      process per_iteration {
         vout->gain=1e5;
      }
   }
}
```

Fig. 7. Parts of theAleC++ description for the amplifier.

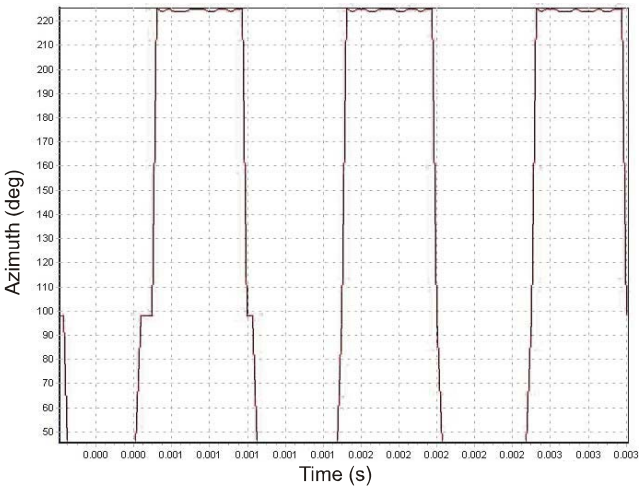

Fig. 8. Alecsis simulation results of the compass with the amplifier module shown in Fig. 6

IV. CONCLUSION

A mission level design language based on hardware description language AleC++ is presented in this paper. It extends powerful modeling capabilities of AleC++ with possibility to use mission level modules. Such design language covers the complete design flow of complex system from mission/operational down to implementation level. Comparing to other design languages, it enables the description of mixed-mode and mixed-signal systems containing various analog, digital and non-electronic components as well as embedded software modules. The language provides the complete system verification and gives the designer an opportunity to combine different levels of abstraction for various system modules and test different implementation solutions at mission level.

REFERENCES

[1] B. Anđelković, V. Litovski, V. Zerbe, "New Aspects in HDL's Performance Evaluation", in *Proc. of IEEE Region 8 EUROCON 2005 Conference*, Belgrade, 2005, pp. 499-502

[2] V. Litovski, D. Maksimović, and Ž. Mrčarica, "Mixed-Signal Modeling with AleC++: Specific Features of the HDL", *Simulation Practice and Theory 8*, 2001, pp. 433-449

[3] *MLDesigner Documentation*, *Version 2.4*, MLDesign Technologies, Inc., 2003.

[4] T. Stork, *Application Note – Electronic Compass Design using KMZ51 and KMZ52*, Philips Semiconductors, Systems Laboratory Hamburg, Germany, 2000. www.web-ee.com/primers/files/AN00022_COMPASS.pdf

[5] *Angular Position Development Kit for the 2SA-10*, *Operation Manual*, GMW, 2005., www.gmw.com/magnetic_measurements/ Sentron/sensors/documents/AN_125KIT_manual.pdf

[6] V. Zerbe, and B. Anđelković, "Design Flow for Automated Programming of FPGA", in *Proc. of IEEE 24th International Conference on Microelectronics (MIEL 2004)*, Vol. 2, Niš, Serbia and Montenegro, 2004, pp. 715-718