HARDWARE IN THE LOOP SIMULATION USING REAL-TIME CD++

Lidan Li
Gabriel A. Wainer
Trevor Pearce
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S5B6, Canada
Email: {lidan, pearce, gwainer}@sce.carleton.ca

KEYWORDS
DEVS formalism, real-time simulation, CD++.

ABSTRACT

Modeling and simulation tools have been used for helping in the early stages of hardware/software systems design. The DEVS formalism is a technique that enables hierarchical description of discrete event models that can be used for this task. The CD++ tool enables the description of discrete event models based on the DEVS formalism, and we have used it to provide hardware-in-the-loop simulation using the CODEC of a DSP board. First, a DEVS model was built using the real-time version of CD++ to simulate the behavior of the CODEC together with a test program using the CODEC. Next, the actual CODEC was deployed as a hardware prototype to replace the CD++ model, integrating the prototype into the original DEVS component. The real-time data communication between the CD++ model and the DSP board was explored in detail. As a result, we are now able to study models in a simulated environment, and to execute them in a hardware surrogate. The hierarchical nature of DEVS permitted to do this without modifying the original models, providing the base for enhanced system development in embedded platforms.

INTRODUCTION

In the early stage of the design of embedded systems, software and hardware are designed independently. The software development team is waiting for the hardware prototypes to be available; however, the hardware development team is waiting for the software environment for hardware prototype verification and testing. It is difficult to decide the trade-offs between the hardware and software solutions in terms of system performance requirements (time, power consumption) and probably delays the product design cycle. In addition, there are few interactions between hardware and software, which restrict the exploration of solutions where some functionality could migrate between both components (Berge 1997). With the emerging of the specific components (I/O, DSP, ASIC, FPGA), a mixed-system design is more efficient to realize specific applications, such as signal processing or telecommunications. The user can test the functionality of the hardware in a very early stage. This is economically efficient, and shortens the product development cycle and time-to-market period.

The DEVS (Discrete EVents Systems specifications) formalism for modeling and simulation (Zeigler. Praehofer and Kim 2000) provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse, reducing development and testing time. The hierarchical and discrete event nature of DEVS makes it a good choice to achieve an efficient product development test. DEVS are timed models, which also enables us to define timing properties for the models under development. Each DEVS model can be built as a behavioral (atomic) or a structural (coupled) model. A DEVS atomic model is described as:

$$\mathbf{M = <X, S, Y,}\ d_{\mathbf{int}},\ d_{\mathbf{ext}},\ l\mathbf{,\ D >}$$

X: the input events set
S: the state set
Y: the output events set
δ_{int} : internal transition function
δ_{ext} : external transition function
λ: output function
D: the elapsed time

A DEVS coupled model is formed by configuring several atomic models or coupled models:

$$\mathbf{CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}\ , select >}$$

X: the set of input events
Y: the set of output events
D: an index of components, each i$\in$ D
M_i: a basic DEVS model, where Mi = < Ii, Xi, Si, Yi, δ_{int}, δ_{xti} , ta >
I_i: the set of influencees of model I, each j $\in$ I_i
Z_{ij}: the i to j translation function
Select: the function prescribes which atomic model should be activated first under simultaneous events.

The CD++ environment (Wainer 2002) is a tool built to implement the DEVS and Cell-DEVS theory. The toolkit has been built as a set of independent software pieces, each of them independent of the operating environment chosen. The defined models are built as a class hierarchy, and each

of them is related with a simulation entity that is activated whenever the model needs to be executed. New models can be incorporated into this class hierarchy by writing DEVS models in C++, overloading the basic methods representing DEVS specifications: external transitions, internal transitions and output functions. CD++ employs a virtual time simulation approach (Rodríguez and Wainer 1999), which allows skipping periods of inactivity. The abstract simulation technique enables defining and using different simulation engines without affecting existing models. The recent real-time extension (Glinsky and Wainer 2002) enables simulation advancing based on the wall-clock, making the simulation process to be quiescent between events. The model being executed must react to external event in a timely fashion. This means when an external event arrives, the model should react within a predefined deadline, and return a result before that time. The real-time extension of the toolkit allows associating deadline with external events.

We show how to gradually incorporate hardware prototypes into a simulated environment using the real-time version of CD++ toolkit (Glinsky and Wainer 2002). The hardware prototype employed is a CODEC (a device performing A/D and D/A operations) embedded in a Digital Signal Processor (DSP) board (Analog Devices 2000). The block diagram in figure 1 below is the 2189M EZ-KITLITE DSP Board (Analog Devices 2000) and some major components used.

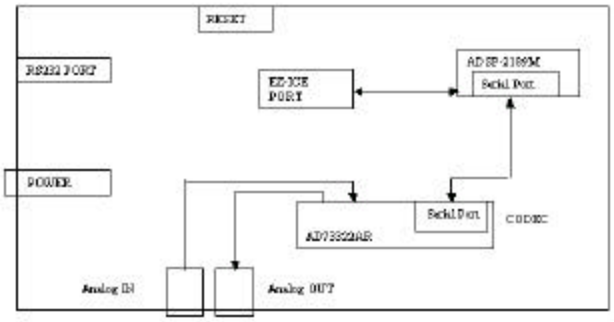

Figure 1: Scheme of the Analog Devices 2189M EZ-KITLITE Evaluation Board

THE APPLICATION MODEL

The prototype was designed in two stages. In the first stage, we built a set of DEVS atomic models using the real-time version of CD++. Figure 2 shows the *test A* experimental frame built in CD++, which was used to test the behavior of the CODEC model.

The *test A* experimental frame includes five atomic models. They are *clock*, *control*, *CODEC*, *analog signal generator* and *display*. A brief description of the these atomic models is given below:

- ***Clock***: generates control signals with a predefined period.
- ***Control***: distinguishes the incoming signal. If a command signal is received, it will invoke the CODEC atomic model.
- ***CODEC***: simulates the behavior of the CODEC.
- ***Analog signal generator***: generates an analog signal.
- ***Display***: updates the results from the CODEC model and displays them.

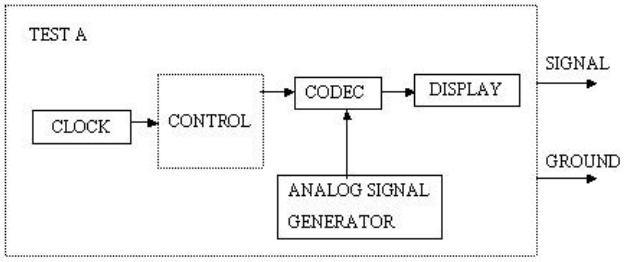

Figure 2: Test A, Experimental Frame Conceptual Model

The detailed definition of these models can be found in (Li, Pearce and Wainer 2002), but we will focus in the description of the *CODEC* model. Later, we will show its implementation using the available hardware. The *CODEC* model is responsible for translating an analog signal into an output digital signal. Figure 3 shows the CODEC DEVS atomic model used in our example.

CODEC = <X, Y, S, δext, δint, λ, D>

```
X = { ControlIn ∈ N+, AnalogIn ∈ R }
Y = { DigitOut ∈ BinaryStream }
S = { AnalogIn ∈ R } ∪ { ControlIn = 1 }

δext ( s, e, x ){
  If ControlIn //check which port is signaling

  case: port = ControlIn
    if control = 1, //start signal conversion
        analog_data = X.AnalogIn;
        digital_data = convert (analog.data);
        holdIn (active, time);
    else
        passivate(); // wait for the next event

  //if AnalogIn signals
  case: port = AnalogIn
        passivate();   // wait for the next event
}
δint ( s, e ) {
        passivate();
}
λ (s) {
        send digital_data to DigitOut;
}
```

Figure 3: CODEC Model Conceptual Definition

When an event arrives at *ControlIn* and its value equals to 1, the CODEC atomic model starts analog to digital signal conversion using the analog received in the *AnalogIn* port. When the signal conversion completes, the CODEC sends all results using the *DigitOut* port. The analog signal is a stream of floats and the digital signal is a stream of binary

strings. Please see the table 1 for a sample analog->digital conversion. The Analog generator is an atomic model and all it does is to randomly generate float representation of the analog signal.

Table 1: Analog to Digital Conversion

Analog Input (voltage)	Digital Output
0	0000
0.2	0001
0.4	0010
0.6	0011
0.8	0100

The *test A* coupled model is built to test the behavior of the software version of the CODEC. Figure 4 is the simulation output of the *test A* experimental frame. According to the figure, the first digital signal is obtained at time *00:00:40:708* through the *signal* output port. Its associated value is *0001* (here we use binary strings representing the digital signal value). At the time *00:00:41:608*, another digital signal is arrived at *ground* output port and its value is *0000*. All digital signals are processed successfully through the software CODEC model.

```
Wall Clock Time    Result      Output Port  Value
00:00:40:708       succeeded     signal      0001
00:00:41:008       succeeded     signal      0110
00:00:41:308       succeeded     signal      0011
00:00:41:608       succeeded     ground      0000
00:00:41:900       succeeded     signal      1010
…
```

Figure 4: Output of Test A Experimental Frame

INTRODUCING HARDWARE-IN-THE-LOOP

The application model we built in the previous section was reused when we replaced the model for the CODEC by the actual hardware. The new experimental frame (figure 5) was built to support interaction with the real CODEC on the board, and existing atomic models were reused. A *clock* model generates periodic signals to awake the *control* model. Then the *control* model will invoke the *TCL* model, which is in charge of initializing the CODEC and start the conversion. When the CODEC finishes the conversion, the *dataTransfer* model will acquire these data and send them to the *control* model. Finally, the *DataTransfer* model will feed the data to the *display* model. A brief description of the new atomic models is given following:

- ***TCL***: invokes and opens the VisualDSP debugger system. Once the debugger system is opened, the different TCL files needed for A/D and D/A access can be invoked to obtain samples and regenerate the analog signal.
- ***DataTransfer***: reads the digital samples and sends them back to the control model. In addition, the data will be written back to the board for display if the DAC is working.

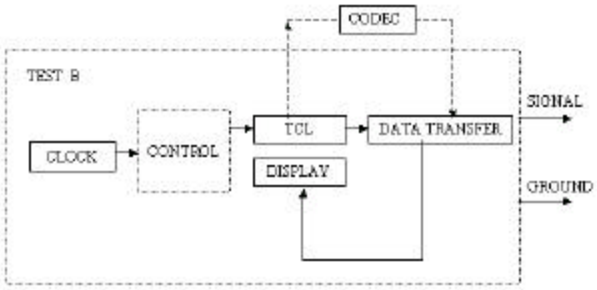

Figure 5: Test B Experimental Frame Conceptual Model with Hardware-in-the-loop

Figure 6 shows the CD++ coupled model definition of the *Test B* experimental frame. The first line in the figure defines the *Top* model, which includes four atomic models and one coupled model. The *Out* port represents two output ports: signal, ground, and the links describe the internal and external coupled schema. The similar configurations are specified for each of the atomic models in the Test B experimental frame conceptual model.

The **[top]** model always defines the coupled model at the top level. As showed in the formal specifications presented earlier, four properties must be configured:

- **`Components`**: describes the models integrating a coupled model. The syntax is `modelName@className`, allowing more than one instance of the same model using different names. The class name reference to either atomic or coupled models (which should be defined in the same configuration file).
- **`Out`**: it defines the names of output ports.
- **`In`**: it defines the names of input ports.
- **`Link`**: it describes the internal and external coupling scheme. The syntax is: `source_port[@model] destination_port[@model]`. The name of the model is optional and, if it is not indicated, the coupled model being defined is used.

```
[top]
components : clock@Clock control tcl@TclCycle
display@DisplayCycle DataTransfer@FileTransferCy
Out : ground signal
Link : out@clock in@control
Link : out@control in@tcl
Link : out@tcl start@transfer
Link : out@transfer in@display
Link : ground@display ground
Link : signal@display signal

[control]
components : queue@Queue central@ControlCycle
In : in
Out : out data_out
Link : in in@queue
Link : out@queue in@central
Link : data_out@central done@queue
…
```

Figure 6: CD++ Coupled Model Specification for Source Code for *Test B*

IMPLEMENTATION

The system was designed using two components: one running in a PC/Workstation, and another on the DSP board. See figure 7.

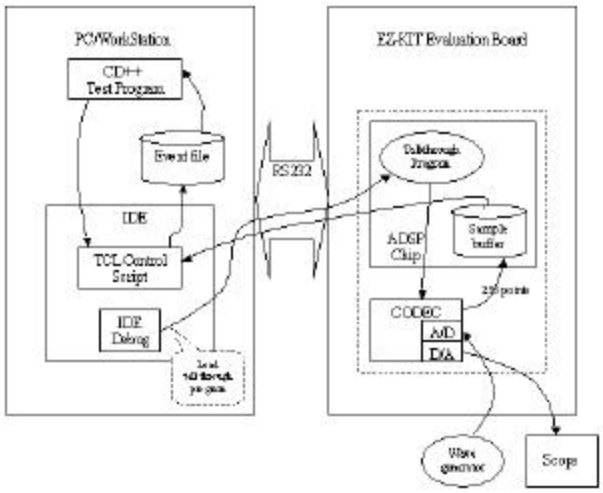

Figure 7: A High-level Architecture of the Simulation

The CD++ test experimental frame communicates with the board through the IDE software, and a built-in TCL script controls the debugger operation. The IDE uses serial communication to access the ADSP chip memory on the DSP board. In the board, a *talkthrough* program is loaded by the IDE, which invokes the CODEC, stores the digital samples in a circular buffer, and sends the data back to the CODEC for output. When the samples are available, the TCL control script gets them and makes them available for CD++ as new external events. The interaction between the components is depicted in Figure 7. There are three cooperating subsystems in the simulation, the DSP board, the IDE and CD++. The DSP board subsystem acquires an analog signal from a 15MHZ function/waveform generator, and digitizes it through an A/D of the CODEC. These digital samples are saved in a predefined location in the DSP chip memory. In addition, the CODEC D/A writes these digital samples back to reproduce the analog signal, and displays them in a digital oscilloscope.

An important issue raised was how to achieve the system's deadline. The real-time CD++ simulator uses the wall-clock to determine when to execute the next event. In this case, when the TCL model invokes the IDE application, we need time to reset the board. The solution is to let the model to wait for the hardware to start. During this period, the model will remain in the active state, and after that, it will generate an internal transition, recording the physical time it takes to this action. Another issue is to ensure that the model is ready to start a new read cycle, and that will depend on the speed for reading. The clock model will generate the read command according to the following diagram (Figure 8):

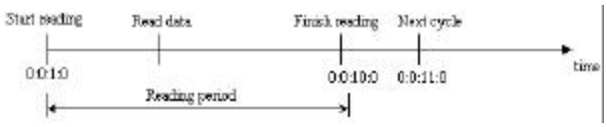

Figure 8: Clock Timing Diagram

Figure 8 shows that at time 0:0:1:0, the model started to read data from the board memory. After 9 time units, the model finished reading the data. The clock model will record this time, and then it will issue the next command at time 0:0:11:0 to start another cycle.

Displaying the analog signal on the real-time digital oscilloscope is another difficult issue. Due to the limitation of the current IDE debugger, only 256 sample points can be written back at a time. These digital points went through the D/A and displayed on the oscilloscope. The following digital pictures show a 1 KHz sine waveform on the scope.

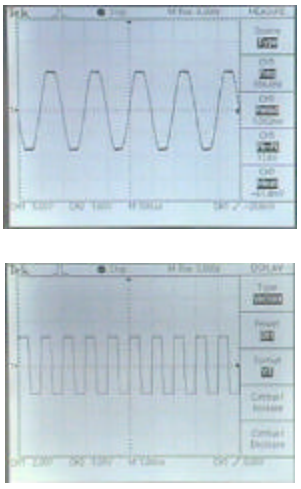

Figure 9: Digital Pictures of 1 KHz Sin Waveform

The top is the input signal in sine waveform; the bottom image is the output from the simulation. Since the output signal is instantaneous, the display of the output signal is caught by freezing the screen. After adjusting the scale of the signal, we can read that the frequency is 1 KHz, which means the output signal is correct (in the picture they seem to be different, because the digital oscilloscope does not have enough signals to acquire in order to auto-set the display).

TEST EXAMPLES

Different tests were conducted to evaluate the software and hardware components, as well as the program as a whole (refer to Test B conceptual frame) to examine the correctness and performance of the simulation.

1) Examining the A/D functionality of the CODEC. At initialization, the clock atomic model generated a control signal to start the *control* atomic model at time 00:00:00:040. The control model then invokes the TCL

model to start the IDE application and to load the TCL script. After pressing the *reset* button on the DSP board, the *talkthrough* program is loaded into the DSP board. At the same time, the *dataTransfer* atomic model is hold in the sleep state and is waiting for the data to be generated by the CODEC on the DSP board. Upon receipt of the converted digital data, the *dataTransfer* model will take these data and send them back to the control model. Finally, through a display atomic model, these samples will be updated and stored in a bin. After one simulation cycle, the clock will generate the next command. The simulation can be terminated by a preset simulation time or manually.

The following execution results show that all the signal conversions are successful and the values are obtained. These results match with those of the *test A* experimental frame and the differences in the value column represent a difference in the format and the analog waveform used.

Wall clock time	results	Output Port	value
00:00:40:708	succeeded	signal	0.50391
00:00:41:008	succeeded	signal	-1.00000
00:00:41:308	succeeded	signal	-0.97546
00:00:41:608	succeeded	ground	0.03967
00:00:41:900	succeeded	signal	-1.00000
…			
00:06:51:210	succeeded	signal	0.99997
00:06:51:210	succeeded	ground	0.03894

Figure 10: Output of Test Model

2) D/A functionality of the CODEC was added into the simulation cycle. The purpose of this test is to verify the correctness of the digital samples obtained from the previous testing case.

CONCLUSION

We presented how to achieve hardware-in-the-loop simulation of discrete event models based on the DEVS formalism. The real-time data communication between the CD++ model and the DSP board was explored in detail. As a result, we are now able to study models in a simulated environment, and to execute them in a hardware surrogate. The hierarchical nature of DEVS permitted to do this without modifying the original models, providing the base for enhanced system development in embedded platforms.

This is the first version of the CD++ simulation platform that employs an actual piece of hardware in the simulation loop. The hardware has been successfully introduced into the CD++ simulation platform. Different test examples and results were described to verify the correctness of the approach. The future development could be done towards making large systems and exploring different ways of hardware/software communications to further enhance and formalizing real-time simulation using CD++.

REFERENCES

Analog Devices, 2000. “ADSP-2189M EZ-KIT Lite Evaluation System Manual”.

Analog Devices, 2000. “AD73322 data sheet”. Analog Devices Technical Staff.

Analog Devices, 2000. “ADSP-218x DSP Hardware Reference”.

Analog Devices, 2000. “VisualDSP User’s Guide for ADSP-21xx DSPs”.

Berge, J.M. 1997. “Hardware/Software Co-Design and Co-Verification”. Kluwer Academic Publishers.

Glinsky, E.; Wainer, G. 2002. "Performance Analysis of Real-Time DEVS models". In *Proceedings of 2002 Winter Simulation Conference*. San Diego, U.S.A.

Li, L.; Pearce, T.; Wainer, G. 2002."An experience in hardware-software codesign using the DEVS formalism". Technical Report SCE-12-02, Department of Systems and Computer Engineering, Carleton University.

Rodríguez, D.; Wainer, G. 1999. "New Extensions to the CD++ tool". In *Proceedings of 31st SCS Summer Computer Simulation Conference*. Chicago, U.S.A.

Wainer, G. "CD++: a toolkit to define discrete-event models". 2002. In *Software, Practice and Experience*. Wiley. Vol. 32, No.3. pp. 1261-1306.

Zeigler, B.; Praehofer, H.; Kim, T. 2000. “Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems”. Academic Press.

BIOGRAPHIES

Lidan Li is a M.A.Sc. student in the Department of Systems and Computer Engineering (SCE) at Carleton University. She received her B. Eng. degree (1998) from Shanghai University, Shanghai, China.

Gabriel Wainer received a M.Sc. (1993) and a Ph.D. (1998, summa cum laude) in Computer Sciences from the Universidad de Buenos Aires (Argentina) and Université d'Aix-Marseille III (France). He is Assistant Professor in the SCE Department, Carleton University. He was Assistant Professor at the Universidad de Buenos Aires, Argentina, a visiting research scholar at ACIMS, University of Arizona and LSIS/CNRS (Marseille, France). He is author of two books and numerous articles on Discrete-Event simulation and real-time systems. He is Associate Editor of the Transactions of SCS. Prof. Wainer was a member of the Board of Directors and he is the chair of Standards Committee of the SCS. He is Associate Director of the Ottawa Center of The McLeod Institute of Simulation Sciences, and the coordinator of an international group on DEVS standardization.

Trevor Pearce completed both a B.Eng. (with High Distinction, 1982) and a M.A.Sc. (1986) in Electrical Engineering at Carleton University, and a Ph.D. in 1994, at Queen's University with the Department of Computing and Information Science. He has been an Assistant Professor at Carleton University, in the SCE Department since 1995, and has served a term as the Associate Chair for Undergraduate Studies. He is researching embedded systems development methods based on modeling and simulation, real-time simulation, and distribuited simulation based on the High Level Architecture (HLA). At Carleton, he is a member of the RADS Lab, and a founding member of the Embedded Systems Group. He is a member of OC-MISS, and the founding Chair of the Standards Activities Committee for SISO Canada.