A System-On-Chip FPGA Implementation of Embedded CD++

Mohammad Moallemi, Gabriel Wainer
Dept. of Systems and Computer Engineering
Carleton University Centre of Visualization and Simulation (V-Sim)
1125 Colonel By Dr. Ottawa, ON, Canada.
moallemi@sce.carleton.ca, gwainer@sce.carleton.ca

Keywords: embedded CD++, DEVS, real-time, FPGA, robot, hardware-in-loop**,** embedded systems

Abstract

The development of embedded systems with real-time constraints has been rapidly advancing in the last 20 years. Most existing methods are still hard to scale up for large systems, or they require expensive testing efforts. Embedded CD++ is a software toolkit that uses model-driven method to develop this kind of applications based on DEVS, a formal technique originally created for modeling and simulation of discrete event systems. Embedded CD++ is a framework to incrementally develop embedded applications, and to seamlessly integrate simulation models with hardware components. We have deployed this tool on a Virtex2 pro FPGA board and made use of different components of an FPGA device to upgrade the hardware control and simulation capabilities of this toolkit. The process of deployment and also execution of a case study control model will be explained in detail.

1. INTRODUCTION

Embedded real-time software construction has usually posed interesting challenges due to the complexity of the tasks executed. Most methods are either hard to scale up for large systems, or require a difficult testing effort with no guarantee for bug-free software products. Formal methods have showed promising results, nevertheless, they are difficult to apply when the complexity of the system under development scales up. Instead, systems engineers have often relied on the use of modeling and simulation (M&S) techniques in order to make system development tasks manageable. Construction of system models and their analysis through simulation reduces both end costs and risks, while enhancing system capabilities and improving the quality of the final products. M&S let users experiment with "virtual" systems, allowing them to explore changes, and test dynamic conditions in a risk-free environment. This is a useful approach, moreover considering that testing under actual operating conditions may be impractical and in some cases impossible. The goal is to develop a simulation environment which can operate in both virtual time and real-time and also capable of transferring the simulated models to the real environment. To achieve this goal the current simulation tool must be modified and new functionalities should be added in order to support real-time and also real environment hardware interactions.

FPGA boards are specific tools for embedded development which are cheap compared to single purchase of embedded equipments and also incorporate variety of embedded tools and components that make them suitable for these kinds of applications. The AP1000 FPGA board [1] is a Programmable logic—specifically field-programmable gate array or FPGA— board manufactured by Amirix™. The rapid and easy design implementation and low cost of such platform enables researchers and students to design and implement variety of embedded projects. These boards consist of memory blocks, microprocessors (soft and hard macros), multiplier and digital signal processing (DSP) blocks, embedded system IP such as bus architectures and peripheral components, and application-specific IP such as in DSP and telecom.

The Virtex-II Pro FPGA Prototyping Station, the predecessor to this station was the System-Level Prototyping Station (SLPS) is based on the Xilinx™ [2] Virtex-II Pro FPGA and includes one AMIRIX AP1000 development board installed in the 64-bit PCI-X slot of an IBM™ Intellisation Z Pro workstation that includes a dual Xeon processor core. With large FPGA gate capacity (44000 logic slices) and two embedded IBM PowerPC hard macros, as well as up to 1.4MB of on-chip RAM, this platform gives university researchers a high-performance, multiprocessor development environment.

The AP1000 PCI platform FPGA development board from Amirix™ is a PCI card that can be slid in to the PCI slot of an IBM™ workstation. The Xilinx™ Virtex2 pro FPGA is connected to DDR SDRAM SRAM, Flash Memory, Ethernet and other interfaces. The AP1000 is configured as a single board computer complete with monitor software and Linux OS.

By using the Amirix™ PCI platform FPGA development board, developers can build embedded systems and explore the architecture in the areas of Networking, Communications, digital Signal Processing, Image Processing, Industrial Controls, Instrumentation, test and measurement and etc.

In the rest of the paper will be introducing a DEVS (Discrete EVent System) based simulator and also a real-

time extension of this tool and its added capabilities and then propose the process and configurations required to port this tool on the AP1000 FPGA board and also run a case study model using this tool on the FPGA board.

2. DEVS FORMALISM

DEVS [3] is an increasingly accepted framework for understanding and supporting the activities of modeling and simulation. DEVS is a sound formal framework based on generic dynamic systems, including well-defined coupling of components, hierarchical, modular construction, support for discrete event approximation of continuous systems and support for repository reuse. DEVS theory provides a rigorous methodology for representing models, and it does present an abstract way of thinking about the world with independence of the simulation mechanisms, underlying hardware and middleware. A real system modeled with DEVS is described as a composite of sub-models, each of them being behavioral (atomic) or structural (coupled).

3. CD++ AND ECD++ TOOLKITS

CD++ [4], [5] is a modeling and simulation tool that implements DEVS [6] models simulation based on an abstract simulator mechanism. Atomic models are defined using a state-based approach (encoded in C++ or an interpreted graphical notation); while coupled models contain atomic models composition and interconnecting information of those atomic models. CD++ has been widely used in various applications from simple queuing systems to complex urban traffic systems or physical systems. CD++ employs the abstract simulators proposed in [3]. Message drives the simulation according to the scheduled time points. CD++ is built as a class hierarchy of models related with simulation processing entities. DEVS Atomic models can be programmed and incorporated onto the Model basic class hierarchy using C++. Once an atomic model is defined, it can be combined with others into a multi-component model using a specification language specially defined with this purpose. Different versions of CD++ have been developed to facilitate various applications.

- Stand alone CD++ implements DEVS and Cell-DEVS simulation.
- Parallel CD++ is aiming to enhance the performance of Cell-DEVS simulation by distributing calculation of different cells over multiple processors.
- Distributed CD++ is developed to facilitate the coordination of the different simulating engines in different sites through the standard distributed computing protocols.
- Real-Time embedded CD++ is constructed especially for Real-Time embedded systems. A timing feature of the Real-Time systems has been included in CD++ to check the timing deadlines of given points of the systems.

E-CD++ [7], [8], [9]has been developed based on RT-DEVS formalism [10], [11] which unlike CD++ that works in simulated environment with simulated time advance function, the former works in real-time manner with real-time time advance function. The inputs to E-CD++ can come from real input ports like sensors, thermometers, timers, or from event files. The outputs can be sent to external devices like motors, transducers, gears, valves or any other component. Hardware-In-The-Loop simulation technique has been used to integrate software with hardware. A comparison between the results of the control model on real hardware with real input and outputs and on the computer with input event files and output files can be done and the model design can be modified to obtain the desired results. Some of the added features to E-CD++ are: supporting GGAD notation, Real-Time functionality by implementing RT-DEVS formalism, flattened coordinator technique and added embedded functional capabilities.

Thus E-CD++ can be used as a controller as it is able to receive real inputs and generate real outputs. One of the advantages of E-CD++ is virtual and simulated model checking prior to real execution on the target.

Working on E-CD++ can be done writing C++ code in a text-based Linux environment with open source tools. E-CD++ will most likely be running on embedded platforms with minimum, or even none output peripherals, therefore the information required during development is rather limited for the developer from the intended platform. In order to improve the development and simulation experience, an IDE is provided for the E-CD++ simulator core that adds Embedded CD++ functionality (the original IDE plus the simulator is called CD++ Builder [12], and is built on the Eclipse Environment [13]as a plug-in).
The IDE for E-CD++ permits code reuse from the original CD++ Standalone version, sharing all the possible resources that the development environment has to offer from the later.

Since E-CD++ will be deployed in a different platform (Target) other than the one where it is being developed, cross-compilation for the Target is provided, as well as means of communication to the Target in order to download executable binary files, run the executable and debug remotely. The tool also remembers important preferences, i.e. last IP Address used if the connection is established through a Local Area Network, and other simulation configuration information that remains constant throughout the development process.

To achieve these features, new processes can be spawned from the CD++ Builder plug-in, each one parallel to the others but also following a certain order among themselves. For instance the project needs to be edited first in order to be compiled and generate an executable file. Only when this file is obtained it can be deployed to the embedded target, and only when this file is present in the

target it can be run remotely thorough a remote shell connection or remotely via a remote command. However, each process is separated from the other to give the user complete control over the development, for instance, the project can be compiled but not deployed and a previous version of such project can be executed for testing purposes.

Given code reuse from the CD++ Standalone version to the E-CD++ version is a central aim, an important development resource to be exploited is the CD++Modeler tool, an application that permits defining DEVS models graphically. CD++Modeler provides an alternative method to create DEVS models for CD++Builder without the need to use programming languages. The IDE extended for E-CD++ with the special features aforementioned, directly benefits from CD++Modeler, as it is a stand-alone application that generates models independently from the target execution environment they will be running on.

The E-CD++ features of the CD++Builder are: *Compile2Target* (a cross-compiling feature that compiles the model for the target environment while the host environment is different with the target one), *Telnet2Target* (opens a telnet connection between the host and the target environment), *Download2Target* (downloads the *modelfils*, *eventfiles* and executable file to the target environment), and *Run Simulation on Target* (runs the simulation remotely on the target). All of them are self contained JAVA classes which are called through the plug-in eXtensible Markup Language (XML) script. When the corresponding buttons in the IDE are clicked on, the XML script launches the corresponding JAVA class, which executes the intended task.

E-CD++ has a real-time simulation mode named *no-hardware* mode, in which all hardware interactions are disabled, thus making the user capable of real-time simulation of his/her model and verify its correctness. The other advantage of this mode is hardware failure detection. In case of external hardware malfunction, the user can disconnect the hardware and simulate the same environment signals using the *eventfile* events and test the model in *no-hardware* mode to keep track of simulation stages in order to find the malfunctioning or time limited hardware component.

4. PROPOSED FPGA IMPLEMENTATION

Figure 1 shows the AMIRIX™ AP1000 FPGA board hardware architecture [14].

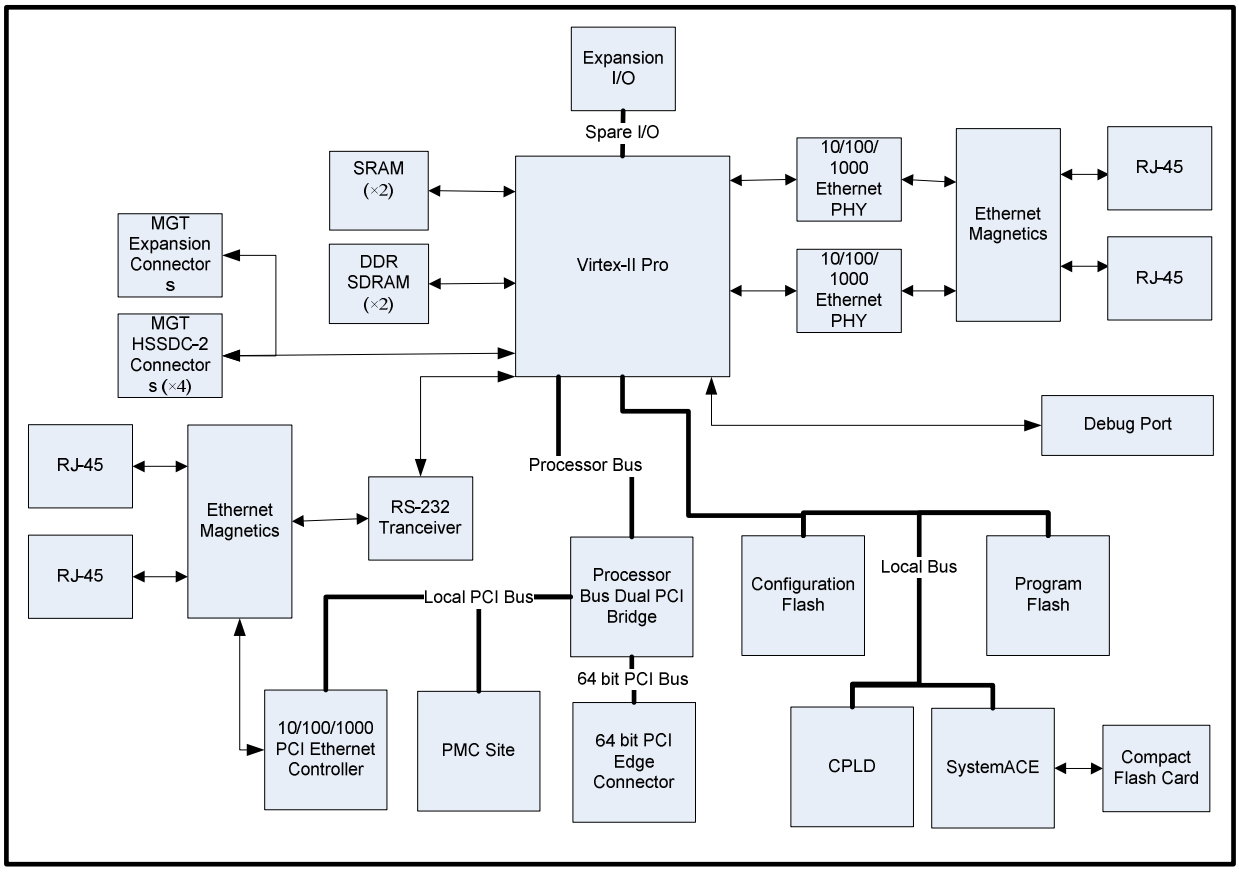

Figure 1- AP1000 FPGA Architecture Block Diagram modified from [15]

The FPGA element in our FPGA board is a Virtex-II Pro central FPGA element and has several interfaces to various devices. It has two DDR SDRAM banks which provide high bandwidth memory interfaces for the on chip processor. Two separate 18Mb synchronous SRAMs are also available, which can be accessed as a single 72-bit memory or as two completely separate 36-bit memory banks. They also provide high-bandwidth to the processor. Various peripheral devices are available through the local bus interface, including Flash memory for both program storage and FPGA configuration data. The SystemACE provides an additional means of FPGA configuration and the Processor Bus Dual PCI Bridge provides a portal to additional functions for the FPGA element. This bus provides the means to include a wide variety of I/O by installing a PMC module, as well as an Ethernet controller for network access. There are two Gigabit Ethernet physical layer devices connected directly to the Virtex-II Pro which provide additional high-speed network interfaces. Expansion I/O port provides additional expansion means, allowing either cabling or custom PCB daughter cards to be directly connected to the Virtex-II Pro. The PCI panel provides accessibility to the CompactFlash, PCI 10/100/1000 Ethernet, and RS-232D connectors. The remaining connectors are accessible from within the system chassis.

The PowerPC405 processor interfaces to the DDR SDRAM controller on the Processor Local Bus (PLB). The 16550 UART is present on the On-chip Peripheral Bus (OPB). The PLB-OPB Bridge allows access from the PLB to the OPB.

There are several ways to program an FPGA board. To provide broader support to a greater range of users, the AMIRIX^{TM} Prototyping Station is supported by two operating systems:

1. Microsoft Windows XP Professional, 32-bit edition is running on the IBM PC installed with the AMIRIX AP1000 board. EDK toolset from Xilinx^{TM} can be used to develop software applications, generate and download a bitstream executable file on to the board.
2. Linux for 32-bit x86 processors. The Linux installation running as a virtual machine using VMware's Player software on the Microsoft Windows XP Professional operating system and using ELDK [16] toolset to develop software applications and using an embedded Linux kernel running on the PowerPC processor on the FPGA board to run the application.

Figure 2 shows different layers of operating system and software applications that can be used on the FPGA board and the IBM host PC.

We used the second approach and downloaded a small configurable Linux kernel to the SDRAM memory blocks on the FPGA board using TFTP server software application and booted the Linux kernel on the PowerPC processor. This way we could have an embedded configurable operating system kernel that can be configured to include drivers and libraries for different FPGA components for any type of hardware control application. The other advantage of using a Linux kernel on the PowerPC is that the operating system kernel provides suitable drivers for all hardware components that are available on the FPGA board, thus making E-CD++ code transfer and reuse from PC to FPGA easier and also provides efficiency for hardware utilization. In our implementation, the Linux kernel operates on the PowerPC processor of the FPGA board, a virtual Linux runs on top of Windows OS using VMware software player which runs ELDK software application, and the Eclipse which provides development environment for E-CD++ and also cross-compilation for E-CD++. Windows OS provides HyperTerminal connection and runs TFTP server and VMware Player.

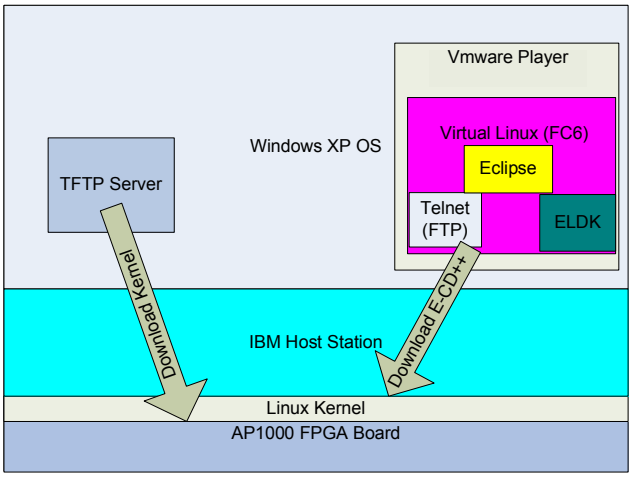

Figure 2- Architecture of the Software Environment

After configuring, building and downloading the Embedded Linux kernel on the board, ELDK tool has been used to compile and generate a compatible executable file for PowerPC processor. ELDK is an open source series of gnu Linux compilers that is used to develop the software application for FPGA board under Linux. Thus, it is suitable for E-CD++ in terms of providing cross-compilation environment in which E-CD++ uses Eclipse environment on the host PC and generates an executable file compatible with FPGA hardware.

The RS-232 UART I/O peripheral on the Virtex2pro FPGA provides communication interface between the AP1000 FPGA board and the Host PC. A serial connection between the RJ45 serial port of the board and the COM port of the

host PC provides I/O communication. HyperTerminal application on Windows OS at the host PC is used to boot the board using *uboot* boot loader application and render the outputs on the screen. A peer to peer Ethernet connection make it possible to establish a telnet connection between the Linux OS on the host PC side and the embedded Linux on the PowerPC side and download the executable binary file and model related files.

4.1. Robocart Case Study

We have implemented a simple control model that we have published previously in [9] using E-CD++ on the FPGA. This example model has been modified which uses a simple DEVS model for control purposes. The main goal of this example model is to test the Embedded functionality of the E-CD++ to control the behavior of the robocart to avoid obstacles in its way and continue operating with real-time inputs until the simulation time ends. We use real-time clock feature of E-CD++, receive real-time inputs and send out real-time outputs to the real hardware.

We have built a simple robocart shape device and have used appropriate C++ open source library [17] to control this device using E-CD++ toolkit. The robocart uses a touch sensor which detects obstacles in front of it. It also has an ultrasonic (sonar) sensor which can measure the distance between the sensor and an obstacle in front of it. The robocart has two motors which the direction and the speed of rotation of the motors can be controlled. Figure 3 shows the robocart from different angles.

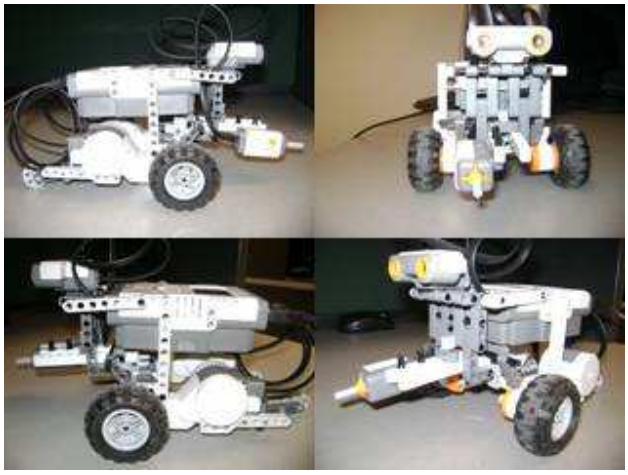

Figure 3- Robocart

Using above capabilities, we defined a simple atomic DEVS model to receive inputs from sensors and send outputs to the motors. For simplicity reasons, the model has only one input port which obtains its input from both sensors and differentiates the inputs of different sensors based on different value codes that is defined for each sensor and given to each sensor by *Input Generator* block (Another ideal case could be one input port per each sensor.) The model has one output port connected to both motors. Figure 4 illustrates the block diagram of the model and its input and output ports connectivity.

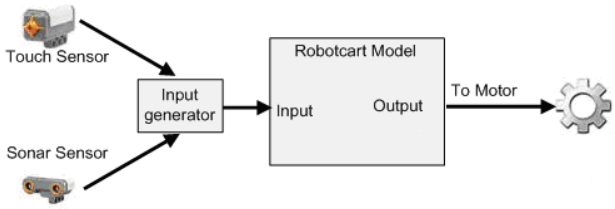

Figure 4- Robocart model

As mentioned before, E-CD++ uses *hardware-in-the-loop* approach to receive real-time inputs. In an iterative manner, the inputs from sensors will be available to the *Input Generator*. The input from touch sensor has higher priority than the one from sonar sensor; therefore, at each iteration of the loop, the input of touch sensor is checked by *Input Generator*. If there is an input from touch sensor, it shows that an obstacle has blocked the robocart. Thus, the input of sonar sensor will be discarded and touch sensor input value code will be sent to the model input port. Otherwise, the sonar sensor value will be obtained and checked and if it is less than a pre-specified value that is necessary to avoid bumping to an obstacle, the *Input Generator* generates an input with the sonar value (which is the distance from the sensor to the obstacle.)

Below is the DEVS model specification of the robocart:

M = <**X**, **Y**, **S**, **ta**, **δint**, **δext**, **λ**>

X: input ports: *in* (connected to *Input Generator*)

Y: output ports: *out* (connected to motors)

S: system states: *Moving Forward*, *Moving Backward*, *Turn Left*, *Turn Right*, and *Stop*.

ta: time advance function: handled externally by the root coordinator.

δint: Internal transition function: specifies the next action and finally resume the forward movement of the robocart.

δext: External transition function: checks for sonar or touch value coming from input port to carry out required state transition.

λ: Output function: sends appropriate commands to the motors.

Whenever there is an input, the external transition function is invoked and checks the input value. If the input is from touch sensor, in this case the robot should resume back to get the appropriate distance from the obstacle in order to do a turn action. Thus, the external transition function changes the state of the robocart to *Move Backward* and introduces a very small transition time (to avoid zero transition time conflict). The output function sends backward moving command to the output port and the internal state transition function changes the state to *Turning Left* or *Turning Right* and introduces a certain transition time for backward motion

to complete. After transition time is passed, the output function will send turning command to the output port and internal transition function will resume the state to *Moving Forward* and start a new pre-specified transition time for turn command to complete. Again after elapse time is expired, the output function will resume *Moving Forward* motion and the internal transition will do nothing.

The other case is the sonar sensor input. In this case again a very small transition time is defined (to avoid zero transition time conflict) and the state of the robocart will change to *Turn Left* or *Turn right*. When the transition time expires, the output function sends appropriate output to the output port for both motors to do the Turn action (one motor floats and the other continues rotating). The internal function resumes the state of the robocart to *Moving Forward* and defines another elapse time which is calculated based on the Turning speed and desired degree of turn in order for the robocart to remain in turning state in this period and complete the turn. When this transition time is finished, the output function sends appropriate commands to move the robocart forward (both motors rotating) and the internal function will do nothing.

4.2. Results

We modified E-CD++ source code to adapt with the specific gcc compiler version included in ELDK tool that is compatible with PowerPC4xx processor series. Then the PowerPC executable binary file has been downloaded to the SDRAM memory block of the FPGA board. The Robocart hardware communicates through USB port. Because AP1000 FPGA port does not have USB port we ran the simulation in real-time and *no-hardware* mode in which the inputs only come from *eventfiles* and outputs get saved in output files. The robocart *modelfile* and *eventfile* have also been downloaded to the SDRAM memory block. Figure 5 shows the execution outputs of the robocart model with three input events at 6^{th}, 9^{th} and 14^{th} seconds after the start of the simulation with 10, 1000, and 3 respective input values. 10 and 3 are considered as sonar sensor distances in centimeters and 1000 is considered as touch sensor input value in robocart model. Below these lines, the state changes of the model based on the input values are shown.

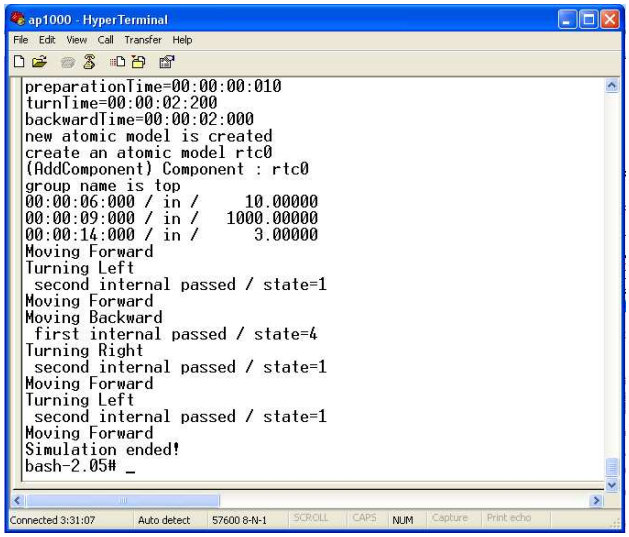

Figure 5- Robocart Model Execution Sequence on HyperTerminal Window

The AP1000 FPGA board is able to work as a single board computer, thus makes it suitable to be mounted on any external hardware as device controller.

Figure 6 shows the output file contents for another execution of robocart model in real-time and *no-hardware* mode with different *eventfile* events. We have defined 1 to 5 numerical values associated with *Moving Forward*, *Moving Backward*, *Turn Left*, *Turn Right*, and *Stop* states respectively. At the beginning of the simulation an internal transition is performed to start the motors to move forward. This output is shown in line 1. At time 00:00:01:633 a touch sensor event has been detected. After passing a small external transition of 10 milliseconds, right at time 00:00:01:643 the backward movement output has been produced (line 2) and after 2 seconds internal transition the next output which is *Turn Left* is generated (line 3). Finally after 2 seconds and 200 milliseconds the *Moving Forward* state is resumed (line 4). As you can see the outputs in lines2 and lines 3 are generated a bit later than the time expected which indicates hardware delay. There was also another touch sensor input at line 7 and a sonar input at line 10.

The concept of deadline for each transition is also visible in the output file.

```
1  Time: 00:00:00:028 (no dealine specified) OutPort: out PortValue: 1
2  Time: 00:00:01:643 DL: 00:00:04:620 (Suc) OutPort: out PortValue: 2
3  Time: 00:00:03:648 (no dealine specified) OutPort: out PortValue: 3
4  Time: 00:00:05:847 (no dealine specified) OutPort: out PortValue: 1
5  Time: 00:00:06:025 DL: 00:00:07:000 (Suc) OutPort: out PortValue: 4
6  Time: 00:00:08:226 (no dealine specified) OutPort: out PortValue: 1
7  Time: 00:00:09:843 DL: 00:00:12:816 (Suc) OutPort: out PortValue: 2
8  Time: 00:00:11:843 (no dealine specified) OutPort: out PortValue: 3
9  Time: 00:00:14:042 (no dealine specified) OutPort: out PortValue: 1
10 Time: 00:00:16:600 DL: 00:00:19:573 (Suc) OutPort: out PortValue: 4
11 Time: 00:00:18:799 (no dealine specified) OutPort: out PortValue: 1
```

Figure 6- Robocart model output file

This example proved Embedded functionality and real-time input capability of E-CD++ in a control project. Same example have been developed using real-time keyboard inputs to steer the robocart through its path.

5. CONCLUSIONS AND FUTURE WORKS

M&S techniques offer significant support for the design and testing of complex embedded real-time applications. DEVS theory can be applied to improve the development of real-time embedded applications. A lot of experiments that have been carried out using CD++, a DEVS tool that has been built following DEVS formal definitions, proved its advantage and integrity. The models were developed independently, and were later integrated at the modeling level. In this paper we proposed system-on-chip FPGA implementation of ECD++ (a real-time and embedded version of CD++ tool) that is able to simulate different real-time models and also be used as a controller in embedded applications. FPGA provides different hardware components and capabilities to our real-time simulator for control applications. It also has the advantage of being used as single board computer and run E-CD++ on any hardware environment for this purpose. We also verified our implementation using a hardware control simulation of a simple robocart and rendered the results.

Future works can include: Developing more complex and applicable embedded models with more advance FPGA boards, Other FPGA components of the board can be used to develop more specific models of specific applications.

6. ACKNOWLEDGEMENT

We wish to thank CMC™ [18] Corporation for the FPGA board donation and support to our lab.

7. REFERENCES

[1] AMIRIX™ Systems, Inc. AP1000 FPGA Development Board Users Guide. [PDF] 2007.

[2] Xilinx™, Inc corporation website available at: http://www.xilinx.com/.

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

[4] Wainer, G. 2002. “CD++: a toolkit to define discrete-event models”. In Software, Practice and Experience. Wiley. Vol. 32, No.3, pp. 1261-130.

[5] Wainer, G. et al. "CD++ A tool for DEVS and Cell-DEVS Modeling and Simulation. User's Guide". Draft. August 2004.

[6] ZEIGLER, B.; KIM, T.; PRAEHOFER, H. "Theory of Modeling and Simulation". Academic Press. 2000.

[7] E. Glinsky, and G. Wainer, "Definition of Embedded simulation in the CD++ toolkit."

[8] Y. H. Yu, and G. Wainer, "eCD++: an engine for executing DEVS models in embedded platforms." pp. 323-330.

[9] M. Moallemi, J. M. Gutierrez-Alcaraz, and G. Wainer, “ECD++ a DEVS based Embedded simulator for embedded systems,” in Proceedings of the 2008 Spring simulation multi-conference, Ottawa, Canada, 2008.

[10] Chow, A.C., and B.P. Zeigler. 1994. “Revised DEVS: A Parallel, Hierarchical, Modular Modeling Formalism”. In the Proceedings of the SCS Winter Simulation Conference.

[11] A. Chow, and B. Kim, "Abstract simulator for the parallel DEVS formalism." pp. 157-163.

[12] C. Chidisiuc, and G. Wainer, “CD++ Builder: An Eclipse-Based IDE For DEVS Modeling.”

[13] ECLIPSE, http://www.eclipse.org, Eclipse 3.2 Online Manual].

[14] CMC. Getting Started with the FPGA Prototyping Station. [PDF] 2007. Available at https://www1.cmc.ca/clients/search/product-details.html?id=48840.

[15] Inc., AMIRIX™ Systems. AP1000 FPGA Development Board Users Guide. [PDF] 2007.

[16] ELDK official website Available at http://www.denx.de/twiki/bin/view/DULG/ELDK.

[17] NXT++ main page available at http://nxtpp.sourceforge.net/index.php.

[18] Canadian Microelectronics Corporation (CMC™) website available at http://www.cmc.ca.