High-Level Circuit and System Simulation with Java

Ulrich Kaiser*

***Abstract* – Java, known from web pages (applets), is also useful for design of application programs.**

This paper presents how the object-orientated language Java makes modeling and simulation of electronic circuits and systems possible, resulting in a portable program containing a graphical user interface as well. Customer demos by fast prototyping and executable specifications can be built by means of this method.

1 Introduction

The increasing design complexity requires higher levels of abstraction; this holds for both hardware and software. Hardware description languages such as VHDL [10] and Verilog [11] are no longer alone sufficient for the specification and design of ASICs and larger systems [12].

Designers need ESL (electronic system level) tools and languages for fast prototyping of complex systems. Available today are [9] e.g. C++, Java, SDL, UML, SLDL, SpecC, SystemC, Superlog, CYN++, and others. (The proprietary design system [12] uses an extended version of C.)

Java is an object-oriented (OO), interpreted, robust, portable, dynamic, multi-threaded language offering high productivity [3-7]. Its object-oriented nature allows effective partitioning of algorithms. Java's high-level libraries (APIs) simplify software development, e.g. creation of graphical user interfaces. - Meanwhile even a Java-to-Verilog compiler (Forge, [8]) is available that allows architectural synthesis.

This paper presents a high-level model of the transponder TMS3792 (DST2) [1,2] together with a reader unit and a graphical user interface (GUI). The GUI consists of several text areas, text entry fields, radio buttons; furthermore, a plot area showing the transponder's supply voltage versus time during receive and transmit phases with the distance as parameter (Fig.1) is implemented.

This kind of executable specification can be executed on all computers providing a Java Virtual Machine (JVM) for interpretation of Java byte-code.

* Texas Instruments Deutschland GmbH,
D 85350 Freising, Germany
Fax: +49 8161 80 4477
Tel.: +49 8161 80 4109
Email: d-kaiser@ti.com
URL: http://www.ti-rfid.com

2 Properties of the Java language

Java is a relatively new language [6,7], built in 1995. In a world of objects Java provides a way to express concepts by means of object-orientated programming (OOP) using the computer as a 'mind amplification tool' [4], as an expressive medium. It allows describing the problem space in a high level of abstraction.

OOP with Java has the following characteristics:

- Classes of objects store data, react on requests.
- The objects interact by sending messages (methods) to each other.
- Each object has its own memory with instances of other objects.
- The objects are strongly typed (class name).
- All objects of a particular type can react to the same messages (methods). This is possible through the concept of 'late binding' and polymorphism.
- Inheritance is supported by means of class extension.
- Classes and their methods can be access controlled (public, private, protected).
- Garbage collection (release of memory of objects no longer in use).
- Exception handling in case of run-time errors.
- Multi-threading (separately running program pieces).

Java can be used to model complex systems. The OOP approach allows control of the complexity by proper partitioning of the problem space. Access restrictions (private, protected) help to avoid unwanted side-effects.

There is also a specialized compiler available, that transforms Java byte-code programs to Verilog HDL descriptions [8]. This allows to design digital circuits in a very high-level language and to perform silicon compilation afterwards with design flows that are already in place.

3 High-level modeling

The development of integrated circuits (ICs) made its way from single transistor design over use of schematic entry tools, hardware description languages (HDLs), and silicon compilation to designs of type 'System-on-a-Chip' (SoC)[12]. For

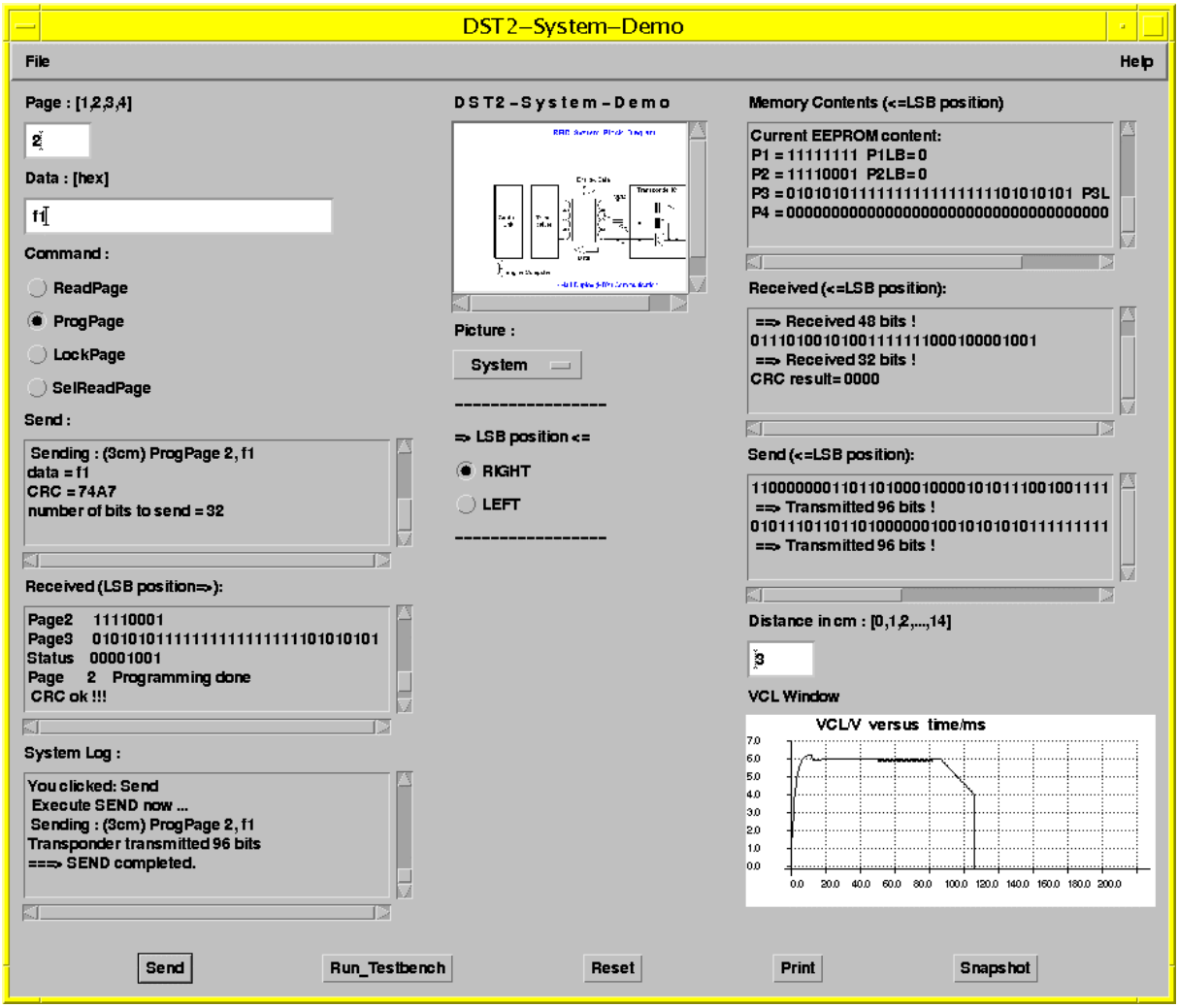

Figure 1: Graphical User Interface (GUI) of the Transponder System simulator

ICs with more than 10 million transistors HDLs alone are no longer sufficient for the design of such complex systems. Additionally, the chip designers are faced with shorter and shorter design cycles in order to meet the market windows of certain products (e.g. hard disk drives). Therefore, solutions are searched for in the area of system design using capabilities that are not available in today's HDLs.

Object-oriented syntax and semantics is of great help when the engineer wants to deal with a design as an ordered collection of objects. He needs tools for description of interfaces and protocols without the necessity to describe any real hardware at that phase of design in order to obtain a high level of abstraction :

– With their classes and methods OO-languages such as C++ and Java are very effective for system design.
– For capturing system structure and early-stage behaviour analysis an OO-diagram language set such as UML [13] is valuable, too.
– To cover all needs for SoC design probably a certain set of design languages instead of a single language must be used [9].

The chip system designer prefers to start with a high-level abstraction model; then the desire follows to organize the collected design requirements into a model that is executable, i.e. can be simulated directly. High-level data flow and mathematical algorithms have to be captured and tested, especially the critical sections. In the next steps partitioning and the mapping to an initial hardware architecture can be performed. Further performed system simulations and iterations lead to refinements of the architecture, constraints and system behavior.

4 DST2-System demo realization

For a system of a reader unit and a transponder the following classes are defined for the circuit and system model (Figure 2):

TirisSystem	top-class, covering all constants, has main method with instances
ReaderUnit	model of the transceiver
CR	set of simple conversion methods
Gui	Graphical User Interface, uses Frame, Canvas, TextField, TextArea, MenuBar, MenuItem, WindowEvent, Panel, Button, ActionEvent, ActionListener, Checkbox, CheckboxGroup, ItemListener, ScrollPane, Choice
Crc16	Cyclic Redundancy Check with CCITT algorithm, is used for both reader and transponder
Encryption	Encryption algorithm for DST2, is used for both reader and transponder
EEPROM92	EEPROM model for the DST2 chip TMS3792
TrpControl	control model of the transponder DST2 for receive, programming and transmit phases.
Transponder	object containing instances of *EEPROM92* and *TrpControl*.
TestBench	stimulates *Transponder* and checks correctness of response; owns a main method, too.

The *Gui* allows the communication with the *ReaderUnit* model. Read- and Write-commands can be sent to the Transponder model. This in turn responds to the requests and sends back data to the *ReaderUnit*. In small panes of the *Gui* the data sent (left) and received (right) is displayed.

On the left side the *ReaderUnit* is represented. It contains the text field for input of page number and data to be sent to the transponder. Four 'radio-buttons' (check box) are designed for selection of commands. In the lower left corner a text area field displays the system simulation messages.

In the middle part a scroll pane is embedded that allows to show some system drawings; they can be chosen by means of a menu field below. Two other 'radio buttons' allow switching between 'LSB right' and 'LSB left' display of the binary strings in the text areas.

On the right side the *Transponder* has its fields. One text area shows e.g. the current memory content. So, the new EEPROM status can be seen immediately after the programming command is executed successfully.

Furthermore, the supply voltage in the transponder is modeled versus time in dependence of the distance between reader unit and transponder. A function plot in a specialized pane shows the different phases during the communication: Charge-up, data transmission (down-link), additional power transmission, data transmission (up-link), discharge (power-down). – The methods of the *TrpControl* have to react also in dependence of the actual distance, entered by the user, e.g. programming of an EEPROM page fails, if the distance is too large.

At the bottom side some rectangular buttons are implemented. The most important one is the 'Send' button, because it is used to stimulate the *ReaderUnit* to send a message to the transponder.

The button "Run_Testbench" starts the class *Testbench*. This in turn reads at first a file containing stimuli data and reference data – it controls the test bench. For every test case a stimulus is sent to the *Transponder* object and its response is collected. Then, this response is checked against the reference data. – It is also possible to run *TestBench* without the *Gui*: The 'main' method of *TestBench* instantiates a *TirisSystem* and a *Transponder*; then, the stimuli are sent to *Transponder* with the parameter setting guiActive = GUI_OFF.

In any case, the programmer can print messages into the transcript of the program. This is helpful during the debugging process.

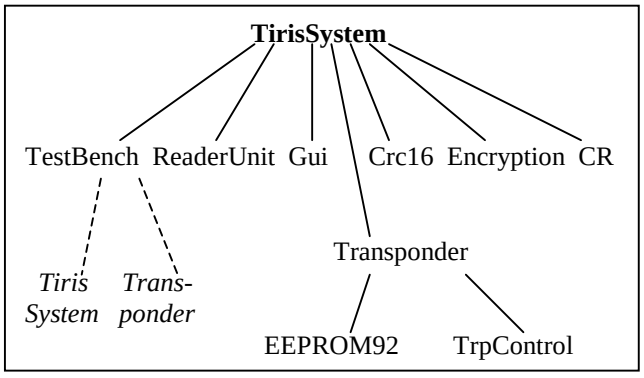

Figure 2: Class Structure and Instantiations

5 Java code examples

The instantiation of a *Transponder* object is:

```
Transponder trp1 = new Transponder();
```

This *Transponder* object `trp1` creates automatically its own instances of *EEPROM92* and *TrpControl* including their related data structures:

```
public class Transponder extends
                                TirisSystem {
  EEPROM92 eeprom = new EEPROM92();
  TrpControl cntl = new TrpControl( this );
```

The keyword 'this' allows here to pass the reference of the created *Transponder* object down to the *TrpControl* object. *TrpControl* contains a variable 'myParent' which is set during its instantiation by means of a so-called constructor method

```
public class TrpControl extends
                                TirisSystem {
  Transponder myParent;
  // constructor
  public TrpControl( Transponder p ) {
     this.myParent = p;
  }
  ....
```

Later, *TrpControl* can get access to the EEPROM by means of that reference; e.g. program page 2

```
myParent.eeprom.setPage2( b8 );
```

or e.g. read EEPROM page4 :

```
b40 = myParent.eeprom.getPage4();
```

The variables `b8` and `b40` are arrays of bytes; each byte holds either 0 or 1 for the bit information, but allows also the representation of other values such as `UNDEFINED` or tri-state values.

The concept of 'late binding' and 'polymorphism' is used e.g. for the second constructor of *Transponder*

```
Transponder( byte[] p3 ){
        eeprom.setPage3( p3 );
}
```

which is called e.g.

```
Transponder trp2 = new Transponder( b32 );
```

and creates an instance with a second object and non-default content of Page3. – One could for example create an array of ten *Transponder*s, each one with a different Page3 content, and the Gui could get a set of ten 'radio buttons' so, that the user might switch easily between different *Transponder* object instances.

6 Conclusions

A high-level simulator, written in Java, has been presented. Java's strength for object-orientated software design and its powerful libraries enable the engineer to built prototypes quickly and portable. And it is possible to build 'executable specifications' by means of Java. – Sure, Java is not **t h e** solution to all problems, but is one useful member in a set of system description languages.

Outlook, next steps planned :

- Increase the number of test cases,
- modify some routines for easier re-use,
- interfacing to HDL simulator for e.g. Verilog or VHDL,
- replace routine 'trp.processing' by "Forge-routines" and generate a Verilog HDL description,
- obtain know-how about UML for Real-Time Systems
- Work on proper description and modeling of 'anti-scenarios', i.e. forbidden scenarios.

References

[1] J. Gordon, U. Kaiser, T. Sabetti, A Low Cost Transponder for High Security Vehicle Immobilizers, Proceedings of ISATA, Florence, Italy, 3 - 6 June 1996, Automotive Electronics, 96AE001

[2] U. Kaiser et al., A Low-Power Digital Signature Transponder IC for High Performance RFID Authentication, Proceedings of European Conference on Circuit Theory and Design, ECCTD'99, Stresa, Italy, Aug. 29-Sep. 02 1999, pp. 45-48

[3] ECS, Teachers Corner, TU Denmark, Javalab, Lars Drud Nielsen, http://eswww.it.dtu.dk/~c49102/javalab.html

[4] B. Eckel, Thinking in Java, 2nd edition, Prentice-Hall, 2000

[5] D. Flanagan, JAVA Examples in a Nutshell, O'Reilly, 1998

[6] Java documentation, http://java.sun.com/docs

[7] Java product, http://www.sun.com/java

[8] Forge Hardware Compiler, Xilinx, http://www.lavalogic.com

[9] G. Moretti, Get a handle on design languages, EDN Europe, June 2000, pp. 54-65

[10] VHDL Language Reference Manual, IEEE Standard 1076, 1993

[11] Standard Description Language Based on the Verilog Hardware Description Language, IEEE Standard 1364, 1995

[12] K. Wakabayashi, T. Okamoto, C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective, IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems, Vol. 19, No. 12, Dec. 2000, pp. 1507-1522

[13] B.P. Douglass, Real-Time UML (2nd Edition), Developing Efficient Objects for Embedded Systems, Addison-Wesley, 2000