STUDIA INFORMATICA
Nr 1(8) Systemy i technologie informacyjne 2007

Monitoring of State in Program Simulator of Electronic Device

Piotr Czeberkus, Alexander O. Timofeev
Institute of Computer Science, University of Podlasie,
ul. Sienkiewicza 51, 08-110 Siedlce, Poland

Abstract. The article presents the interactive method of state monitoring of electronic device in autonomous simulated application. Form of electronic device state presentation is discussed. The information about state is presented by means of a special template, which can include both text and graphics. Author of model can decide about the format of displayed results. Implementation of the subsystem for monitoring electronic devices states, simulation course and observation of results are described. It is coming in sight, that the interactive monitoring of simulated devices state by means of text and graphics is readable, intelligible and more user friendly, and also it helps to understand better the processes, which take place in simulated devices and it is a good solution for didactics reasons.

Keywords: simulation, electronic device, state, monitoring of state.

1 Introduction

The question of computer simulation of dynamic device is current from for a few dozen of years. The multiplicity of various program simulators has already appeared.

However, the number of good solutions enabling to display simulation results interactively and originally is still insufficient. It is difficult to use them for teaching since they are complicated and considerably complex.

It is desirable, that the system would enable user to observe internal state of the simulated model in each discrete period. Often, the simulator is expected to do more than show the simulation results in rigid tabular form without additional explanation, especially when it is used in didactics.

The user of simulator will carry benefit maximally from application, when it presents results in clear and readable way with elements of graphics and with comments.

It is recommended, that each device component shows its own state in separate window and the user can control simulation process in the way he needs.

Simulating system Amethyst meets mentioned expectations. It generates the autonomous device simulator, which stands out above the other solutions [1, 2].

2 Form of electronic device state presentation

In Amethyst system, the information about state is presented by means of a special template, which can include both text and graphics. The template is composed of 256 ASCII symbols in the form of pseudo graphics with capability to insert images from file.

Author of model can decide about the format of displayed results. He can easily built the template in the form of device scheme and affix bitmaps with photos or drawings.

Each properly constructed simulator must have a simulation monitor. The monitor is usually a huge and powerful program that registers model event times and directly executes appropriate actions in specified order.

In order to decrease the size of monitor, thus also the size of entire application, processing of delays and filtering of signals in Amethyst system have been decentralized.

It is assumed that the model of each component monitors events on its inputs and outputs. In each step, it must calculate time of a next event and send the result to higher level in hierarchy. Once calculated and sent to the top of component hierarchy the time of a next event is applied to entire model. The monitor compares this time with the global system time and unless the end of simulation is reached it proceeds to the next step.

The application provides following simulation control function:

- start of simulation in continuous mode,
- perform transition to the next event,
- stop of simulation process,
- start of simulation for definite period,
- perform immediate transition to the last step of simulation,
- return back to initial state.

Simulation results are also observed in decentralized manner. Components of model show their state independently in specific format. It requires designing separate state presentation windows for each component. Components, which are not placed in library of system, must be first defined by the Amethyst's standard description, and then placed in the system library.

3 Method of description of electronic device

Accepted method of standard describing electronic devices consists in creating a text file with component definition and program files. Program files are written in high-level programming language, e.g. C++, and include procedures, which simulate the functionality of components.

The file with component definition has an extension “.ame”. Its content is divided into base text and control text.

Control text describes types of scheme fields and determines how they should be displayed.

Base text contains initial values of dynamic fields. It is usually made as a pseudo graphical drawing that can also mark places for external images.

A subsystem for state monitoring gets the information from proper sections of file *ame*. These sections contain data in the form of text and drawings that will be displayed during simulation.

System must read all fields, process and change them depending on course of simulation, and then display them in state presentation windows.

In order to know how to interpret particular rows of data they must be properly marked and described. In the end of each row containing such data, there is a description encoded in the way intelligible to system. This description is called *control text*.

Dynamically changing row fragments are called *lexemes*. Some rows are not the source of information used in simulating process and do not change dynamically. They can be just fragments of template and include drawings, borders of maps, tables etc., therefore, they do not require the control text.

After preparation of model definition file it is necessary to program the simulated object. It requires creating source files named identically as file *ame*.

Choice of programming language should depend on environment and compiler used later for generating a project of an autonomous application.

Programming functionality requires preparation of two structures responsible for storage of model parameters and variables displayed in state presentation windows.

The variables must strictly correspond to lexemes from model definition file. It means that variable types must match appropriate lexeme types and should be declared in the same order.

The next step is declaring the model class containing pointer on structure with state variables, pointer on structure with parameters and standard simulation procedures described in Amethyst's documentation: *fCreateUnit*, *fSet*, *f1* and *f2*.

Function *fCreateUnit* should contain operations being executed during first initialization of model. Since this function overrides its significant base class equivalent the inherited function should be called explicitly within user function body.

Procedure *fSet* is executed every time the simulation monitor is demanded to initialize the model. It is a proper place to assign values from parameters section to variables representing model's state.

Procedure *f1* assigns values to output variables on the base of model internal variables. It also can contain instructions of decentralized monitor for checking and setting time variables.

Procedure *f2* applies all internal changes of simulated model. It is an intended place to include a code reading data from inputs and calculating values of internal variables comprising the model's state.

Model implementation files can also include many auxiliary variables and procedures.

4 Implementation of the subsystem for monitoring electronic devices states

Both the system's module responsible for displaying simulated objects and the simulation monitor function together creating a subsystem for monitoring states.

The information presented to user is updated directly in each step of simulation and is initiated by monitor's thread. It is stored in special program structures and must be correctly refreshed and showed whenever is changed.

Because the forms of presentation of individual elements and variables could be different it is necessary to store the separate information about them. Hence, the information of simulated device state is divided into two types: essential information and metadata.

The essential information concerns dynamically changing values of simulated device, e.g. tensions on pins, times of events, values of buses and registers etc. Usually it is represented by numerical or text variables and it is placed in source files defined by model's author.

The metadata concerns types and styles of displayed values. It includes the information about used units of measure, colors and typefaces of fonts, positions on screen etc. The information is taken from file *ame* with hypertext definition of component.

It is also important to arrange and store mentioned information in organized form, so that operations of reading and updating can be intuitive and fast.

This goal has achieved by introducing classes of models containing fields describing state and methods responsible for simulation. As a result each simulated component has represented by an object of specific class.

Before being displayed the information presented to user takes many intermediate forms. In order to finally display results in readable and intelligible manner the special functions were defined to convert information from one form to another.

Process of conversion has been illustrated on Figure 1.

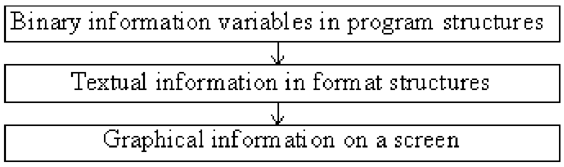

Binary information variables in program structures
Textual information in format structures
Graphical information on a screen

Figure 1. The sequence of conversion of component state information

The simulation monitor invokes procedures updating binary values of variables in program structures, i.e. internal variables of simulated object. However, the subsystem for displaying state does not operate these structures explicitly. It uses intermediate form shown as a middle block on Figure 1.

At this stage, the information takes a textual form, which can be easily illustrated graphically. Each row of text is placed in separate structure along with data about its format, that's why it is called “format structure”.

Conversion of binary information into text is performed just before displaying. Operation of displaying is performed in graphical mode and requires efficient method of drawing on a screen.

Solution currently implemented in Amethyst consists in drawing components of model directly into the bitmap memory, and next displaying whole image in a window. This method is fast because it performs low-level operations on memory blocks and cells and uses functions *API Windows*. It omits whole compound structure of *VCL* (*Visual Component Library*) objects.

Due to double buffering, undesirable flickering effect and on-screen picture distortion usually appearing during refreshing and scrolling window contents have been eliminated.

Direct writing to on-screen image memory requires using low-level instructions and special functions of graphic card driver. This is a fastest possible solution, but poorly universal and involving specialized graphics libraries, such as DirectX or OpenGL.

In case of this application, it is redundant relative to real requirements of program, whereas utilization of basic graphics interface *GDI (Graphics Device Interface)* has turned out to suffice and provide relatively efficient communication between program and output device.

Convenient method of presenting device's state graphically is using bitmaps *DIB (Device Independent Bitmap).* As the name indicates, this is device independent format, besides it is well - documented and implemented in standard libraries of Windows operating system.

Writing to bitmap memory is performed in the order determined by occurrence of format structures and its lexemes, row-by-row, column-by-column. Each symbol from format's text is drawn separately according to defined typeface of font.

Fonts are stored in special files in which each row corresponds to one ASCII character and consists of sequence of hexadecimal numbers. The result of loading font file into memory is an area of bytes where each single bit decides about visibility of one pixel.

Covering bitmap with characters is made by finding a proper place in font memory area, and then using nearest area of bits as a Boolean mask (Figure 2). An offset of the proper place in memory is calculated on the base of character position in ASCII table.

A size of the mask depends on loaded typeface. Pixels are drawn in places where corresponding bits take a value of „1”.

The color of each pixel is defined in lexemes. If particular fragment of text does not belong to any lexeme, then default color is applied.

An exception to the rule illustrated on Figure 2 occurs in case of inserting external images. The fragment of bitmap memory is filled directly with data coming from loaded bitmap file by copying it pixel by pixel in the ratio of one to one. Designer of model should choose such bitmaps that perfectly fit to the rest of image.

The last operation in process of displaying is moving created DIB bitmap to the context of device associated with presentation window. After all the actions relating to showing state have been done, the simulation monitor takes over the control and determines next cycle of computations and presentation.

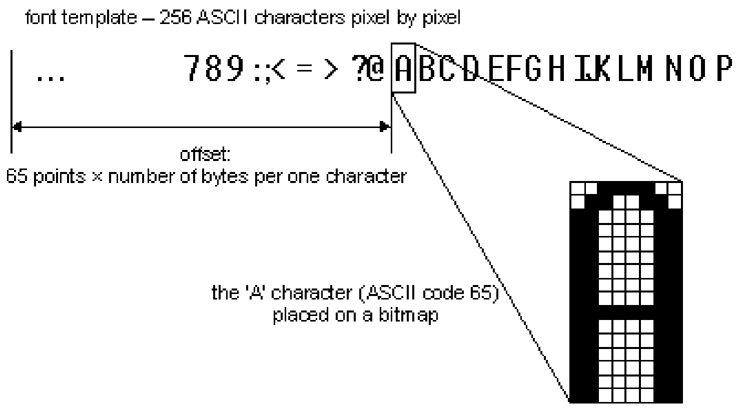

font template – 256 ASCII characters pixel by pixel
...
789:;< = > ?@ A BCDEFGHIJKLMNOP
offset:
65 points × number of bytes per one character
the 'A' character (ASCII code 65)
placed on a bitmap

Figure 2. Painting of text in bitmap memory

5 Simulation course and observation of results

A simulation is preceded always by the operations of reading and analyzing model definition file. They are performed every time the monitoring process is instructed to initialize model and restore initial values. A special procedure performs parsing the file and fills format structures with data.

The format structures contain base text and information about lexemes, i.e. dynamic fields relating to symbolic or physical elements of device. This information includes lexemes types, values, position in base text and preferred methods of presentation.

The structures are filled in on the base of control text. When binary form of model comprised of format structures is ready the program performs initialization of simulated object by invoking particular procedures programmed by model's author.

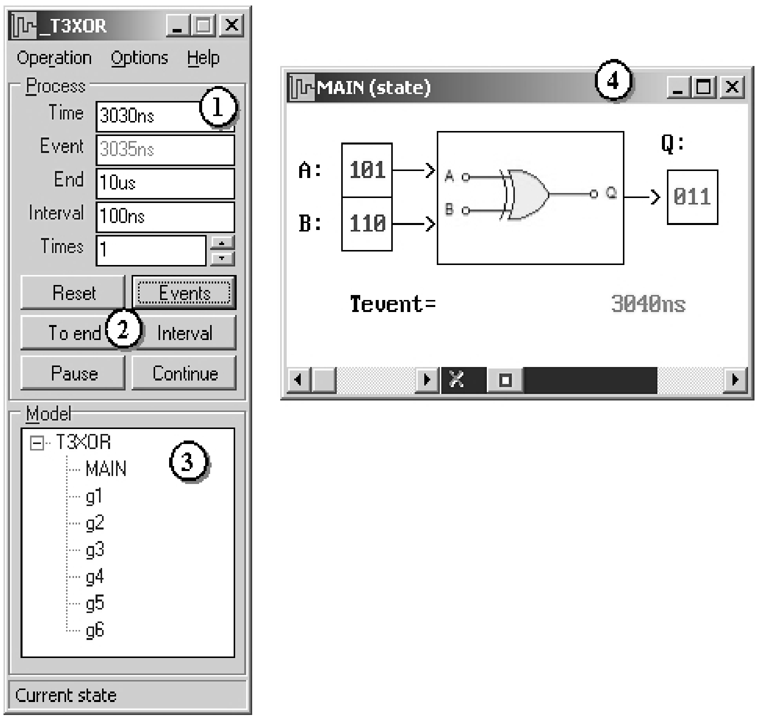
After successful initialization, the monitor's thread is started up in suspended mode. Resuming the thread is caused by a proper user action. A monitor's body constitutes a loop, which is running until the maximal defined simulation time is exceeded. Inside the loop, procedures of two phases of simulation: *f1* and *f2* are executed.

At the end of each simulation step, i.e. after executing single loop of the monitor, the content of the user interface is updated. If the monitor works in “to end state" mode then refreshing of interface is performed just once after the end of simulation and only state that is showed is the final state.

The interface presenting device state is user - friendly. It consists of several windows invoked separately for each atomic component of a device. User decides, which windows he wants to open and observe.

The content of each currently opened window is refreshed dynamically on the base of updated format structures.

The user can control to simulation using buttons placed in the main application window. In order to observe simulation results and follow dynamic model state user opens separate presentation windows for particular device elements from a component tree (Figure 3).



Time 3030ns
Event 3035ns
End 10us
Interval 100ns
Times 1

A: 101
B: 110
Q: 011
Tevent= 3040ns

Figure 3. Autonomous XOR gate simulator created in Amethyst system.
1 - fields to editing discreet time parameters, 2 - simulation monitor control buttons,
3 - components tree, 4 - state presentation window

The meaning of interface parts is as follows:

- *Time* - field showing current simulation time;
- *Event* - field displaying time of next event;
- *End* - field showing a maximal simulation time;

- *Interval* - field defining a period, after which simulation has to be stopped;
- *Events* - button causing transition to next state;
- *Interval* - button starting up simulation during;
- *Times* - if button *Interval* is pressed it denotes a multiplier for time value from field *Interval*; if button *Events* is pressed it specifies a number of events that have to occur before simulation is suspended. For example, if field *Interval* has a value of 10ns and field *Times* has a value of 4 then clicking on button *Interval* will cause start of simulation for 4 * 10ns= 40ns;
- *Reset* - button resetting device model;
- *To end* - button causing start of simulation without stops until time entered in field *End* is reached;
- *Pause* - button suspending simulation;
- *Continue* - button resuming simulation.

Within a device state presentation window (Figure 3), it is possible to observe dynamically changing values of model variables embedded in the symbolic device scheme.

6 Conclusion

The interactive monitoring of simulated devices state by means of text and graphics is readable, intelligible and more user friendly. It helps to understand better the processes, which take place in simulated devices and it is a good solution for didactics and other cognitional reasons. Such method is implemented in the simulation system Amethyst being a generator of interactive autonomous simulators.

References

1. A.O. Timofeev (2004): Computer production of programs for simulation of dynamic systems. Proceedings of the 15th International Conference on Systems Design (7-10 September 2004, Wroclaw, Poland). Vol. 2. Wrocław, Oficyna Wydawnicza Politechniki Wrocławskiej, 2004. Pp. 91-95.
2. P. Czeberkus (2007): Design and implementation of an electronic circuits simulator. A subsystem for monitoring circuits states. Master's thesis (in Polish). Akademia Podlaska, Siedlce, 2007.