The Virtual Printer Designer

Mikio Takashi

In recent years, there has been growing emphasis on methodologies aimed at rationalizing design, such as parallel design or collaborative design methods, and at Oki Data we have developed and implemented a design support tool for this purpose – the "printer simulator" (or "virtual printer")[1].

As well as outputting statistical data indicating the cache error rate, bus occupancy rate, and so on, the virtual printer also provides a visualization of the internal data processing inside the printer, allowing designers to identify the processor load and the operational state of the hardware modules. This has been very beneficial in analyzing the data processing capacity of the printer.

Meanwhile, printer system design has shifted to systems which improve data processing capacity by building some of the processes into hardware modules to achieve higher processing speeds, and running these hardware modules in parallel with the processor.

However, in order to run the hardware modules in a virtual printer, the procedures must be converted into models and encoded so that they function as a component of the virtual printer. In the past, hardware modules have been used sparingly and manual encoding for them has not proved to be a problem. Now, however, they have come to form a much larger part of printer design and manual encoding represents an impediment to the original object of parallel design. Therefore, virtual printers are currently limited to use in confirming appropriated design elements, such as tuning firmware after product design, or changing the processor operator frequency or cache size.

To resolve this problem, we have built two tools which cut the manual coding tasks. One of these is a virtual printer designer, which depicts the combination of the various components – processor, bus, hardware modules, etc. – in block diagrams and generates virtual printer source code, and the other is a logic editor which creates models of the hardware modules by means of an input method based on state transition diagrams.

Virtual printer designer

The virtual printer is a type of command level simulator, which comprises a simulator control unit for managing the whole simulator, software models for replacing the hardware components, such as the processor, bus, logic, etc., and a viewer for depicting the information relating to each model.[2] Conventionally, after designing the hardware system, in the LSI design stage, source code written with Verilog is used as a basis for manual definition of C++ source code. The virtual printer designer (hereafter, "Designer") uses state transition diagrams and block diagrams to define the hardware, instead of Verilog source code, and it generates the C++ source code automatically. (Fig. 1)

The Designer requires the same input tasks as a normal hardware design process, namely, block diagrams, memory maps, I/O settings, etc., using window screens.

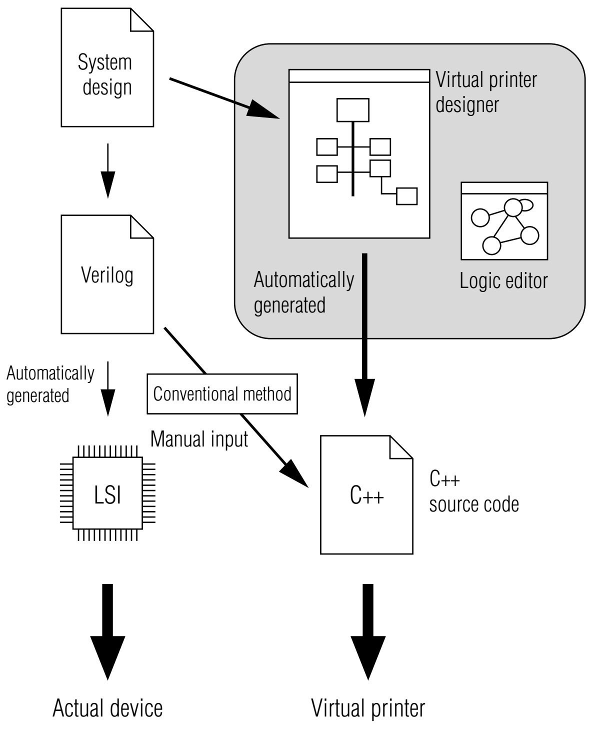


Fig. 1 Flow for creating virtual printer

Fig. 2 shows the window layout in the Designer. As well as the graphic editing window for positioning and wiring the various models, there is also a tool selection window, memory map window, and different logic definition windows for each model. A "logic" component is a new hardware module created by the user, and the logic definition window is used to set the various parameters for that hardware module. In Fig. 2, the I/O register definition window, which is an extension of the logic definition window, is displayed, along with a detailed settings window. These various windows are described in more detail below.

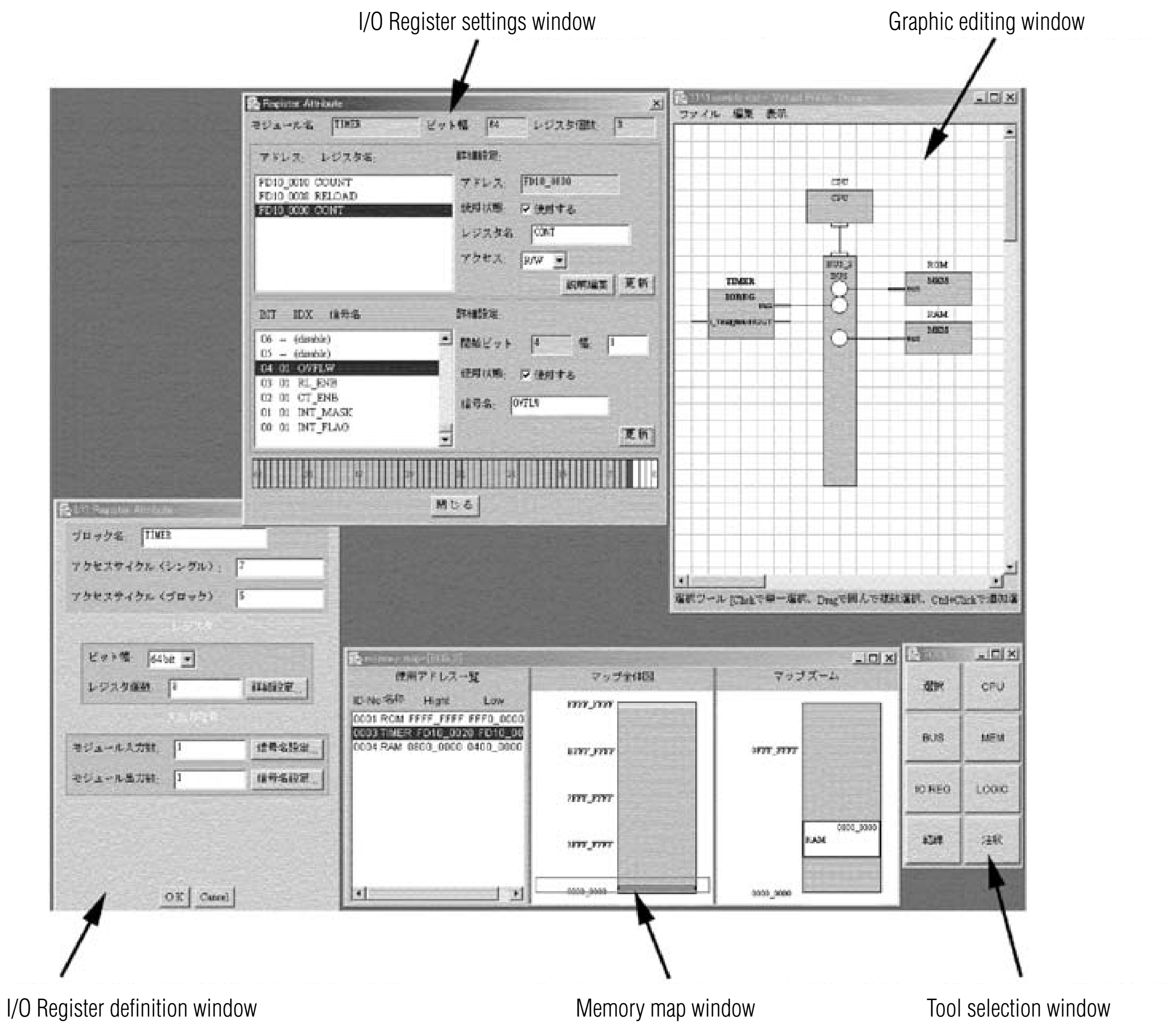


Fig. 2 View of Virtual Printer Designer

(1) Memory map window

A memory map defines the address space of the processor bus. Generally, a number of hardware modules are connected to the same bus, and the processor manages these modules by allocating each one to a certain range of the address space. In the virtual printer, a bus model is used for the function of selecting the models connected to the address space, on the basis of the address information. When creating the source code, the settings in this window are generated automatically by deduction from the bus model classes.

(2) Logic definition window

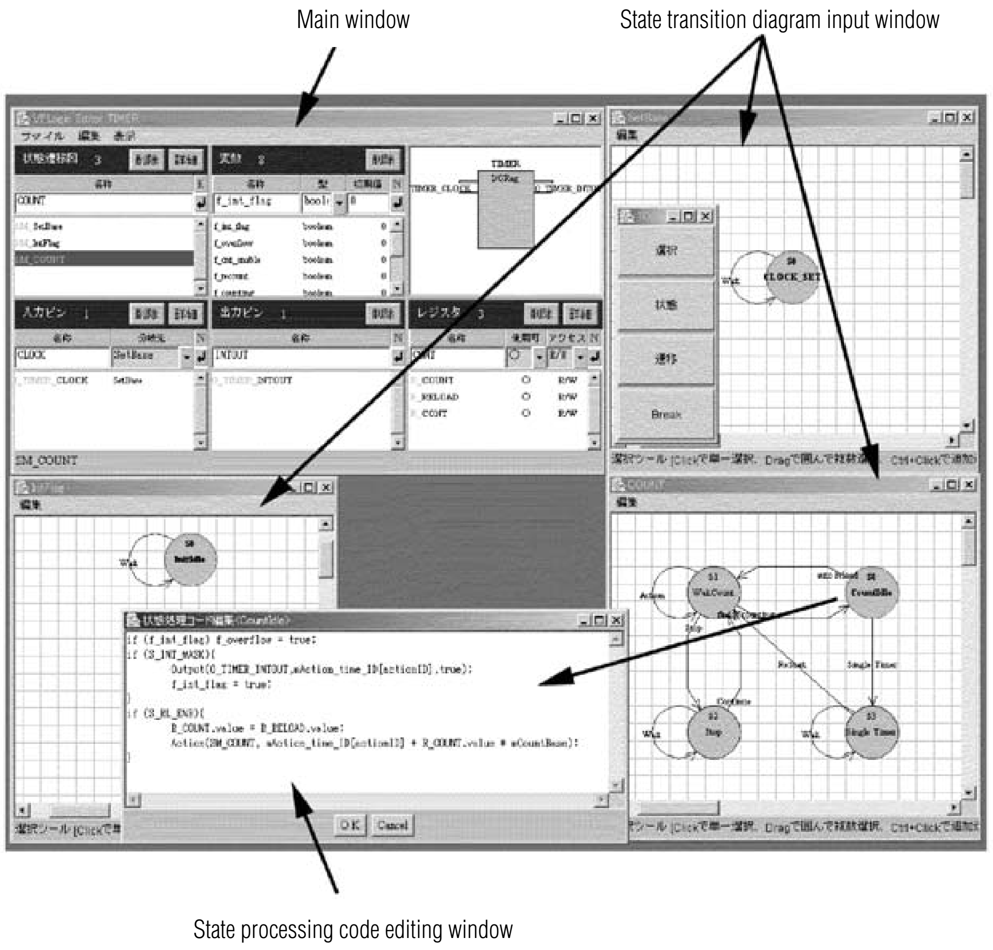
This window is used to define the number of signal input and output ports between logic components. Essentially, the models handled by the virtual printer are all logic models, apart from the processor and bus. The concept of logic models is described in more detail in the section on the logic editor.

As well as the electronic components, the models for the engine, printing paper, and so on, are also derived on the basis of these logic models, so that they can be run as constituent parts of the virtual printer.

The logic editor described below is used to create the operations inside each logic component.

(3) I/O register definition window

A logic component connected to the bus must always have a register which allows it to be accessed by the processor. In the I/O register definition window, besides the logic definition, settings are also made for the bit width of the register, the access time, and the number of logic components.


Main window
State transition diagram input window
State processing code editing window
OK
Cancel

```
if (f_int_flag) f_overflow = true;
if (S_INT_MASK){
        Output(O_TIMER_INTOUT,mAction_time_ID[actionID],true);
        f_int_flag = true;
}
if (S_RL_ENB){
        R_COUNT.value = R_RELOAD.value;
        Action(SM_COUNT, mAction_time_ID[actionID] + R_COUNT.value * mCountBase);
}
```

Fig. 3 View of Logic Editor

Logic editor

The virtual printer is made up of the simulator control unit and the hardware models, but the basic hardware classes for the processor, bus and logic components are handled in a synchronized fashion by the simulator control unit. Since the simulator control units exchanges signals between these basic hardware classes, a new hardware model can be operated as a component part of the virtual printer by deriving it from these basic classes.

The models used for the processor and bus are nearly all predetermined and are not created by the user. Therefore, a separate tool is provided for creating new hardware models on the basis of basic logic classes. The basic logic classes are already defined with the minimum levels of code required for them to run in the virtual printer, and a logic component defined by the user makes tacit use of this previously defined code. In the logic editor, the user should only write definitions for the internal operations of the hardware model, and does not need to be conscious of the fact that they are models for a virtual printer. The internal operations of a model are defined as a state machine, so that the whole device can be defined in terms of state transition diagrams.

Fig. 3 is a view of the logic editor screen, showing a main window where the user can add input pins and output pins for the exchange of input and output signals between the hardware models, state transition diagrams for defining the internal operations, and internal variables, a state transition diagram input window for inputting state transition diagrams, and a state processing code editing windows for defining the internal processing for each state.

A number of independent state machines can be defined within a single hardware model. To make internal definitions for a state machine, a name is registered for the state machine, and the settings button is then selected to open a state transition diagram input window, where the machine can be described graphically. To edit the processing for the states in the state machine, the operator can double-click on one of the state labels to open a state processing code editing window, where the processing can be described in C++ source code. In addition, signals representing internal signal states are prepared as internal variables, similarly to actual hardware, and when converting to source code, these can be transferred directly into the objects as class variables.

In the virtual printer, in order to achieve synchronization and high-speed operation in the different models, when one state machine has entered standby, the printer switches to processing another state machine. Therefore, it is necessary to record the state at the time that processing was transferred, so that processing can be restarted from the previous state, when the next process comes around. In order to achieve this kind of system, "input standby" and "process first" are prepared as state transition labels.

"Input standby" is a label which halts at the same state as long as particular conditions are satisfied, maintaining the current state and diverting processing from the state machine. This corresponds to a condition change standby state, dependent on the input pins, register access, or the like.

"Process first" is way of speeding up operation and anticipating the simulation time in a way which does not allow the state machines to affect external operations. By sending the anticipated time to the simulator control unit, the state machine is restarted when the simulation time has reached the recorded time. By adopting this system, it is possible to reduce clock by clock execution of the states, thus preventing any decline in simulation speed.

Creating a virtual printer

A virtual printer is created using this Designer tool, by linking together the various component blocks arranged in the graphic editing window. Generic type components, such as the processor, bus and memory are already stored in Designer, and these can be selected and arranged as desired by the user. To add separate hardware modules, models created using the logic editor are imported into the Designer.

Via the commands on the "Create virtual printer" menu, the information specified in the graphic editing window is converted to C++ source code for the whole virtual printer, which is embedded with registration code for the generation of new models, and model grouping information.

Evaluation

One example of a hardware module was a timer module we built. The state machine COUNT created in this timer module has four states. In order to model the functions of the module, the processing in each of these states, and the state transition conditions, should be defined. Since the code defined here is copied directly to the C++ source code generated by the logic editor, it obeys C++ syntax and when any one of the state labels or transition labels is double-clicked, an editing window opens and the processing details and transition conditions can be described. The various state and transition labels can be given names which make them easier for a human operator to distinguish.

When the hardware module has been created in this way, it is converted to C++ source code via the Create Code menu command. The timer module thus generated is based on code derived from the basic logic classes, and is able to run directly in the virtual printer.

Conclusion

By using basic hardware classes and state transition diagrams, the tasks of manual code input are reduced massively. What is more, since the same state transition diagrams used in physical hardware design can be applied directly to this system, the operation of the module is very easy for the user to follow. In the future, we hope to accelerate virtual printer development by proceeding to describe models with the logic editor.

At the present time, processing within states, and state transition conditions, are described using C++ source code, but we could like to simplify this further for the user, by converting to use of icons and macros.

References

1) Mikio Takashi, Nobuhiro Matsushiro: "Printer Simulators ... Virtual Printers" Oki Technical Review No. 178, Vol.65 No.2, p.83-86, May 1998

2) Mikio Takashi, Nobuhiro Matsushiro: "Profiler type simulator for optimizing firmware in integrated devices", Information Processing Society of Japan, Proceedings of 54th National Conference (1), p.247, 248, 1997

Authors

Mikio Takashi: Oki Data Corp., NIP Div.,
Image Development Group, Team Leader