Visualising Reconfigurable Libraries for FPGAs

Wayne Luk and Scott Guo
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ, England
wl@doc.ic.ac.uk, srg@doc.ic.ac.uk

Abstract

This paper describes a framework and tools for visualising hardware libraries for Fleld-Programmable Gate Arrays (FPGAs), which should also be useful for circuit design in general. Our approach integrates the visualisation of design behaviour and structure, supports various simulation modes, and assists the development of run-time reconfigurable designs in FPGAs such as Xilinx 6200 devices. Our tools can automatically generate a block diagram from a concise parametrised description. Design operations are animated by projecting a dataflow model on the block diagram. The user can select to view data values on specific input and output ports and internal paths. Numerical, symbolic and bit-level simulation and their combination are supported, and the animation speed can be adjusted. The tools should benefit both library users and suppliers, since they can be used (a) to show the internal structure of a design, (b) to illustrate effective usage of library components, and (c) to present the consequences of parametrising designs in different ways.

1. Introduction

Efficient and flexible libraries have been widely recognised as the key to accelerating the production of optimised FPGA designs [1]. As libraries are becoming increasingly complex, it is no longer straightforward to understand the design trade-offs necessary for their effective use. Moreover, while the run-time reconfigurability [2] of some devices inspires novel optimisation techniques such as morphing [3], the additional complexity also provides much room for error.

Existing development tools can either simulate design behaviour or illustrate design structure, but they seldom combine the two views effectively. Few commercial tools support symbolic simulation or run-time reconfiguration [4]. Moreover errors in library designs could result in widespread damage; it is paramount to be able to validate libraries before their release. Our tools address these issues, and they should be useful for both library users and library suppliers. A prototype system that assists the evaluation, validation and documentation of designs will be described in the following section.

2. Our System

A snapshot of the graphical user interface of our visualisation system is shown in Figure 1. Our tools can automatically generate a block diagram from a concise description in the Ruby language [1]. There are two simulation modes: one supports simulating a design cycle by cycle, and the other supports sub-cycle simulation, showing how signals could propagate through a combinational network. Figure 1 shows the visualiser running symbolic simulation cycle by cycle.

The buttons at the top left-hand corner allow the selection of simulation modes and input options; stimulation data can be provided from a file or from the command line input at the bottom. Various controls are used to magnify designs, to choose step-by-step, continuous or cyclic simulation, and to adjust the simulation speed.

Figures 2 to 4 demonstrate the visualisation of an adder array in the sub-cycle mode, when a visual frame is constructed in each sub-cycle. For each simulation cycle in this example, there are five visual frames. Figures 2 to 4 depict an animation sequence in the first simulation cycle with the inputs X0,...,X4. Note that all wires are initialised to an unknown value "?".

3. Requirements

Our work is intended to provide versatile and informative tools which help users to improve their design.

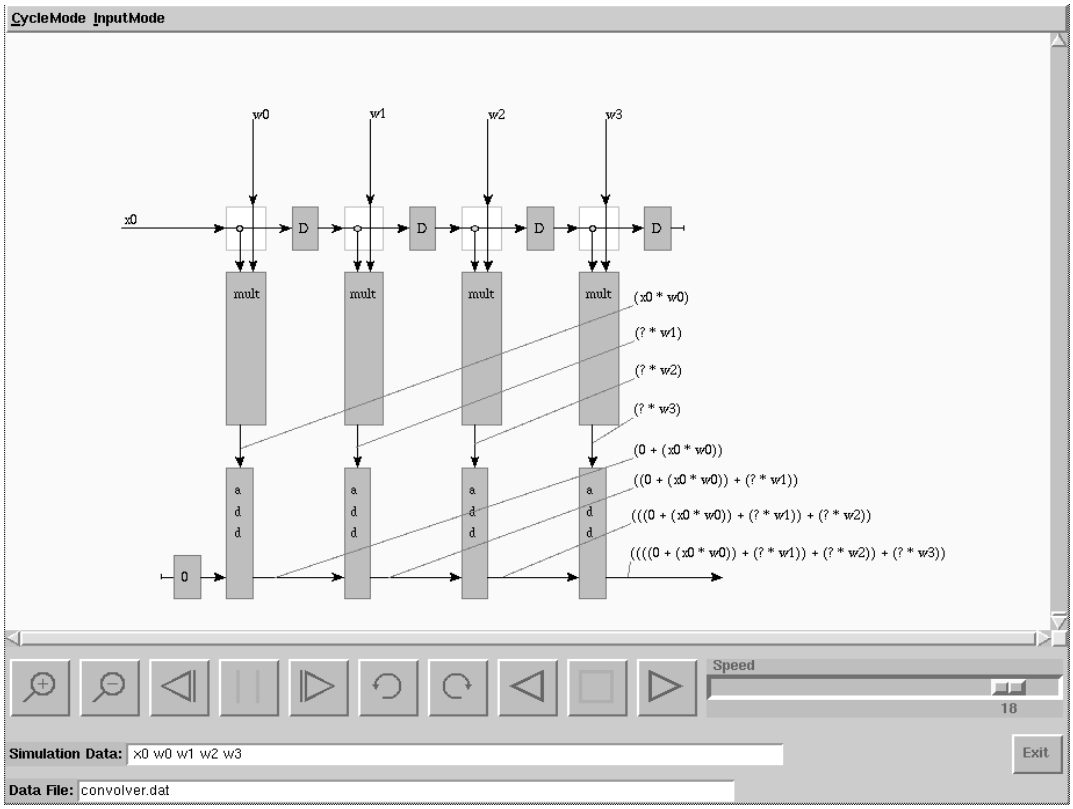

Figure 1. A snapshot of our visualiser carrying out symbolic simulation. D represents a register.

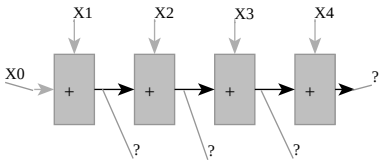

Figure 2. Frame 1 (sub-cycle 1 of simulation cycle 1).

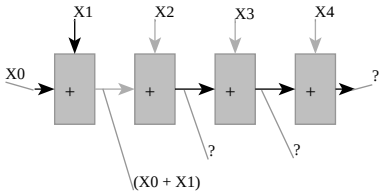

Figure 3. Frame 2 (sub-cycle 2 of simulation cycle 1).

This requires our visualisation system to provide:

- a view of the design structure, which should be concise, regular and easily understood,
- a view of the design behaviour, including state transition and the interaction between components in a design,
- an effective integration of the structural and behavioural views,
- support for run-time reconfiguration, and
- a good graphical user interface.

From experience, a visualiser becomes more useful if it supports both numerical and symbolic simulation. Furthermore, the visualiser should provide values not only for the input and output ports, but also for the internal nodes of the design. Such values can be projected dynamically on the circuit diagram to provide a coherent integration of design structure and behaviour. It is also important to be able to view the simulation results in different ways to cover, for example, both cycle-by-cycle simulation and sub-cycle simulation.

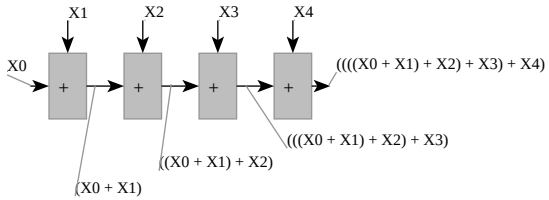

Figure 4. Frame 5 (sub-cycle 5 of simulation cycle 1). Data have been propagated to the output port of the whole design.

4. System Architecture

A system designed to satisfy the above requirements has been prototyped and includes the following major components: a sketcher, a simulator and a visualiser. Figure 5 shows the relationship between the three tools in our system.

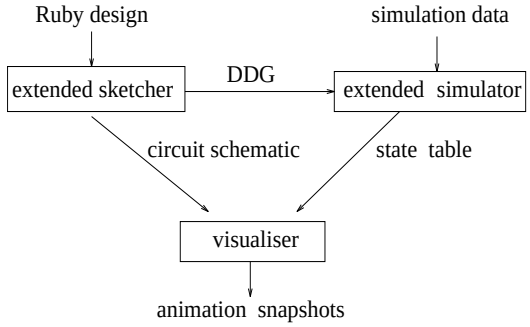

Figure 5. The three main components of the visualisation system. DDG stands for directed dataflow graph.

Given a description in the Ruby language, our sketcher produces a circuit schematic and a directed dataflow graph. The directed dataflow graph is used by a simulator to perform both numerical and symbolic simulation. The simulation results are used to construct a state table, which contains the values on external ports and internal paths of the design at different instants. The visualiser can project the numerical or symbolic data values onto the circuit schematic supplied by the sketcher (Figure 5). The circuit changes state with new inputs arriving cycle by cycle over the total simulation period, and the visualisation sequence is produced accordingly.

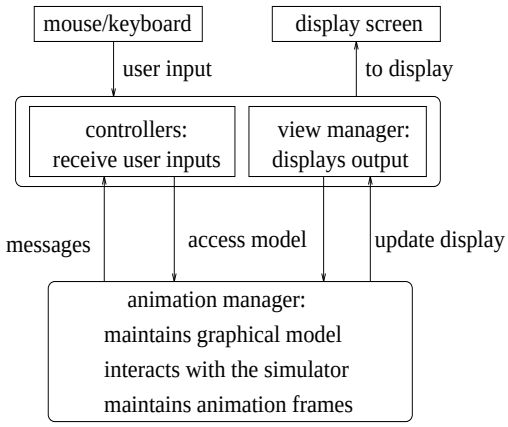

Figure 6. Software architecture of the visualiser.

The software architecture of the visualiser is shown in Figure 6. Users of the visualiser interact with various controllers, which communicate with the animation manager responsible for maintaining the graphical display.

In our current system, the view manager and the controllers are combined into a single module. This module handles mouse and keyboard input, and deals with the screen display in a system-independent manner. The animation manager handles all aspects of internal storage and interacts with both the simulator and sketcher. It also maintains the visualisation sequence.

5. Implementation

The implementation of the visualiser is based on the Rebecca system [1], an environment for Ruby designs that we have been developing. Work is underway to provide similar facilities for other design languages. Our environment contains a design sketcher [5] which automatically produces block diagrams, and a simulator which supports numerical and symbolic simulation. Since the sketcher and the simulator have been developed independently, numerous extensions have been

made so that they can be integrated seamlessly in a coherent system.

The user interface of our visualiser is based on the Tcl/Tk software package [6]. One of the key components in the visualiser is the visual frame manager, which supports two circuit models. In the cycle-by-cycle model, there is no time delay in combinational components of the circuit. In the sub-cycle model, every component is given the same propagation delay. Each simulation cycle contains several sub-cycles, each having a visual frame, to provide the illusion of data propagation.

6. Reconfigurable Designs

Many designers are beginning to exploit the flexibility of FPGAs by reconfiguring them at run time. Recent FPGAs, such as Xilinx 6200 devices, support partial reconfiguration: part of the circuit can be reconfigured while other parts continue normal operation. Such capability improves resource sharing by reconfiguring the FPGA on demand, at the expense of complicating the development of designs.

A simple model [2] has been proposed for visualising reconfigurable designs. Reconfiguration is described using virtual control elements, such as multiplexers and demultiplexers, which connect a specific component to its environment at a particular time. Our visualisation tool can animate reconfigurable designs either by including the virtual control elements and the associated control signals, or by dynamically changing the component at the appropriate instant. The former method provides a bird's eye view of all possible configurations, but the resulting display tends to be cluttered. The other method produces simpler diagrams, but the signals controlling the reconfiguration are hidden.

These two methods will be illustrated using a pattern matcher example [4]. Component A in Figure 7 top represents an AND gate with its bottom input inverted, while component B represents a normal AND gate. This design can be used to match the pattern FTFT. Figure 7 bottom shows an instant when the above pattern matcher has been reconfigured to match the pattern TFTF. The design in Figure 8 is the same as the one in Figure 7 bottom, but with the virtual control elements – in this case the fanouts and the multiplexers labelled mux – made explicit.

We have been exploring a library-based approach [1] for developing reconfigurable designs. Our visualisation tool should contribute to the exploration and evaluation of different reconfiguration possibilities for both library providers and users.

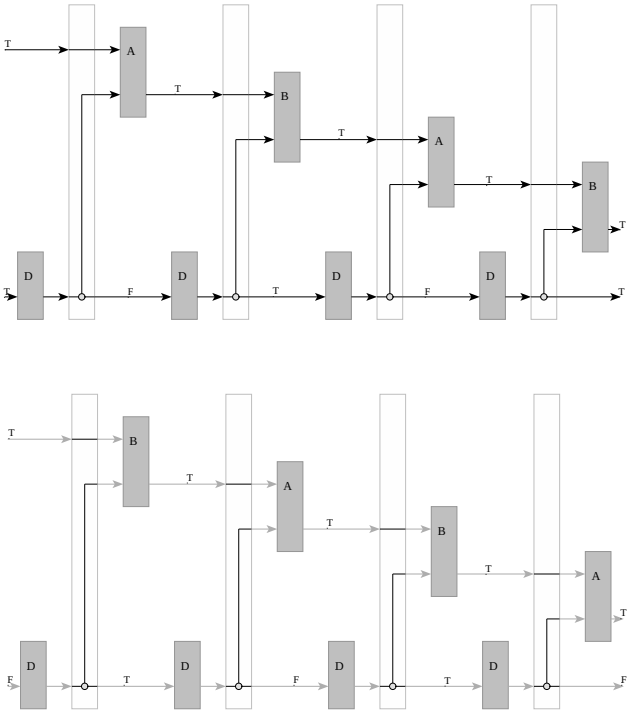

Figure 7. Top: pattern matcher for the pattern FTFT. Bottom: pattern matcher for the pattern TFTF. A represents an AND gate with inverted bottom input, and B represents an AND gate.

7. Summary

A framework and tools for visualising reconfigurable hardware have been presented. The key elements of our approach include the integration of visualising design behaviour and structure, the availability of various simulation modes such as symbolic and bit-level simulation, and the support for run-time reconfigurable designs. Current and future work involves automating the production of animation sequences for FPGA library documentation, integrating the visualisation tools with other circuit analysis and synthesis tools [4], and extending our facilities to support design languages in addition to Ruby.

Acknowledgements

Thanks to Steve McKeever, Richard Sandiford and Nabeel Shirazi for their comments and suggestions.

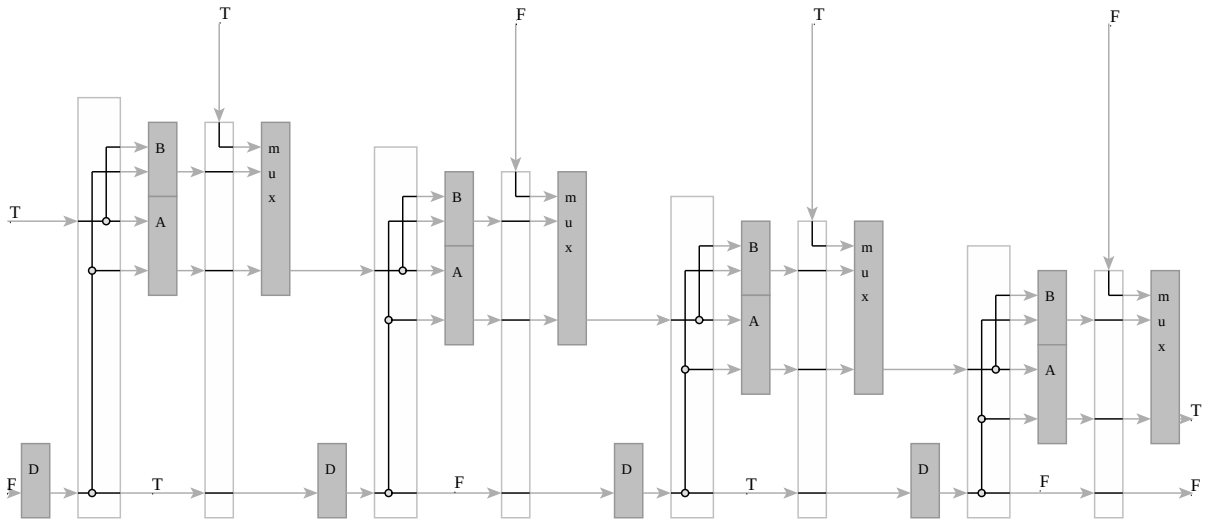

Figure 8. Reconfigurable pattern matcher for the pattern TFTF using mux, a virtual control element.

The support of Xilinx Development Corporation and UK Engineering and Physical Sciences Research Council (Grant Number GR/L24366 and GR/L54356) is gratefully acknowledged.

References

[1] W. Luk, S.R. Guo, N. Shirazi and N. Zhuang, "A framework for developing parametrised FPGA libraries," in *Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, R.W. Hartenstein and M. Glesner (eds.), LNCS 1142, Springer, 1996, pp. 24–33.

[2] W. Luk, N. Shirazi and P.Y.K. Cheung, "Modelling and optimising run-time reconfigurable systems," *Proc. IEEE Symposium on FCCM*, K.L. Pocek and J. Arnold (eds.), IEEE Computer Society Press, 1996, pp. 167–176.

[3] W. Luk, N. Shirazi, S.R. Guo and P.Y.K. Cheung, "Pipeline morphing and virtual pipelines," in *Field-Programmable Logic and Applications*, W. Luk, P.Y.K. Cheung and M. Glesner (eds.), LNCS 1304, Springer, 1997, pp. 111-120.

[4] W. Luk, N. Shirazi and P.Y.K. Cheung, "Compilation tools for run-time reconfigurable designs," in *Proc. IEEE Symposium on FCCM*, K.L. Pocek and J. Arnold (eds.), IEEE Computer Society Press, 1997.

[5] S.R. Guo and W. Luk, "Producing design diagrams from declarative descriptions," in *Proc. Fourth Int. Conf. on CAD/CG*, S. Yand, J. Zhou and C. Li (eds.), SPIE, 1995, pp. 1084-1093.

[6] J.K. Ousterhout, *Tcl and Tk toolkit.* Addison-Wesley, 1996.