An Interactive Datasheet for the Xilinx XC6200

Gordon Brebner

Department of Computer Science
University of Edinburgh
Mayfield Road
Edinburgh EH9 3JZ
Scotland

Abstract. In 1997, Lechner and Guccione introduced JERC: the Java Environment for Reconfigurable Computing. This consisted of a library of functions that permit logic and routing to be specified and configured in a reconfigurable logic device, making it possible for all design data to reside in a single Java source file. The first version of JERC involved some abstraction and simplification of the XC6200 architecture, which meant that many hardware resources of the XC6200 could not be harnessed. This paper reports on work that makes good the deficiencies, by integrating JERC with the author's SPODE environment. The resulting enriched environment is used to support a demonstration application. This is a useful tool that can described as an 'interactive datasheet'. This allows users to navigate technical information about the XC6200, and also to perform small experiments with a real XC6200 device, via a graphical user interface.

1 Introduction

In [4], Lechner and Guccione introduced JERC: the Java Environment for Reconfigurable Computing. This represented an interesting approach to hardware/software co-design for coprocessors based on reconfigurable logic. Specifically, the JERC environment consists of a library of functions that permit logic and routing to be specified and configured in a reconfigurable logic device. By making calls on these functions, circuits may be configured and reconfigured; additionally, programs can be written that interact with the reconfigurable hardware. Thus, it is possible for all design data to reside in a single Java source file. The platform for JERC is the XC6200DS development system, a PCI bus board containing a Xilinx XC6200 FPGA.

The first version of JERC had some drawbacks. These included the fact that abstraction and simplification meant that many hardware resources of the XC6200 could not be harnessed. Also, the question of adding some kind of graphical front-end was left as possible future work. This paper reports on work that has addressed both of these issues.

The solution to the problem of making the full hardware resource set of the XC6200 available was to make use of the author's own SPODE (Software

Person Oriented Design Environment) tool [3] to underpin JERC. SPODE is targeted at the XC6200, and is the descendant of an earlier, similar, programming environment (outlined in [1]) produced by the author for the Algotronix CAL technology. It is akin to JERC, in that it consists of a set of C functions that allow circuitry to be specified by calling the functions. The original version of SPODE, as reported in [3], caused configuration data for the XC6200 to be output to a file. A separate C library was then provided to allow the data to be loaded from the file, and the circuitry to be exercised under program control. In short, the overall effect was thus not unlike that of JERC, although lacking in the modernity of being Java-based.

In [4], it was suggested that JERC should be investigated as the basis for a traditional graphical CAD tool. There is no doubt that it is suitable as a fairly direct back-end for supporting a graphical tool for design at a low level, where the user specifies cell configurations and interconnectivity directly. Such a tool would have benefits not only for design, but also as a debugging aid for completed circuits. However, such a graphical design style is really only suitable for very small-scale designs, perhaps for educational purposes. The low-level specification method is most defensible when used in program mode, where programs can use function calls to specify and configure large, but regular, circuitry very efficiently. Indeed, the author has successfully designed non-trivial circuitry for both the Algotronix CAL and Xilinx XC6200 architectures using such methods (e.g., [1,2]).

As an extension to the idea of just having a useable (if somewhat low-level) design tool, the work described here has resulted in a novel tool that can be described as an 'interactive datasheet'. This tool allows a user to navigate information about the XC6200 part, and also perform simple experiments with the real part to check that they have correctly understood the supplied information. This represents a significant advance on having to fight with the terse and non-interactive descriptions contained in conventional hardcopy datasheets.

2 JERC plus SPODE

A new combined system has been produced, in which SPODE is used to underpin JERC, and the abstractions of JERC have been amended and extended to allow full use of the XC6200 hardware resources. Referring to [4], the original JERC system was organised as a two-level system. Level 0 consisted of abstractions corresponding to the bit-level reconfigurability of the XC6200. Level 1 consisted of abstractions corresponding to logic gates, registers and clock routing; it was implemented in terms of the Level 0 facilities. In the combined system, SPODE replaces JERC Level 0 as a 'Level 0.8' (approximately) facility. That is, it allows full access to the XC6200 resources, but is pitched at a level of abstraction not far below that of JERC Level 1. In particular, SPODE works at a configurable logic gate level, rather than a XC6200-specific configurable multiplexor level.

The major changes to SPODE and JERC to create a combined system capable of fully using the XC6200 were as follows:

- SPODE was enhanced so that it could write configuration data directly to a XC6200 chip, rather than just to an output file (this was very straightforward, given that appropriate code already existed);
- Java wrappers for SPODE's C functions were developed, so that the revised JERC could call them (this required a little specialist knowledge);
- JERC Level 1 was modified, so that it called appropriate SPODE functions, rather than directly accessing configurations (this was fairly intricate, to ensure a complete conversion — particular problems were that JERC stored configuration data for cells individually, and that JERC assumed certain default configuration settings);
- JERC Level 1's set of abstractions were extended in a manner consistent with existing abstractions, so that they addressed all of the XC6200's facilities (this required significant design work, motivated by the existing abstractions used by SPODE).

Unfortunately, space restrictions do not allow a full description of the final extended JERC facilities here, but documentation is available.

The result of the exercise is a substantially enhanced version of JERC (with working name of 'JERCng' — JERC next generation — pending further acronym development). This is suitable as a solid basis for higher-level design tools. Meanwhile, given the regrettable decision by Xilinx not to continue development of the XC6200 family, within Xilinx JERC has acquired the new, and more tasteful, name of 'JavaBits' and evolved into an environment targeted at the XC4000 and Virtex architectures instead.

3 Interactive data sheet

To demonstrate the new facilities of JERCng, a novel tool has been created. This can be used for graphical low-level design, but its main use is envisaged as an 'interactive datasheet'. In fact, the tool could have equally well been implemented directly over SPODE, omitting JERC, but this approach allowed the tool to be written in Java, based on an all-Java environment: JERC to access the XC6200 part, and the full range of Java APIs.

Any reader of FPGA datasheets will know how hard it is to describe the features of today's elaborate chips in print — the XC6200 datasheet is no exception in this respect. The idea behind the interactive datasheet is that, given a computer is equipped with the FPGA chip in question (here, the XC6200), a user can experiment with the information about the FPGA's properties by carrying out configuration activities and getting immediate feedback on their effects via a screen display. The author envisages that such a facility might become a standard part of the documentation distributed with FPGA chips. Such a facility is made particularly practical when FPGA technologies allow partial configuration, as does the XC6200. Without this, it is necessary to write full chip configurations, with only extremely small changes between writes.

The idea behind the interactive datasheet tool is that the user is presented with a cellular array, which can be viewed at several levels of detail:

– individual cell;
– 4×4-cell block;
– 16×16-cell block;
– whole array.

This choice was made to reflect the routing hierarchy of the XC6200. As well as allowing the exposition of length-4, length-16 and chip-length routing wires, it also allows a user to investigation interactions between neighbouring cells and neighbouring blocks. The whole-array level allows an exposition of input/output blocks.

For the individual cell level, the function is viewed at the logic-gate level of abstraction, rather than the lowest configuration level where interconnected multiplexors implement logic functions. This is consistent with the way the FPGA is presented in the XC6200 datasheet, for example. Experimental circuit configuration by the user is done by clicking on the displayed array, with menus indicating the allowable possibilities in each case. Basically, these menus are the interactive equivalent of the configuration options listed in the hardcopy datasheet. Additional textual information about any menu item is also available to the user, corresponding to the detailed descriptions of the options in the hardcopy datasheet. As a further bonus, the user may exercise the circuitry being constructed, to check that her/his interpretation of the datasheet is, in fact, correct. That is, particular inputs can be supplied by the user and the resulting outputs observed.

The individual-cell level will serve as a concrete example of the types of function presented to a user. In any cell, there are the following configurable components:

– the boolean or multiplexor function performed;
– the source of the inputs to the function unit;
– whether the D-type flip-flop is unused, read-only or read-write;
– the sources of the north, south, east and west outputs of the cell;
– the sources of the 'magic' output of the cell.

When the user chooses configrations for any these, the choice is superimposed on the generic cell diagram on screen. In the case of the function choice and flip-flop choice, this is a symbolic annotation. In the case of the various signal sources, this is shown as an explicit wire with the appropriate orientation in the cell.

For cells located on the boundaries of 4×4 and 16×16 blocks, there are extra choices for the north, south, east and west outputs. A cell with this more specialised nature can be selected for individual study by clicking on the appropriate cell at the 4×4-cell or 16×16-cell block levels. The 4×4-cell block level also allows a full presentation of the effect of the 'magic' wires, since these are not uniform at the individual cell level.

4 Conclusion

Both JERC and SPODE originated from similar viewpoints on how reconfigurable circuit design might best be carried out. These move beyond traditional hardware design-style tools, targeted at fixed architectures, and explore the ways in which software programs might intimately interact with soft circuitry. The combination of JERC and SPODE has led to a Java-based environment for the XC6200 that allows its complete functionality to be exercised. Hopefully, this will tempt more users from software backgrounds into attempting the design of reconfigurable circuitry.

The electronic datasheet concept seems a great advance in providing information about FPGA technologies to a user, not only aiding navigation of the information, but also allowing safe interaction with the real part to check understanding. Early experiments, using students unfamiliar with the XC6200, have shown that the new tool is of immense benefit in clearing up misconceptions. Tools like this should be a further way of tempting those from software backgrounds into a world that appears alien at first sight.

Acknowledgement

The author thanks Xilinx Inc., in particular Steve Guccione, for providing him with the source code and documentation for JERC, together with encouragement to develop it further. Thanks also to Richard Morrell for his efforts in helping to bridge the divide between the Java and C languages.

References

1. Brebner, "Configurable Array Logic Circuits for Computing Network Error Detection Codes", Journal of VLSI Signal Processing, **6**, 1993, pp.101–117.
2. Brebner and Gray: "Use of reconfigurability in variable-length code detection at video rates," Proc. 5th International Workshop on Field Programmable Logic and Applications, Springer LNCS 975, 1995, pp.429–438.
3. Brebner, "CHASTE: a Hardware-Software Co-design Testbed for the Xilinx XC6200", Proc. 4th Reconfigurable Architecture Workshop, ITpress Verlag, 1997, pp.16–23.
4. Lechner and Guccione, "The Java Environment for Reconfigurable Computing", Proc. 7th International Workshop on Field-Programmable Logic and Applications, Springer LNCS 1304, 1997, pp.284–293.