GRAPHICAL REPRESENTATIONS OF SELF-MODIFYING CIRCUITRY

N. Macias 27 Apr 2009

Abstract

The role of self-configurable hardware is described with respect to current and future SOC challenges. The dynamic nature of such hardware is illustrated, and the need for an appropriate design representation is discussed. An extension to traditional schematic diagrams is introduced, and examples of using this extension to describe dynamic circuitry are given. Conclusions and plans for future work are presented.

I. INTRODUCTION

Self-configurable hardware – devices similar to FPGAs, but with the added capability of reconfiguring *themselves* – are a next logical step in the progression from custom ASIC to configurable to reconfigurable hardware. As system complexity continues to increase, the challenges posed in dealing with manufacturing defects, run-time defects, and post-manufacture maintenance are growing beyond what can be reasonably handled with a simple compile-then-run methodology.

As the number of available transistors increases, and SOC designs include more and more components, the opportunities for runtime, on-the-fly hardware modification are increasing: just-in-time compilation of recurring algorithms into hardware; architecture tuning; on-chip network modification; and so on.

Moreover, when configurable SOC grow to be 10x to 100x of what they are today, conventional FPGA-like configuration strategies will simply break down. Limited configuration stream bandwidth and fully serial strategies may lead to unacceptable power-up times. Self-configuration strategies can have a dramatic impact here, as techniques such as parallel configuration move the processing burden inside the system itself.

One major impediment to the development of such self-configurable systems is a methodology for representing their dynamic behavior. Standard tools for representing electronic circuits, such as schematic diagrams or netlists, are sufficient for static circuits, but not for dynamic ones. Just as a textual listing of a computer program may not make clear the full dynamics of that program, capturing the full dynamic behavior of an arbitrary self-modifying circuit may not be possible in a static representation. However, there are cases where certain aspects of self-modification may be representable with a single, fixed schematic. The details of such a schematic representation are the subject of this paper.

Section 2 discusses background information on self-configurable hardware, and describes one such system called the "Cell Matrix." Section 3 describes particular self-modifying circuits of interest ("wires" and "cell replicators"). Section 4 introduces a circuit representation scheme called "Magic Polygons," for capturing certain kinds of dynamic behavior in this Cell Matrix. Section 5 gives specific examples of representing useful dynamic circuits, and Section 6 discusses future work.

II. BACKGROUND

Traditional electronic circuitry can be represented as a set of components, and a collection of connections among those components' inputs and outputs. Such a representation is easily made as a schematic drawing, which is well-suited for human analysis, but not very appropriate for automated processing. Circuit information can be more-easily processed by a computer if it is represented as a graph, and stored as a *netlist*. Such representations are sufficient for simple circuits, while for more-complex systems, a hardware definition language (HDL) such as VHDL [1] may be used, allowing the designer to more-directly specify run-time behavior of the system. In these cases though, the specification of the system hardware is still limited to a static hardware design. This is sufficient though for hardware systems which do not change at runtime, including designs implemented on most reconfigurable systems such as FPGAs [2].

The past ten years have seen the advent of new types of reconfigurable devices which are not only reconfigurable, but are also self-configurable [3]. Such devices show promise for a range of applications, including highly-scalable parallel processing [4], fault-

correcting hardware [5], and the development of autonomous systems [6]. In these and other applications, the ability of the system to self-modify and, in some cases, self-analyze, plays a critical role in the initial layout, run-time operation, and/or long-term maintenance of the system. It is thus desirable to be able to design systems which employ such self-configuration.

The next sections will discuss uses and representations of self-modifying circuitry on an architecture called the Cell Matrix. We must first, however, discuss the basic design and configuration mechanisms of this architecture.

A Cell Matrix is comprised of a large number of small, simple processors called *cells*. Each cell is connected to a fixed set of neighbors, and has two inputs (C and D) and two outputs (C and D) attached to each neighbor. Each cell operates in one of two modes: C-mode (if any C inputs are 1), or D-mode (if all C inputs are 0).

In D-mode, a cell's C and D outputs in response to its set of D inputs is determined by a truth table stored inside each cell. This is a basic combinatorial mode, where cells simply map inputs to outputs. Configuration of a cell amounts to changing the cell's truth table. This is done by asserting a C input (i.e., putting the cell into C-mode), and then shifting in the truth table's bits, one at a time, through the D input on the same side on which the C input is asserted. [7] has further details on the architecture of the Cell Matrix. For the present discussion, the above description should suffice.

III. SELF-MODIFYING CIRCUITS

Figure 1 shows a simple circuit called a *remote-cell replicator*. The cell on the right is the *source cell*: it's truth table is going to be copied to the *destination cell* on the left. When the GO input is set to 1, the top row of cells transmit that 1 to the C inputs of the source and destination cells, thereby placing them in C-mode. The C-mode source cell thus begins outputting its truth table's bits to its Western D output. The bottom row of intervening cells transmits these truth table bits to the destination cell's Eastern D input. Since the destination cell is also in C-mode, these bits are used to populate the destination cell's truth table. After all bits have been copied, the destination cell is now an exact copy of the source cell.

Cells are directly connected only to their immediate neighbors : in the case of 4-sided cells, each cell has four neighbors. Therefore, to pass information to non-adjacent cells, intervening cells must be used to implement wires, as illustrated in Figure 1. Based on experience, the most common use for wires is actually to build more wires, while the second most common use is to transfer truth table bits from one cell to another. Both of these are exhibited in the circuit of Figure 1.

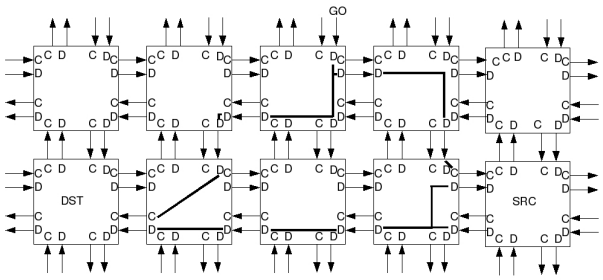

Figure 1. A set of cells configured to replicate a SRC cell to a DST cell.

While Figure 1 is useful for discussing the general concepts of wires and cell replication, it's not actually a very practical circuit, for two reasons: (1) the source and destination cells have fixed locations; and (2) the circuit operates on only a single source and destination pair, while in general we would like to build complex circuits consisting of many cells. In practice, one uses wires with a more-complex structure, which supports configuring cells near the end of a wire so as to extend the wire, or cause it to change directions [7]. Such wires also allow one to raster-scan a copy of a multi-cell source circuit, and transfer the collection of truth tables to a destination region, thereby achieving multi-cell circuit replication.

An important point to note in all this is that cells can transmit, effectively, *two* different types of information: simple data, such as the GO signal; and configuration information, i.e., bits which correspond to truth tables. In either case though, *the intervening cells have no knowledge of what type of information they are transmitting*: there is no way to tell if a bitstream contains data or configuration information, nor does it matter.

IV. MAGIC POLYGONS

A circuit such as the one in Figure 1 is basically handling data which is, in fact, a collection of truth tables corresponding to the cells of some circuit. One way of viewing such circuits is thus to think of them as processing a new type of information: in addition to processing numbers, characters, boolean values, and so on, we can consider that these circuits also process *hardware itself.*

Cell or circuit replication consists of three main steps:

1. converting a source circuit's truth tables into a bitstream (“serialization”);
2. processing of the serial bitstream; and
3. converting the bitstream back into a new

destination circuit (“de-serialization”).

These three operations are sufficient to describe both wire building and circuit replication.

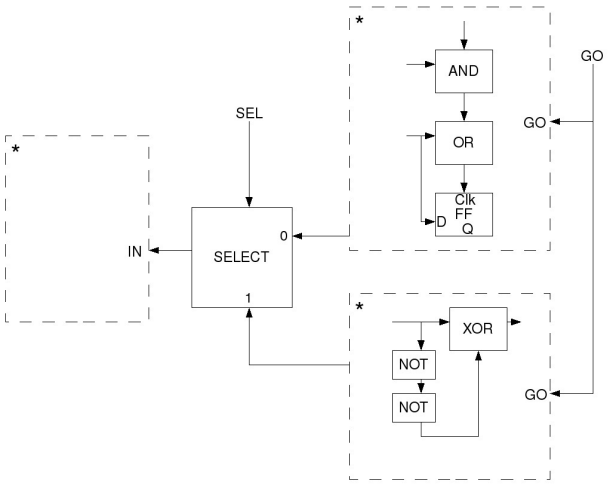

Figure 2. Simple hardware library example, which selects one of two circuits to replicate.

Figure 2 shows a graphical scheme for representing these steps. The two circuits on the right represent a pair of source circuits. The dashed rectangles surrounding each circuit are called *Magic Polygons* (MP). An MP is basically a circuit serializer: it scans the cells of the circuit it contains, reading their truth tables one by one, and sending them out serially through the polygon's output. Serialization is initiated by pulsing the *GO* input. The selector in Figure 2 receives the two serial streams, and transmits one of them to its output, based on the *SEL* input. Finally, the selector's output is sent to another MP, where it is de-serialized, i.e., turned back into a configured circuit. Figure 2 thus depicts a simple hardware library: a system which, at run time, selectively synthesizes one of two possible circuits.

The (*) inside each polygon is an anchor point. When the stream is de-serialized, the reconstituted circuit will be located such that it's (*) aligns with the (*) from the source circuit.

V. EXAMPLES

Figure 3a shows a wire-extension circuit. In this case, the large MP surrounds a circuit containing a wire and another, smaller MP. This is perfectly legal, and the behavior is exactly as you would expect. The wire and smaller MP are serialized, sent through the wire labeled W1, and reconstituted at the end of W1. Figure 3b shows the resulting circuit following a serialize/transmit/de-serialize (STD) cycle. As can be seen, a new piece of wire (“W2”) has been adjoined to the end of W1, but other than this, the circuit of Figure 3b looks the same as that on Figure 3a. This is an example of wire extension. Repeated STD cycles will further extend the wire to the West. Similar circuitry can be used to make turns, and a selection circuit such as shown in Figure 2 can be employed to select among straight wires, corners, and so on. It is thus possible to use Magic Polygons to describe circuits which include general wire-building subsystems.

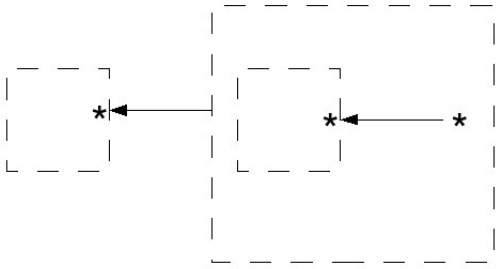

Figure 3a. Wire-extension circuit, prior to Serialize/Transmit/De-serialize (STD) cycle

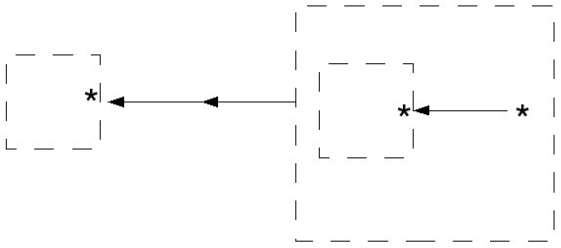

Figure 3b. Extended wire following STD cycle

Figure 4 shows another example of a dynamic circuit, called an “Expanding Counter.” It consists of a simple binary counter, composed of cascaded toggle flip-flops. When the count gets close to the counter's capacity, the circuitry on the right automatically builds a new flip flop to the left of the counter, thus doubling the counter's capacity. This circuit operates in basically the same way as the circuit in Figure 3a: each STD cycle creates a new piece of wire and moves the lefthand MP to the new end of the wire. Since the large MP also contains a flip flop, a new flip flop is also added to the left of the flip flop array.

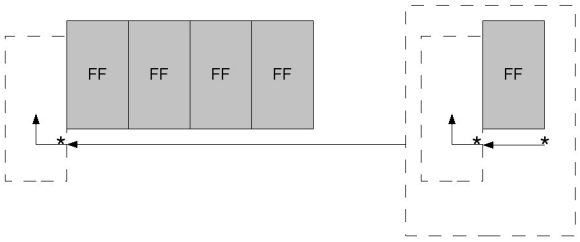

Figure 4. An expanding counter.

VI. CONCLUSIONS AND FUTURE WORK

Dynamic circuitry based on self-configurable hardware offers new opportunities for using today's large transistor budgets, and are well-suited to capitalize on future, potentially huge increases in device count. Describing the dynamic behavior of such circuits is one challenge to making them more-generally useful. Magic Polygons are a simple enhancement to standard schematic diagrams, and offer a means for describing dynamic behavior such as wire building and circuit replication. They are based on the methodology employed in actual dynamic circuitry on the Cell Matrix, and thus offer the promise of, perhaps, being able to capture more-general types of self-modification, including self-modification strategies for fault handling [5].

The present discussion has introduced the basic MP concept, but there remains more work, and more potential enhancements. The anchoring scheme, by which replicated circuits are located in the matrix, must be further-refined to avoid confusion when the enclosed circuit includes MPs with their own anchor points. The option of re-orienting the new circuit would be a useful embellishment. When a new wire is placed next to a pre-existing MP, the MP should, in some cases, be de-activated (this is what happens in an actual wire-extension circuit), so a means for indicating that must be devised.

The circuitry which supports MPs – circuitry for scanning a circuit, serializing it, and so on – has been designed in particular cases, including an expanding counter, and a *self*-replicating circuit. This support circuitry still needs to be generalized though. Also, a compiler which reads MP-enhanced schematics and creates dynamic circuitry (including the necessary support circuitry) needs to be developed. Finally, extension of the MP concept from schematics to more of a hardware definition language or netlist representation would make the concept more generally usable.

VII. REFERENCES

[1] Lin, Y-L, "Survey Paper: Recent developments in high-level synthesis," ACM Transactions on Design Automation of Electronic Systems, Vol. 2, No. 1, January, 1997.

[2] de Lorimier, M. and DeHon, A., “Floating point sparse matrix-vector multiply for FPGAs,” Proceedings of the ACM International Symposium on Field Programmable Gate Arrays, Monterey, CA, 2005.

[3] Fong, R., Harper, S. and Athanas, P., “Versatile Framework for FPGA Field Updates: An Application of Partial Self-Reconfiguration,” Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping (RSP’03), 2003.

[4] Cret, O. et al, "FPGA-Based scalable implementation of the general Smith-Waterman algorithm," Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, 2004.

[5] Zheng, W., Marzwell, N., Chau, S., “In-System Partial Run-Time Reconfiguration for Fault Recovery Applications on Spacecrafts,” 2005 IEEE International Conference on Systems, Man, and Cybernetics, 2005.

[6] Malik, A., Qureshi, T. and Ali, M., “Implementation of an autonomous tracking system for UGVs using FPGAs,” Proceedings of the 7th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, Cambridge, 2008.

[7] Macias, N. and Durbeck, L., “Self-Organizing Digital Circuits,” in Advances in Applied Self-Organizing Systems, M. Prokopenko, Ed., Springer, 2008.