Bridging the Genotype-Phenotype Mapping for Digital FPGAs

Pauline C. Haddow and Gunnar Tufte

The Norwegian University of Science and Technology
Department of Computer and Information Science
O. S. Bragstadsplass 2E, 7491 Trondheim, Norway
pauline@idi.ntnu.no, gunnart@idi.ntnu.no

Abstract

To solve the genome complexity issue and enable evolution of large complex circuits, the need to move away from a one-to-one genotype/phenotype mapping is becoming generally accepted. This involves development of new forms of representation with features such as growth. Shrinking the size of the genotype in effect moves complexity from the genotype representation to the genotype/phenotype mapping.

The field of digital evolvable hardware is relatively young but already researchers have not only had to move through different technology platforms i.e. 6200, 4000 and Virtex series, but also evolution friendly features have disappeared. A mass produced evolution friendly reconfigurable platform is not likely to be ahead of us and a newer technology more evolution friendly than traditional reconfigurable platforms is not around the corner. To be able to reuse results and lessons learned from today's technology on tomorrow's technology and exploit the power of evolution, one solution is to provide a virtual evolution friendly reconfigurable platform which may be mapped onto a given technology.

We propose a two stage genotype/phenotype mapping using our virtual evolvable hardware FPGA as the bridge. The two stages simplify the genotype/phenotype transition at the same time as the virtual evolvable hardware FPGA bridge provides a more evolution friendly platform, further reducing the complexity of the genotype representation.

1. Introduction

The field of evolvable hardware (EHW) is relatively young but already there have been many changes in the technology platforms used. The appearance of Xilinx's 6200 series enabled Thomson [14] to introduce the electronic design community to the idea of evolvable hardware. The 6200 series offered many evolution friendly features such as partial reconfiguration; access to a single cell; configuration unit of a single bit; open architecture and protection against illegal configurations. This opened for a new and exciting design possibility - evolvable hardware. Many other researchers have worked on the 6200 [2, 3, 10]. However, with the demise of the 6200 series many researchers worked on the Xilinx 4000 series [12, 16]. The 4000 series is not partially reconfigurable but available unlike the promised Virtex series which finally replaced Xilinx 6200 series. Unfortunately when Virtex came, although it offers some of the features of the 6200, not all of the evolution friendly features were retained. However, as shown by Hollingworth [7], virtex may still be used as an EHW platform, Virtex 2 is around the corner, still not evolution friendly but possibly an improvement on Virtex due to the promised inbuilt processor core. As such, not only have researchers had to move through different technology platforms but also features have disappeared.

Some researchers have attacked this problem by building their own platforms. One approach aimed at trying to find a suitable technology for evolution is that of Layzell [8] involving the evolvable motherboard. Different logic elements may be placed in the logic blocks for evaluation and a programmable crossbar switch allows a number of interconnection topologies to be realised.

Other approaches involve the creation of chip prototypes for use in EHW. One such approach is the Processing Integrated Grid (PIG) re-configurable platform [9]. Individuals are not stored in their genotype form but implemented in hardware. As such, genetic operations applied to individuals change the implemented configuration of these individuals. This places a requirement on the technology that neighbouring cells may change the configuration of a given cell. As such, this technology offers self-reconfiguration, although somewhat limited.

Other solutions in the digital field include ETLs GRD chip [11]. This chip combines different technologies — a RISC core processor combined with a re-configurable area on a single chip. At JPL a field programmable transistor array (FPTA) [13] has been developed for evolvable hardware offering reconfiguration at the transistor level. This chip provides a platform for evolution of both analogue, digital and mixed signal designs.

Although these platforms may be much more evolution friendly than standard platforms, what we need is broadly available technology platforms for research in this area. A mass produced evolution friendly reconfigurable platform is not likely to be ahead of us and a newer technology more evolution friendly than traditional reconfigurable platforms is not around the corner. To be able to reuse results and lessons learned from today's technology on tomorrow's technology and exploit the power of evolution, one solution is to provide a virtual evolution friendly reconfigurable platform which may be mapped onto a given technology.

One of the key challenges in evolvable hardware for complex circuits is finding new forms of representation i.e. the genotype challenge. Generally in EHW, a one-to-one mapping has been chosen for the genotype-phenotype transition. This means that the genotype needs to include all the information required to program the phenotype technology i.e. the FPGA. This includes both logic and routing information. Larger and larger FPGAs require more and more configuration data thus increasing the size of the phenotype. Using the one-to-one mapping implies an equivalent increase in the size of the genotype. A large genotype, increases resource requirements for the evolution process and, as such, is a disadvantage for evolution.

To solve the genome complexity issues and enable evolution of large complex circuits, the need to move away from a one-to-one genotype-phenotype mapping is becoming generally accepted. We can expect newer features such as incremental evolution [6], 'divide and conquer' approach [15] or growth [1] to become more common in the work within the field of EHW so as to attack the representation problem.

In this work we assume that representation uses a form of growth. Adding growth to the representation effectively moves the complexity problem over to the genotype-phenotype mapping thus increasing the complexity of the mapping.

We propose to simplify the complex mapping process by splitting the mapping process into two stages using a virtual EHW FPGA as the bridge between the genotype and the phenotype.

In this paper we present our virtual EHW FPGA, based on the structure proposed in [4]. We propose a two stage genotype-phenotype mapping using our virtual EHW FPGA as a bridge in the mapping process. The phenotype technology chosen is Xilinx Virtex and a detailed description of the mapping to Virtex is given.

This paper is laid out as follows. In section 2, we describe our bridge, the virtual EHW FPGA and in section 3 we describe the phenotype technology - Xilinx Virtex, chosen as an example of today's digital platforms. The two stage mapping is presented in section 4 and section 5 presents the mapping from our virtual EHW FPGA to Virtex in more detail. Finally, in section 6 we discuss the benefits of this approach and in section 7 we describe ongoing work.

2 Virtual EHW FPGA

The Virtual EHW FPGA contains blocks — *sblocks*, laid out as a symmetric grid where neighbouring blocks touch one another. There is no external routing between these blocks except for the global clock lines. Input and output ports enable communication between touching neighbours. Each sblock neighbours onto sblocks on its 4 sides.

Each sblock consists of both a simple logic/memory component and routing resources. The sblock may either be configured as a logic or memory element with direct connections to its 4 neighbours or it may be configured as a routing element to connect one or more neighbours to non-local nodes. By connecting together several nodes as routing elements, longer connections may be realised.

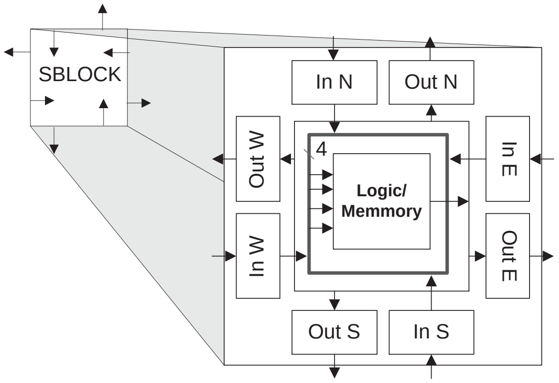


Figure 1. Sblock — Routing and Logic/Memory Block

Figure 1 illustrates the internal routing of the sblock. Each sblock is responsible for processing input signals and routing signals to its neighbouring sblocks. There are 4 pairs of unidirectional wires at each interface. The input wires are attached through routing logic to the input routing channel which provides a dedicated wire for each input. This channel is then fed into the logic/memory block. Output from the logic/memory block is a single wire which feeds the output routing ring. All the 4 outputs of the sblock are attached to this output routing ring through the output routing logic.

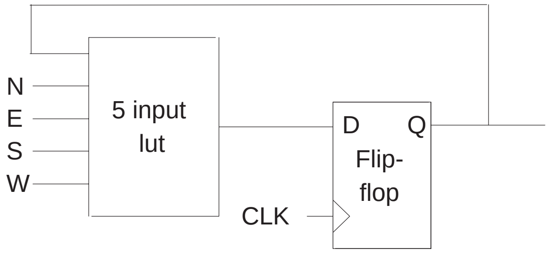
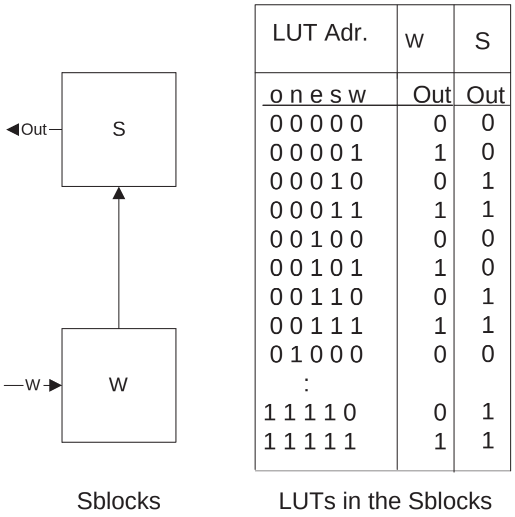


Figure 2. Sblock Logic

A more detailed view of the logic/memory block is shown in figure 2. Inputs from neighbouring sblocks and a feedback from the output are connected to a 5 input look up table (LUT). The LUT may be configured to hold a function. When Don't Care (DC) bits are placed at a given input, then that neighbouring sblock is not connected to it. In this way the LUT is programmed not only for functionality but external connectivity of the sblocks. Therefore, to alter the functionality and connectivity of an sblock only the LUT content need be reprogrammed.



LUT Adr.	W	S
o n e s w	Out	Out
0 0 0 0 0	0	0
0 0 0 0 1	1	0
0 0 0 1 0	0	1
0 0 0 1 1	1	1
0 0 1 0 0	0	0
0 0 1 0 1	1	0
0 0 1 1 0	0	1
0 0 1 1 1	1	1
0 1 0 0 0	0	0
:		
1 1 1 1 0	0	1
1 1 1 1 1	1	1

Figure 3. Sblock through Routing

When the sblock is used for routing, all inputs apart from the incoming input to be routed are don't care inputs. The LUT is thus set up as shown in figure 3. In this example the west input is to be routed to the north. The output reflects the values of the west input thus implying don't cares at the other inputs. However, the output values will appear at all outputs not just the north output. It is up to the sblock to the north to read its input at this port i.e. its south input, as shown.

Although this is a virtual FPGA, for a realistic FPGA it is important that illegal configurations are avoided where we define illegal configurations as configurations that may cause damage to the chip. The internal routing of each sblock only allows inputs to be routed to outputs and the interface between sblocks only allows outputs to be routed to inputs.

3 Virtex

A Virtex FPGA consists of an array of configurable logic blocks (CLBs), input/output blocks (IOBs) and routing resources. A CLB contains 4 LUTs and 4 flip-flops as well as multiplexers for internal routing. These are arranged to form two slices, each with 2 LUTs and 2 flip flops. External routing consists of local, non-local and dedicated clock routing.

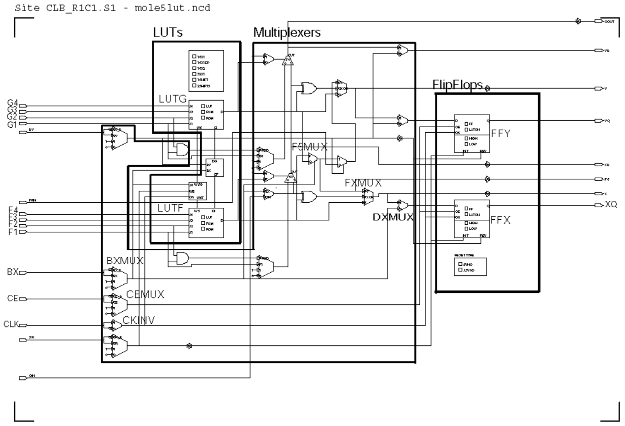


Figure 4. One Slice of a Virtex CLB

Figure 4 illustrates the structure of one slice of a CLB. To illustrate the different functional groups the slice is divided into three.

- LUTs: The two LUTs **LUTF** and **LUTG** are function generators that can be configured to operate as RAM or logic. The LUTs can also be used to create shift registers. When used as logic, each LUT can be a four input logical function. The output from each LUT is available as an output signal from the slice.

- Flip-Flops: The two output flip-flops **FFX** and **FFY** can be configured to be positive or negative edge-sensitive. The reset may be configured as synchronous or asynchrous. Each flip-flop's output signal is available as an output from the slice.

- Multiplexers: There are two types of multiplexers available: those controlled by logic signals i.e varying control value, and those controlled by the FPGA configuration data i.e. fixed control value.

Each CLB consists of a General Routing Matrix (GRM) for external routing and two switch boxes for internal routing. Figure 5 illustrates two neighbouring CLBs. As shown, the GRM connects to its 4 neighbours and the internal switch boxes. Note that other routing resources are also available but are not the focus of this work.

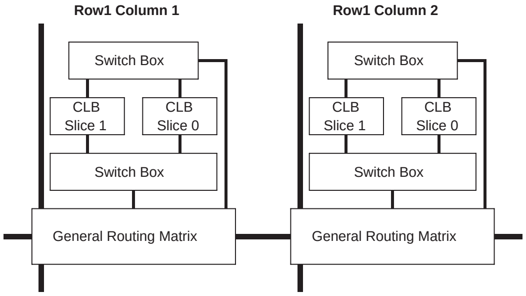


Figure 5. Virtex CLBs with Switch Boxes

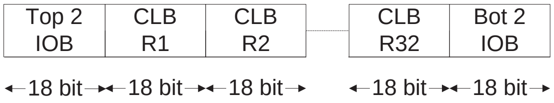


Figure 6. Configuration Data Frame

In Xilinx 6200 partial configuration was achieved through accessing either a particular cell or a range of cells. That is that the configuration unit i.e. the smallest amount of data that can be written to or read from a device, was a single cell. In the new Virtex series, the configuration unit is a frame. A frame is organised as a vertical array one bit wide. A frame represents part of the configuration data for the top two IOBs, all CLBs and for the bottom two IOBs of a column. Figure 6 illustrates the structure of the configuration data frame. As shown, the frame contains 18 bits of configuration data for each element in the column. 48 frames are needed to configure a column.

An LUTs configuration data is stored in 16 of the 48 configuration frames for the CLB. As such, it takes 16 frames to configure one or more LUTs of a column. However, to alter a single bit in one or more LUTs of a column, only one frame of configuration data is needed.

Virtex can be used as a partial reconfigurable device in two different modes, JTAG or SelectMap. JTAG is a serial protocol and not considered here. The SelectMap mode uses the SelectMap port on the Virtex. Configuration is done by writing configuration data and configuration commands to the 8 data bit parallel SelectMap port interface.

4 Two Stage Mapping

Our genotype-phenotype mapping is achieved in two stages using the virtual EHW FPGA as our bridge. We begin with our genotype representation, map to our virtual EHW FPGA and finally map to our phenotype representation — Virtex.

4.1 Genotype Representation

Our aim is to be able to represent complex circuits. One possible solution is a 2D structure with connected nodes. Each node has a re-programmable function depending on the surrounding nodes and a relative x,y placement. It is possible to add or remove nodes and to grow side chains to create possible solutions. That is we are not restricting the representation to a tree structure as in genetic programming. As the representation is not the focus of this paper we draw the readers attention to [5] for further information. The mapping of the representation to the Virtual EHW FPGA is of course dependent on the actual representation used and therefore will not be discussed further in this paper.

4.2 Virtual EHW FPGA Representation

After mapping to the virtual EHW FPGA bridge, representation consists of a number of sblocks, programmed to reflect the mapped individual, within the constraints of the sblock architecture. That is, each sblock has a maximum of 4 neighbours, a 5-input LUT and an output flip flop.

Sblocks are connected i.e. inputs/outputs activated, in the pattern given in the individual. The interconnection thus illustrates the relative placement of the sblocks with respect to each other. In addition, the sblocks are programmed as either routing or logic blocks. For routing, only one input is active. For logic, the chosen inputs are active and the functionality of the sblock is programmed.

Figure 7, provides an example of a simple digital design mapped onto a sblock structure. The actual contents of the LUTs is not shown but a description of their content is provided. As shown, 3 of the sblocks are used for through routing and two for logic.

4.3 Virtex Representation

The Virtex representation is a standard physical FPGA chip. The chip we have chosen is from the Virtex series. Here two sblocks may be mapped onto a single CLB. Direct

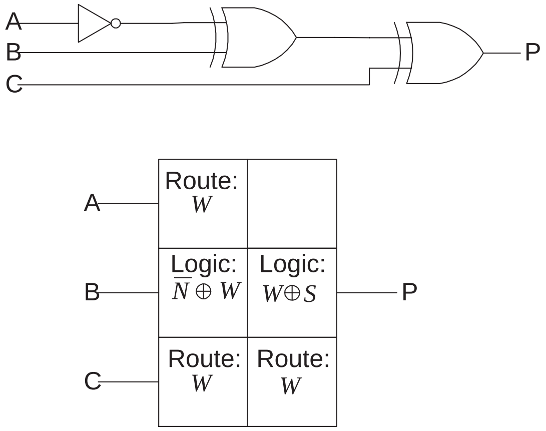


Figure 7. Sblock Representation of a Digital Circuit

connections between CLBs along with some of the internal routing in the CLB are the only routing resources used for the sblock architecture.

A circuit mapped onto the Virtex chip will now be represented as actual rather than relative placement. Inputs and outputs of the virtual architecture are mapped to IO blocks.

5 Mapping the Virtual EHW FPGA to Virtex

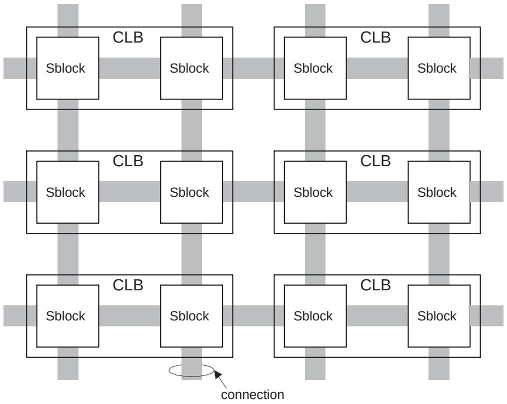


Figure 8. Sblock Structure in a Virtex

To simplify the mapping process further, the Virtex chip is initialised by configuring it as a sblock grid — see figure 8, where each sblock occupies one CLB slice.

5.1 Initialisation: Setting up the Sblock Grid

Configuring a Slice

To implement an sblock, a 5 input LUT and a flip-flop are needed. As illustrated in figure 9, this is achieved by using both LUTs, multiplexers and one of the flip flops in the CLB slice shown in figure 4.

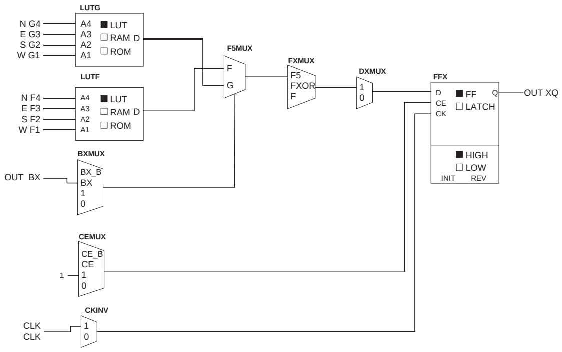


Figure 9. Sblock in CLB Slice

The inputs from the neighbouring sblocks connect to both LUTF and LUTG. The outputs of these LUTs are connected to F5MUX. Control of F5MUX is provided by the slice output fed back through BXMUX thus providing a 5 input LUT. LUTG provides the 16 low address locations and LUTF provides the 16 high address locations of the 32 address LUT.

Several multiplexers are used for the sblock internal routing. BXMUX routes the 5th LUT line to F5MUX; CEMUX routes a logical ”1” to the FFX flip-flop Clock Enable (CE); CKINV selects FFX to be positive-edge triggered and FXMUX and DXMUX are used to route the 5 input LUT to the output flip-flop.

Configuring CLB Routing

External routing to an sblock involves routing between it and its neighbours and feedback of its own output. An example of routing for a sblock is shown in figure 10 where routing for the sblock in slice 0 is shown. Signals from the surrounding sblocks are labelled **N, E, S, W** and **Out** for the sblock output. The inputs are routed from the GRM to the lower switch box where they are duplicated and forwarded to the 2 LUTs within the sblock. **Out** is routed through the top switch box to the GRM where it is fed back into the lower switch box as well as being available to the neighbouring sblocks.

5.2 Mapping Individuals

At this stage, sblocks have no programmed functionality. The FPGA can now be configured to reflect a given individual by addressing sblocks in the pre-configured sblock architecture.

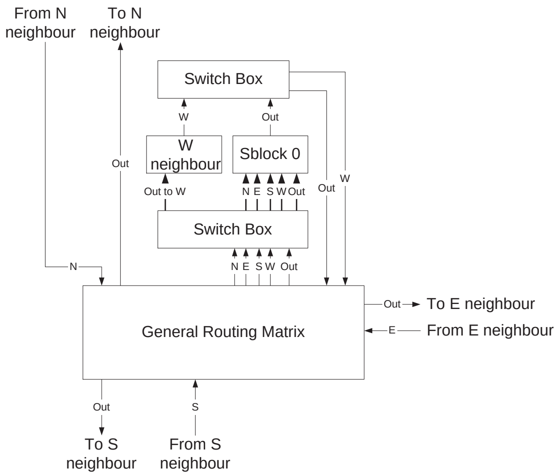


Figure 10. Routing in Virtex for Sblocks

Each sblock is placed in a particular CLB slice so as to reflect the relative placement of the given sblock in the individual. The functionality of the sblock and its connectivity to its neighbours is achieved by configuring its LUT.

Figure 11 illustrates implementation of an individual representing a circuit which is the OR of two 2 input AND gates. The number of gates required i.e. sblocks, their relative placement and the inputs and outputs required are given in the Virtual EHW FPGA representation. The sblock structure is already initialised and configuration at this stage involves programming the respective sblocks as an AND or OR gate and placing don't cares at the connections that are not required. Considering the OR gate, only the north and south inputs are required, all others will get don't cares.

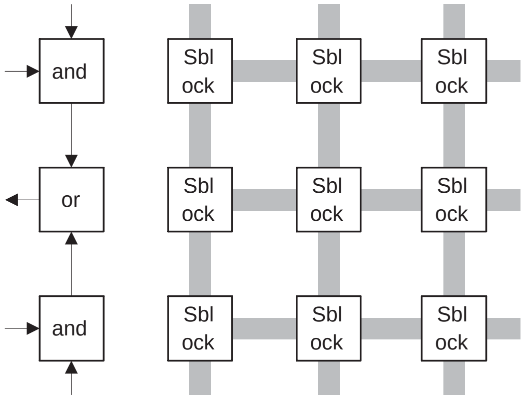
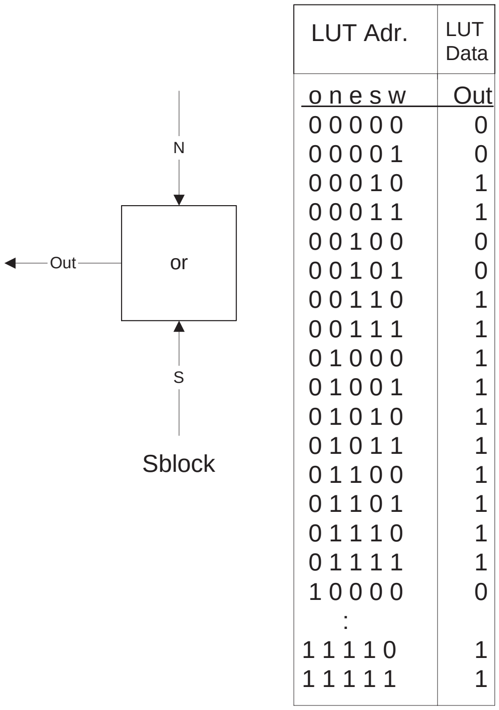


Figure 11. Configeration of Connectivity and Function

Figure 12 illustrates the programmed LUT for the OR gate of figure 11. From the contents of the LUT it is clear that only the North and South signals are effecting the LUT output. The don't care for East and West means that the sblock does not respond to these signals and therefore can be considered unconnected to its East and West neighbours. Note that physically the complete sblock structure is still present in the Virtex chip.



LUT Adr.	LUT Data
o n e s w	Out
0 0 0 0 0	0
0 0 0 0 1	0
0 0 0 1 0	1
0 0 0 1 1	1
0 0 1 0 0	0
0 0 1 0 1	0
0 0 1 1 0	1
0 0 1 1 1	1
0 1 0 0 0	1
0 1 0 0 1	1
0 1 0 1 0	1
0 1 0 1 1	1
0 1 1 0 0	1
0 1 1 0 1	1
0 1 1 1 0	1
0 1 1 1 1	1
1 0 0 0 0	0
:	
1 1 1 1 0	1
1 1 1 1 1	1

LUT in the Sblock

Figure 12. Sblock with LUT configuration

As stated, to alter a function in a sblock or change its connectivity, the LUT may be reprogrammed. Considering again the OR gate of figure 12, a reconfiguration to an AND function is achieved by altering the LUT to only output logical ”1” when both North and South inputs are logical ”1”. Similarly by altering the don't care conditions in the LUT it is possible to make the sblock sensitive to other input signals thus alter the connectivity of the sblock.

6 Evaluation

The virtual EHW FPGA is a virtual i.e. not physically built FPGA, designed for evolvable hardware. That is it includes features desirable for an evolution platform — see [4], not available in today's technology. As this FPGA may be said to be 'evolution friendly' then mapping from a new genotype representation to the virtual EHW FPGA representation is simpler than that expected from a non evolution friendly FPGA such as Xilinx Virtex. Since the virtual EHW FPGA has many of the characteristics of the physical FPGA then this stage of the mapping process is much simpler than that which would have been the case with a direct genotype-phenotype mapping.

Simplicity of the mapping is not the only benefit of bridging the genotype phenotype mapping. As stated, many researchers are investigating the genotype representation problem and we can expect new and better methods of representation in the near future. Different representations may be tested out against the bridge, enabling a chip independent assessment to be achieved. If an actual physical chip evaluation is required then the representation may be mapped further to the actual technology platform. The second stage of the mapping, on the other hand, is independent of the representation which will vary as different representations may be expected to be tried out to achieve complex designs. This stage maps a virtual FPGA to a given FPGA.

In the current FPGAs, a vast amount of configuration data is attributed to routing. All routing must be configured and with the vast amount of routing available this entails a lot of routing configuration data even if a lot of the routing resources are unused. The virtual EHW FPGA is designed with just neighbouring sblock connections and clocking. When the genotype reflects configuration data, as is often the case, substantially less configuration data is required for the virtual EHW FPGA due to the lack of external routing. As such, a much smaller genotype results than that which would be the case for a Virtex genotype. The virtual EHW FPGA thus not only simplifies the genotype-phenotype mapping but also further simplifies the genotype representation itself. Note that all unused Virtex routing is configured as unused in the initialisation of the Virtex chip before mapping of individuals begins.

As illustrated in section 5.2, manipulating functionality or connectivity of sblocks is easily achievable. As such, individuals are easily mapped from the virtual EHW FPGA to Virtex.

7. Ongoing work

We are currently working on a new genotype representation based on L-systems. Each individual will we mapped through the Virtual EHW FPGA to the Virtex chip to evaluate the new genotype representation on a physical hardware platform.

References

[1] P. J. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 35–43, 1999.

[2] T. C. Fogarty, J. F. Miller, and P. Thomson. Evolving digital logic circuits in xilinx 6000 family fpgas. In *Soft Computing in Engineering Design and Manufacturing*, pages 299–305. Springer, 1998.

[3] F. Gers, H. de Garis, and M. Korkin. Codi: A simplified cellular automata based neuron model. In *Artificial Evolution Conference 1997 (AE97)*, 1997.

[4] P. Haddow and G. Tufte. An evolvable hardware fpga for adaptive hardware. In *Congress on Evolutionary Computation(CEC00)*, pages 553–560, 2000.

[5] P. Haddow, G. Tufte, and P. Van Remortel. Shrinking the genotype: L-systems for ehw? In *submitted to 4th International Conference on Evolvable Systems (ICES01)*, 2001.

[6] I. Harvey et al. Why evolutionary robotics? *Robotics and Manufacturing: Recent Tends in Research and Applications*, 6:293–298, 1996.

[7] S. Hollingworth, G. Smith and A. Tyrrell. Safe intrisic evolution of virtex devices. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 195–202. IEEE, 2000.

[8] P. Layzell. A new research tool for intrinsic hardware evolution. In *2nd International Conference on Evolvable Systems (ICES98)*, Lecture Notes in Computer Science, pages 47–56. Springer, 1998.

[9] N. Macias. The pig paradigm: The design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In *The 1st NASA/DoD Workshop on Evolvable Hardware*, pages 175–180, 1999.

[10] J. Miller and P. Thomson. Aspects of digital evolution: Evolvability and architecture. In *Fifth Intern. Conf. on Parallel Problem Solving from Nature (PPSN'98)*, 1998.

[11] M. Murakawa, S. Yoshizawa, I. Kajitani, X. Yao, N. Kajihara, M. Iwata, and T. Higuchi. The grd chip: Genetic reconfiguration of dsps for neural network processing. *IEEE Transactions on Computers*, 48(6):628–639, June 1999.

[12] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. The firefly machine: Online evolware. In *Proc. of 1997 International Conference on Evolutionary Computation (ICEC97)*, pages 181–186. IEEE, 1997.

[13] A. Stoica, D. Keymeulen, R. Tawel, and C. W.-T. L. Salazar-Lazaro. Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital cmos circuits. In *The 1st NASA/DoD Workshop on Evolvable Hardware*, pages 76–84, 1999.

[14] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *1st International Conference on Evolvable Systems (ICES96)*, Lecture Notes in Computer Science, pages 390–405. Springer, 1996.

[15] J. Torresen. A divide-and-conquer approach to evolvable hardware. In *2nd International Conference on Evolvable Systems (ICES98)*, Lecture Notes in Computer Science, pages 57–65. Springer, 1998.

[16] G. Tufte and P. Haddow. Evolving and adpative digital filter. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 143–150, 2000.