Two Approaches for High-level Evolvable Hardware using Function Blocks

Magnus Ekman
Department of Computer Engineering
Chalmers University of Technology
Göteborg, Sweden
mekman@ce.chalmers.se

Andreas Magnusson and Peter Nordin
Department of Physical Resource Theory
Chalmers University of Technology
Göteborg, Sweden
{andmag, nordin}@fy.chalmers.se

In general, approaches to genetic programming are more efficient if the implementation is close to hardware. For instance, evolution of binary machine code is very efficient time-, space- and energy-wise. In AIM-GP [1] binary machine code is directly evolved resulting in a speed up of several orders of magnitude, while also enabling compact implementation on low-end computer architectures. Here we present work taking GP implementation one step closer to the hardware. The logical step for increased efficiency is to move below the level of processors into reconfigurable logic chips, such as FPGAs. This is quite straight forward as long as the evolution and fitness function is limited to logical functions [2]. However, in reality many domains require evolution of functionality composed of blocks at a higher level. Hardware evolution can thus be performed at various abstraction levels. Work has been done at low levels by Thompson et al. [3] who have evolved configuration strings for an FPGA. Higuchi et al. have investigated hardware evolution at function level [4], using more complex functions instead of just primitive two-input gates. In this paper we present two other approaches to evolvable hardware using an FPGA:

1. The main objective for the first approach was to develop a method for more efficient individual evaluation in genetic programming, enabling evolved FPGA configurations to be used for arithmetic tasks in weak architectures such as those used for mobile computing. The individuals are represented by a list of instructions. Instead of executing these instructions in a general processor, several specialized functional blocks are placed after each other in a pipeline, implementing a data-flow machine that tests one individual each clock cycle. The functional blocks that represent the instructions are constructed as VHDL objects and are only routed once, before evolution. All of the blocks have the same number of inputs and outputs and they all occupy the same physical size. This makes it possible to switch the blocks in the data path without a new "place and route" pass. All evaluations were performed on Xilinx chips.

2. In many applications it is desirable to address a segment for abstraction between that of a numerical register machine and the pure logic function. For this requirement we have designed the second approach, which is based on state-machines implemented in VHDL. We evolve the code using a structure consisting of several state-machines of various sizes that co-operate to solve the problem. The reason to have several machines instead of one big complicated one is that no efficient method to mix two machines and still keep the properties of both exists. We thus use several state-machines to make crossover easier to implement — an alternative future approach would be to evaluate the system without crossover using only mutation, e.g. the way state-machines are treated in early work of Evolutionary Programming [5]. The structure we use is built from several synchronous state-machines and one combinatorial net that are connected to a bus. Each state-machine and the combinatorial net are assigned a constant number of bits on the bus that it can write to. The way the inputs to the machines and the net are connected is evolved. The crossover operator exchanges machines and mixes the net between individuals. Mutations modify the state-transitions.

In summary, we have implemented and evaluated the first method for evolution of high-level register machine structures directly in reprogrammable hardware, while the second method has been simulated. Both variants have application areas where efficiency in the physical system is critical but evolution of pure logical functions is too crude. Such architectures exist in mobile computing and handheld devices as well as other areas which require low-power but adaptive applications.

For full paper and references see:
http://www.ce.chalmers.se/staff/mekman/

Proposal for a Register Transfer Level Evolution on a Microprocessor Incorporated Flash Memory

Yuji Sato

Department of Computer Science
Hosei University
3-7-2, Kajino-cho, Koganei-shi, Tokyo 184-8584, Japan
E-mail: yuji@k.hosei.ac.jp

Abstract

A new idea for evolvable hardware based on a microprocessor is proposed. In recent years, there has been much research using Programmable Logic Devices (PLD) and Field Programmable Gate Arrays (FPGA). In particular, the application of digital circuit evolution to engineering fields has already begun. On the other hand, long learning time, difficulty to predict when an effective capability will appear, large chip size and other such problems have hindered progress in diffusion into engineering fields. Here, we propose register transfer level evolution performed on a microprocessor as a means of addressing these problems.

1 BASIC APPROACH OF THE PROPOSED EVOLVABLE HARDWARE SCHEME

With conventional evolvable hardware, for both the PLD scheme and the FPGA scheme, the target functions are implemented by a logic circuit whose configuration evolves in a process where the configuration bits of the logic serve as the chromosomes of a GA [Higuchi 1999]. With the evolvable hardware that we propose here, the target functions are by an execution unit that have fixed configurations. The evolution takes place in the control signal that indicates how the execution unit is used. That is to say, register transfer level evolution takes place on the LSI chip. A conceptual diagram that illustrates the relationship of the execution unit and control signal is shown in Fig. 1. The execution unit comprises a register file group, an arithmetic and logic unit (ALU), a comparator, a flag control and other such functional modules connected by a data bus. This configurational element is controlled by a control vector of control signals s_1 through s_n that is obtained by decoding microinstruction code. For example, when the value of control signal s_1 in Fig. 1 is 1, the data in the register that specifies an address signal is read by bus A. In the same way, when the value of control signal s_2 is 1, the address-specifying register content is read by bus B. When the values of control signals s_4 and s_5 are 1, the contents of data bus A and data bus B are input to the ALU. When control signals s_6 = 0, s_7 = 1 and s_8 = 1, for example, represent an ADD instruction and the value of control signal s_9 is 1, the computation results of the ALU are output to data bus C. Accordingly, if the control signal bit string (s_1, s_2, …, s_n) is regarded as a GA chromosome, then the target function can be implemented through genetic manipulation of the control signal.

A number of reports concerning research on the synthesis of circuits on the register transfer level through genetic manipulation at the HDL (Hardware Description Language) level have already. All of those, however, involve evolution at the HDL level and require changes in specifications and revision of the chip manufacturing.

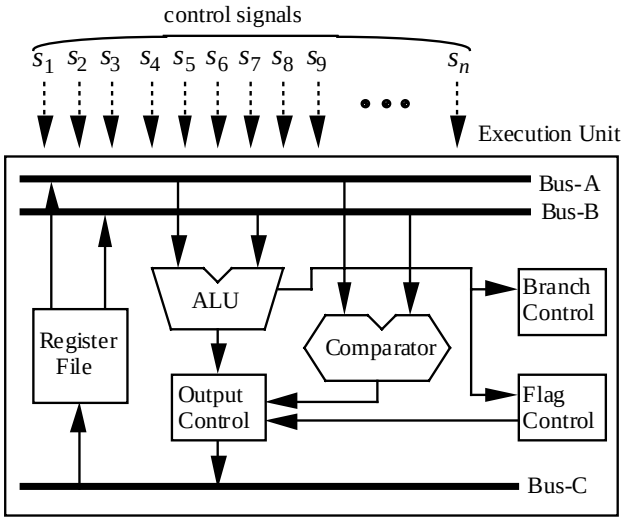

Fig. 1 A conceptual diagram that illustrates the relationship of the execution unit and control signals.

References

[Higuchi 1999] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, and E. Takahashi, “Real-World Applications of Analog and Digital Evolvable Hardware,” IEEE Trans. on Evolutionary Computation, Vol. 3, No. 3, pp. 293-308, September (1999).