Acceleration of Transistor-Level Evolution using Xilinx Zynq Platform

Vojtech Mrazek and Zdenek Vasicek
Brno University of Technology
Faculty of Information Technology
Brno, Czech Republic
Email: imrazek@fit.vutbr.cz, vasicek@fit.vutbr.cz

***Abstract*—The aim of this paper is to introduce a new accelerator developed to address the problem of evolutionary synthesis of digital circuits at transistor level. The proposed accelerator, based on recently introduced Xilinx Zynq platform, consists of a discrete simulator implemented in programmable logic and an evolutionary algorithm running on a tightly coupled embedded ARM processor. The discrete simulator was introduced in order to achieve a good trade-off between the precision and performance of the simulation of transistor-level circuits. The simulator is implemented using the concept of virtual reconfigurable circuit and operates on multiple logic levels which enables to evaluate the behavior of candidate transistor-level circuits at a reasonable level of detail. In this work, the concept of virtual reconfigurable circuit was extended to enable bidirectional data flow which represents the basic feature of transistor level circuits. According to the experimental evaluation, the proposed architecture speeds up the evolution in one order of magnitude compared to an optimized software implementation. The developed accelerator is utilized in the evolution of basic logic circuits having up to 5 inputs. It is shown that solutions competitive to the circuits obtained by conventional design methods can be discovered.**

I. Introduction

In recent years, many authors demonstrated the merits of evolutionary design techniques in the field of digital circuit design. For example, efficient implementations of various combinational circuits unreachable by conventional design approaches were obtained using cartesian genetic programming (CGP) representing probably the most efficient evolutionary technique applied in this area [1], [2], [3], [4]. While the gate-level evolutionary synthesis represents an intensively studied research area, the synthesis of transistor-level digital circuits remains, in contrast with design of transistor-level analog circuits investigated for example in [5], [6], on a peripheral concern of the researchers despite the fact that even some basic logical expressions can be implemented much effectively at transistor level. For example, the number of transistors that are required to implement a 4-input circuit known as AND-OR-INVERT can be reduced by 60% when the circuit is designed directly at transistor-level.

Only a few papers were devoted to the evolutionary design of digital circuits from scratch directly at transistor level. The lack of interest is probably caused by an extremely time consuming evaluation of candidate solutions. Usually, a third-party SPICE-like analog circuit simulator is applied to evaluate the behavior of candidate solutions. Though there are some approaches addressing the problem of acceleration of transistor-level circuit simulation (e.g. [7]), they are hardly applicable in the field of evolutionary design due to the overhead which arises from the necessity to create a new instance of a hardware accelerator for each candidate solution.

Since the evolutionary design is based on the generate-and-test principle which typically requires to evaluate many candidate solutions to discover a satisfactory solution, it is obvious that the performance of the utilized circuit simulator has a substantial impact on the scalability of the whole evolutionary approach. In order to address this issue, Zaloudek et al. proposed an approach that utilized a simple simulator designed for rapid evaluation of candidate solutions [8]. A rough approximation of transistor behavior unfortunately caused that this approach tended to produce incorrectly working circuits. Trefzer used another technique to evolve some basic logic gates [9]. Instead of using a time consuming analog circuit simulator, a reconfigurable analog transistor array was employed. However, it was shown that the performance of many solutions decreased in simulation; about 50% of the discovered circuits failed in the simulation. These findings indicate that the evolved solutions are highly dependent on the utilized platform. Trefzer also noticed that even the basic logic gates get harder to evolve as the complexity increases. While the evolutionary design of the NAND and the NOR circuits was successful for almost all runs, only 2 out of 50 runs produced a fully working solution for XOR and XNOR circuits. Later, Walker et al. adopted another technique to evolve variability tolerant transistor-level circuits [10]. The time needed to calculate the fitness function was reduced using a cluster of SPICE simulators. Despite the fact that it was possible to evolve correct solutions, only relative small problem instances were investigated. It is necessary to note, however, that the design of variability tolerant circuits is a more complex task compared to the aforementioned ones.

One of the goals of the early pioneers of evolvable hardware was to evolve complex circuits. From the beginning, however, the researchers struggle with various issues which prevented them from achieving this goal. The scalability of evaluation of candidate solutions probably represents the most limiting factor. This phenomenon is even worse at transistor-level where more complex behavior have to be evaluated. To reduce its effect, various hardware accelerators were proposed in literature. Authors of the first FPGA-based accelerators used the programmable logic primarily as a coprocessor for fast evaluation of candidate solutions [11]. The evolutionary algorithm was usually executed on a personal computer that was connected with FPGA and provided configuration

bitstreams representing candidate circuits. With emerging of more advanced FPGA technology, the authors started implementing the entire evolvable systems in FPGAs [12], [13], [14]. The first accelerators were based on a complete hardware implementation of genetic as well as fitness engine. Despite the fact that the authors reported a significant reduction of time needed to evaluate a single candidate solution, the hard-wired genetic engine did not enable to implement a more sophisticated search strategy. Later, with the development of a deep sub-micron semiconductor technology, new FPGAs equipped with PowerPC processors operating at 400 MHz were introduced. As the processors were directly connected via a fast local bus to programmable elements of the FPGA, the evolvable hardware systems could simultaneously benefit from the fast evaluation of candidate circuits directly in the FPGA and software implementation of evolutionary algorithm (EA) which was more sophisticated than in the previous circuit implementations [15], [16]. Recently, Xilinx introduced a new family of programmable SoCs denoted as Zynq-7000 equipped with a dual-core ARM processor tightly coupled with 7-series Xilinx programmable logic which has a great potential for evolvable hardware.

In this paper, a novel approach to the evolution of transistor-level digital circuits is proposed. In order to reflect behavior of real transistors and simultaneously keep the complexity of fitness computation at a reasonable level, a discrete high-level simulator of transistor circuits is introduced. The main feature of the proposed simulator is the support of bidirectional signal flow and ability to handle multiple logic levels. The goal is to obtain a tool which will be able to simulate digital circuits not only faster than by means of an analog SPICE-based simulator, but also accurately. In order to further improve the performance and scalability of fitness evaluation, a hardware accelerator which implements the proposed simulator in connection with evolutionary algorithm is developed in FPGA.

The paper is organized as follows. Section II introduces Xilinx Zynq Platform and discusses the principles of evolvable systems. Section III describes the method designed for the evolutionary synthesis of transistor-level circuits. Section IV discusses the architecture of the proposed FPGA-based accelerator. Section V contains the evaluation of the accelerator and discusses the obtained results. Concluding remarks are given in Section VI.

II. Xilinx Zynq-7000 Platform

Recently, a new family of programmable SoCs (system-on-chip) denoted as Zynq-7000 was introduced by Xilinx. In contrast with common FPGAs, the Zynq platform is a processor-centric system equipped with ARM processor tightly coupled with 7-series Xilinx programmable logic. As the architecture suggests, this platform targets the applications that benefit from the software based control and hardware-based processing.

The processing system (PS) consists of a dual-core ARM Cortex-A9, memory interfaces and I/O peripherals. The Zynq processors always boot first, programmable logic can be configured as part of the boot process or configured at some point in the future. ARM processor is equipped with two separate 32kB L1 caches for instruction and data, shared 512kB L2 cache, 256-kB on-chip memory, memory management unit and two NEON co-processors extending the 32-bit instruction set of ARM processor by multimedia instructions operating over 128-bit data widths. Each core can operate on 1 GHz operating frequency in both symmetrical as well as asymmetrical multiprocessing configuration.

The architecture of programmable logic (PL) fabricated with 28 nm technology is similar to Xilinx's Artix-7 and Kintex-7 families with the usual programmable resources consisting of configurable logic blocks (CLBs), on-chip block memories (BRAMs), DSP blocks and configurable I/Os. Each configurable logic block is equipped with four 6-input lookup tables (LUTs) and eight flip-flops (FFs). The LUTs in 7-series FPGAs can be configured as either one 6-input LUT (64-bit ROMs) with one output, or two 5-input LUTs (32-bit ROMs) with separate outputs but common addresses or logic inputs.

A. Evolvable systems on Xilinx Zynq

From the viewpoint of evolvable hardware, the processor-centrinc Zynq combining powerful processors with flexible programmable logic represents an interesting and potentially beneficial platform. The evolutionary algorithm can be executed on ARM processor while the candidate solutions can be efficiently evaluated in programmable logic. In addition to that, the EA can run either on top of a high-level OS or directly on a dedicated processor core.

The insufficient support for reconfiguration of former FPGA families was the key factor for introducing virtual reconfigurable circuits (VRCs) [17]. The VRC implemented using the programmable logic is, in fact, a second reconfiguration layer developed on the top of an FPGA used to evaluate candidate solutions. The main benefit of VRC can be seen in very fast reconfiguration capabilities and possibility to define application-specific programmable elements.

As the performance of the internal reconfiguration system of recent 7-series FPGAs noticeably increased, it seems to be useful to employ the dynamic partial reconfiguration. Dynamic partial reconfiguration enables to quickly redefine behavior of a part of programmable logic while the rest of programmable logic is still working. Even if the research in this area is still in the beginning, it has been already shown that an evolutionary system utilizing the dynamic reconfiguration can achieve the performance comparable with VRC [18]. As a consequence, more dynamically reconfigurable fitness units can be placed within a single FPGA chip by employing this technique. Thus, additional speedup can be achieved because multiple fitness units are able to evaluate more candidate solutions in parallel.

III. Evolutionary design of transistor-level circuits

In order to evolve digital circuits at transistor level, a suitable representation enabling to encode complex graph structures containing multiple connections (junctions) is needed. Various approaches to encode the candidate solutions were adopted in literature. For example, Zaloudek et al. encoded the circuits directly using a CGP-like representation. No loops were allowed. Walker et al. utilized representation which separates the directed graph topology of the genotype from

the transistor circuit topology of the phenotype [10]. The main advantage of that approach is that it allows loops and multiple connections to occur within the phenotype, whilst maintaining the feed-forward nature of the representation. However, an extra decoding step is required to convert between the floating-point representation and phenotype.

In this paper, we utilize an integer-based encoding that is inspired by CGP [4]. The proposed encoding has the ability to encode loops and multiple connections and simultaneously it does not require a decoding phase. Hence, the encoding can directly be utilized in the FPGA-based accelerator.

A. Circuit Representation

A candidate circuit is represented by means of an array of elementary nodes arranged in n_c columns and n_r rows. Each node consists of a single output pin and two source pins that can independently be connected either to the output of a node placed in previous l columns or to one of the primary circuit inputs. According to the configuration, each node can act as a wire, junction, n-mos transistor or p-mos transistor. The utilized nodes are shown in Figure 1.

Presence of a junction node represents the main feature of the proposed technique. This node is able to combine two input signals and one output signal together. As a consequence of that, loops and multiple connections are natively supported.

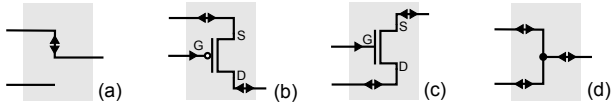


Fig. 1. Basic building blocks of transistor-level circuits: (a) wire connecting the first pin with output, (b) p-mos transistor, (c) n-mos transistor, and (d) junction that combines two signals together. If a proper level (voltage) is applied on the gate electrode (V_{ss} for p-mos, V_{dd} for n-mos), transistor connects its source electrode with drain. Possible directions of signal flow which have to be considered during the evaluation are shown.

The following encoding scheme is utilized. The primary inputs as well as node outputs are labeled from 0 to $n_i + n_c.n_r - 1$, respectively, where n_i denotes the number of primary inputs. A candidate solution is represented in the chromosome by $n_c.n_r$ triplets (x_1, x_2, f) determining for each node its function f, and labels of nodes or primary inputs (x_1 and x_2) that are connected to the source pins. The last part of the chromosome contains n_o integers specifying the labels of nodes where the n_o primary outputs are connected to. The first and last primary input is reserved for power supply rails.

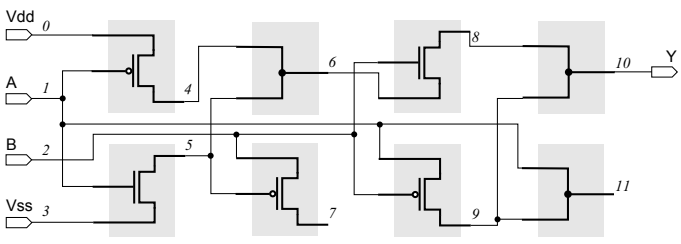


Fig. 2. Example of a candidate circuit implementing function $Y = A \oplus B$ (XOR) using four transistors. Parameters are as follows: n_i=4, n_o=1, n_c=4, n_r=2, l=1. Chromosome: (0,1,p-mos)(1,3,n-mos)(4,5,junct)(2,5,p-mos)(2,6,n-mos)(1,2,p-mos)(8,9,junct)(1,9,junct)(10).

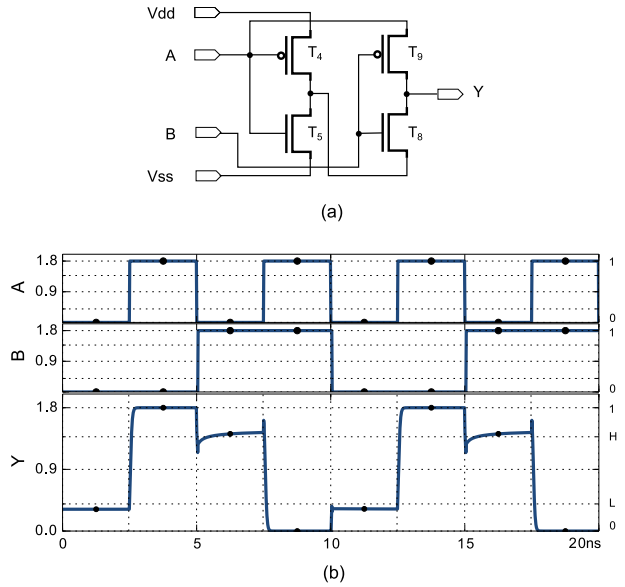


Fig. 3. (a) Schematic of the circuit encoded using candidate solution illustrated in Figure 2 and (b) its output waveform obtained using an analog SPICE simulator, a TSMC 0.18μm CMOS technology and 1.8V power supply. It can be seen that although the chromosome contains five transistor nodes, only four transistors are utilized in the resulting circuit. As it was discussed, there are two of four input combinations (A='0', B='0' and A='0', B='1') with reduction in output voltage swing.

Figure 2 demonstrates the principle of utilized encoding on a XOR circuit implemented using pass-transistor logic. This chromosome encodes a candidate circuit using eight nodes, however, only some of them contribute to the phenotype and are active. The activity of a node is determined as follows. A node is active if its output is (a) connected to any of the primary outputs or (b) to the input of an active node. It means, that node 7 and node 11 represent inactive nodes.

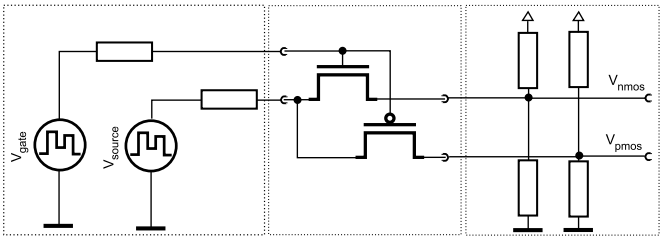
Schematics of the corresponding phenotype constructed using the active nodes is given in Figure 3a. The resulting circuit contains four transistors, two p-mos transistors and two n-mos transistors. The first complementary pair of transistors which comprises T4 and T5 represents a common CMOS inverter. The second pair of transistors (i.e. T8 and T9) is used to pass the appropriate logic level (direct or inverted version of input line A) to the primary output. Even if this XOR gate implementation is very compact, it is rarely used in practice because of the connection of source electrode of p-mos transistor T9. The electrode is connected to the input line instead of being connected to power lines which causes a reduction in output voltage swing (i.e. there is a threshold loss at the output node for certain input combination). This effect is visible in the output waveform shown in Figure 3b.

B. Evaluation of a candidate circuit

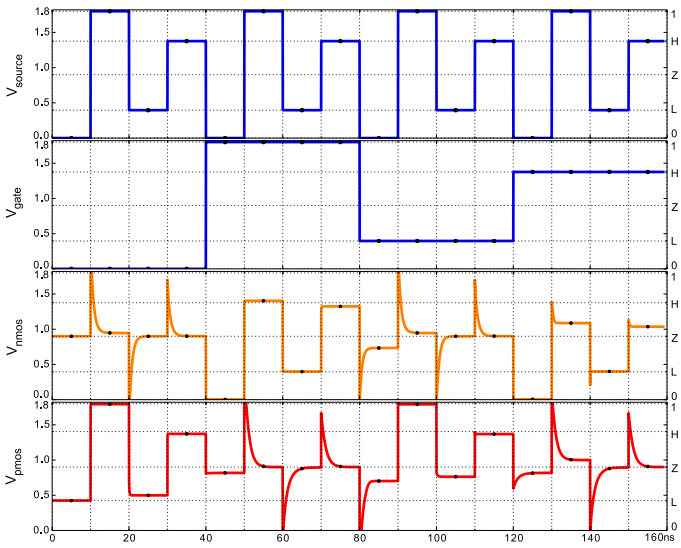
Evaluation of a candidate circuit consists of two steps. Firstly, the set of active nodes is determined. Only the active nodes are considered during the evaluation. The inactive nodes are ignored. Potentially unwanted nodes causing short-circuits can be removed in this step (e.g. node 11 in Figure 2). Note that if there is a requirement to make the evaluation efficient, this step is present also in a common CGP implementation.

Secondly, multi-level discrete event-driven simulator is utilized to determine responses for each input combination.

In order to reflect the behavior of real transistors, the simulator operates on seven discrete levels: '0' (V_{ss}), '1' (V_{dd}), 'L' (degraded logic 0, i.e. $V_{ss}+V_t$), 'H' (degraded 1, i.e. $V_{dd}-V_t$), 'Z' (high-impedance), 'X' (shortage), 'U' (unknown). The last value is used to identify already calculated values and helps in avoiding worthless events. As a consequence of the discretization, the behavior of each elementary node can be defined by extended truth table. For example, an open n-mos transistor is known to pass logic 0 well but logic 1 poorly. This loss is known as threshold drop. An attempt to pass logic 1 (V_{dd}) never gives value above $V_{dd}-V_t$, where V_t is threshold voltage. Among others, the extended truth table thus includes the following rule: G='1' $\wedge$ S='1' $\wedge$ D='Z' $\Rightarrow$ D='H'.



(a)



(b)

Fig. 4. (a) Test bench and (b) output waveforms for p-mos and n-mos transistors utilized to derive the content of extended truth table which is embedded in discrete simulator. The waveform was obtained using an analog SPICE simulator, a TSMC 0.18μm CMOS technology and 1.8V power supply. The corresponding discrete values are shown on the right side. Note that the peaks are caused by charging and discharging of parasitic capacitance.

Threshold drop is a well-known phenomenon which is well understood. However, there can occur a situation whose understanding and proper evaluation requires deeper knowledge of utilized technology. In order to avoid errors in the design of extended truth table, we utilized an analog simulator to determine the output value for a given state of input electrode and gate. Figure 4a shows the SPICE test bench utilized to evaluate the response of p-mos and n-mos transistors. The test bench consists of two piecewise linear voltage sources, transistors and output stage. The output stage consists of voltage divider that ensures that a load is applied to the test circuit. In addition to that, the divider helps to identify presence of high impedance at transistor outputs. Figure 4b shows the obtained output waveform. Each voltage source generates four discrete values – two levels corresponding with full voltage and two levels representing degraded values. As each source utilizes a different clock period, every possible steady-state combination of input values is evaluated during the analysis. It can be easily verified that the n-mos transistor outputs H value for $G =' 1' \wedge S =' 1'$ (see the V_{nmos} waveform at 55 ns).

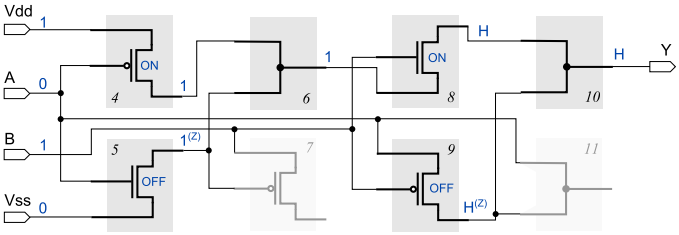


Fig. 5. Example from Figure 2 accompanied with the informations obtained from discrete simulator for A='0', B='1'.

Let us describe the process of evaluation of the situation shown in Figure 5. Firstly, all nodes are initialized to value 'U' which indicates that the nodes have not been yet evaluated. Then, the primary inputs are forced to A='0', B='1'. This change invokes evaluation of nodes 4, 5, 8 and 9. Node 4 is evaluated as an open p-mos transistor which connects drain with V_{dd}. Thus, value '1' appears on its output and node 6 is planned to be evaluated. Then, node 5 is evaluated as closed p-mos transistor providing 'Z' to its output. As the output of this node is connected to node 6, this node should be recalculated. However, as it is already at the end of the update queue, it is not appended again. Node 8 represents an open p-mos transistor, however its source is connected to node 6 which produces 'U' value, thus the output value remains unchanged. Node 9 evaluates as closed p-mos transistor and changes its output from 'U' to 'Z'. This change causes recalculation of node 10. Then, the junction encoded by node 6 evaluates as '1' and propagates this change to node 5 and node 8. The output value of node 10 remains unchanged because the first input contains unknown value 'U'. Node 5 changes its output value from 'Z' to '1' and invokes an update of node 6. Node 8 is an open n-mos transistor that degrades value '1' to 'H'. As the resulting value of node 6 remains stable, none additional update is needed and the evaluation continues in the same manner with node 10 and 9. Finally, the update queue is empty and the primary output provides stable value 'H'.

C. Search strategy

As a search algorithm, the $(1+\lambda)$ evolutionary strategy is utilized [4]. The initial population is randomly generated. Every new population consists of the best individual and λ offspring. In the case when two or more individuals have received the same fitness score in the previous population, the individual which did not serve as a parent in the previous population will be selected as a new parent. The evolution

is terminated when a predefined number of generations is exhausted or a required solution is found.

The search is guided by the fitness function which determines how good the current candidate circuit is. For evolution of logic circuits, all possible input combinations have to be applied at the candidate circuit inputs. The output values are collected and the goal is to minimize the difference between obtained responses and required truth table. In order to smooth the search space, the fitness value is constructed as follows. If the obtained output value equals to the expected one, 5 points are added to the fitness value. If the calculated value exhibits the same polarity but represents degraded voltage, 2 points are used. Otherwise, no point is added because the response is invalid. The weights were chosen experimentally. Additional penalties may be applied. If there is a node that evaluates during simulation to 'X', the simulation is terminated and penalty is applied to the total fitness value. Similarly, if the simulator exceeds the predefined number of steps (i.e. node outputs are not in stable state), the simulation is terminated and the fitness value is penalized.

IV. Architecture of the Proposed Accelerator

The architecture of the proposed accelerator is shown in Figure 6. The evolutionary algorithm is implemented using the processing system equipped with 1GB of DDR3 memory. The processing system is connected through point-to-point bi-directional AXI4 interface with programmable logic which implements an acceleration unit consisting of the controller, fitness calculation unit and array of reconfigurable nodes implemented using the systolic array (i.e. VRC).

The controller is responsible for communication with the ARM processor, reconfiguration of the VRC and controlling the simulator. A candidate circuit, represented by bitstream, is generated on ARM processor and transmitted via AXI interface to the controller which simultaneously reconfigures VRC. As soon as the VRC is configured, active nodes are detected. To accomplish this task, the VRC elements are switched to the 'activation mode' for a specified number of clocks. Then, each element, which is evaluated as an active node, proceeds to the 'evaluation mode' and the fitness unit is activated. The fitness unit resets the fitness value register, initializes VRC elements, generates the first input combination and waits for a specified amount of time to enable the input data to propagate through VRC. Then, the output value is sampled for three consecutive clock cycles. This sampling is used to detect a stable output value. The obtained output is compared with the required value and the fitness value is incremented as described above. Then, the VRC is reinitialized, the next input combination is generated and the evaluation is repeated. When the response for the last input combination is calculated (i.e. 2^{n_i-2} input vectors were evaluated), the fitness unit is deactivated and the controller sends the calculated fitness value to ARM processor.

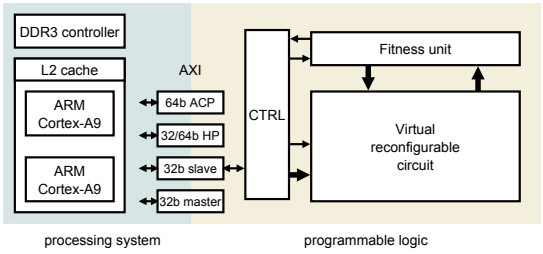


Fig. 6. Structure of the accelerator implemented using Zynq platform

A. Array of reconfigurable elements

The structure of the reconfigurable array utilized to evaluate the candidate solutions is shown in Figure 7. It consists of programmable elements (E_{ij}) placed in a grid of n_c columns and n_r rows that can be configured to implement one of the elementary functions described in Section III-A. Each element can be connected either with a primary input or the output of an element situated in the preceding column. The primary outputs can be connected to the output of any element. The reconfiguration is performed column by column, one column is configured in a single clock cycle.

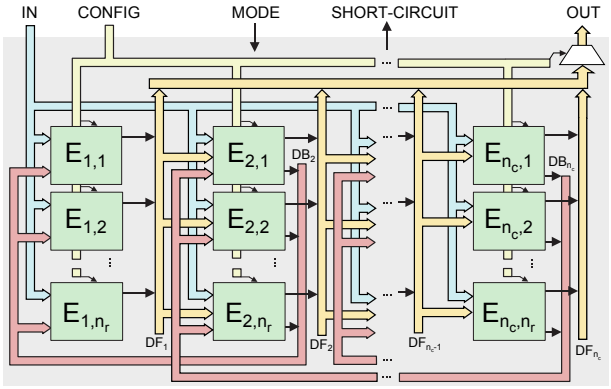


Fig. 7. Architecture of the bi-directional virtual reconfigurable circuit.

In contrast with VRC utilized in [16], forward and backward data path are established between adjacent columns. The forward path (DF_i) supplies the data from outputs of the elements of a certain column to the inputs of the elements placed in the succeeding column. The backward path (DB_i) is used to propagate the changes of element's states back through theirs input pins to the elements in the preceding column. In addition to that, the backward path is utilized to detect active nodes during the activation mode. The proposed implementation allows us to simulate transistor-level circuits exhibiting bi-directional data flow.

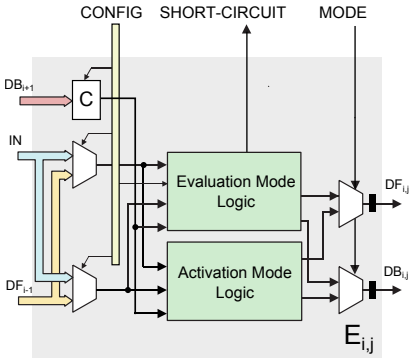


Fig. 8. Structure of the VRC's programmable element.

Due to the inherent parallelism of VRC, it is not necessary to implement event-based message passing to maximize the

TABLE I. AVERAGE TIME (t) NEEDED TO EVALUATE A SINGLE CANDIDATE SOLUTION AND ACHIEVED SPEED UP (s) FOR VARIOUS IMPLEMENTATIONS AND VARIOUS NUMBER OF PRIMARY INPUTS n_i OF THE BENCHMARK CIRCUIT

implementation	platform	frequency	$n_i = 4$		$n_i = 5$		$n_i = 6$		$n_i = 7$	
			t	s	t	s	t	s	t	s
HW accelerator (PL+PS)	Zynq	50MHz (PL), 667MHz	41 us	0.8	44 us	1.7	49 us	3.7	54 us	4.7
HW accelerator (PL only)	Zynq	50MHz	2 us	17.0	3 us	24.3	6 us	30.0	11 us	23.0
ngspice simulator	Xeon	2.3GHz	49 ms	-	60 ms	-	100 ms	-	237 ms	-
discrete simulator	Xeon	2.3GHz	34 us	1.0	73 us	1.0	180 us	1.0	253 us	1.0
discrete simulator	Zynq	667MHz	130 us	0.3	320 us	0.2	762 us	0.3	950 us	0.3

simulator's performance as it was inevitable in the software implementation. Instead, the programmable elements are equipped with output registers and operate in parallel. The simulation of a candidate solution is implemented as follows. Each element calculates its output value according to the rules specified by the extended truth table. Then, with raising edge of clock signal, all elements synchronously update their output registers. The final state of the nodes is resolved after performing a certain number of clock cycles. The number of cycles is determined for each candidate solution in advance according to the number of occurrences of each node type.

The structure of the programmable element is shown in Figure 8. The element consists of two logic blocks that are dedicated to the computation of output values in the evaluation and activation mode. In the activation mode, a backward path is used to determine the active nodes. The value of DB_{ij} is calculated using the data provided by input DB_{i+1} which contains information about active nodes in the succeeding column. Each active node and node connected directly to a primary output generates '1', otherwise 'Z' is generated. The block denoted as C represents a logic that computes a single value according to the knowledge of the values provided by the active elements that are connected to the node E_{ij}. If this block returns '1', it means that the output of the element is connected to an active node. On the other hand, value 'Z' means that no active node utilizes the output of the element.

In the evaluation mode, two possible situations can occur. If a node is active, it determines its output values according to the actual configuration and information from forward and backward data paths. Otherwise, value 'Z' is generated by both outputs. The output value reflects the behavior of the current node.

V. EXPERIMENTAL RESULTS

A. Results of Synthesis

The evolutionary platform was described in VHDL and synthesized using Xilinx Vivado. The results of synthesis showing the amount of PL resources required to implement various VRCs are summarized in Table II. It can be seen that the utilization of look up tables (LUTs) noticeably increases with the increasing size of VRC while the number of flip-flops (FFs) remains stable. This behavior is caused by the fact that the LUTs are occupied mainly by (a) multiplexers ensuring reconfigurability and (b) the backward path logic resolving the output value of each element. Interestingly, the number of primary inputs n_i has a negligible impact on the amount of occupied resources. According to the results of synthesis, the worst-case operational frequency is 55 MHz.

B. Performance of the proposed accelerator

In order to evaluate performance of the proposed accelerator, we chose the problem of the evolutionary design of a multiple-input NAND gate. The following setup was utilized. The processing system operates at 667 MHz, programmable logic runs at 50 MHz, VRC consists of 10×8 elements, $\lambda = 4$, n_i varies from 4 to 7, and the maximum number of generations is set to $500 \cdot 10^3$.

The obtained results are summarized in Table I. The performance is expressed as the average time needed to evaluate a single candidate solution. The time is averaged over 10 independent experimental runs and over all evaluations. The performance of the proposed hardware accelerator is compared with three SW-based implementations – the analog circuit simulator ngSPICE running on Intel Xeon E5-2630@2.30 GHz and the proposed event-based discrete simulator, described in Section III, running on Intel Xeon and Zynq ARM processor.

The performance of the discrete simulator is significantly higher (approx. three orders of magnitude) comparing to performance of ngSPICE. In addition to that, the performance decreases exponentially with the increasing number of primary inputs n_i. This effect is caused by doubling of the number of input combinations that have to be evaluated. If we compare the ARM-based and Xeon-based implementations, the ARM-based one requires approximately three times more time to evaluate a candidate solution. This corresponds with the fact that the operating frequency of ARM is approx. three times lower.

According to the experimental evaluation, the proposed accelerator provides the speed up from 0.8 to 4.7 compared to a software-based implementation running on a common CPU (see 'HW accelerator (PL+PS)' in Table I). However, theoretical performance of the acceleration unit implemented in PL should be noticeably higher. Thus we performed an in-depth analysis and identified that AXI interface represents the main bottleneck. The communication introduces substantial overhead which cannot be sufficiently overlapped with evaluation phase as the time needed to evaluate a candidate solution represents only a fraction of time needed to send

TABLE II. AMOUNT OF XC7Z020 RESOURCES REQUIRED TO IMPLEMENT VARIOUS VRCS

VRC	$n_i = 4$		$n_i = 5$		$n_i = 6$		$n_i = 7$	
($n_c \times n_r$)	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
7×4	16 %	2 %	15 %	2 %	15 %	2 %	15 %	2 %
8×6	29 %	3 %	31 %	3 %	28 %	3 %	30 %	3 %
10×8	54 %	4 %	55 %	4 %	53 %	4 %	55 %	4 %

the chromosome into PL. In particular, the communication introduces approx. 95% overhead for $n_i = 4$ and 20% for $n_i = 7$. In order to mitigate the communication overhead and increase the performance, we modified the architecture of the accelerator. The modification consists in an implementation of a hardware circuit which is able to generate the required chromosomes directly in PL. As a consequence of this modification, the amount of data transfers between PS and PL is significantly reduced. The obtained speedup is given in Table I, see 'HW accelerator (PL only)'. The speedup of the accelerator increased by a factor of 5-20.

C. Evolutionary design of logic circuits

The proposed method and the implemented accelerator were experimentally evaluated on the evolutionary design of basic logic circuits having up to 5 inputs. Two experiments are presented in this paper – evolutionary design of 2-input XOR gate and 4-input AND-OR-INVERT gate. The goal of the experiments was to evolve fully functional implementations exhibiting full voltage swing. It means that no degradations are allowed on primary inputs and outputs. The XOR gate as well as AND-OR-INVERT gate with full voltage swing can be implemented using 8 transistors in CMOS logic.

Firstly, we investigated the impact of CGP parameters such as l-back parameter, the number of nodes (i.e. VRC size) and mutation rate h. Note that the l-back was investigated only in software implementation. In order to evaluate the effect of these parameters, we calculated *success effort* which measures the expected number of generations before a solution is found. Success effort was calculated as suggested in [19]. The results were obtained from 50 independent runs using the following experimental setup: $g_{max} = 500 \cdot 10^3$, $\lambda = 4$, $h = \{1, 5, 15\}$, $n_c \times n_r = \{10 \times 8, 7 \times 4\}$.

Interestingly, the l-back parameter does not have any significant impact on the success effort of the investigated problems if a sufficient mutation rate is provided ($h \approx 5\%$). Otherwise, the higher values of l-back caused deterioration of the success rate for $h < 5\%$. Hence, it seems to be beneficial that the proposed accelerator has the l-back parameter fixed to one.

The obtained success effort plots for different setting of CGP parameters are shown in Figure 9 and 10. It can be seen that the evolutionary design was successful in both cases. More than 80% of evolutionary runs successfully discovered a fully functional solution of a XOR gate. A run is successful if the evolved circuit obtains the maximal possible fitness score which means that it exhibits full voltage swing. More generations, however, are required to achieve the same success rate in the case of the AND-OR-INVERT gate design. Over 70% of the evolved XOR gate solutions occupy 6 transistors. The largest solution consists of 10 transistors. The best discovered AND-OR-INVERT implementation containing 8 transistors was found in 24% successfull evolutionary runs. The largest implementation utilizes 14 transistors.

By observing the success effort across different mutation rates, we can identify that $h = 5$ ($h = 15$) provides the best results for array 7×4 (10×8). This result complies with a general recommendation that advises to set h to be equal to 5% of chromosome size [4]. The chosen setup $h = 5$ ($h = 15$) ensures that up to 6.2% (5.8%) genes are modified.

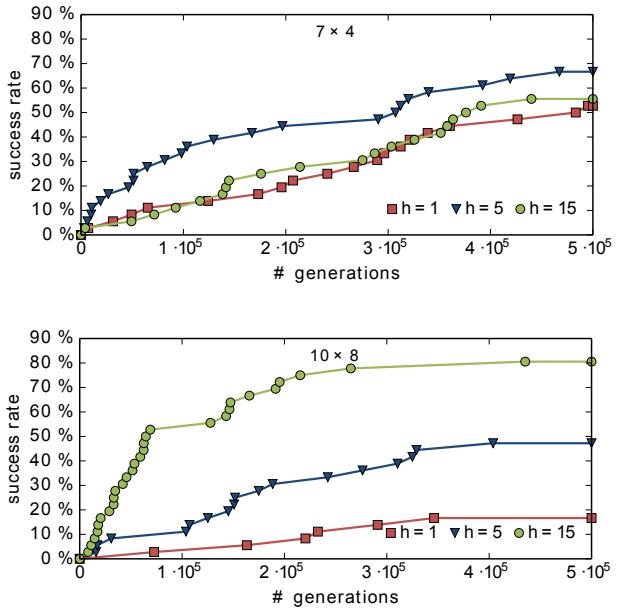


Fig. 9. Success effort of evolutionary design of the 2-input XOR gate for different sizes of VRC

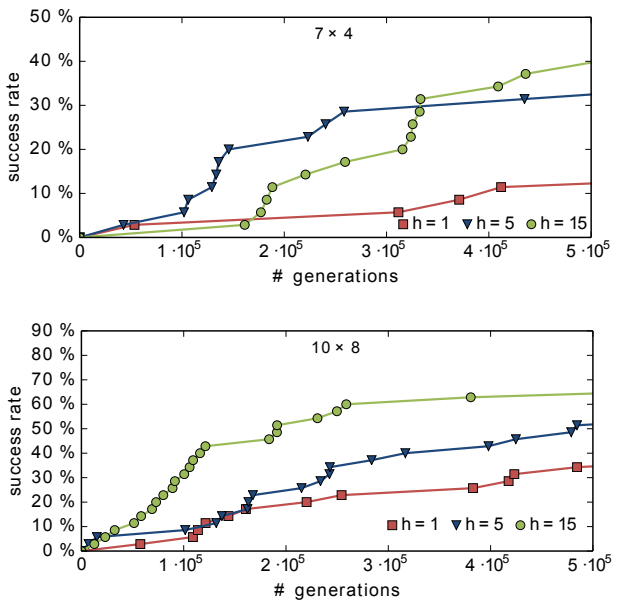


Fig. 10. Success effort of evolutionary design of the 4-input AND-OR-INVERT logic circuit for different sizes of VRC

The increasing VRC size has a positive impact on the success effort in both cases. The lower number of generations are required to achieve the same success rate.

Example of the evolved XOR circuit is shown in Figure 11. Note that the inactive nodes were omitted due to the limited space. While a common CMOS implementation requires 8 transistors, the evolved circuit consists of 6 transistors, providing a full voltage swing at the outputs and exhibiting high operating frequency. The evolution discovered a solution which is known as the transmission-gate XOR circuit.

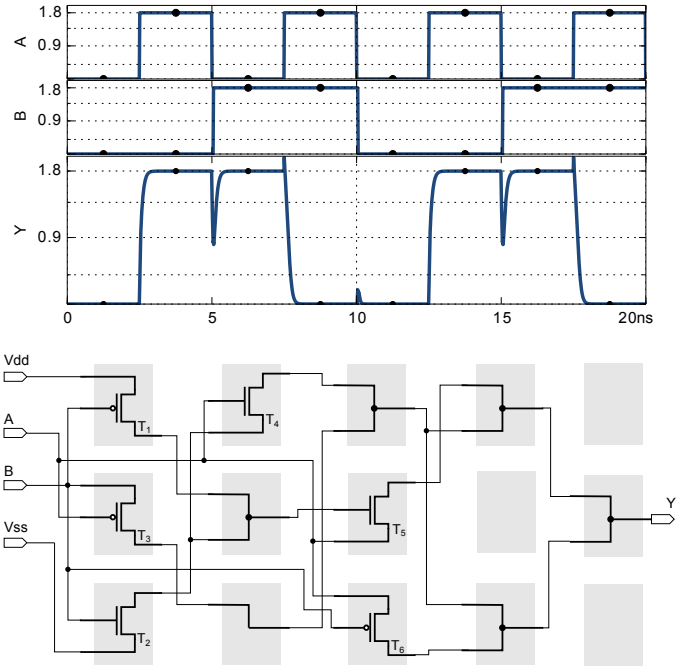


Fig. 11. Evolved XOR circuit (bottom) and its output waveform obtained using analog SPICE simulator (top). The circuit consists of an inverter realized using T1 and T2, a transmission gate implemented using T5 and T6 and two additional transistors T3 and T4.

VI. Conclusion

A new approach suitable for the evolutionary design of transistor-level digital circuits based on event-driven discrete simulator was introduced in this paper. In addition to that, a hardware accelerator implemented using Xilinx Zynq platform was proposed. The accelerator consists of an array of reconfigurable nodes supporting a bidirectional data flow and processing system running the evolutionary algorithm. We evaluated the proposed method in evolutionary design of basic logic circuits and demonstrated that it is tractable to evolve logic circuits directly at transistor-level. We showed that the hardware implementation is able to provide a reasonable speedup (30) w.r.t. an optimized software implementation even for those relatively small circuits.

However, we identified that the communication interface represents a bottleneck which noticeably hurts the overall performance. Thus, future work has to be conducted to eliminate this problem and exploit all the features and advantages of Zynq platform. Some further changes in the proposed method are expected in order to increase the achieved speedup. For example, the reconfigurable elements could be implemented more efficiently for a cost of BRAM memories. As a consequence, multiple reconfigurable circuits could be placed in the programmable logic and more candidate circuits could be evaluated in parallel. Another possibility is to utilize a dynamic partial reconfiguration which enables to reduce the area overhead of VRC.

Acknowledgments

This work was supported by the Czech science foundation project 14-04197S.

References

[1] V. Vassilev, D. Job, and J. Miller, "Towards the Automatic Design of More Efficient Digital Circuits," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. Los Alamitos, CA, USA: IEEE Computer Society, 2000, pp. 151–160.

[2] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 8–35, 2000.

[3] ——, "Principles in the Evolutionary Design of Digital Circuits – Part II," *Genetic Programming and Evolvable Machines*, vol. 1, no. 3, pp. 259–288, 2000.

[4] J. F. Miller, *Cartesian Genetic Programming*. Springer Verlag, 2011.

[5] A. Stoica, C.-S. Lazaro, D. Keymeulen, and K. Hayworth, "Evolution of CMOS circuits in simulations and directly in hardware on a programmable chip," in *1999 Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. Morgan Kaufmann, 1999, pp. 1198–1203.

[6] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, Y. Klimeck, R. Tawel, and V. Duong, "Evolution of analog circuits on field programmable transistor arrays," in *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*, 2000, pp. 99–108.

[7] A. Maache, "A prototype parallel multi-FPGA accelerator for SPICE CMOS model evaluation," Ph.D. dissertation, University of Southampton, 2011.

[8] L. Zaloudek and L. Sekanina, "Transistor-level evolution of digital circuits using a special circuit simulator," in *Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 5216. Springer Verlag, 2008, pp. 320–331.

[9] M. Trefzer, "Evolution of Transistor Circuits," Ph.D. dissertation, Ruprecht-Karls-Universitt Heidelberg, 2006.

[10] J. Walker, J. Hilder, and A. Tyrrell, "Towards evolving industry-feasible intrinsic variability tolerant cmos designs," in *IEEE Congress on Evolutionary Computation*, 2009, pp. 1591–1598.

[11] A. Thompson, "Silicon evolution," in *Proceedings of the First Annual Conference on Genetic Programming*, ser. GECCO '96. Cambridge, MA, USA: MIT Press, 1996, pp. 444–452.

[12] Y. Zhang, S. Smith, and A. Tyrrell, "Digital Circuit Design using Intrinsic Evolvable Hardware," in *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*. Seattle, USA: IEEE Computer Society, 2004, pp. 55–62.

[13] L. Sekanina and S. Friedl, "An evolvable combinational unit for FPGAs," *Computing and Informatics*, vol. 23, no. 5, pp. 461–486, 2004.

[14] L. Sekanina and T. Martinek, "Evolving image operators directly in hardware," in *Genetic and Evolutionary Computation for Image Processing and Analysis, EURASIP Book Series*, vol. 8. Hindawi Publishing Corp., 2007, pp. 93–112.

[15] K. Glette and J. Torresen, "A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device," in *Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 3637. Springer, 2005, pp. 66–75.

[16] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.

[17] L. Sekanina, *Evolvable components: From Theory to Hardware Implementations*, ser. Natural Computing. Springer Verlag, 2004.

[18] R. Dobai and L. Sekanina, "Image filter evolution on the xilinx zynq platform," in *Proc. of the 2013 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE Circuits and Systems Society, 2013, pp. 164–171.

[19] M. Walker, H. Edwards, and C. H. Messom, "Success effort and other statistics for performance comparisons in genetic programming," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 4631–4638.