Fast Evolution of Large Digital Circuits

XIAOXUAN SHE
State key Laboratory of ASIC & System
Fudan University
825 Zhangheng Road, Shanghai
CHINA
Chamb_she@hotmail.com

Abstract: - Evolvable hardware (EHW) refers to self-reconfiguration hardware design, where the configuration is under the control of an evolution algorithm. One of the main difficulties in using EHW to solve real-world problems is scalability, which limits the size of the circuit that may be evolved. This paper outlines evolvable hardware based on a 2-LUT (2-input lookup table) array, which allows the evolution of large circuits via decomposition. The proposed EHW has been tested with multipliers and logic circuits taken from the Microelectronics Centre of North Carolina (MCNC) benchmark library. The experimental results demonstrate that the proposed scheme improves the evolution of logic circuits in terms of the number of generations, area and delay, reduces computational time and enables the evolution of large circuits. The proposed EHW automatically generates a complete circuit netlist in a SDRAM. Because of the low cost and large data storage of a SDRAM, the evolvable hardware provides a good platform to evolve large circuits.

Key-words: - Evolvable hardware, Evolutionary computation, Adaptive systems, Digital circuits, Problem decomposition, Circuit design

1 Introduction

Evolvable hardware (EHW) [1] is a technique to automatically design circuits (sensors [2], digital, analogue, optics, antennas, and robots [3]), where the circuit configuration is under the control of evolution algorithms (EAs) [4]-[8]. In these evolution algorithms, first all the chromosomes are randomly initialized. Second, the fitness value of each individual is computed. Third, the fittest individual is selected. Fourth, if the fitness value of the chromosome is 100% or the number of generations has reached the maximum value set by the user for that particular experiment, the evolution process stops. If the conditions are not met, a new population will be generated by crossovering and mutating the best chromosomes (selected at the third step). In the next cycle, all newly created chromosomes are evaluated and operated in the same way. The evolution process continues until the fitness value of the chromosome is 100% or the number of generations has reached the maximum value set by the user for that particular experiment.

The chromosome defines the structure of the logic circuit, the connectivity between logic cells, and cell types. The chromosome of EHW can be subdivided into gate-level [9]-[11] and function-level [12]-[15] representations. In gate-level evolution, the design of digital circuits is based on primitive hardware gates such as AND and OR. Gate-level EHW can only evolve solutions for simple problems. In function-level evolution, the design of circuits is based on higher functions such as sin, adders, etc. Function-level EHW can evolve solutions for complicated problems, but it requires human intervention to select the most appropriate functions

for specific problems.

In order to evolve large and complex circuits, Stomeo, Kalganova, and Lambert introduced a divide-and-conquer approach for the evolution of a complex task [16]. The principle is to divide a complex task into simpler subtasks in order to evolve each of these subtasks and then to merge the evolved subsystems, reassembling a new evolved complex system. Hong and Chou divided a complex circuit into modules based on outputs [17]. This scheme reduces the number of required generations of evolution, but this separation of modules based on outputs is only possible when there are many outputs in hardware. It has little effect when the hardware has few outputs. Stomeo *et al*. decomposed a complex system into two subsystems G and H based on inputs [18]. The subsystem H is generated with MUXes and does not participate in the evolution process. The subsystem G, which has fewer inputs and more outputs than the original system, is evolved in an AND-OR PLA (Programmable Logic Array) structure. This decomposition scheme reduces the number of generations of evolution, but the AND-OR PLA have only AND gates and OR gates for evolution and hence may require more generations than evolvable hardware with a bit more types of logic gates.

Evolvable hardware can be commercial FPGAs such as Xilinx devices [19], VRCs (Vitual Reconfigurable Circuits) [20], [21] or ASICs [22]. Ideally, evolvable hardware needs fine grained configuration and be random configuration safe. However, current devices such as Xilinx FPGAs only have course partial reconfiguration capabilities [23]. The older XC6200 [24] series was an exception, having a more flexible configuration system, however, it is no longer manufactured. A VRC solution is to implement a custom reconfigurable architecture on top of a commercial FPGA, but implementing one reconfigurable circuit upon another is inefficient limiting the size of circuit available for use. An ASIC solution provides the maximum flexibility to the designer, but the cost of ASIC fabrication is great. The proposed evolvable hardware is an ASIC solution.

This paper presents a new type of evolvable hardware based on a 2-input lookup table array. A 2-LUT (2-input lookup table) is a 4-bit memory module, which uses the address lines as its 2 inputs and returns the contents of the addressed location as the output of a Boolean function. The proposed evolvable hardware has the following merits.

- It is capable of evolving large combinational circuits and sequential circuits with a small number of lookup tables. It can automatically decompose a large circuit into smaller sub-circuits based on both inputs and outputs, and combine these evolved sub-circuits together to generate a complete circuit netlist.
- It is capable of evolving area and delay optimized circuits. It can evolve combinational circuits and sequential circuits with the smallest area or the shortest delay.
- It reduces the number of generations to evolve a large circuit significantly. It requires fewer generations than direct evolution, the decomposition scheme based on outputs [17] and the decomposition scheme based on an AND-OR PLA [18].
- It automatically generates a complete circuit netlist in a SDRAM. Because of the low cost and large data storage of a SDRAM, the evolvable hardware provides a good platform to evolve large circuits

This paper is organized as follows: the next section describes the principles and architecture of proposed evolvable hardware. Section 3 gives the experimental results, followed by the conclusions.

2 Proposed Evolvable hardware

2.1 Decomposition

The proposed evolvable hardware comprises lookup tables and Muxes. It can tackle a hard-to-evolve system that cannot become evolved within a certain

number of generations. The proposed scheme reduces the number of inputs and outputs of the evolved logic circuit, so that the number of required generations decreases to speed up the evolutionary process. The proposed method is based on rewriting the truth table in such a way that the inputs needed to describe the system are decomposed in two parts. Supposing that a system with *n* inputs and *m* outputs, see Fig.1(a), should be evolved using an evolutionary algorithm. The functionality of this system can be described by the truth table given in Fig.1(b), where $p = 2^n$ is the number of products (or the so-called number of input-output combinations). The system depicted in Fig.1(a) can be decomposed into two subsystems as shown in Fig.1(c). The subsystem (lookup tables) with *r* inputs and *s* outputs is the evolvable part of the newly created system, where $s = m \times 2^{n-r}$. The subsystem (Muxes) with *s* data inputs, *n-r* control inputs and *m* outputs is the fixed part of the circuit that is generated using multiplexers. This part does not participate in the evolutionary process.

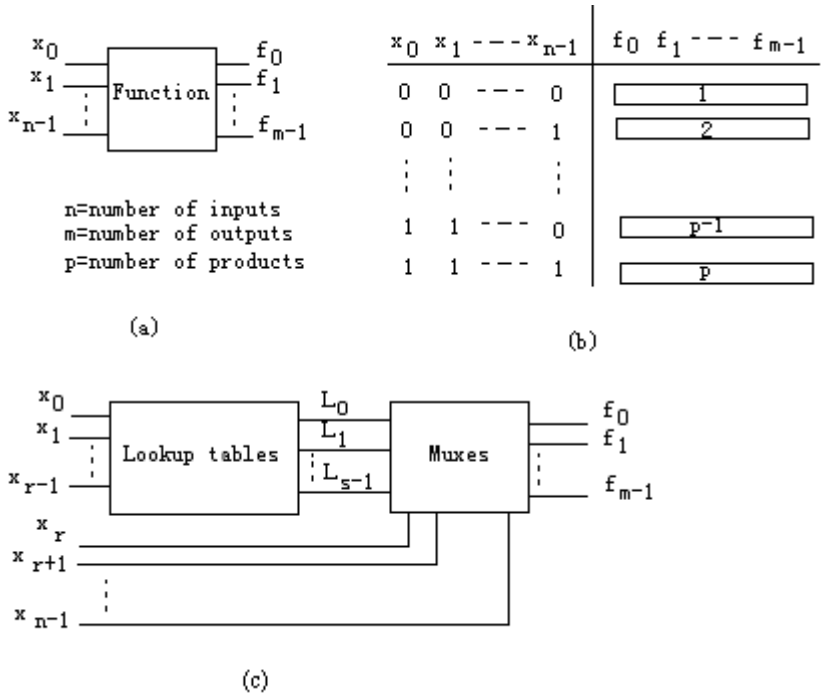
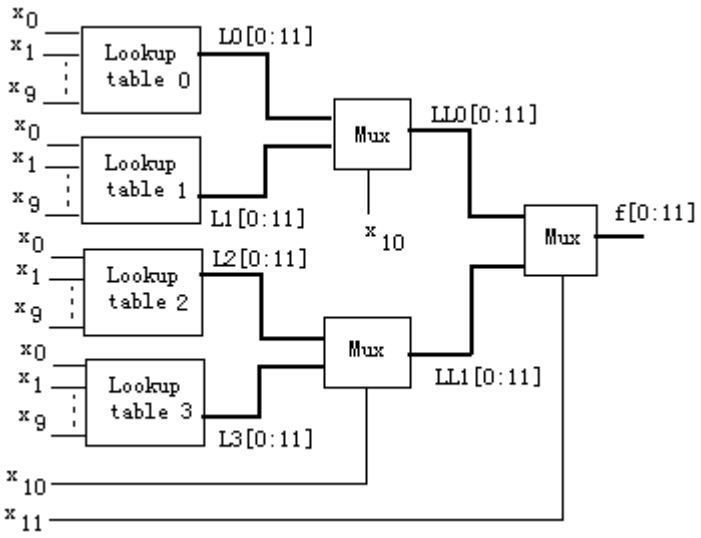


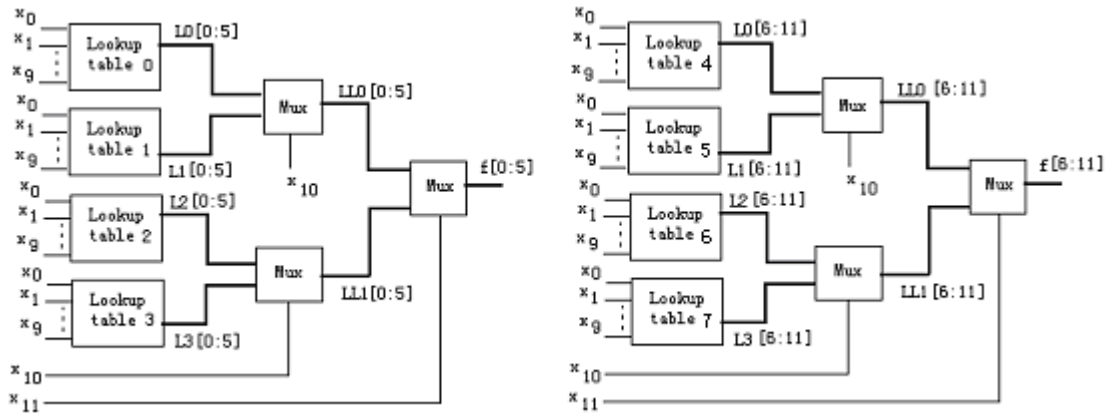
Fig.1. General description of logic circuits and decomposition of the initial logic circuit

Here is an example to describe how this system works. Supposing that a combinational logic circuit with 12 inputs and 12 outputs should be evolved. To reduce the number of required generations, using the above decomposition approach, this logic circuit is decomposed into four 10-input/12-output lookup tables as shown in Fig.2(a). A Mux tree is used to generate the final outputs as shown in Fig.2(a). In Fig.2(a), [0:11] represents a 12-bit aggregated output {output 0, output 1, …, output 11}. The 10 inputs {x_0, x_1, …, x_9} are input to 4 lookup tables (Lookup table 0-3) concurrently. The resultant four 12-bit outputs (L0-3[0:11]) are input into two Muxes (their control inputs are both x_{10}) to generate two 12-bit outputs (LL0-1[0:11]). The final Mux controlled by x_{11} outputs the final 12-bit result of the logic circuit *f*[0:11]. In Fig.2(a), the Mux tree does not participate in the evolutionary process. The four lookup tables (Lookup table 0-3) can be evolved sequentially or concurrently. If there is only one EA (Evolution Algorithm) engine, these four lookup tables can be evolved sequentially. If there are four EA engines, they can be evolved in parallel to reduce the evolution time further.

To reduce the number of required generations further, this 12-input/12-output logic circuit can be decomposed based on both inputs and outputs. For example, the decomposed logic circuit depicted in Fig.2(a) can be further decomposed into two 6-output subsystems as shown in Fig.2(b). If there is only one EA engine, theses lookup tables (Lookup table 0-7) can be evolved sequentially. If there are multiple EA engines, they can be evolved in parallel to reduce the evolution time further.



(a) Decomposition based on inputs



(b) Decomposition based on inputs and outputs

Fig.2. Decomposition of a 12-input/12 output combinational circuit

A complex logic circuit, which is difficult to evolve, can be decomposed (in inputs and outputs) using the method discussed above until the EA is able to evolve it. In Fig.3, an example is given to show how a system is decomposed into smaller subsystem. The EA is applied until a subsystem is small enough to be evolved. If it will not become evolved within a certain number of generations, it will be decomposed into subsystems.

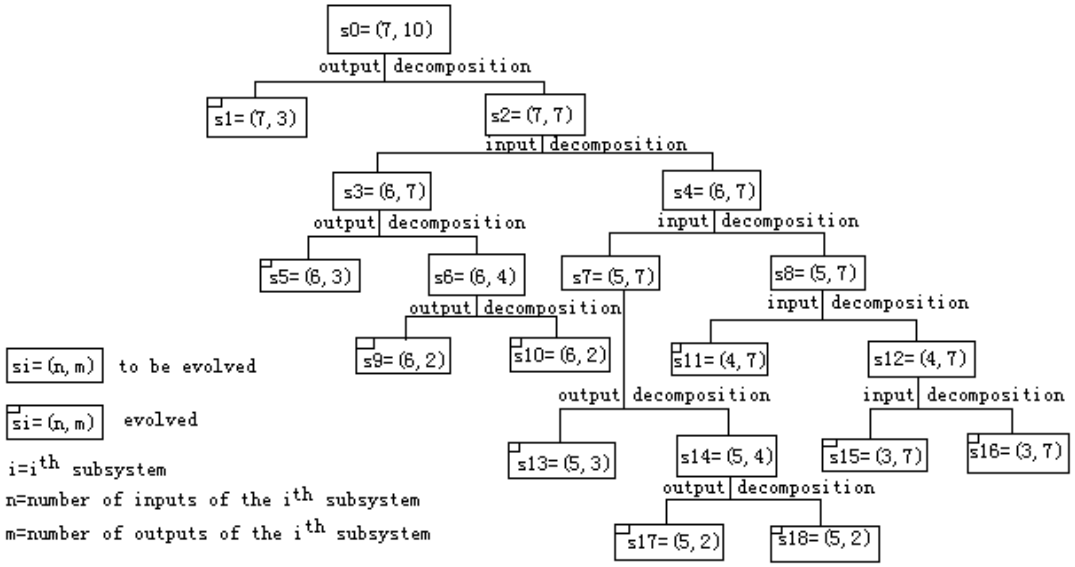


Fig.3. Example of input and output decomposition of a system

A subsystem (a multi-input multi-output lookup table, e.g. lookup table 0 in Fig.2(a) or s17=(5,2) in Fig.3) can be built with an array of 2-LUTs (2-input 1-output lookup tables) as shown in Fig.4. In Fig.4, primary inputs (Input [0:n-1]) are connected to the inputs of all Muxes in the first column. Two Muxes will select two inputs for a 2-input LUT. The outputs of LUTs in a column are connected to the inputs of all Muxes in the next column. Some of the LUTs in the final column generate primary outputs (Outputs [0:m-1]). The control bits of Muxes and contents of LUTs are chromosomes in evolution.

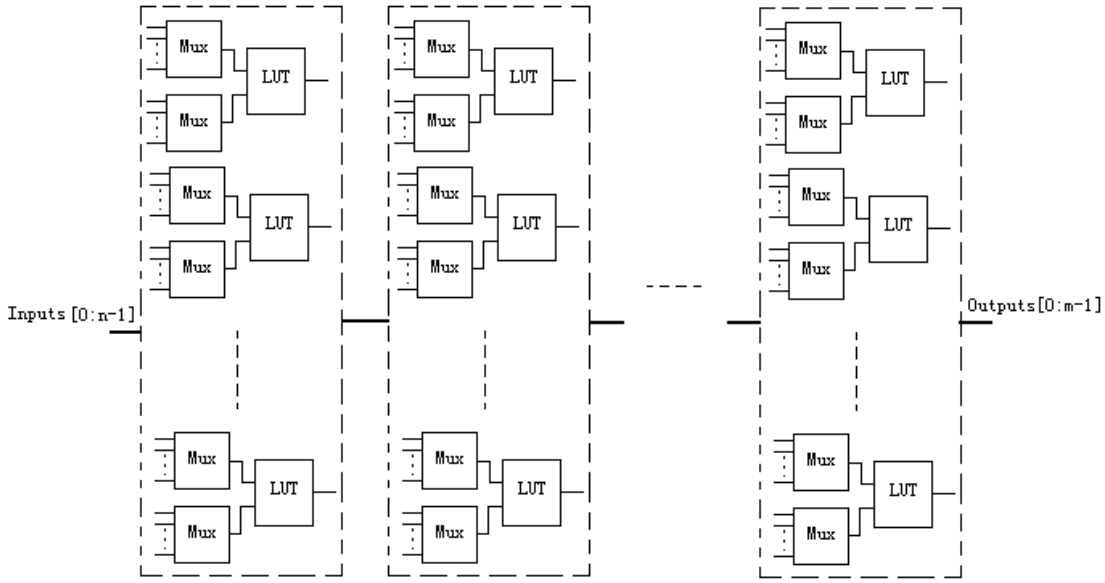


Fig.4. Array of 2-LUTs

2.2 Area and delay optimization

The fitness function evaluates the evolved circuits in terms of their functionality. In our experiment, a dynamic fitness function has been considered. It has two main criteria: first design and second, once the circuit is fully functional evolved, optimization which leads to shorter delay and reduced number of active LUTs used in the circuit configuration. The dynamic function *f* is calculated as:

$$f = \begin{cases} f1 & \text{if } f \leq 100 \text{ circuit design} \\ f1 + f2 & \text{if } f > 100 \text{ circuit optimization} \end{cases}$$

where $f1$ is a design criterion that defines the percentage of correct bits in the evolved circuit, $f2$ is the optimization criterion for the optimization stage.

The fitness function for the functionality of the evolved circuit $f1$, or so-called design criterion is calculated as follows:

$$f1 = 100 - \frac{100}{m2^n} \sum_{fc=1}^{2^n} \sum_{i=0}^{m-1} | y_i - d_i | \quad (1)$$

where m and n are the number of outputs and inputs of the given logic circuit respectively, 2^n is the number of input-output combinations, y_i is the ith digit of the output combination produced by the evaluation of the circuit, d_i is the ith digit of the desired output combination for the fitness case fc, and $| y_i - d_i |$ is the absolute difference between the actual and the required outputs. The fitness function for the optimization stage is calculated as follows:

$$f2 = W_{area}(N_{LUT} - N_{ALUT}) + W_{delay}(N_{LUT} - N_{MDPALUT}) \quad (2)$$

where N_{LUT} is the total number of LUTs present in the chromosome, N_{ALUT} is the number of active LUTs, $N_{MDPALUT}$ is the number of active LUTs in the maximum delay path, W_{area} and W_{delay} are the weights of area optimization and delay optimization. $W_{area} + W_{delay} = 1$.

The method to decide if a LUT is active is as follows. A 2-input LUT is a 4-bit memory addressed by its inputs. This LUT can be represented as Y=memory [A, B], where A and B are inputs, and Y is the content of the location addressed by the inputs A and B. In other words, assuming that the 4-bit LUT contents are represented by *abcd*, when inputs AB are 00, 01, 10, and 11, Y is equal to *a*, *b*, *c*, and *d* respectively. When the LUT contents are 0000 or 1111, Y=0 or Y=1. This LUT is inactive because a LUT generating a constant needs not be used in the actual construction of a logic circuit. When the LUT contents are 0011 or 0101, Y=A or Y=B. This LUT acts as a wire and needs not be used in the actual circuit construction, so it is also inactive. In addition, some redundant LUTs may exist in the fully functional evolved circuit. Each LUT is eliminated to see if this circuit is still fully functional. If so, this redundant LUT is also regarded as inactive. Hence a 2-input LUT is regarded as active if its contents are not 0000, 1111, 0011 or 0101, and it is not a redundant LUT.

The delay of each LUT is a constant. In delay optimization, only active LUTs are counted in delay calculation because the LUTs acting as wires and constants and redundant LUTs need not be used in the actual construction of a logic circuit. The interconnect delay (for wires) can be optimized in actual circuit layout, and hence is not considered in circuit evolution. The number of active LUTs in the maximum delay path decides the delay of the evolved circuit. All outputs of the evolved circuit are backtracked to inputs (via the control bits of Muxes) to find the maximum delay path which has the largest number of active LUTs.

2.3 Evolution of synchronous finite state machines

Sequential circuits, or simply finite state machines have two main characteristics: (*i*) there is at least one feedback path from the system output signal to the system input signal; and (*ii*) there is a memory capability that allows the system to determine current and future output signal values based on the previous input and output signal values. A finite synchronous state machine is shown in Fig.5(a), wherein the feedback signals constitute the machine state, the control logic is a combinational circuit that computes the state machine output signals (also called primary output signals) from the state signals (also called current state) and the input signals (also called

primary input signals). It also produces the signals of new machine state (also called next state). The combinational circuit of the control logic can be evolved (using the method discussed above) as shown in Fig.5(b). Various delay and area optimizations can be achieved via adjusting the values of W_{area} and W_{delay} in equation (2). In Fig.5(b), primary inputs and assigned current states are the inputs of the evolved circuit, and primary outputs and assigned next states are the outputs of the evolved circuit. The complexity of this control logic will vary for different assignments of current states and next states. In this paper, we focus on evolving control logic for a given state transition. We assume that the best state assignment has been found. It is clear that finding the best state assignment is a NP-complete problem. It can be solved using a genetic algorithm [25].

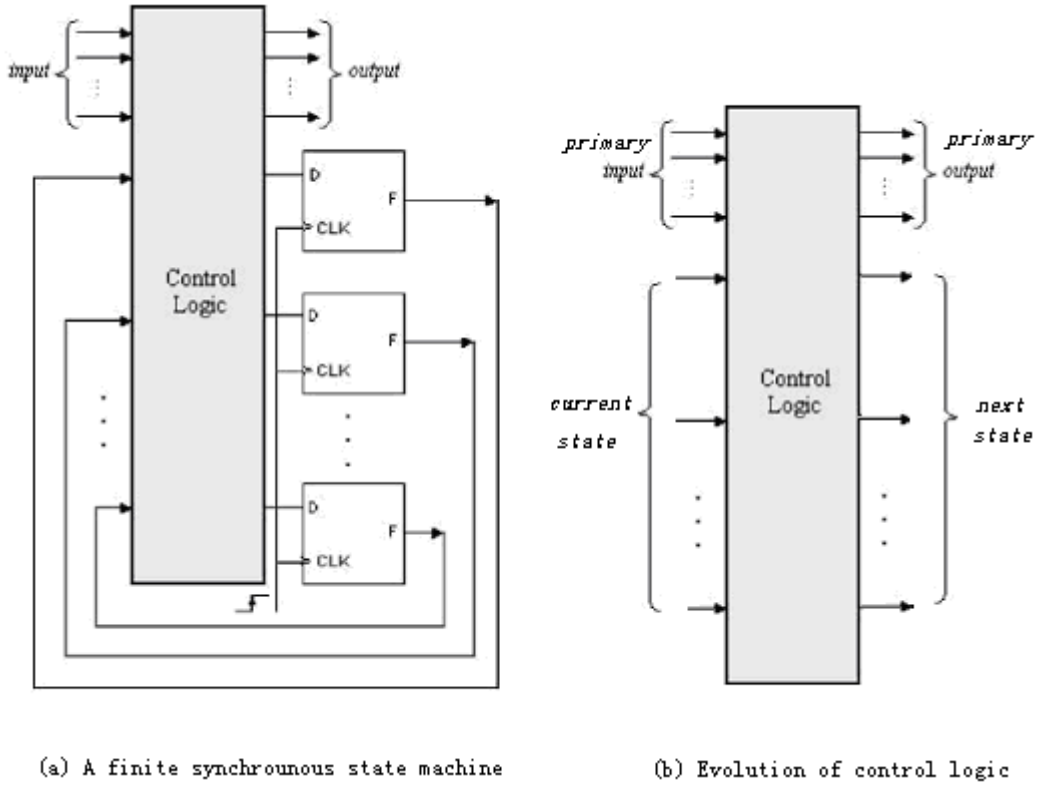


Fig.5. A finite synchronous state machine and evolution of control logic

2.4 Proposed architecture

As Fig.6 shows, the proposed evolution architecture comprises an EA (Evolution Algorithm) engine, a LUT array, a netlist generator, a combiner, a SDRAM and a controller. The controller plays the role of master and controls the entire system. In the EA engine, the population generator generates populations of chromosomes. The initial population is generated randomly. These chromosomes are performed by genetic operators selection, crossover and mutation to generate offspring. These offspring chromosomes are sent to a LUT array (in Fig.6) for computation. The computation results are sent to the EA engine to perform fitness evaluation. The fittest individuals are put into the population generator for the next evolution. The evolution proceeds until the fitness value of the chromosome meets expectation or the number of generations has reached the maximum value set by the user for that particular experiment. If the fitness value of the chromosome meets expectation, the netlist generator (in Fig.6) generates the circuit netlist according to this

chromosome and stores it in a SDRAM. If the circuit cannot become evolved within the maximum number of generations set by the user, the controller (in Fig.6) decomposes the evolved circuit into sub-circuits that are small enough to be evolved using the decomposition method discussed above. When the netlist of all sub-circuits are generated and stored in the SDRAM, the combiner (in Fig.6) combines them together via Muxes (if the evolved circuit is a sequential circuit, the combiner will also add D flip-flops as shown in Fig.5(a).), and writes the reassembled netlist back to the SDRAM. The complete circuit netlist is stored in the SDRAM now. A random number generator is required for population generation, crossover and mutation in the EA engine. The proposed architecture uses multiple Cellular Automata (CA) [26] to generate random numbers. To extract a random number from a bank of CAs, one composes the number by taking a single bit from each CA.

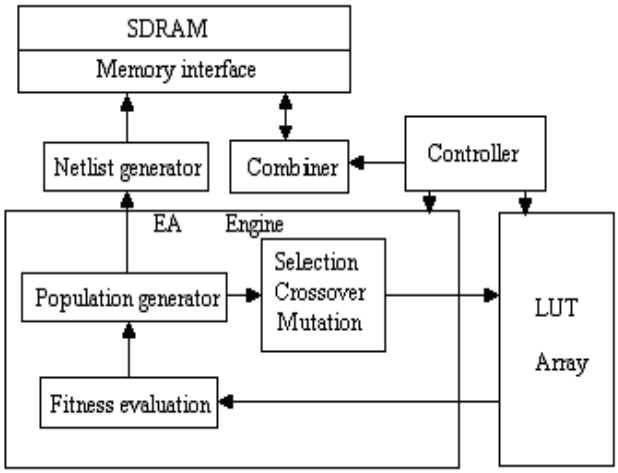


Fig.6. Proposed evolution platform

According to the chromosome (representing the circuit structure in the LUT array), the netlist representing connections between active LUTs, circuit inputs, and circuit outputs is acquired via the control bits of Muxes associated with LUTs. When a circuit or a sub-circuit cannot become evolved within the maximum number of generations set by the user, the controller decomposes it based on inputs and outputs. Supposing that a circuit or a sub-circuit to be decomposed has n inputs and m outputs, if $n \geq m$, this circuit is decomposed into smaller sub-circuits with n-1 inputs and m outputs, else if $n < m$, this circuit is decomposed into smaller sub-circuits with n inputs and $\left\lceil \frac{m}{2} \right\rceil$ outputs.

The proposed architecture (except the SDRAM) is fabricated in an ASIC. The ASIC has a total area of 2.5 mm^2 and its maximum operational frequency is 250 MHZ. The target technology is 0.12 μm CMOS (the HCMOS9 process from STMicroelectronics). In our experiments, the proposed architecture uses a 6×8 2-LUT array and a 128MB SDRAM for evolution.

3 Experiments

The aim of the experiments is to prove that the proposed method requires less generations in comparison with the following evolution computation methods: direct evolution (which does not use any decomposition technique), the decomposition scheme based on outputs [17], the decomposition scheme based on an AND-OR PLA [18], and the decomposition scheme based on a 4-LUT (4-input lookup table) array (that is the variation of the proposed scheme in which the 2-LUT array is replaced by a 4-LUT array). The aim of the experiments is also to illustrate the performance of the proposed scheme during area and delay optimization processes.

The well-known evolution algorithm $(1+\lambda)$ ES [27] is applied to all evolution schemes, where λ

represents the population size. First, all the chromosomes are randomly initialized. Second, the fitness value of each individual is computed. Third, the fittest individual is selected. Fourth, the previously selected individual is used to test if the conditions to stop the process have been met. These conditions are: the fitness value of the chromosome is 100% or the number of generations has reached the maximum value set by the user. If the conditions are not met, a new population will be generated by crossovering and mutating the best 2 chromosomes (selected at the third step) λ times in order to obtain other 2λ individuals. In the next cycle, all newly created chromosomes are evaluated, the fitness value of each of them is compared with the fitness value of the best chromosome of the previous generation, and the best $(1+\lambda)$ individuals are selected. In our experiments, the population size is 5, crossover rate is 1, mutation rate is 0.05, the maximum number of generations is 1,000,000, and the termination criterion for all decomposition schemes is 2000 generations without any improvement in the fitness function.

3.1 Experimental results: fully functional evolution

The experimental results obtained by using the proposed scheme, direct evolution, the decomposition scheme based on outputs [17], the decomposition scheme based on an AND-OR PLA [18], and the decomposition scheme based on a 4-LUT array are shown in Table 1. In Table 1, all the characteristics of the circuit are given. For example, by looking at the multiplier circuit Mult3, it has 6 inputs and 6 outputs. Then the average number of generations required to evolve this circuit is reported. The last column gives the average time spent for each experiment. The evolved circuits are (3-5 bit) multipliers and benchmark circuits from the MCNC benchmark library [28].

Table 1. Experimental results from the proposed scheme, direct evolution, the decomposition scheme based on outputs, the decomposition scheme based on an AND-OR PLA, and the decomposition scheme based on a 4-LUT array, where In and Out are the number of inputs and outputs in the given logic function. "Not evolved" indicates the case when it does not find a solution within the maximum generation.

Hardware	In	Out	Evolution method	Average generations	Total time spent for each experiment in seconds
Multiplier circuits					
Mult3	6	6	Direct evolution	507899	6765
			Decomposition in outputs	21948	288
			Decomposition in a 4-LUT array	13774	180
			Decomposition in an AND-OR PLA	9156	123
			Proposed scheme	6875	92
Mult4	8	8	Direct evolution	Not evolved	
			Decomposition in outputs	146663	1718
			Decomposition in a 4-LUT array	132118	1471
			Decomposition in an AND-OR PLA	87411	1040
			Proposed scheme	65658	790
Mult5	10	10	Direct evolution	Not evolved	
			Decomposition in outputs	740164	16033
			Decomposition in a 4-LUT array	607616	13161
			Decomposition in an AND-OR PLA	506347	10968
			Proposed scheme	381978	8278

Logic circuits taken from MCNC benchmark library					
Add2_7	7	4	Direct evolution	554598	5405
			Decomposition in outputs	28121	269
			Decomposition in a 4-LUT array	17587	130
			Decomposition in an AND-OR PLA	11665	90
			Proposed scheme	8836	72
Addm4	9	8	Direct evolution	Not evolved	
			Decomposition in outputs	168053	10713
			Decomposition in a 4-LUT array	157896	5952
			Decomposition in an AND-OR PLA	132414	4908
			Proposed scheme	103582	3839
Rd84	8	4	Direct evolution	989758	8928
			Decomposition in outputs	87752	781
			Decomposition in a 4-LUT array	57899	375
			Decomposition in an AND-OR PLA	38533	250
			Proposed scheme	29199	196
Cm162	14	5	Direct evolution	Not evolved	
			Decomposition in outputs	Not evolved	
			Decomposition in a 4-LUT array	205618	18639
			Decomposition in an AND-OR PLA	136412	12167
			Proposed scheme	107837	9718
Add6	12	7	Direct evolution	Not evolved	
			Decomposition in outputs	Not evolved	
			Decomposition in a 4-LUT array	528656	32585
			Decomposition in an AND-OR PLA	352361	21652
			Proposed scheme	265789	17332
5×pl	7	10	Direct evolution	808454	35788
			Decomposition in outputs	43643	1878
			Decomposition in a 4-LUT array	36880	1365
			Decomposition in an AND-OR PLA	24560	884
			Proposed scheme	18487	670

Based on the results found, one may conclude that the main advantages of using the proposed scheme are:

- a smaller amount of generations are required during evolution
- it solves the tasks quicker than other evolution schemes.

The reason why the proposed scheme requires fewer generations than the decomposition scheme based on an AND-OR PLA may be that the AND-OR PLA have only AND gates and OR gates for evolution, but a 2-LUT (in the proposed scheme) provides 16 Boolean functions for evolution. The reason why the proposed scheme requires fewer generations than the decomposition scheme based on a 4-LUT array may be that a 4-LUT provides 2^{16} Boolean functions for evolution and a 2-LUT provides 16 Boolean functions for evolution, which means the search space of a 4-LUT array is much larger than a 2-LUT array. The reason why the proposed scheme requires fewer generations than the decomposition scheme based on outputs may be that the proposed scheme can decompose circuits based on both inputs and outputs.

3.2 Experimental results: optimization evolution

Section 3.1 gives the experimental results at the design stage (for fully functional evolution, whose fitness function is equation (1)). This section will give experimental results at the area and delay optimization stage (whose fitness function is equation (2)).

In this section, those combinational circuits (in Table 1) are further optimized in area. Table 2 gives the experimental results at the design stage (for fully function evolution) and the area optimization stage in evolution. Only active LUTs are counted in area optimization because inactive LUTs (redundant LUTs and the LUTs acting as constants or wires) are not used in the actual construction of a logic circuit. As Table 2 shows, the proposed scheme reduces active LUTs by 35-45% at the optimization stage.

Table 2. Area optimization

Hardware	Number of active LUTs (design stage)	Number of active LUTs (optimization stage)	Reduced active LUTs (percentage)
Mult3	45	25	44%
Mult4	180	115	36%
Mult5	684	412	40%
Add2_7	135	88	35%
Addm4	740	429	42%
Rd84	548	339	38%
Cm162	8486	4667	45%
Add6	4193	2516	40%
5×pl	270	176	35%

Three state machines *shiftreg*, *lion9* and *train11* [29] are generally used as benchmarks. The state assignments used (as shown in Table 3) are the best ones so far. These benchmark state machines are evolved at the design stage and the delay optimization stage. The delay optimization evolution can improve maximum operational frequencies of these state machines. The delay of each LUT is a constant. In delay optimization, only active LUTs are counted in delay calculation because inactive LUTs (the LUTs acting as wires and constants and redundant LUTs) need not be used in the actual construction of a logic circuit. The interconnect delay (for wires) can be optimized in actual circuit layout, and hence is not considered in circuit evolution. The number of active LUTs in the maximum delay path decides the delay of the evolved circuit. As Table 3 shows, the proposed scheme reduces active LUTs in the maximum delay path by 43-67% at the optimization stage.

Table 3. Delay optimization, where In and Out stand for the number of primary inputs and the number of primary outputs respectively.

State machine	In	Out	State assignments	Active LUTs in maximum delay path (design)	Active LUTs in maximum delay path (optimization)	Reduced active LUTs in maximum delay path (percentage)
Shiftreg	1	1	[4,0,3,7,5,1,2,6]	6	2	67%
Lion9	2	1	[10,8,12,9,13,15,7,3,11]	8	4	50%
Train11	2	1	[2,6,1,4,0,14,10,9,8,11,3]	7	4	43%

4 Conclusion

In this paper, the evolvable hardware based on a 2-LUT array, which can automatically decompose a large combinational or sequential circuit into smaller sub-circuits and combine them together to generate a complete circuit netlist, has been presented and compared with other EHW techniques.

The proposed scheme has been tested on the evolution of multipliers and logic circuits taken from the MCNC benchmark library. The experimental results have confirmed the proposed scheme requires fewer generations to evolve fully functional solutions, reduces the time for an experiment, and allows the evolution of large circuits. The proposed scheme has also been tested in area and delay optimization evolution. The experimental results have confirmed that the proposed scheme can achieve the area and delay optimization of large combinational circuits and sequential circuits.

The proposed evolvable hardware automatically generates a complete circuit netlist in a SDRAM. Because of the low cost and large data storage of a SDRAM, the evolvable hardware provides a good platform to evolve large circuits.

References:

[1] X. Yao, T. Higuchi. "Promises and challenges of evolvable hardware", *IEEE Trans. Systems, Man and Cybernetics, Part C*, vol.29, Feb. 1999, pp. 87 – 97.

[2] Jinchuang Zhao, Xuekun Song, Wenli Fu, Jingjie Lei, "Preliminary Research of Evolvable Sensor", *Third International Conference on Natural Computation*, Vol.5, 24-27 Aug. 2007, pp.195 – 198.

[3] J.D. Lohn, G.S. Hornby, "Evolvable hardware: using evolutionary computation to design and optimize hardware systems", *IEEE Computational Intelligence Magazine*, Vol.1, Issue 1, Feb. 2006, pp.19 - 27.

[4] Oliveira, Tiago Carvalho Junior, Valfredo Pilla, "An Implementation of Compact Genetic Algorithm on FPGA for Extrinsic Evolvable Hardware", *4th Southern Conference on Programmable Logic*, 26-28 March 2008, pp.187 – 190.

[5] E. Benkhelifa, A. Pipe, G. Dragffy, M. Nibouche, "Towards evolving fault tolerant biologically inspired hardware using evolutionary algorithms", *IEEE Congress on Evolutionary Computation*, 25-28 Sept. 2007, pp.1548 – 1554.

[6] Wei-Rong Guan, Hai-Yun Zhou, Bin Song, "The Optimization Speed of Eitist Evolutionary Algorithms in Off-Line EHW", *International Conference on Machine Learning and Cybernetics*, Aug. 2006, pp.2154 – 2158.

[7] Y. Jewajinda, P. Chongstitvatana, "A Cooperative Approach to Compact Genetic Algorithm for Evolvable Hardware", *IEEE Congress on Evolutionary Computation*, 16-21 July 2006, pp.2779 – 2786.

[8] E. Stomeo, T. Kalganova, C. Lambert, "A Novel Genetic Algorithm for Evolvable Hardware", *IEEE Congress on Evolutionary Computation*, 16-21 July 2006, pp.134 – 141.

[9] J. Lee, J. Sitte, "Gate-level Morphogenetic Evolvable Hardware for Scalability and Adaptation on FPGAs", *First NASA/ESA Conference on Adaptive Hardware and Systems*, 15-18 June 2006, pp.145 – 152.

[10] L. Sekanina, "Evolutionary Design of Digital Circuits: Where Are Current Limits", *First NASA/ESA Conference on Adaptive Hardware and Systems*, 15-18 June 2006, pp.171-178.

[11] L. Sekanina, T. Martinek, Z. Gajda, "Extrinsic and Intrinsic Evolution of Multifunctional Combinational Modules", *IEEE Congress on Evolutionary Computation*, 16-21 July 2006, pp.2771-2778.

[12] D. Dhanasekaran, K. Boopathy Bagan, "Fault Tolerant Dynamic Antenna Array in Smart Antenna System Using Evolved Virtual Reconfigurable Circuit", *21st International Conference on VLSI Design*, 4-8 Jan. 2008, pp.77 – 83.

[13] B. Karunya, R. Uma, "Functional Level Implementation Of Evolvable Hardware Using Genetic Algorithms", *International Conference on Mixed Design of Integrated Circuits and System*, 22-24 June 2006, pp.671-674.

[14] K. Glette, J. Torresen, M. Yasunaga, Y.

Yamaguchi, "On-Chip Evolution Using a Soft Processor Core Applied to Image Recognition", *First NASA/ESA Conference on Adaptive Hardware and Systems*, 15-18 June 2006, pp.373 – 380.
[15] Z. Vasicek, L. Sekanina, "An Area-Efficient Alternative to Adaptive Median Filtering in FPGAs", *International Conference on Field Programmable Logic and Applications*, 27-29 Aug. 2007, pp.216 – 221.
[16] E. Stomeo, T. Kalganova, C. Lambert, "Generalized Disjunction Decomposition for Evolvable hardware", *IEEE Trans. Systems, Man and Cybernetics*, Vol.36, No.5, Oct. 2006, pp.1024-1043.
[17] Jin-Hyuk Hong, Sung-Bae Cho, "MEH: modular evolvable hardware for designing complex circuits", *IEEE Congress on Evolutionary Computation*, Vol.1, 8-12 Dec. 2003, pp.92 – 99.
[18] E. Stomeo, T. Kalganova, C. Lambert, N. Lipnitsakya, Y. Yatskevich, "On evolution of relatively large combinational logic circuits", *NASA/DoD Conference on Evolvable Hardware*, 29 June-1 July 2005, pp.59 – 66.
[19] R.S. Oreifej, R.N. Al-Haddad, H. Tan, R.F. DeMara, "Layered Approach to Intrinsic Evolvable Hardware using Direct Bitstream Manipulation of Virtex II Pro Devices", *International Conference on Field Programmable Logic and Applications*, 27-29 Aug. 2007 pp.299 – 304.
[20] L. Sekanina, S. Friedl, "On routine implementation of virtual evolvable devices using COMBO6", *NASA/DoD Conference on Evolvable Hardware*, 24-26 June 2004, pp. 63 – 70.
[21] L. Sekanina, "Towards evolvable IP cores for FPGAs", *NASA/DoD Conference on Evolvable Hardware*, 9-11 July 2003, pp.145 – 154.
[22] A.J. Greensted, A.M. Tyrrell, "RISA: A Hardware Platform for Evolutionary Design", *IEEE Workshop on Evolvable and Adaptive Hardware*, 1-5 April 2007, pp.1 – 7.
[23] Two flows for partial reconfiguration: Module Based or Difference Based – *Xapp 290*, *Xilinx*, 2004.
[24] XC6200 Field Programmable Gate Arrays – *Data Sheet*, *Xilinx*, 1997.
[25] N. Nedjah, L.M. Mourelle, "Evolutionary state assignment for synchronous finite state machine", *Proc. of International Conference on Computation Science, LNCS,* Springer-Verlag, 2004.
[26] S. Bonissone, R. Subbu, "Evolutionary Multiobjective Optimization on a Chip", *IEEE Workshop on Evolvable and Adaptive Hardware*, 1-5 April 2007, pp.61 – 66.
[27] T. Kalaganova, J. Miller, "Evolving more efficient digital circuits by allowing circuit layout and multi-objective fitness", *1st NASA/DoD Workshop on Evolvable Hardware*, July 1999, pp.54-63.
[28] S. Yang, "Logic Synthesis and Optimization Benchmark User Guide Version 3.0", *MCNC*, 1991.
[29] Collaborative Benchmarking Laboratory, North Carolina State University, *www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth89/fsmexamples/*, Nov. 2003.