

# Chapter 2

## A Hardware Implementation of Evolvable Embedded System for Combinational Logic Circuits Using Virtex 6 FPGA

C. Ranjith and S.P. Joy Vasantha Rani

### 2.1 Introduction

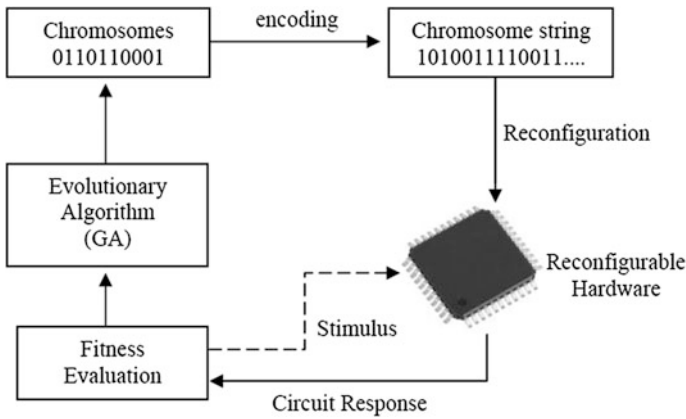
The traditional method of designing digital systems was in the form of schematic or programming by hardware description languages (HDL). The above two methods require an in-depth analysis of the system under design. The manual design through schematic or programming can be sometimes complex and may lead to monotonous solutions. Another, novel method of circuit design is by evolvable hardware, wherein the circuits is evolved automatically based on some optimisation algorithm. The mode of digital design is now getting due importance from the research community and the industry. Research in this field received a boom with major chip industries innovating novel tools and chips for the application and testing of “evolvable” circuits.

Evolvable hardware (EHW) was a term coined by Hugo De Garis in the year 1992 for circuits which could configure its own hardware structure dynamically depending on the changes in the environment or on design parameters [1]. This capability of configuring the hardware was achieved by employing efficient search algorithms, like genetic algorithms (GA) [2, 3]. Evolvable systems for digital domain are mainly designed using field programmable gate array (FPGA) chips. FPGA-based EHW can be classified based on the evaluation of the solutions. The first method is recognized as *extrinsic evolution*, where the development of circuits uses a simulation approach of determining the best evaluation; such solutions are then implemented in the device. The second approach is *intrinsic evolution*, where

---

C. Ranjith (✉) · S.P. Joy Vasantha Rani  
Electronics Engineering Department, M.I.T. Campus,  
Anna University, Chennai, Tamil Nadu, India  
e-mail: ranjith.kmct@gmail.com

S.P. Joy Vasantha Rani  
e-mail: joy\_mit@annauniv.edu



**Fig. 2.1** Basic structure of EHW

each candidate solution is physically tested on the hardware device. The latter offers better accuracy and operation of self-evolved circuits [4]. The basic concept of the combination of GAs with FPGA in EHW is with regard that the configuration bit strings to the FPGA is chromosomes of the GA. The fitness function is designed, such that the GA can autonomously find the best hardware for the design to be implemented in the FPGA. Figure 2.1 depicts a simple example of the EHW concept [5]. In conventional works on EHW, the GA operations were conducted in computer or workstations, which would make the system robust and slow. In recent works, the GA operation is performed on the same FPGA chip either by hardware or by software in a dedicated core.

This paper describes the implementation of the evolvable embedded system which uses the evolutionary algorithm to dynamically modify some of the system components or parameters in order to adapt to the changing environment [2]. The use of a soft processor (MicroBlaze) for computing the optimal search algorithm (GA) and hardware architecture for evolution are integrated in a single FPGA, therefore termed *evolvable embedded system*. An evolvable embedded system architecture of combinational circuits using the intrinsic mode of evolution is discussed. The hardware was changed based on the fitness evaluation of the GA. The evolvable embedded hardware architecture was implemented on Virtex 6 (XC6VLX240T-1FFG1156) ML605 Evaluation Kit. Combinational circuits with 8 inputs and outputs could be evolved in this architecture with GA results displayed onto a PC. A 2-bit adder and multiplier circuits were evolved as an example using this architecture.

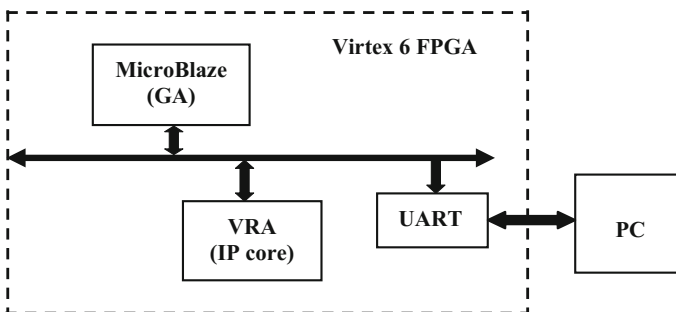
The paper is organized as follows: Section 2.2 describes the structure of EHW system, with Sect. 2.3 giving a complete view of the modeled system and its design. Section 2.4 describes the specifications and implementation process. Section 2.5 analyses the results, and final conclusion of the paper is discussed in brief.

## 2.2 Structure of Evolvable Embedded System

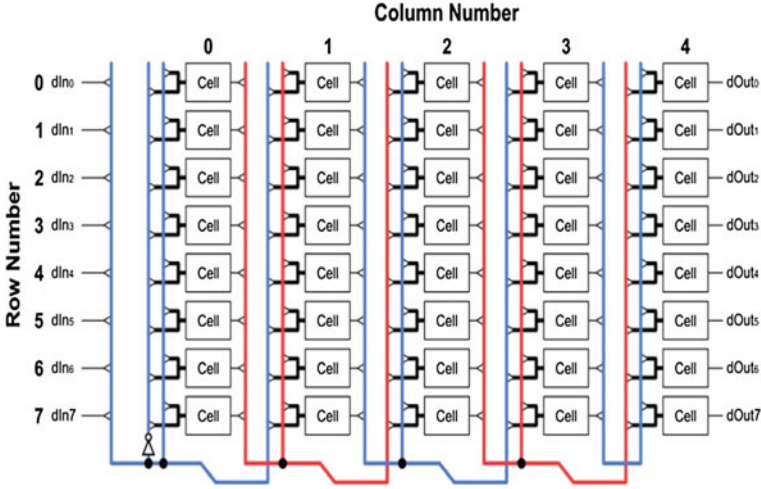
The architecture of the EHW uses the concept of the virtual reconfigurable architecture (VRA) [6]. A hardware description of the architecture was layered over the reconfigurable chip, to implement the evolutionary structure. The GA program was fused in the MicroBlaze soft processor where the computed fitness measure was displayed on the PC through a configured UART peripheral. The complete structure is as shown in Fig. 2.2.

### 2.2.1 Study of VRA

VRA is modeled in HDL and is taken as a second reconfigurable layer on the FPGA. The primary advantage of this concept is to provide a simple and efficient mode of intrinsic evolution [6]. Figure 2.3 shows the VRA structure implemented for the design of evolvable combinational circuits. An array of HDL-defined, configurable cells are arranged in rows and columns, where each cell's input is connected to the outputs of the two previous columns with the exception of the first-array column which is connected to the inputs and its invert as shown in Fig. 2.3. The number of rows and columns selected depends on the complexity of the problem. Configuration bits from the GA provide the connectivity and logic based on the 16-bit input combinations to the cell. The outputs are checked for problem logic (truth table conditions) to determine the fitness criteria. The VRA approach is widely employed in the implementation of EHW systems as the configuration bit formats of the FPGA are company proprietary. The VRA concept is similar to the Cartesian genetic programming (CGP) and provides other benefits, including (1) array of configuration cells directly connected to the hardware of the GA in the same FPGA, making the communication faster; (2) the VRA is modeled as an HDL source code which makes it easier to modify and synthesize in other



**Fig. 2.2** Structure of evolvable embedded system

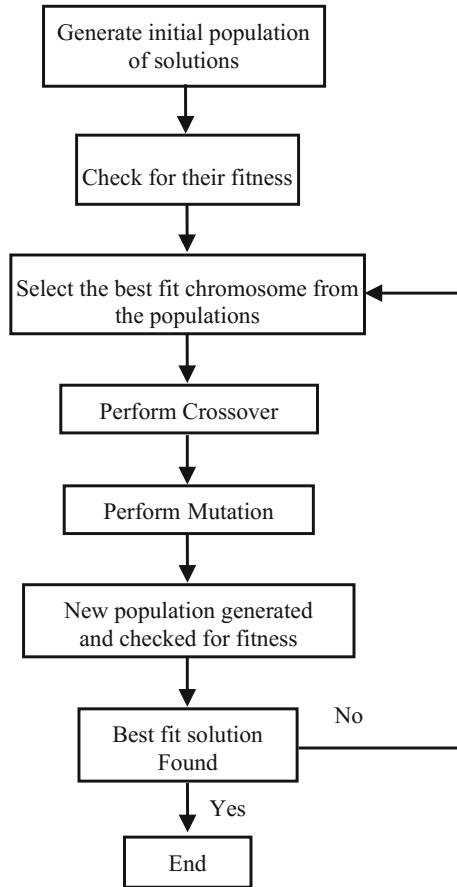


**Fig. 2.3** An  $8 \times 5$  VRA structure

target platforms; and (3) The VRA architecture modeled can be utilized for similar set of problem definitions.

### 2.2.2 Genetic Algorithm Flow

In this paper, we make use of a simple GA to search for the best solution [7]. The algorithm runs on the principle of population size of individuals (candidate) solutions to the optimization problem. These individuals consist of a string (chromosomes) of genes (genotype). The genotype is encoded to produce the configuration set (phenotype). The operations of GA include selection, reproduction, crossover, and mutation on the individuals to get a better solution [3]. The initial step is to prepare a chromosome format for the given problem, the length of which corresponds to the total number of decision variables in the search. Each of these chromosomes is passed on their fitness probability and the best fit chromosomes are selected. The best chromosome strings from the pool of populations undergoes genetic operations like crossover, mutation, and selection. New populations are generated at each iteration and checked for their fitness to produce new solutions. The cycle repeats till an optimized and best solution is found. The complete flow diagram of GA process is shown in Fig. 2.4.



**Fig. 2.4** Flow graph of simple GA structure

### 2.2.3 Reconfigurable FPGA Chip

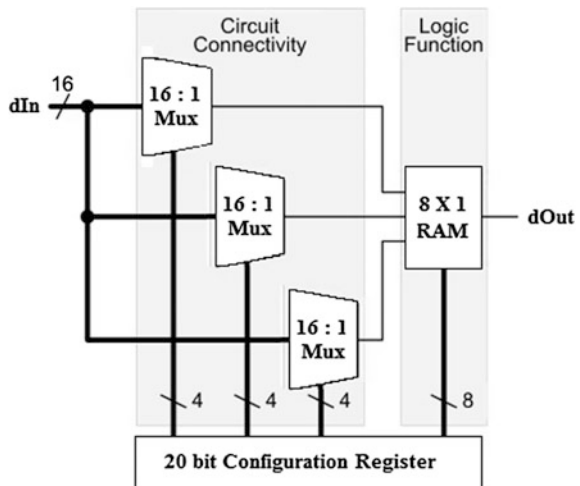
The evolvable embedded architecture is implemented on a ML605 Evaluation board with Virtex 6 FPGA. The Virtex-6 family is built on a 40-nm process for high computational electronic systems. This series of FPGAs have integrated features that include DSP blocks, PCI-Express controllers, Ethernet MAC blocks, and high-speed transceivers [8]. The use of Virtex 6 FPGA was mandatory due to simple and flexible implementation of a 32-bit soft core MicroBlaze processor [9]. This processor computes the GA, therefore faster evaluation time can be achieved. The main advantage is that the complete EHW process can be implemented on a single chip, thereby making area efficient and flexible.

### 2.3 Evolvable Embedded System Design

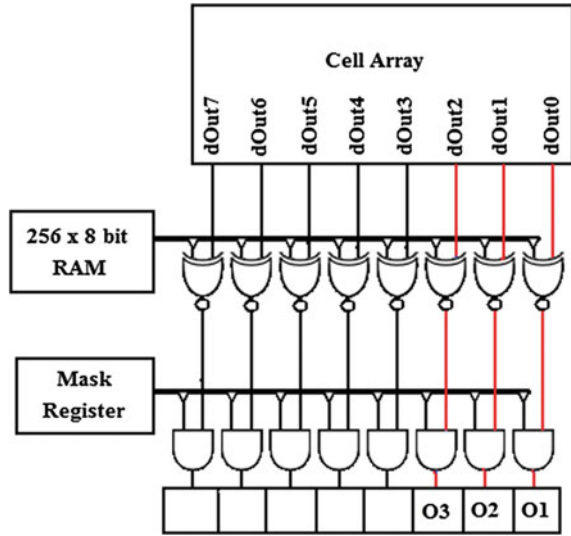
This section gives an overall view of the complete evolvable embedded system design. The VRA architecture of combinational circuits is modeled and coded by a hardware description language. The architecture of the VRA is modeled as shown in Fig. 2.5. The heart of the architecture unit is named as configuration *cell*, consisting of three 16:1 multiplexers and an  $8 \times 1$  bit RAM. This unit offers logical function and interconnectivity for the system under design. Three multiplexers are used to select the inputs to the lookup table (LUT), which are driven from one of the set of sixteen inputs. The configuration cell is driven by 20 configuration bits ( $3 \times 4$  select lines of MUX + 8 selectable lines to RAM). The configuration cells are interconnected in '*m*' rows and '*n*' columns to form a *Cell Array*. A simple interconnection principle is followed where each cell input is connected to the output of the two columns, except for the first cell array column, which are connected by the inputs and its invert. This interconnection principle simplifies the cell-to-cell routing. For the design, an  $8 \times 5$  cell array was selected, with a prospect of evolving simple combinational circuits with a maximum of 8 inputs and outputs. Here, a 400 configuration bits ( $40 \text{ cells} \times 20 \text{ configuration bits}$ ) were required to perform the complete logical functions and interconnection between cells. The configuration bits are provided by the GA taking the fitness function criteria. A reconfigurable hardware is tested using a hardware setup of simple logical gates to check the functionality of the design.

The major advantage of performing fitness evaluations in hardware was that multiple evaluations in parallel could be achieved easily. In this design, we have combined 8 cell arrays into a single block. The block contains extra circuitry to help determine the fitness of an individual. Figure 2.6 shows one such cell array with the additional circuits to perform a fitness evaluation. The RAM is used to store the

**Fig. 2.5** Architecture of the cell



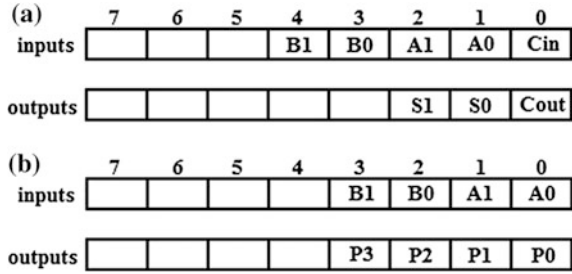
**Fig. 2.6** Single cell array with fitness check connections (showing the outputs for a 2-bit adder circuit)



truth table of the target circuit (in this case, a 2-bit adder/multiplier). The output bits of the cell array after configuration bit selection are compared with the correct values of the 8-bit RAM outputs, using XNOR gate (last 3 bits are the truth table entries of Cout, S0, S1 for a 2-bit adder), and final values are masked by keeping one input of AND gate to high depending on the number of outputs of the evolved circuit. Match outputs are obtained (O1, O2, O3) as shown for a 2-bit adder circuit. Here, O1, O2, O3 symbolizes Cout, S0, S1 of a two-input adder. The same could be employed for a 2-bit multiplier where 4 outputs are valid, so the last 4 AND gates input are held high.

The evolutionary system was developed using Xilinx Platform Studio (XPS) tool. An embedded system with both hardware and software elements was created in the EDK (Embedded Development Kit) and SDK (Software Development Kit), respectively; a 32-bit MicroBlaze soft core processor (for performing the GA) with a clock frequency of 50 MHz and an 8-cell array VRA core structure (for evaluating candidate solutions). The system also includes a configured serial UART (universal asynchronous receiver/transmitter) to send GA computations to the PC. The cell array VRA block was imported as an IP core (intellectual property) and integrated as a peripheral to the Microblaze processor. The IP core was plugged into the processor bus using the generated macros in situ with an interface program [9]. The system netlist (ngc files) and bitstream (bit files) are generated from the EDK. The C program for the GA optimization was hand-coded, encoding the inputs and outputs of the cell array. The input and output cell array configuration for a 2-bit adder and 2-bit multiplier is as shown Fig. 2.7a, b. The GA parameters such as crossover rate, mutation rate, and number of selections are assigned by trial and error. The evaluation is based upon minimum iterations to converge to a better fit for the optimization of the circuit. The program calls for

**Fig. 2.7** Input and output cell array configuration **a** 2-bit adder, **b** 2-bit multiplier

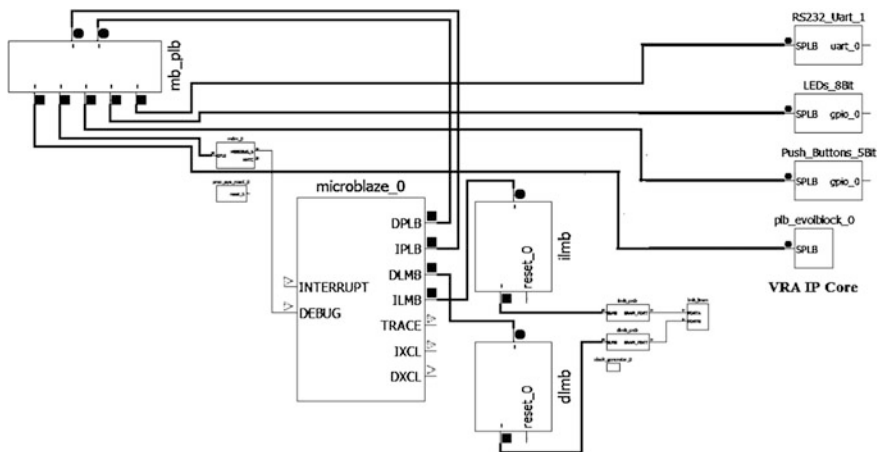


additional header files, to hold parameters required to communicate the application program with the devices to the embedded system. This application program creates an executable and linkable format file (elf file) for the program to be realized by the FPGA. This file along with the bit file is integrated to be programmed into the FPGA (ML605) through a JTAG cable.

## 2.4 Implementation

The VRA architecture was coded in VHDL and synthesized in Xilinx ISE Design Suite 14.6. An inbuilt MicroBlaze soft processor having 32 bit, Reduced Instruction Set Computer (RISC) architecture is utilized for modeling the system [9]. The MicroBlaze processor with its peripherals for the design include UART (for communication with PC), 4 seven-segment LEDs, push buttons for inputs and reset options, realized from XPS 14.6 EDK platform tool. The peripherals are linked to the processor via the processor local bus (PLB). The synthesized VRA module files are exported to realize as a user IP core. This IP core is connected to the processor via the PLB. The generated macros are modified to interconnect the user IP core to the PLB bus. The complete embedded processor structure with its peripherals realized as block schematic is presented in Fig. 2.8. The microprocessor hardware specification file (mhs file) and peripheral analysis order file (pao file) has to be updated to include the changes accommodated by the addition of new IP core, along with the creation of black box definition file (bbd file). Once the following peripherals are connected, new netlist files (*system.ngc*) and bit files (*system.bit*) are generated.

The hardware system created in EDK is exported to SDK tool to develop the GA program to operate in association with the hardware. A simple GA program with



**Fig. 2.8** MicroBlaze processor with peripheral interconnections (viewed through XPS tool)

optimization for the evolution of circuits is coded in C language. The GA program is developed with the following subroutines:

- (i) The truth table of the target circuit to be stored in the  $256 \times 8$  bit RAM. Here, 2-bit adder/multiplier truth tables are initially stored occupying the first 16 locations of the RAM, whereas the rest is default taken to be nil.
- (ii) Configure the input and output registers of the cell array with the bit placing as shown in Fig. 2.7a, b. For a 2-bit adder, the first 5 bits of the input register represent the 2-bit input, and a carry (Cin, A0, A1, B0, B1), whereas the first 3 bits of output register represent the outputs (C0, S0, S1) of the 2-bit adder circuit. Similar logic holds for a 2-bit multiplier, having of 4 inputs (A0, A1, B0, and B1) and generating 4 outputs (P0, P1, P2, and P3).
- (iii) A simple GA program to find a best fit configuration is to be applied to the cell. The fitness function is the truth table of the 2-bit adder/multiplier circuit. The fitness function can be calculated as:

$$\text{fitness} = 2^i, \quad \text{where } i = \text{inputs of the system} \quad (2.1)$$

From the criteria, the fitness of the 2-bit adder circuit is summed to 96 (sum of the fitness of C0, S0, S1), and that of 2-bit multiplier to be 64. The GA program is developed and tested for the following parameters:

No. of iterations: 500

Population size: 128 with 16 sets of population for each cell array (8 cell arrays in parallel)

Crossover rate: 55%

Mutation rate: 0.1%

Selection method: Tournament selection.

The GA program communicates with the devices of the embedded system through the header files provided by the vendor. Two header files, namely *xparameters.h* and *xgpio.h*, holding parameters for the communication are added to the application program. The *.elf* file (software) from the SDK and the *system.bit* file (hardware) from the EDK are combined to the Virtex 6 FPGA (XC6VLX240T-1FFG1156). The combinations of these two files integrate to form *download.bit*. This bit files are used to program the FPGA through the JTAG cable.

## 2.5 Results

The results are monitored on a PC connected through the UART peripheral. The results display the GA program evaluating the fitness calculations for a 2-bit adder/multiplier circuits. The program is terminated once the fitness is evaluated and displays the number of generations and configurations for each cell array as shown in Fig. 2.9a, b for 2-bit adder and multiplier, respectively. From the results, it can be analyzed that a 2-bit adder was evolved over 88 generations, and 135 generations were required to evolve a 2-bit multiplier. The number of generations differs with the number of inputs and outputs of the system. The experiments were repeated for different values of crossover rate, mutation rate, and selection criteria.

Optimized results were obtained for crossover rate = 55%, mutation rate = 0.1%, and tournament selection of 5 individuals at a time. The hardware interconnections could be analyzed from the configurations of the individual cells. The complete floor plan of the evolvable embedded system for the combinational circuit using the VRA concept is shown in Fig. 2.10 using the PlanAhead tool. From the figure, it can be seen that around 40% of the implemented chip area were occupied by the MicroBlaze processor and its peripherals and the rest, by the VRA IP core. From the total chip area, around 6% of LUTs, 4% IOBs and 3% of registers were used from the available resources as tabulated in Table 2.1. The timing criteria were met with an implementation time of 17.063 ns (maximum frequency of 58.606 MHz).

(a)

```

C/C++ - Xilinx SDK
File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help

Serial: (COM3, 9600, 8, 1, Even, None - CONNECTED) - Encoding: (UTF-8)

Gen: 73, fittest: 22 (92), Config: 5A180151, leastFit: 43 (25), average 76
Gen: 74, fittest: 24 (92), Config: 5A180151, leastFit: 114 (35), average 77
Gen: 75, fittest: 16 (92), Config: 5A180151, leastFit: 59 (35), average 76
Gen: 76, fittest: 0 (92), Config: 5A180151, leastFit: 110 (27), average 76
Gen: 77, fittest: 32 (92), Config: 5A180151, leastFit: 78 (35), average 77
Gen: 78, fittest: 0 (92), Config: 5A180151, leastFit: 103 (31), average 75
Gen: 79, fittest: 4 (92), Config: 5A180151, leastFit: 122 (28), average 75
Gen: 80, fittest: 2 (92), Config: 5A180151, leastFit: 122 (40), average 77
Gen: 81, fittest: 1 (92), Config: 5A180151, leastFit: 110 (28), average 78
Gen: 82, fittest: 3 (92), Config: 5A180151, leastFit: 51 (34), average 76
Gen: 83, fittest: 2 (92), Config: 5A180153, leastFit: 31 (31), average 77
Gen: 84, fittest: 4 (92), Config: 5A180173, leastFit: 101 (34), average 76
Gen: 85, fittest: 0 (92), Config: 5A182111, leastFit: 3 (24), average 75
Gen: 86, fittest: 4 (92), Config: 5A180111, leastFit: 37 (38), average 75
Gen: 87, fittest: 3 (92), Config: 5A180153, leastFit: 1 (41), average 77
Gen: 88, fittest: 80 (96), Config: 5A18015B, leastFit: 51 (35), average 75
Solution Found, Gen: 88, fittest: 80 (96)

```

(b)

```

Gen: 121, fittest: 0 (63), Config: 1FA90351, leastFit: 1 (34), average 56
Gen: 122, fittest: 4 (63), Config: 1FAD0351, leastFit: 62 (36), average 56
Gen: 123, fittest: 0 (63), Config: 1FAF0351, leastFit: 126 (30), average 56
Gen: 124, fittest: 2 (63), Config: 1FAD0351, leastFit: 85 (30), average 57
Gen: 125, fittest: 0 (63), Config: 1FAD0351, leastFit: 79 (37), average 57
Gen: 126, fittest: 0 (63), Config: 1FAD0351, leastFit: 109 (36), average 57
Gen: 127, fittest: 1 (63), Config: 1FAD0351, leastFit: 21 (38), average 57
Gen: 128, fittest: 1 (63), Config: 1FAD0351, leastFit: 114 (38), average 57
Gen: 129, fittest: 1 (63), Config: 1FAD0751, leastFit: 105 (38), average 57
Gen: 130, fittest: 9 (63), Config: 1FAD0351, leastFit: 54 (35), average 56
Gen: 131, fittest: 0 (63), Config: 1FAD0351, leastFit: 35 (40), average 57
Gen: 132, fittest: 0 (63), Config: 1FAD0351, leastFit: 26 (34), average 56
Gen: 133, fittest: 4 (63), Config: 1FAD0355, leastFit: 107 (40), average 57
Gen: 134, fittest: 6 (63), Config: 1FA50351, leastFit: 83 (41), average 58
Gen: 135, fittest: 102 (64), Config: 1FAD0351, leastFit: 91 (36), average 57
Solution Found, Gen: 135, fittest: 102 (64)

```

Fig. 2.9 GA fitness evaluation results a 2-bit adder b 2-bit multiplier

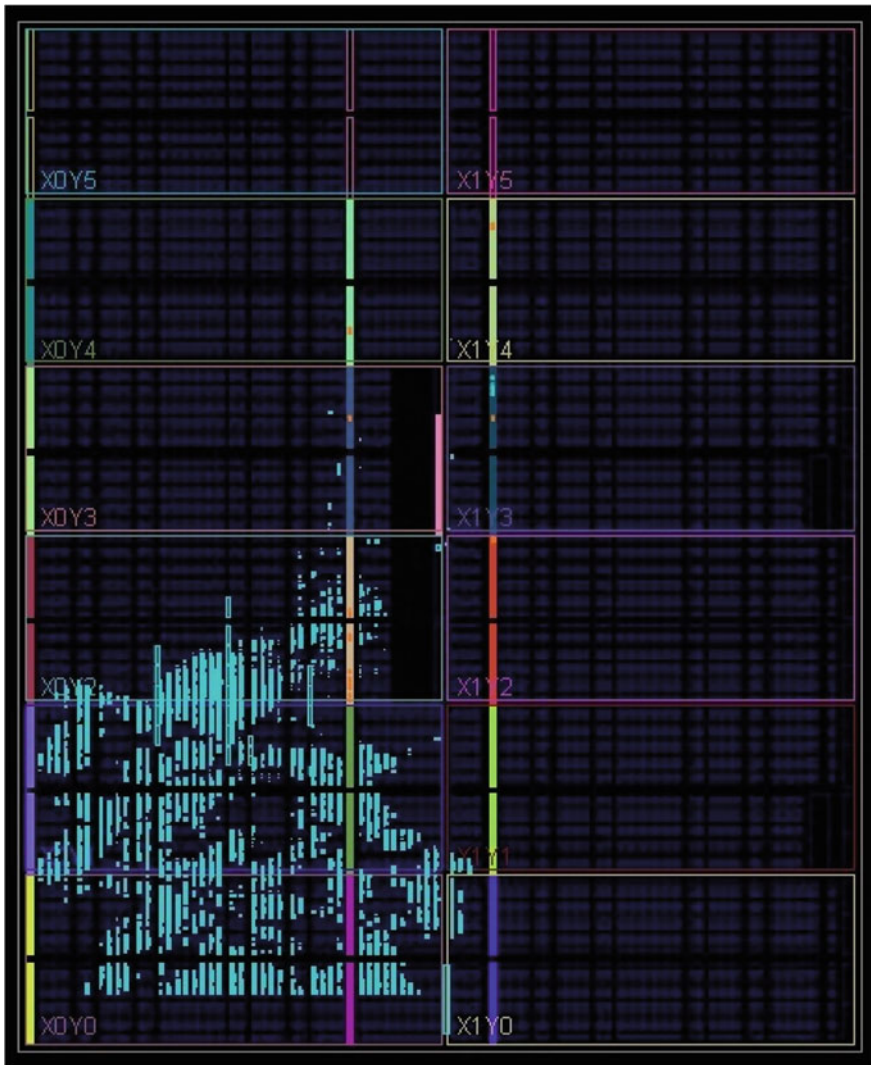


Fig. 2.10 Complete floor plan of the evolvable embedded system

## 2.6 Conclusions

The concept of the evolvable embedded systems is introduced through this paper which could provide automation of complex circuits like filters, fault tolerance circuits, and adaptive circuits. The complete evolvable hardware process can be conceptualized in a single FPGA, in which evaluation and evolution are performed in parallel in the same FPGA. The MicroBlaze soft processor is used for the

**Table 2.1** Total resource utilization of Virtex 6 (XC6VLX240T-1FFG1156) for evolvable embedded system structure

Description	Available	Used	Utilization (%)
No. of slice registers	301,440	8413	3
No. of slice LUTs	150,720	8496	6
No. of bounded IOBs	600	22	4
No. of RAMB36E1	416	16	4
No. of RAMB18E1	832	1	1
No. of DSP48E1	768	3	1

computation of genetic algorithm and communication link between the PC and FPGA system. Optimized utilization of FPGA resources was created by selecting optional peripherals for the CPU. The computation of the GA program in the processor has speeded up the evaluation process compared to the conventional use of evaluating through the PC. The evaluation time could still be accelerated up by programming the FPGA through a Compact Flash card rather than a JTAG cable. Here, the bitstreams are stored on a Compact Flash and interconnected to the processor using the SysACE peripheral.

Here, an evolvable embedded system was implemented on Virtex 6 (XC6VLX240T-1FFG1156) ML605 evaluation kit. The GA performed 88 and 135 generations, to evaluate the fitness of the 2-bit adder and multiplier circuits, respectively. The optimized generations were obtained after a manually varying the GA parameters. The system evolved through this process give novel or optimized systems. The advantage of this architecture is that any combinational circuit whose inputs and outputs not exceeding 8 bits could be evolved from the same architectural model. The GA program can also be programmed to adaptively change the parameters so as to optimize the number of generations effectively. This could solve the problem of manually varying the GA parameters for optimization. Memetic algorithm (MA) is another optimization algorithm which could be employed in the evolutionary process to obtain faster and better convergence [10]. This optimization process is still under research, but able optimize circuits with large number of inputs or circuits which require large search space.

## References

1. Lambert C, Kalganova T, Stomeo E (2007) FPGA-based systems for evolvable hardware. *World Acad Sci Eng Technol* 1:743–749
2. Sekanina L, Drabek V (2004) Theory and applications of evolvable embedded systems. In: *Proceedings of the 11th IEEE computer-based systems (ECBS'04)*. IEEE Computer Society Press, Aug 2004
3. Stomeo E, Kalganova T, Lambert C (2006) A novel genetic algorithm for evolvable hardware. In: *IEEE congress on evolutionary computation*, Vancouver, Canada, July 2006, pp 134–141

4. Bartolini DB, Cancare F, Carminati M, Sciuto D (2011) HERA: hardware evolution over reconfigurable architectures. In: 1st international workshop on computing in heterogeneous, autonomous 'N' goal environments (CHANGE 2011), March 2011, pp 1–6
5. Vasicek Z, Sekanina L (2007) An evolvable hardware system in Xilinx Virtex II Pro FPGA. *Int J Innov Comput Appl* 1(1):63–73
6. Wang J, Chen QS, Lee CH (2008) Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware. *IET Comput Digital Tech* 2(5):386–400
7. Torreson J (2004) An evolvable hardware tutorial. In: *Field programmable logic and applications. Lecture notes in computer science*, vol 3203, pp 821–830
8. Xilinx (2012) ML605 hardware User Guide, Application UG534, Oct 2012
9. Jesman R, Vallina FM, Saniie J (2006) MicroBlaze tutorial creating a simple embedded system and adding custom peripherals using Xilinx EDK software tools. *Embedded Computing and Signal Processing Laboratory, Illinois Institute of Technology*
10. Mo H, Meng L (2013) Research on evolution hardware design based on memetic algorithm. In: *IEEE workshop on memetic computing*, April 2013, pp 32–36

Proceedings of the International Conference on  
Nano-electronics, Circuits & Communication Systems

Nath, V. (Ed.)

2017, XV, 415 p. 251 illus., 166 illus. in color.,

Hardcover

ISBN: 978-981-10-2998-1