IEICE TRANS. INF. & SYST., VOL.E101–D, NO.3 MARCH 2018

PAPER

# A General Low-Cost Fast Hybrid Reconfiguration Architecture for FPGA-Based Self-Adaptive System

**Rui YAO**[†a)], **Ping ZHU**[†], **Junjie DU**[†], ***Nonmembers***, **Meiqun WANG**[†], ***Student Member***, ***and*** **Zhaihe ZHOU**[†], ***Nonmember***

**SUMMARY** Evolvable hardware (EHW) based on field-programmable gate arrays (FPGAs) opens up new possibilities towards building efficient adaptive system. State of the art EHW systems based on virtual reconfiguration and dynamic partial reconfiguration (DPR) both have their limitations. The former has a huge area overhead and circuit delay, and the later has slow configuration speed and low flexibility. Therefore a general low-cost fast hybrid reconfiguration architecture is proposed in this paper, which merges the high flexibility of virtual reconfiguration and the low resource cost of DPR. Moreover, the bitstream relocation technology is introduced to save the bitstream storage space, and the discrepancy configuration technology is adopted to reduce reconfiguration time. And an embedded RAM core is adopted to store bitstreams which accelerate the reconfiguration speed further. The proposed architecture is evaluated by the online evolution of digital image filter implemented on the Xilinx Virtex-6 FPGA development board ML605. And the experimental results show that our system has lower resource overhead, higher operating frequency, faster reconfiguration speed and less bitstream storage space in comparison with the previous works.

***key words:*** *adaptive system, hybrid reconfiguration, evolvable hardware (EHW), bitstream relocation, discrepancy configuration*

## 1. Introduction

Embedded systems engineering is nowadays facing an enormous challenge derived from the ever increasing demand for highly versatile electronic devices, high performance, increasing complexity, flexibility, low power, abilities of fault tolerance and reprogram. Low time to market and long operational life are also required. These stringent requirements, which oppose each other, have caused an exponential increase in the underlying complexity of embedded systems, since, among other concerns, the need for systems able to adapt to very diverse operating conditions throughout its lifetime arise. And this adaptation should be possible without human intervention. Besides, to obtain the aforementioned adaptation capabilities, the challenge is to combine hardware performance with the flexibility required by adaptive systems, while keeping design times as low as possible. Therefore, an adequate combination of techniques and technologies is needed to obtain this expected self-adaptive behavior [1].

The emergence of evolvable hardware (EHW) opens up new possibilities towards building efficient adaptive system. EHW is a method for circuit or device design (within some media) that uses inspiration from biology to create techniques that enable hardware designs to emerge rather than be designed. In 1993, EHW was proposed by Higuchi [2] as the automatic design of hardware using an EA. The goal is to create system able to adapt to their environment without human intervention. This line of research brings together reconfigurable hardware, autonomous systems, artificial intelligence, and automatic design [3]. In the mid-1990s, the online EHW was first implemented on Xilinx multiplexer-based XC6200 FPGA family. This family support dynamic partial reconfiguration (DPR) and the bitstream format is open, so it can implement evolution by direct bitstream manipulation. Later, this family was discontinued; and manufacturers reconfiguration technologies proved not yet mature enough to provide a seamless tool to be embedded in the final SoC. This was mainly due to the fact that unconstrained evolution in commercial FPGAs by direct bitstream manipulation was considered not possible because random modifications of the bitstream were dangerous for the device integrity. And its huge bitstream size turned the search space unmanageable. An alternative trying to overcome these limitations was proposed in the early 2000s, which is usually referred to as virtual reconfiguration circuit (VRC) [4].

A VRC within an FPGA is a (virtual) reconfiguration layer built on top of the device fabric that reduces the complexity of the reconfiguration process while increasing speed, creating a kind of application specific programmable elements. It consists of an array of processing nodes where each mode contains all the intended required functions (selected by multiplexers) along with certain inter-node connectivity configuration. The configuration information of each node is expressed in the chromosome of the candidate solution. Although the reconfiguration speed is very fast, because it only involves writing a big register, this approach suffers from a huge area overhead (all functions are implemented in every node of the graph) and large circuit delay introduced by multiplexers.

Aiming at the disadvantages of VRC, the DPR technology of modern SRAM-based FPGAs [5] can be utilized to construct EHW systems. The DPR implemented in FPGA adopts native reconfigurable array that based on reconfigurable module (RM) instead of virtual reconfiguration layer. The connection between all the RMs is fixed (each RM only connects to its four neighbors). So the reconfigurable ar-

Manuscript received July 18, 2017.
Manuscript revised October 26, 2017.
Manuscript publicized December 18, 2017.

[†]The authors are with the College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 211106, China.
a) E-mail: yaorui@nuaa.edu.cn

DOI: 10.1587/transinf.2017EDP7231

ray can implement different circuits by changing the configuration bitstreams of RMs according to the information expressed in the chromosome of each candidate solution. At any time, only one functionality is required to implement in a RM, hence the resource overhead of the DPR approach is lower than the VRC approach. And the circuit delay is also reduced since no multiplexer is required. However, this approach suffers from low flexibility owing to the fixed connection between RMs. And its reconfiguration speed is slow, because the bitstreams to configure all the RMs are often stored in external non-volatile memory. E.g., a self-reconfigurable adaptive FIR filter system design using DPR was presented in [6], in which the Compact-Flash (CF) memory was adopted to store the partial bitstreams. Moreover, usually a large amount of memory is needed, since total $n \times p$ partial bitstreams are intended to be stored, where n is the number of RMs in the array, and p is the number of possible functionalities of a RM.

In order to accelerate the configuration speed, a system architecture combining 2D data processing arrays and an enhanced DPR engine is proposed by Otero [7], in which external DDR memory instead of the CF memory was utilized to store the partial bitstreams and high reconfiguration speed was achieved by over-clocking and the dedicated data link from the external DDR2 memory to the RM. This method achieves relatively faster reconfiguration speed. However, the fast reconfiguration depends on the customized DPR engine [8], which limits its applications. And the disadvantages of low flexibility and need for large amount of external memory still exist. Moreover, since the DDR memory is volatile, the system cannot work independently.

Thereafter, various pros and cons can be observed in VRC and DPR, and a detailed comparison between these two methods was given in [9]. Then, a hybrid configuration method of VRC and DPR was adopted to build a image filter evolution system implemented on the Zynq 7000 platform [10]. The system is based on an hybrid configuration architecture, the native configuration is adopted to change the function of each RM, and the virtual configuration is adopted to modify the connection between RMs. So the flexibility of the DPR array has been enhanced, at the same time, the huge area overhead of the VRC array has been avoided. However, this method is still at the cost of a lower reconfigurable speed and a larger external memory. Therefore, in [11] the authors presented a low-level architecture for evolvable hardware in the Zynq-7000 platform, in which higher reconfiguration time is limited by the dense placement of the proposed processing elements (PEs) and the difference-based reconfiguration approach. However, it needs direct bitstream manipulation to modify the configuration bit, which is not documented by the manufacturer and may be dangerous for the integrity and correctness of bitstream. So it is always necessary to repeat the manual design flow to generate the differential bitstreams, which in turn increases the configuration time. Therefore, in fact, the reconfiguration time is not shortened by the difference-based reconfiguration as expected, when considering the time needed to generate the differential bitstreams on-line. An alternative is to pre-generated all the possible differential bitstreams off-line and store them in the external memory. But thus may introduce two main shortcomings: (1) there are too many differential bitstreams to be stored; (2) measures should be taken to determine which one should be loaded into the reconfiguration region. Moreover, the total amount of memory is not reduced. And the generality of this approach is not satisfactory.

Therefore, in this paper, we propose a general low-cost fast hybrid reconfiguration architecture for stand-alone embedded self-adaptive systems based on EHW. In our research, together with the hybrid reconfiguration architecture combining the VRC and the module-based DPR, the bitstream relocation technology is introduced to reduce the amount of memory required [12]–[14], and the discrepancy configuration technology is adopted to accelerate reconfiguration process. In addition, an embedded RAM core implemented by BRAM resource of the FPGA is included in the system except for the external CF memory. During the evolution process, any bitstream needed to reconfigure each RM is read from the internal RAM core instead of the external CF memory, thus the reconfiguration time and the evolution time are reduced significantly as well.

The rest of the paper is organized as follows. In the next section, the proposed adaptive system, together with the architecture of evolution core as well as some implementation details of bitstream relocation is described. Section 3 features the software design of the system. Finally, in Sect. 4, the design instance and experiment results are provided. And conclusions are drawn in Sect. 5.

## 2. The System Hardware Design

### 2.1 General Structure of the Adaptive System

A system design trend that seems to be consolidating as a de facto standard in the last years is the reuse and integration of IP cores created and validated independently, as a way to manage complexity. Therefore, as is shown in Fig. 1, the general structure of the adaptive system presented here is a system-on-programmable-chip (SOPC) composed of a set of IP cores, some of which are adaptive, connected through a common bus interface. An adaptive IP core is a standard IP

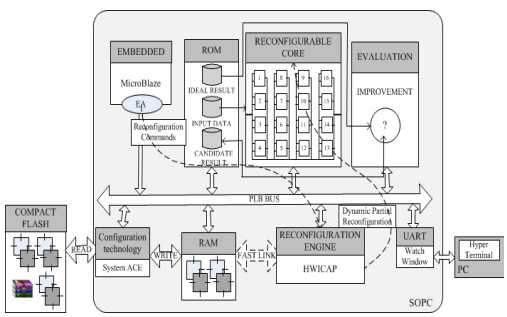


**Fig. 1** Overview of the self-adaptive system.

core with the capability of adjusting its internal processing features as commanded by an adaptation engine (AE). The AE acts upon the measured component performance to fulfill the requirements in a changing operating environment. In this proposal, the adaptability of the IP core is implemented by the DPR of the FPGA, and the AE is an evolutionary algorithm (EA) running on the on-chip processor.

In Fig. 1, the core function of the system is implemented by a customized adaptive IP core, the reconfigurable core (RC). An embedded MicroBlaze processor acts as the AE, and issues reconfiguration commands to configure the RC to implement the candidate solutions. And a reconfiguration engine (RE) is utilized to (re)configure the RC via the internal ICAP configuration port. The combination of self-reconfiguration and the EA provide the system with the required adaptability. The System ACE IP is adopted to read the ACE file and partial bitstreams from an external CF memory. The UART module provides a means of communication between the application running in the embedded processor and the user, via a hyper-terminal interface. The ROM core is used to store data to be processed, and the RAM core is used to store partial bitstream. Both cores are connected on Processor Local Bus (PLB) and can be visited by other equipment.

Once the power is on, the System ACE controller reads the ACE file from the CF card through JTAG port to initialize the system. Meanwhile, all the partial bitstreams are read from the CF card and stored in the RAM core. When evolution starts, the population is initialized and reconfiguration commands are sent by MicroBlaze to the RE according to the individual chromosome information. And the RE obtains the related bitstreams from the RAM core to configure the corresponding RMs in the RC. Then stimulus data are fed to the input of the RC; and the respond data from the output of the RC are compared with the expected data to obtain the fitness value of each individual. Thereafter, the next population is generated. Repeat the operation above, until the ideal fitness or the max generation is achieved.

### 2.2 Architecture of the Reconfigurable Core

Targeting at SRAM-based FPGAs with characteristics of two-dimension reconfiguration, the RC core can be designed as array of PEs. The size of the PE array is determined according to the complexity of the target application. A too small array may prevent a successful evolution or limit the quality of solutions. A too large array, where only a small portion of PEs are used, will have a negative impact on the evolution time because the search space will become too big. E.g., for an application of image filter, a $4 \times 4$ array is preferred, as is shown in Fig. 2.

Figure 3 shows the interconnection details and functionality configuration of a PE. Each PE can be dynamically configured to have different functionality and input mapping by means of hybrid configuration. A PE has two inputs, Ain and Bin, as well as one output. Each input can be connected with any previous column output (PCO) or the primary inputs (PIs) of the circuit under evolution, via multiplexers (MUX A and MUX B). The interconnection via multiplexers is determined according to the appropriate bit positions in the chromosome and is modified by virtual reconfiguration, which is faster than native reconfiguration. Therefore, certain flexibility in the adaptation of the data transmission flow is allowed. The functionality reconfiguration of each PE is implemented by dynamical reconfiguration based on function modules pre-synthesized and stored as independent bitstreams in the RAM core. E.g., the function modules of an image filter constitute a library shown in Table 1. If virtual reconfiguration is adopted to perform this configuration, it will be less effectively in term of area overhead and operational frequency. The selection of functionality is also based on the corresponding gene bits in the chromosome.

The output of the array is obtained from one of the last column PEs or the PIs, by using a multiplexer controlled by evolution. The smallest candidate solution can be constructed from only one PE, while the largest one may contain almost all the PEs. The PE array is the sandbox of the

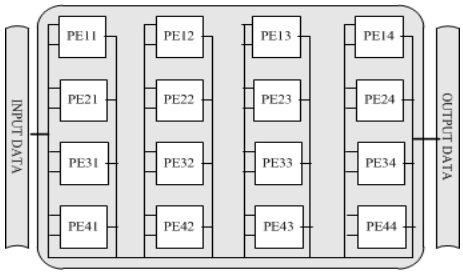


**Fig. 2** Architecture of the 2D processing array.

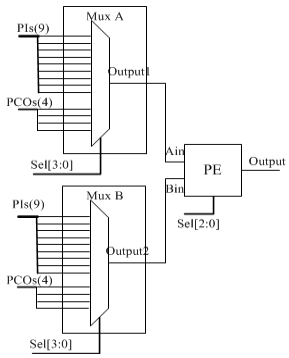


**Fig. 3** Interconnection and function configuration of a PE.

**Table 1** Functions of a PE and their encodings.

| | Description | Function | encoding |
|---|---|---|---|
| 1 | aa | ain | 000 |
| 2 | ainver | 255-ain | 001 |
| 3 | average | (ain+bin)≫1 | 010 |
| 4 | binver | 255-bin | 011 |
| 5 | adder | ain+bin | 100 |
| 6 | abs | \|ain-bin\| | 101 |
| 7 | max | max(ain,bin) | 110 |
| 8 | min | min(ain,bin) | 111 |

EA where the EA can find the solution best suited for the given specification. The search space is determined by the possible inter-connections and operations of PEs offered by the PE array.

In the hybrid architecture in [11], a reconfiguration region (RR) contains multiple PEs, which can reduce the overall area overhead; the RR is divided by the clock region column and multiple PEs can be configured at one time, which limits the reconfiguration times; but it also has some drawbacks. Firstly, a RR contains several PEs which would generate multiple combinations of configuration function. It is necessary to generate one bitstream for each combination. As a result, a RR produces a large number of bitstreams. E.g., assume that a RR contains 4 PEs, and each PE can be configured as 4 functions. Then there will be $4 \times 4 \times 4 \times 4 = 256$ configuration modes and 256 bitstreams need to be generated and stored. This will take up an amount of storage space and also add complexity to the coding of chromosomes..

Therefore, in our architecture, a RR only contains one PE module, which allows designing each PE independently, using the Xilinx modular design flow. The use of a standard commercial flow reduces the dependency with the specific details of the device. This also facilitates the use of bitstream relocation and discrepancy configuration technology. In addition, reconfiguration time is kept low because discrepancy configuration technology is adopted to reduce the number of configuration bitstream and a embedded RAM core is adopted to accelerate the reading speed of the bitstream; The bitstream storage space is also kept low because the bitstream relocation technology reduces the number of bitstreams to be stored.

The proposed architecture is a generic evolvable processing framework, and its suitability for different processing tasks depends on the utilized functionality library. Adaptation is achieved by directly configuring the required function in each PE, taking it from the library of pre-synthesized functionality. This process can be seen as placing pieces in a puzzle. For each piece (PE to reconfigure), the RR places the required element as commanded by the processor in the correct position of the array.

### 2.3 Design of RAM Core for Bitstream Storage

A customized RAM core is utilized in the proposed framework that enables fast reconfiguration to make online adaptation feasible. There are three reasons adopting the embedded RAM core to store bitstreams: (1) the data transmission speed of embedded memory is much faster than the external memory; (2) the FPGA contains BRAM resource to construct the RAM core; (3) the Xilinx officials provide the tools to generate the embedded RAM core.

Thanks to the bitstream relocation technology adopted, only those bitstreams with different functions are to be stored. In order to reduce the memory size further, the header of the bitstream in the RAM core is excluded. Hence, there is no need to parse the header during the reconfiguration process, which can reduce evolution time in turn. Here a single-port RAM core is customized which can work independently under two operation states: read state and write state. Compared to the approaches that read bitstream from external memory (such as CF or DDR), this approach possess faster data transmission speed, and it also ensures that the system is more independent, facilitate overall system migration.

### 2.4 Bitstream Relocation Design

#### 2.4.1 Reconfiguration Region Design

The partial bitstream generated by traditional DPR technology can only be configured to a fix RR of the FPGA. As is shown in Fig. 4, the partial bitstream A1 can only be configured to the RR1, and cannot be configured to the RR2 or RR3. In order to relocate the partial bitstream A1, each RR design has to be uniform. The design points of uniforming are the followings:

−Amount of reconfigurable resources
−Relative layout of reconfigurable resource
−Relative placement of proxy logic
−Relative routing path between proxy logic and static region
−Rejection of the wire from RRs, which does not through proxy logic

Take the design of an image filter as a case study. There are 16 RRs in the evolution core. In order to meet the requirements of bitstream relocation, each RR is assigned a uniformed region. Consider the following two points: Firstly, a frame is the minimum unit of configuration memory addressing in Xilinx FPGA. Secondly, all the configuration functions of a RR only use a small amount of slice resources. So, each RR is assigned to a minimal reconfigurable region.

In order to reduce the possibility that the signals of static region go through the reconfigurable area, the static resource are assigned to fixed region to implement constraints. The specific resource module distribution of the system is shown in Fig. 5, where each reconfigurable area includes 4 RRs. The interface between reconfigurable area and static region is implemented by proxy logic. Using PIN constraint to modify the position of proxy logic manually, the relative position of all the proxy logic must be consistent in the reconfigurable regions. Using LOC and BEL constraints to

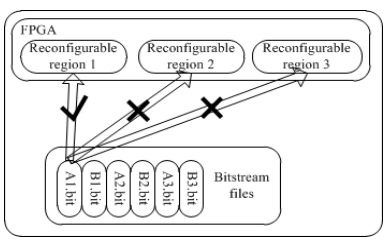


**Fig. 4** Traditional partial bitstream configuration.

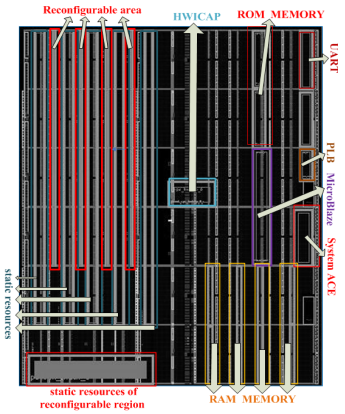


**Fig. 5** Resource distribution of the system.

```
(a) Constraints for proxy logics:
PIN "math_0/math_0/USER_LOGIC_I/LH1.ain1[0]" LOC = SLICE_X68Y59;
(b) Constraints for D-FF:
INST "math_0/math_0/USER_LOGIC_I/ain1_0" BEL = DFF;
INST "math_0/math_0/USER_LOGIC_I/ain1_0" LOC = SLICE_X18Y46;
(c) Constraints for routing:
NET "math_0/math_0/USER_LOGIC_I/ain1[0]" ROUTE = "{3;1;
6vlx240tff1156;8ee6acdd!-1;-178242;325080;S!0;-683;-272!1;""1008;
1520!2;3144;-1136!3;683;-464;L!}";
```

**Fig. 6** Three kinds of constraint commands.

add D-FF in the static region, the interconnection between proxy logic and static module can be constrained. Finally after finishing global routing, check the path information and adjust irregular routing using NET and ROUTE constraints. Figure 6 shows an example of the different constraint statements. In constraint (a), the proxy logic of ain1[0] port in user_logic module Lh1 is fixed to the position of slice resource in the 68th row and 59th column. In constraint (b), the DFF of the slice resource (Row 18, Column 46) is used to lock the external signal ain1_0 of user_logic module. In constrain (c), the numbers between S and L are relative address of signals, the numbers between -1 and S are absolute address, the rest are the information of the development board; and the signal layout can be changed by modifying the relative address.

#### 2.4.2 Configurable Bitstream Modify

After the uniform design of RRs, the partial bitstreams generated can be relocated among them. However, the configuration commands of partial bitstream have to be modified before relocation, including the old frame address register (FAR) and the cyclical redundancy check (CRC) values.

The FAR is placed in the configuration bitstream described after $(30{,}002{,}001)_{16}$ command. It indicates the start address of the partial RR. The bitstream of Xilinx Virtex-6 FPGA has three FARs which store the start addresses of different resource configuration block in the RR respectively. Among them, the second FAR store the starting address of CLB module. The design in this paper only assigns the CLB module for RRs, so just to modify the second FAR.

The CRC parameter is placed in the footer represented as $(30{,}000{,}001)_{16}$, and is utilized to check the validity of the bitstream. When creating relocation bitstream, we have to replace the frame address of the original RR by that of the target RR at first. Then we must recalculate the CRC syndrome according to Eq. (1). Finally, we have to modify the access command to the CRC register to invalidate the CRC.

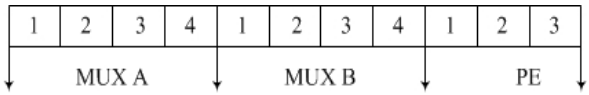


**Fig. 7** A segment of chromosome encoding.

$$X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1 \quad (1)$$

This way, one partial bitstream can be written to any RRs which significantly reduces the number of bitstream stored, thus the overall bitstream storage space get decreased.

## 3. The System Software Design

The main task of the system software design is to run the EA on the MicroBlaze to effectuate the control of the adaptive system. In this paper, the genetic algorithm (GA) is selected as the self-adaption algorithm for the reason that it have proved to be suitable for solving problems with a huge number of parameters and with an objective function characterized by several local minima (or maxima) [15]. The basic steps include the chromosome encoding, population initialization, individual chromosome configuration, fitness evaluation, genetic operator, etc. Here we still take the image filter as a case study.

### 3.1 Chromosome Encoding and Configuration

The candidate solution is described by the chromosome. The system calls population initialization function to generate the chromosomes of the initial population. The customized evolution core is attached to the bus, so the configuration of chromosome is implemented through the bus piecewise, as is shown in Fig. 3.

The reconfigurable core of the image filter consists of an array of 16 PEs, each PE has eight possible functions and multiple connection modes. Segmented binary coding scheme is applied to encode the PE array, including configuration functionality and interconnection within the reconfigurable core. For each PE, 3 binary bits is required to select the functionality and $4 \times 2$ binary bits are necessary to select the connection, as is shown in Fig. 7. A total of 11 binary bits are needed for each PE, and extra 4 binary bits are required to select the final output for the system via a 13-to-1 multiplexer, so the chromosome length is $11 \times 16 + 4 = 180$ bits. Chromosome length can be defined by macro definition. Table 1 shows the relationship between functionality

of a PE and the corresponding encoding.

### 3.2 Fitness Evaluation

Generally, Mean Difference per Pixel (*Mdpp*) is used to evaluate the filtering result of an image filter. *Mdpp* describes the difference between the filtered image and the noise-free image, which represents the quality of the solution. The *Mdpp* is calculated by the following equation:

$$Mdpp = \frac{\sum_{i=1}^{c} \sum_{j=1}^{r} |ideal(i, j) - filtered(i, j)|}{c \times r} \quad (2)$$

Where $c$ is the number of image columns, $r$ is the number of image rows, $filtered(i, j)$ is the filtered pixel, and $ideal(i, j)$ is the noise-free pixel. As is expected, the *Mdpp* measures the absolute difference between the filtered and noise-free images. The smaller the *Mdpp* is, the better the filtering result is. And the *Mdpp* value of an ideal filter able to suppress all the noise in the image would is close to 0.

### 3.3 Evolution Process

The flow of evolution operation is shown in Fig. 8. The genetic operators are used to create the new generation of population which includes selection and mutation. In addition, there are two termination evolution conditions, one is to find the expected individual (E.g., the filter whose *Mdpp* is zero), namely to find the optimal solution, the other is to arrive at the maximum generation preset.

### 3.4 Design of Discrepancy Configuration

When evaluating a candidate solution, it is necessary to decode the chromosome to determine the functionality of each RR in the array, and configure it with the corresponding bitstream. In the previous DPR-based EHW systems, all the RRs are intended to be configured during the configuration process, without consideration for the similarity of the new and the old chromosome (namely complete configuration).

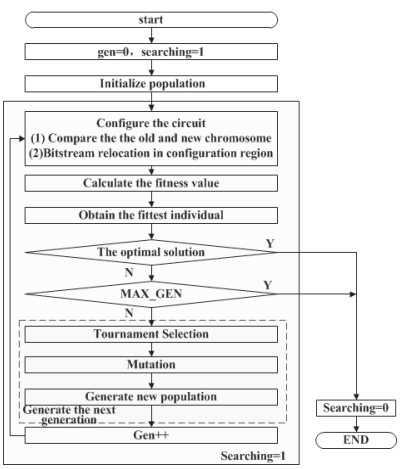


**Fig. 8** The flow of adaptive evolution.

Assume that it takes time $t$ to configure a RR and the evolution core is made up of an array of $m \times n$ RRs, it will need $t \times m \times n$ to configure a candidate, when complete configuration is adopted.

In fact, the new chromosome is usually not totally different with the old one. And the difference between them may become slighter and slighter during the evolution process. So a great deal of time can be saved if we only reconfigure the RRs whose functionalities are different. Therefore, the discrepancy configuration technology is adopted in this paper, so as to reduce the configuration time and accelerate the evolution speed.

To do this we define three configuration matrices, i.e., new matrix A, old matrix B and differential matrix C. If the evolution core is an array of $m \times n$ PEs, the new matrix A is established according to the new chromosome.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad (3)$$

Similarly, the old matrix B is established according to the old chromosome.

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix} \quad (4)$$

In Eqs. (3) and (4), $a_{ij}$ and $b_{ij}$ $(i = 1, 2, \dots, m,\ j = 1, 2, \dots, n)$ are the functionality encoding of the corresponding PEs. Then the differential matrix C can be obtained by comparing the corresponding elements of matrix A and B.

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{pmatrix} \quad (5)$$

Where $c_{ij}$ $(i = 1, 2, \dots, m,\ j = 1, 2, \dots, n)$ is the comparison value of $a_{ij}$ and $b_{ij}$; if they are not equal, $b_{ij} = a_{ij}$; otherwise, $c_{ij}$ is set as a fixed value different to all the possible values of the functionality encoding of each RR.

Then the configuration process of a candidate is as follows. Firstly, determine matrix A and B according to the new and old chromosome, respectively. Secondly, compare corresponding elements between the two matrices and get matrix C. Finally, configure the RRs according to matrix C. If $c_{ij}$ is the pre-defined fixed value, the corresponding RR keeps unchanged; otherwise, it needs to be reconfigured again. Thus discrepancy configuration is implemented. Since there is no need to configure all the RRs one by one when evaluating each chromosome, the whole configuration time get reduced, and the evolution speeded up.

## 4. Experiment Result and Discussion

### 4.1 Design Instance and Hardware Platform

The proposed self-adaptive system with hybrid fast reconfiguration architecture is a general system architecture, and it can implement any adaptive data-processing or decision-making applications (e.g., data-parallel image processing, machine vision, robot path planning and target recognition), on any FPGA that supports DPR and has enough resources to host a soft microprocessor. Here an adaptive system for switching image filter evolution was implemented on a virtex-6 FPGA platform ML605 using the proposed hybrid reconfiguration architecture. In the evolution process, there are a mass of data, including the image to be processed, the ideal image and the filtered image, and all of which should participate in the fitness calculations. In order to ease the tasks of data storage and reading, the ideal data and the data to be processed is stored in customized ROM cores on the FPGA. As input of the evolvable core, the data to be processed is read under the control of the software in the evolution process. Attached to the architecture is a kernel of $3 \times 3$ pixels, which implies nine inputs for the filter: the current pixel and its eight neighbors. The pixel of filtered image is defined as a variable for which a temporary memory space is assigned. And the processed data is compared with ideal data to get fitness according to Eq. (5).

### 4.2 Resource and Timing Analysis

The overall resource utilization of the adaptive system for switching image filter evolution is shown in Table 2. As can be seen in Table 2, the utilization of various resources (except the ICAP and BSCAN resources) is comparatively small, i.e., less than 10%. Therefore, more complex applications can be implemented using the proposed system architecture. In fact, for different applications, the ICAP and BSCAN resources are the same, and the main difference in hardware lies in the reconfigurable core that composed of PE array. And the size of the PE array to be evolved in the proposed system depends on the amount of hardware resource of the FPGA utilized and the size of each PE as well. If the hardware resource allows, any size of PE array can be evolved theoretically.

As is shown in Table 3, it is the resource utilization comparison in the design of user IP core between our approach and VRC approach. As it can be observed, the proposed method has lower resource overhead in comparison with the VRC approach. This is mainly because that in our hybrid reconfiguration architecture, there is no resource redundancy in each PE; i.e., the amount of resource assigned to each PE is not intend to implement all the possible functionalities as VRC does, but is just to implement the most complex functionality among them. In addition, the signal interfaces between static and reconfigurable regions are performed through Proxy logic, not Bus macros, which does not need to reserve resources and the flexibility of the system design gets enhanced.

**Table 2** System resource utilization.

| Resource | Estimation | Available | Utilization |
|---|---|---|---|
| Register | 5,745 | 301,440 | 1% |
| LUT | 7,920 | 150,720 | 5% |
| Slice | 2,695 | 37,680 | 7% |
| IO | 24 | 600 | 4% |
| RAMB36E1 | 44 | 832 | 5% |
| RAMB18E1 | 58 | 832 | 6% |
| ILOGICE1 | 9 | 720 | 1% |
| OLOGICE1 | 19 | 720 | 2% |
| BSCAN | 1 | 4 | 25% |
| DSP48E1 | 3 | 768 | 1% |
| ICAP | 1 | 2 | 50% |
| MMCM_ADV | 1 | 12 | 8% |
| Global clock buffer | 3 | 32 | 9% |

**Table 3** Device resources utilization comparison.

| | Available | VRC | Proposed array |
|---|---|---|---|
| Slice Registers | 301,440 | 757 | 396 |
| Slice LUTs | 150,720 | 2,796 | 828 |
| Logic LUTs | 150,720 | 2,668 | 700 |
| Memory LUTs | 58,400 | 128 | 128 |

The design platform is based on Virtex-6 ML605.

The timing reports comparison between the designs of our approach and VRC approach is shown in Table 4. As can been seen in Table 4, the maximum frequency of our PE array is higher than that of the VRC array. The reason lies in that no large numbers of multiplexers are used in our array. In addition, the maximum frequency of each PE, reported in the synthesis process, using ISE 12.4 tool, is over 1,200 MHz. However, the maximum frequency of the full array is 504 MHz after the mapping and implementation phases. This increase in the delay is due to the inter-connection of the different PEs through the Proxy logic and the interface to the static design area.

### 4.3 Bitstreams Storage Analyses

Table 5 shows the comparison of the number of bitstreams and the size of the memory storage using bitstream relocation technique (our method) and the non-bitstream-relocation technique (traditional method), respectively. For the evolution of the image filter, each RR can be configured to implement 8 possible functionalities, so it only need to store 8 bitstream files when adopting our method, which is much less than the 120 bitstream files when adopting the traditional method. And the size of storage space is reduced significantly, total 1,510Kb, i.e., about 93.5% of storage space is reduced.

In fact, the sizes of storage space that can be saved depend on the number of RRs. There are 16 RRs in the reconfigurable core of the image filter; it only needs to generate 8 kinds of bitstream files for one RR when our bitstream relocation technology is adopted. Rather than the traditional methods, it needs to generate 8 bitstream files for each RR.

**Table 4** Timing comparison.

| | Propose array | Array in VRC | Proposed PE | PE in VRC |
|---|---|---|---|---|
| Minimum period (ns): | 1.908 | 5.471 | 0.785 | 1.523 |
| Maximum Frequency (MHz): | 504.032 | 182.782 | 1,273.885 | 656.599 |
| Minimum input arrival time before clock (ns): | 3.917 | 3.845 | 0.434 | 4.872 |
| Maximum output required time after clock (ns): | 2.729 | 2.729 | 0.375 | 0.375 |
| Maximum combinational path delay (ns): | 1.895 | 1.985 | —— | —— |

**Table 5** Comparison of two configuration techniques.

| FUNC | Number of bitstream file | | Size of storage space | |
|---|---|---|---|---|
| | Relocation | No relocation | Relocation | No relocation |
| aa | 1 | 16 | 13Kb | 208Kb |
| ainver | 1 | 16 | 13Kb | 208Kb |
| average | 1 | 16 | 13Kb | 208Kb |
| binver | 1 | 16 | 13Kb | 208Kb |
| adder | 1 | 16 | 13Kb | 208Kb |
| abs | 1 | 16 | 13Kb | 208Kb |
| max | 1 | 16 | 13Kb | 208Kb |
| min | 1 | 16 | 13Kb | 208Kb |
| Total | 8 | 128 | 104Kb | 1,664Kb |

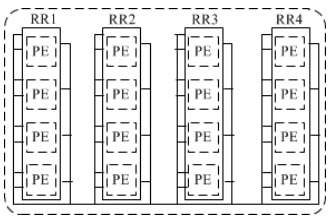


**Fig. 9** The array architecture of literature [11] with 4 RRs.

If the difference-based reconfiguration method in paper [11] is adopted, the array architecture of the 16 PEs is shown in Fig. 9. The array includes 4 RRs, each RR consists of 4 PEs, and each PE can be configured to implement 8 kinds of functionalities. In order to perform difference-based reconfiguration, first of all, 4 complete partial bitstreams need to be created for each RR. Secondly, based on these partial bitstreams, $8^4 - 1 = 4{,}095$ kinds of differential bitstreams need to be generated. Although these differential bitstreams only modify some LUTs and the size is small, its quantity is huge, which brings inconvenience for storage. There are at least 4 partial bitstreams and 4,095 differential bitstreams that need to be stored. In addition, the design of difference-based reconfiguration requires an understanding of how to make logic changes using the FPGA_Editor tool and to produce a bitstream using the BitGen tool, which makes the design more dependent on the external development environment. Therefore, in terms of bitstream storage, the bitstream relocation based on hybrid reconfiguration is superior to the difference-based one in [11] for the requirements of EHW.

In addition, the size of storage space also lies on the FPGA family utilized. And the configuration bitstream files for each FPGA family are different. If the FPGA family utilized changes, the bitstream files have to be regenerated.

**Table 6** Comparison of configuration time.

| Memory | Configuration Time |
|---|---|
| CF card | 23.911ms |
| RAM core | 2.076ms |
| DDR3 SODIMM | 2.296ms |

### 4.4 Evolution Time Analysis

The evolution time of the system depends on the time needed to evaluate an individual and the total number of individuals to be evaluated. The time required to evaluate a candidates mainly lies on two factors, configuration time and fitness calculation time. The latter depends on the selected application. And the former depends on the area size of a RR, the number of RRs to be reconfigured (the number of changed PEs) and the frequency at which configuration data is sent to the ICAP. In our system, the birstreams are stored in the embedded RAM, so that the data transfer frequency is consistent with the bus frequency; and the mutate rate is small, hence the number of changed PEs is less. In addition, to reconfigure a RR made up of a CLB column requires the change of 40 frames in the configuration memory; and the partial bitstream is composed of 3,142 configuration words with the header omitted. In the case of Xilinx HWICAP, these configuration words have to be transmitted via the PLB bus.

Here some contrast experiments are conducted based on the same hardware platform and software system as well as the same parameters setting. Some parameters are set as follows: population size PS = 64, tournament size TS = 10, mutation rate PM = 3/256, terminated generation = 10,000 and the soft core working frequency is 100MHz.

Firstly, experiments are conducted by reading configuration data from the external CF card, the embedded RAM core and the external DDR3 SODIMM, respectively. And the time statistics for configuring 3142 words is shown in Table 6. Obviously, the configuration time of reading configuration data from embedded RAM is much shorter than the time reading data from the external CF card (about 90% of configuration time is reduced); and it is also a little shorter than reading data from DDR3 SODIMM (about 10% configuration time can get reduced). Figure 10 illustrates the join methods of these memories and the bus. The configuration data in CF card and the DDR3 SODIMM are transferred to the PLB via the system ACE controller and the muti-port memory controller (MPMC), respectively, which increases the configuration time; and the RAM core is directly connected to PLB which get the fastest data configu-

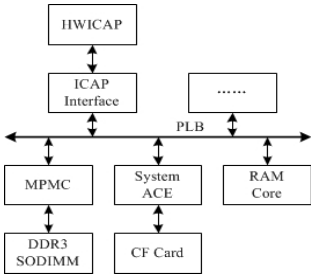


**Fig. 10** Block diagram of memory connected on PLB in system.

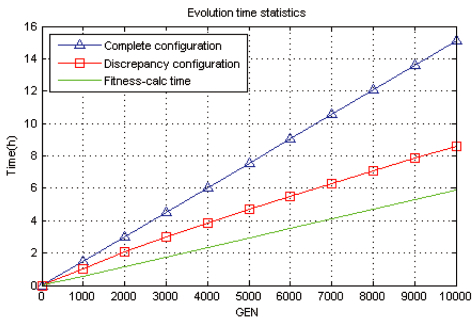


**Fig. 11** Time comparison of two configuration approaches.

ration speed.

Then experiments are conducted by adopting the complete configuration and the discrepancy configuration approaches respectively, with the embedded RAM core adopted to store bitstreams. And the evolution time comparison is shown in Fig. 11. As can be seen in Fig. 11, there is a direct proportion relationship between the evolution time and the evolution generation when the complete configuration is adopted, because all the RRs corresponding to each chromosome need to be reconfigured, which caused redundant configuration. Therefore, it takes a little more time to finish the evolution. When the discrepancy configuration is adopted, only those RRs whose functionalities changed need to be reconfigured, without redundant configuration. Therefore, it takes less time (save more than 40% time in 10,000 generation) to finish the evolution.

In conclusion, the evolution speed of the adaptive system can be accelerated significantly by adopting embedded RAM core to store bitstreams and the discrepancy configuration technology to perform DPR.

In addition, the size of storage space also lies on the FPGA family utilized. And the configuration bitstream files for each FPGA family are different. If the FPGA family utilized changes, the bitstream files have to be regenerated.

In order to analysis the degree of configuration time to evolution process further, we have calculate the fitness calculation time in the evolution process and plotted it in Fig. 11 too. As can be seen from Fig. 11, when the evolution generation up to 10,000, the fitness calculation time accounts for 75% of the time in discrepancy configuration approach and it accounts for 40% of the time in complete configuration approach. Therefore, it can greatly reduce the configuration time when discrepancy configuration adopted.

**Table 7** *Mdpp* for a set of experimental images.

| | The *Mdpp* of polluted image | The *Mdpp* of filtered image |
|---|---|---|
| 5% S&P noise | 6.55 | 1.01 |
| 10% S&P noise | 13.167 | 2.223 |
| 15% S&P noise | 19.671 | 8.373 |
| Gaussian noise with variance 0.01 | 20.495 | 10.085 |
| Gaussian noise with variance 0.05 | 43.363 | 16.669 |
| Gaussian noise with variance 0.1 | 56.493 | 26.602 |


**Fig. 12** Result of image filtering.

### 4.5 Effect of Image Filtering

To validate the generality of the image filter system, experiments are conducted by filtering 125 × 124 Lena images polluted by different levels of Gaussian noise and salt and pepper noise. Table 7 shows the comparison of *Mdpp* between the polluted image and the filtered image. Figure 12 shows the contrast results of polluted images and filtered images.

In Fig. 12, a1, a2 and a3 represent images polluted by different levels of salt and pepper noise, and b1, b2 and b3 represent the corresponding filtered results; c1, c2 and c3 represent the images polluted by different levels of Gaussian noise, and d1, d2 and d3 represent the corresponding filtered results.

As can be seen in Table 7, the *Mdpp* after filtering is smaller than the one before filtering. And all the results in Table 7 and Fig. 12 indicate that the system is able to evolve very different type of filters. So we can come to conclusion that the self-adaption system designed in this paper can filter image noise in a certain extent.

## 5. Conclusions

In this paper, a general low-cost fast hybrid reconfiguration architecture is proposed to construct stand-alone em-

bedded self-adaptive systems based on EHW. This architecture merges the advantages of the virtual reconfiguration and the module-based DPR. The use of virtual reconfiguration for setting the interconnections between PEs enhances the flexibility of the DPR approach. And the utilization of DPR for reconfiguring the functionality of PEs reduces the area overhead of the virtual reconfiguration approach. Moreover, our module-based DPR approach is more general compared with the difference-based one, since it is independent on the external software development environment and simplifies the encoding and decoding of the bitstreams. In addition, in view of the bitstream storage involved by DPR, the bitstream relocation technology is introduced to save the bitstream storage space, which reduces the amount of memory needed to store the bitstreams.

Furthermore, the discrepancy configuration technology is adopted to reduce reconfiguration time. And an embedded RAM core implemented by BRAM resource of the FPGA is included in the system except for the external CF and DDR memory. During the evolution process, any bitstream needed to reconfigure each RM is read from the internal RAM core instead of the external CF and DDR memory, which accelerates the reconfiguration speed significantly.

The proposed architecture is evaluated by the online evolution of digital image filter implemented on the Xilinx Virtex-6 FPGA development board ML605. And the experimental results demonstrate advantages over the previous work considering flexibility, generality, area overhead, operating frequency, reconfiguration speed and bitstream storage space in comparison with the previous works.

Although the reconfiguration speed has been accelerated significantly by our approach, the total evolution time to find a satisfactory solution is still long, which limits the real-world applications of EHW-based embedded adaptive systems. The evolution time mainly lies on the speed of fitness calculation and the efficiency of the EA, except for the configuration time. Therefore, in the future, we will explore the parallel pipeline processing capabilities of FPGAs to accelerate the fitness calculation process, and will introduce new EAs to increase the search efficiency.

## Acknowledgments

This work was supported by National Natural Science Foundation of China (NO. 61402226).

### References

[1] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Self-reconfigurable evolvable hardware system for adaptive image processing," IEEE Trans. Comput., vol.62, no.8, pp.1481–1493, Aug. 2013.

[2] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu, "Real-world applications of analog and digital evolvable hardware," IEEE Trans. Evol. Comput., vol.3, no.3, pp.220–235, Sept. 1999.

[3] A. Stoica, "Evolvable hardware for autonomous systems," Proc. Congress Evolutionary Computation, pp.1–125, 2004.

[4] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware," Proc. Fifth Int. Conf. Evolvable Systems: From Biology to Hardware, Lecture Notes in Computer Science, vol.2606, pp.186–197, Springer, Berlin, Heidelberg, 2003.

[5] Xilinx Inc., Partial Reconfiguration of a Processor Peripheral (ug744), 2010-9-21.

[6] C.-S. Choi and H. Lee, "A self-reconfigurable adaptive FIR filter system on partial reconfiguration platform," IEICE Trans. Inf. & Syst., vol.E90-D, no.12, pp.1932–1938, Dec. 2007

[7] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "A fast reconfigurable 2D HW core architecture on FPGAs for evolvable self-adaptive systems," NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2011, pp.336–343, June 2011.

[8] A. Otero, A. Morales-Cas, J. Portilla, E. de la Torre, and T. Riesgo, "A modular peripheral to support self-reconfiguration in SoCs," 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2010, pp.88–95, Sept. 2010.

[9] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Implementation techniques for evolvable HW systems: Virtual vs. dynamic reconfiguration," 22nd International Conference on Field Programmable Logic and Applications, FPL 2012, pp.547–550, Aug. 2012.

[10] R. Dobai and L. Sekanina, "Image filter evolution on the Xilinx Zynq platform," NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2013, pp.164–171, 2013.

[11] R. Dobai and L. Sekanina, "Low-level flexible architecture with hybrid reconfiguration for evolvable hardware," ACM Trans. Reconfigurable Technology and Systems, vol.8, no.3, pp.20:1–pp:24, May 2015.

[12] K. Zhang, H. Lu, S. Xiao, and W. Hu, "Runtime bitstream relocation based intrinsic evolvable system," Elektronika ir Elektrotechnika, vol.20, no.6, pp.93–99, 2014.

[13] S. Corbetta, M. Morandi, M. Novati, M.D. Santambrogio, D. Sciuto, and P. Spoletini, "Internal and external bitstream relocation for partial dynamic reconfiguration," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.17, no.11, pp.1650–1654, 2009.

[14] M.A. Ponrani, G. Manoj, and R. Rajesvari, "Module based partial reconfiguration on bitstream relocation filter," Int. Journal of Computer Applications, vol.66, pp.23–28, 2013.

[15] F. Cancare, D.B. Bartolini, M. Carminati, D. Sciuto, and M.D. Santambrogio, "On the evolution of hardware circuits via reconfigurable architectures," ACM Trans. Reconfigurable Technology and Systems, vol.5, no.4, pp.22:1–22:22, 2012.

**Rui Yao** received the B.S. and M.S. degrees in optoelectronics and physical electronics from Harbin Institute of Technology in 1996 and 1998, respectively, and the Ph.D. degree in Testing and Measuring Technology and Instrumentation from Nanjing University of Aeronautics and Astronautics (NUAA) in 2008. She is currently an associate professor with College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

**Ping Zhu** is currently pursuing the Master degree in Instrumentation engineering, from NUAA. Her current research direction is the computer measurement and control system.

**Junjie Du** is currently pursuing the Master degree in Instrumentation engineering from NUAA.

**Meiqun Wang** is currently pursuing the Master degree in Instrumentation engineering from NUAA.

**Zhaihe Zhou** received the M.S. and Ph.D. degrees in navigation, guidance and control from NUAA in 2002 and 2010, respectively. He is currently an associate professor with College of Automation Engineering NUAA China.