Runtime Bitstream Relocation based Intrinsic Evolvable System

Kaifeng Zhang[1], Huanzhang Lu[1], Shanzhu Xiao[1], Weidong Hu[1]
[1]*ATR key lab, National University of Defense Technology, Changsha, China*
zkf0100007@163.com

***Abstract*—Earlier evolvable hardware (EHW) platforms suffer from major drawbacks such as high area and delay overheads, high configuration memory overhead, low configuration speed and lack of flexibility. In this paper, we propose an intrinsic evolvable system on dynamic partial reconfigurable (DPR) platform using bitstream relocation technique to address these limitations. This relieves the overhead of configuration memory required to save the partial bistream (PB). In addition, the data transfer time between the configuration memory and the field programmable gate array (FPGA) is also reduced, which leads to a relatively high configuration speed. We implemented the proposed evolvable system using an FPGA board with an application of an adaptive finite impulse response (FIR) filter. The experimental result shows that the proposed system can achieve 116 × configuration speedup and 85 % configuration memory saving.**

***Index Terms*—Evolvable hardware, dynamic partial reconfiguration, bitstream relocation, bitstream compression.**

I. INTRODUCTION

In the past decades, evolvable hardware (EHW) has received increasing attention from all over the world [1]–[6]. EHW refers to hardware that can change its architecture and behaviour dynamically and autonomously by interacting with its environment [7]. EHW can be classified into two major divisions, extrinsic EHW and intrinsic EHW. In the intrinsic EHW approach, one of the most interesting features is that the calculation of fitness value of each candidate circuit, which is the most time consuming part in the whole evolutionary process, can be evaluated more quickly than in software simulation based extrinsic EHW [8]. Another visible advantage is that it can exploit physical properties of the electronic platform or environment and thus to provide some innovative features such as fault tolerance.

An efficient and flexible platform plays a vital important role in the research domain of intrinsic EHW. Owing to no commercial evolvable platform available, much research work has been undertaken on evolvable platform since the birth of EHW. The earliest evolvable platform includes generic array logic (GAL) [9] and programmable logic array (PLA) [10], due to the limited logic resource of these two simple programmable logic devices (PLD), only relatively simple circuit could be evolved. The Xilinx XC6200 series field programmable gate array (FPGA), with the feature of known bitstream format and safe configuration, has made great promotion in intrinsic EHW. A number of evolutionary experiments were conducted on XC6200 [11]. Unfortunately, XC6200 was withdrawn and replaced by subsequent Virtex family. To prevent from reverse engineering, the bitstream format of Virtex FPGA is unknown. Although Xilinx provided a Java based bitstream manipulation tools named JBits, new devices after Virtex II were not supported by JBits. Moreover, JBits running on Java virtual machine which have the disadvantage of greater complexity and higher computation cost are not suitable for embedded systems. A well-known alternative is the use of virtual reconfigurable circuits (VRC), a reconfigurable layer built on top of the reconfigurable fabric that reduces the complexity of the reconfiguration process, creating a kind of application specific programmable elements [12]. Main problems of VRCs are area and delay overheads, as well as power consumption [13]. Other custom evolvable platform includes evolvable motherboard [14], RISA [15] and POEtic [16] et al. Although great progress has been made in the field of custom evolvable platform, the existing platforms suffer from high cost and less generality.

Fortunately, the promising feature of partial reconfiguration at runtime, which provided by the Virtex FPGA has open a brand new paradigm for intrinsic EHW. Previous researchers have presented evolvable system using dynamic partial reconfiguration (DPR). Upegui introduced an evolvable system by dynamically changing the look-up table (LUT) contents while keeping a safe predefined fixed routing [17]. This approach is mainly constrained by its very specific purpose and the coarse granularity of the modules. A stand-alone self-reconfigurable adaptive FIR filter system using the DPR method was presented in [18]. In this case, an external processor was employed, and the DPR was implemented by system ACE (Advanced Configuration Environment) using JTAG interface which suffered from low speed. A system architecture combining 2D data processing arrays and an enhanced DPR engine is proposed by Otero [13]. Although this alternative achieves relatively higher reconfiguration speed, the two dimensional mesh type systolic arrays focused on data processing applications, so the generality was constrained. However, high reconfiguration speed was achieved by over-clocking, which constrained its use in other applications. Further, bitstream compression is

Manuscript received July 31, 2013; accepted March 10, 2014.
This research was funded by a National 863 High-Tech Research and Development Plan of China (No. 2009AA8050701).

not considered in Otero's work.

In this paper, a novel intrinsic evolvable system based on DPR platform is proposed that is using bitstream relocation technique. The purpose of this evolvable system is to construct a general-purpose and flexible platform for EHW research. More specifically, the novel features of our proposed intrinsic evolvable system are given below.

1. Runtime relocatability – eliminate the need of pre-design all possible configuration bitstream for each partial reconfigurable partition (PRP).
2. High configuration speed – by introducing a bitsream compression technology, the data transfer time between configuration memory and FPGA was reduced which lead to a high configuration speed.
3. Low memory overhead – the bitstream relocation and compression technology both contribute to low memory overhead.

The remainder of this paper is organized as follows. Section II gives a brief review of DPR and bitstream relocation. Section III presents an intrinsic evolvable system based on dynamic partial reconfigurable platform using bitstream relocation technique, and the design methodology and hardware implementation details are described in detail. In Section IV, experiments were conducted to verify the validity of our proposed evolvable system. Finally, conclusions are drawn in Section V.

II. DYNAMIC PARTIAL RECONFIGURATION AND BITSTREAM RELOCATION

This section discusses the architecture of Xilinx Virtex-5 FPGAs and the reconfiguration mechanism. Although we addressed with Virtex-5, our system is rather general and can be applied for other FPGA platforms.

A. Dynamic Partial Reconfiguration

DPR means parts of FPGA can be changed at runtime while the rest parts are still functioning. Several approaches have been provided by Xilinx for DPR, such as difference-based, module-based, early access partial reconfiguration (EAPR), and the newest partition-based technique. The difference-based flow is only applicable for small design change. The module-based and EAPR flow are not supported by new design tools. The partition-based flow has several promising features compared to its predecessors. Bus macros instantiation are no longer required in partition-based flow, and the counterpart partition pins are automatically inserted by the design tools.

B. Partial Bitstream and Relocation

A partial bitstream (PB) is used to configure a PRP, which contains dummy words, synchronization word, commands, data, and cyclic redundancy check (CRC) values. The PB is composed of configuration frames, and a frame has a unique address. In principle, a PB is only usable for an associated PRP which has assigned during design phase. In the case of a design that contains N PRPs and M modules, N*M PBs should be designed, which brings high configuration storage overhead and leads to a time consuming job. In order to solve this problem, a technique named bitstream relocation was developed [19], [20]. Bitstream relocation means that a PB assigned to PRP_A could be used to configure PRP_B at runtime. By introducing bitstream relocation into a design scenario mentioned above, only M PBs are needed. The studies of bitstream relocation can be divided into bitstream manipulation and relocatable partition design [21], [22]. The former includes REPLICA, REPLICA2Pro and BiRF [19] et al. To design a relocatable partition is the key point of relocating bitstream at runtime. On the basis of a relocatable partition, the bitstream relocation could be simply conducted by modifying the frame address and CRC values. The modification could be performed by the means of software or hardware. The detailed description of bitstream relocation is elucidated in the following section.

III. PROPOSED INTRINSIC EVOLVABLE SYSTEM

In this section, we discuss the proposed intrinsic evolvable system (IES). This section is further divided into four subsections. First, we give a brief introduction to the overview of intrinsic evolvable system. In view of improving the reconfiguration speed and reducing configuration storage overhead, a custom ICAP supporting bitstream decompression and relocation was designed. Finally, the whole design flow was given, and attention was focused on creating relocatable partitions.

A. Overview of Intrinsic Evolvable System

The intrinsic evolvable system we proposed is composed of the following modules: a PowerPC based embedded system, a custom internal configuration access port (ICAP) with the feature of bitstream decompression, and relocatable partitions.

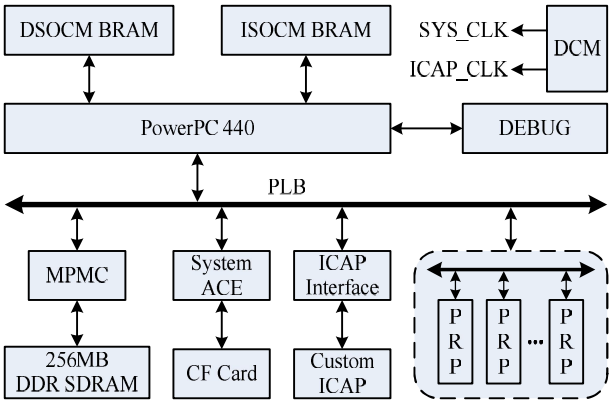

Fig. 1. Block diagram of intrinsic evolvable system.

Figure 1 shows the block diagram of the proposed evolvable system. The Power PC 440 controls the whole system work flow, and the evolutionary algorithm (EA) is also running on it. A 256 MByte DDR SDRAM is attached to the processor local bus (PLB) via the multi-port memory controller (MPMC) IP core. The configuration information includes full/partial bitstream are stored in a nonvolatile compact flash (CF) card. The System ACE controller is in charge of data transfer between the FPGA and the CF card. The DPR is performed using the custom ICAP. The compressed PB read from the CF card is decompressed by the decompression logic. The decompressed bitstream is manipulated by a relocation filter to modify the target frame address. Finally, the relocated bitstream is loaded into the

FPGA to configure the PRPs.

B. Custom ICAP

By analysing the reconfiguration process, most time is spent on data transfer between the CF card and the configuration port such as joint test action group (JTAG), SelectMAP or ICAP. Due to the nature of the CF card as non-volatile memory which suffers from low speed, using bitstream compression to reduce data transfer time is a feasible way. A custom ICAP was developed to fulfil this purpose. As depicted in Fig. 2, the custom ICAP consists of two FIFOs, bitstream decompressor, bitstream relocation filter (BiRF) and ICAP_WRAPPER.

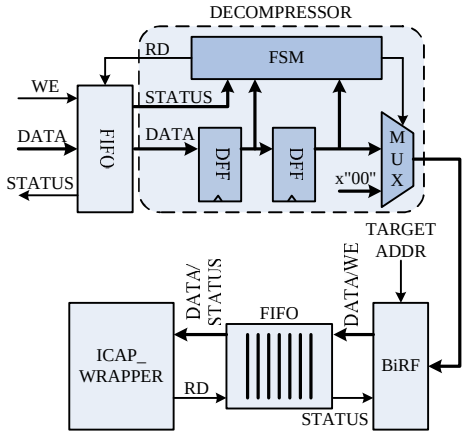

Fig. 2. Block diagram of custom ICAP.

In view of the fact that the PB contains large amount of consecutive zeroes, a relatively high compression ratio could be obtained. Among a number of data compression algorithms, run-length encoding is a simple technique to compress a sequence of identical tokens belonging to a data stream [23]. In this paper, we proposed a simplified run-length bitstream compression (SRLBC) method. In our approach, we treat the input data stream as a byte-width manner, and only zeroes are compressed, the non-zero values are not compressed. A pseudocode of the proposed SRLBC is provided in Fig. 3.

```
Repeat until reaches the end of bitstream
    Read in a byte from bitstream file, named Current_byte
    If Current_byte != 0x00
    If Zero_flag == 0
      Output_byte = Curren_byte
    Else
      Output_byte = 0x00
            Output_byte = Num_zeros
            Output_byte = Curren_byte
    End
        Zero_flag = 0
    Else
        Zero_flag = 1
        Num_zeros++
  End
End
```

Fig. 3. Pseudocode of the proposed SRLBC.

The decompression is relatively simple, if the input byte is non-zero, output it directly; if the input byte is zero, the following byte *L* refers to the run-length (number of consecutive zeroes), just output *L* zeroes. Repeating the process until it reaches the end of the bitstream. Owing to the simplicity of our proposed SRLBC, it is quite suitable for FPGA implementation. In order to validate the SRLBC, several commonly used function modules were implemented as reconfigurable components. The PB generated by Xilinx tools was compressed by SRLBC which was implemented using Matlab. The decompressor was implemented on FPGA. As shown in Fig. 3, the decompressor is composed of DFFs, multiplexer and a simple finite state machine (FSM). The waveform of bitstream decompression was shown in Fig. 4.

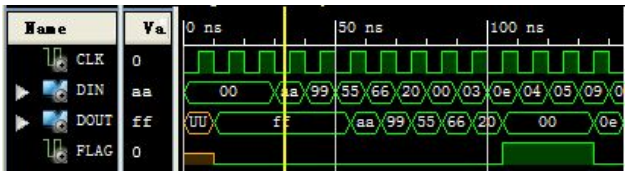

Fig. 4. Waveform of bitstream decompression.

As shown in Fig. 4, DIN indicates the input of compressed bitstream, DOUT indicates the decompressed bitstream and FLAG indicates the decompression status which can be used to control the RD of the input FIFO.

Table I shows the performance of SRLBC. Compression ratios vary with the target device family and the target design contents. For a same partition, a 46-tap low pass FIR filter that occupied 90 % logic resource yields a 59.24 % compression ratio, while the all pass filter yields a 2.29 % compression ratio. The reason lies in that the all pass filter only occupied less than 5 % logic resource. Under most conditions, the logic utilization is less than 90 %, a compression ratio under 60 % could be obtained.

TABLE I. PERFORMANCE OF SRLBC.

Configuration	Original bitstream size(bytes)	Compressed bitstream size(bytes)	Compression ratio
Adder	45445	5548	12.2 %
Multiplier	45445	4490	9.88 %
FIR-46 tap (all pass)	285906	6539	2.29 %
FIR-46 tap (low pass)	285906	169357	59.24 %

A BiRF [19] proposed by Corbetta is employed to carry out bitstream relocation at runtime. The BiRF consists of a logic unit, a CRC calculator and a FSM. The FSM is used to identify the frame address register (FAR) and CRC commands in the bitstream, in order to modify the corresponding parameters.

The CRC is used to check the data integrity of the bitstream. Although no device damage has ever been reported for loading a faulty PB, it is not a certainty that damage could never occur, nor that the system that the FPGA device is controlling would not be damaged [24]. If the correctness of bitstream can be guaranteed, the CRC could be bypassed. Under such circumstance, a default CRC value 0xDEFC is used.

C. Relocatable Partition

Relocatable partition means that although the two partitions implemented different logic functions, they have same partition pins and relative routings between proxy logic and static region. In other word, the two partitions have uniform configuration and can be relocated at runtime. In

general, the two relocatable partitions should comply with several disciplines [21] shown below:

– 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 PRR, which does not through proxy logic

To achieve the aforementioned five disciplines, some constraints and design consideration should be followed. The details of how to create relocatable partitions were discussed in the following subsection.

D. Creating the Proposed IES

Figure 5 depicts the software work flow of creating a partial reconfiguration design. This work flow has integrated a partition-based flow adapted from the standard Xilinx flow. The steps containing in the solid box are standard Xilinx flow, and the Step 4 enclosed in the dashed box is an additional flow. The additional flow is used to create uniform relocatable partitions, and special attention is focused on it. The steps required to build the proposed intrinsic evolvable system are described below.

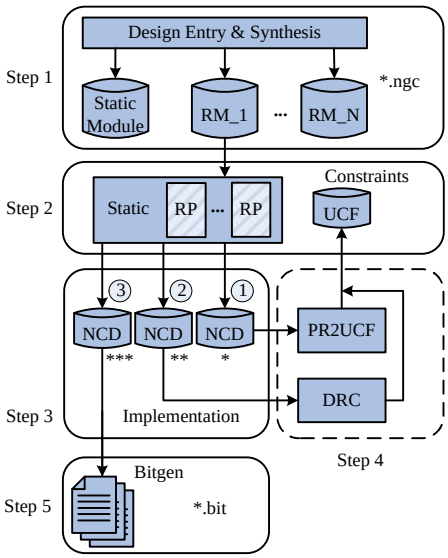

Fig. 5. Partial reconfiguration design flow.

Step 1 Design entry and synthesis. A PowerPC system corresponded to static module was created using Xilinx EDK. In addition to the basic elements of the PowerPC based embedded system, three peripheral IPs are added, including MPMC, System ACE and custom ICAP. After the hardware platform is created, a software project was created using Xilinx SDK tool. The PRPs were described in hardware description language (HDL) and synthesized using XST tool.

Step 2 Create constraints. A reconfigurable design was created using PlanAhead. The reconfigurable design is composed of static module and reconfigurable modules. The shape and size of reconfigurable partition are set by GUI tool or user constraints. Several reconfigurable partitions named PRP_1 to PRP_N were created. Those PRPs are same in shape and size.

Step 3 Run the PAR flow. After the reconfigurable design was created, the placement and routing (PAR) flow was invoked to implement the design. In the standard flow, the routed design was ready for bitstream generation. However, in our design, for the purpose of creating relocatable partitions, an additional step was employed.

Step 4 Create relocatable partitions. This step is responsible for creating relocatable partitions. As shown in Fig. 5, after the first run of the PAR flow completed, a NCD* file was generated. Although the PRPs created in Step 2 both have same shape and size, the proxy logics and routings may be different. In order to make these PRPs relocatable, the proxy logics and routing of them must be uniformed. A PRP was selected as baseline design, and other PRPs must be compatible with it. The detailed steps are shown below:

1. Use "pr2ucf -bel -o partition_pins_bel.ucf config_1.ncd" to extract the placement information of the PRPs. Add the placement constraints to the UCF file and re-run the PAR flow. After the PAR flow, the NCD** file was generated.
2. Open the generated NCD** file using FPGA Editor, and extract the routing information using directed routing constraints (DRC) command. A PRP was selected as the baseline design. To uniform the routings, constraints of the baseline PRP can be applied to other PRPs by modifying the offset of X coordinate. Add the routing constraints to the UCF file and re-run the PAR flow. After the PAR flow, the NCD*** file was generated.

Finally, no routings cross the PRR without the proxy logic should be guaranteed. The routings disobey the rules should be rerouted manually.

Step 5 Generate bitstream.

The full bitstream and partial bitstream is generated using Bitgen tool. In the case of default CRC was used, the –g CRC: Disable option is required. For the purpose of booting from CF card, a system.ace containing the C program was needed. The partial bitstream is also stored on the CF card. DPR was performed by loading the partial bitstream from the CF card to the ICAP at runtime.

IV. EXPERIMENTS AND RESULTS

An adaptive finite impulse response (FIR) filter was employed to demonstrate the effectiveness of our proposed evolvable system. In the EHW system, the adaptive feature includes parameter adaptation and structural adaptation. In the case of adaptive FIR filter, parameter adaptation refers to coefficient modification, and structural adaptation consists in topology modifications, for instance, the order or type of filters. The coefficient could be reloaded by modifying the coefficient memory through the write operation of the PLB bus. The structural adaptation is implemented by modifying the filter order dynamically. The architecture of adaptive FIR filter was shown in Fig. 6.

The basic module is an 8-tap FIR filter, which is composed of F00, F01, F02 and F03. The h[15:0] are filter coefficients, and FO[3:0] control the filter order. For example, if FO[3:0] = 0001, it means that an 8-tap FIR filter is needed. In this case, the M03 is the last level, so the MUX selects the DFF output. Under other conditions, the M03 is not the last level, so the MUX selects the adder output.

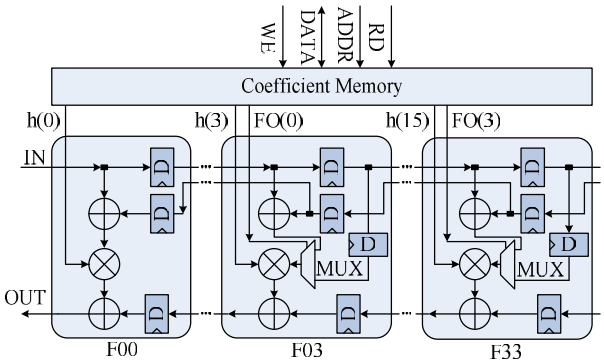

Fig. 6. Architecture of adaptive FIR filter.

The compact genetic algorithm (CGA) is employed to control the adaptive FIR filter. Unlike the simple genetic algorithm (SGA) [25], [26], the CGA represents populations of candidate solutions as probability vectors (PV) rather than as sets of bit strings [27]. By doing so, significant memory saving can be realized, which make it very suitable for embedded systems. The parameters and structure including filter order and filter coefficients are encoded as chromosome. The chromosome structure is shown in Fig. 7.

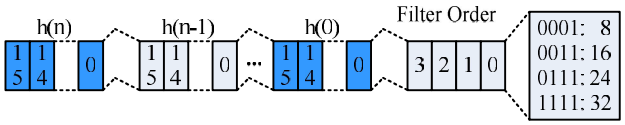

Fig. 7. Chromosome structure of CGA.

The chromosome is composed of filter order and filter coefficients, the four LSBs indicate the filter order, and the filter coefficients are represented using 16 bits two's complement. Owing to the symmetric feature of FIR filter, for a N-tap filter (N is even), only N/2 coefficients were needed. So, the total length of the chromosome is 16 × 16 + 4 = 260 bits.

The parameters of CGA are as follows: population size is 20, mutation rate is 0.05. The genetic operators include reproduction and mutation, no crossover was used. In order to protect the optimal individual of each generation, an elitism based reproduction strategy was employed to ensure that the currently best candidate remains in the next generation.

Figure 8 shows the work flow of adaptive FIR filter system. After power on, the full bitstream was loaded to the FPGA from the CF card, and an initial population was randomly generated. Due to the decoded chromosome, the filters were configured. Then the input data was processed to get the fitness value. The genetic operation including mutation and reproduction was performed to generate a new population. The above cycle of filter configuration, data processing, fitness evaluation and genetic operation is repeated in every generation until the solution is found. If the expected solution was found, the CGA exits the optimization cycle and outputs the best individual.

The experimental system was based on a Xilinx ML507 board containing a XC5VFX70T-1-FF1136. The Xilinx ISE Design Suite 13.2 System Edition with an additional PR License was used to implement the whole evolvable system. The implementation results of adaptive FIR filter system were shown in Fig. 9.

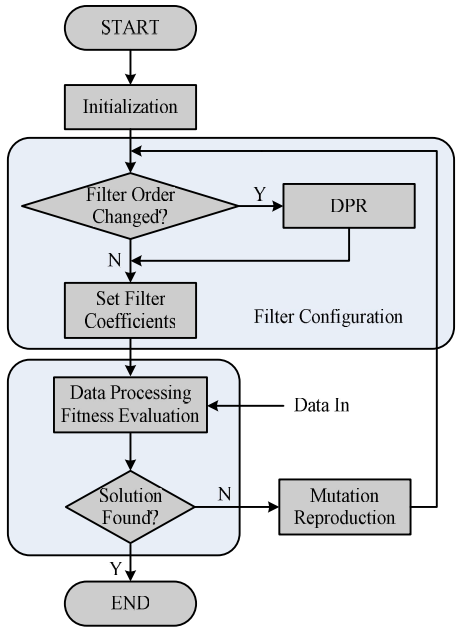

Fig. 8. Workflow of adaptive FIR filter system.

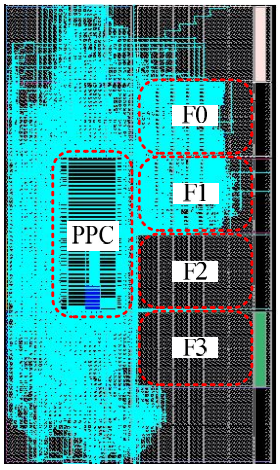

Fig. 9. Implementation results of adaptive FIR filter system.

As shown in Fig. 12, a 16-tap filter was implemented, only F0 and F1 were involved, F2 and F3 were leaving unused.

The electrocardiogram (ECG) data from MIT [28] was used to train our adaptive FIR filter. The sampling rate is 200 Hz. The fitness function is defined as follows:

$$fitness = \frac{1}{\sum_{i=1}^{N} \left| filter\_output_i - ref_i \right|}, \quad (1)$$

where *filter_output* is the filtered result, *ref* is the reference signal and N is the sample length of the input data. The reference signal may be uncorrupted signal or pre-computed output according to the filter specification. For the simplicity, we chose uncorrupted signal as the reference signal here.

The convergence criteria are defined as follows

$$\left|PV_i\right| \le 0.05 \quad or \quad \left|PV_i - 1\right| \le 0.05. \quad (2)$$

When the PV converged to 0 or 1, an optimal solution was found. The filtering result of the evolved FIR filter was shown in Fig. 10.

As shown in Fig. 10, the upper figure shows the ECG signal corrupted by a 50 Hz noise, and the lower figure shows the filtered result. The filtering result shows that the noise was removed effectively.

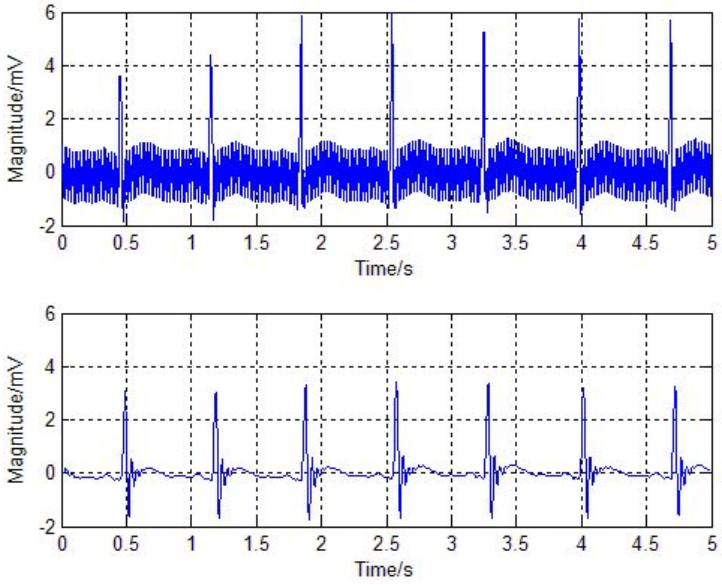

Fig. 10. Filtering result of evolved FIR filter.

TABLE II. PERFORMANCE OF SRLBC.

Method	Original bitstream size (bytes)	Compressed bitstream size (bytes)	Configuration time (ms)
proposed	84244	44011	0.71
[18]	360800	-	350

The bitstream size is not provided in [18], in order to make a fair comparison, the size is indispensable. Fortunately, as referred in [18], a FIR macro occupied four clock regions, so the PB size could be estimated using PlanAhead. As can be seen from Table II, our method achieved a speedup of approximately 116 × over the method proposed in [18]. In the case of FIR macro [18], three PBs are needed for three partitions. By using the bitstream relocation technique, although our design contains four partitions, only one PB is needed. Therefore, great configuration memory saving is achieved. Further, configuration memory overhead can be reduced by bitstream compression. For a FIR system containing 4 partitions, the PB size of each partition is 8.4 KB, the total configuration memory requirement is 8.4 × 4 = 33.6 KB. While in our system, the configuration memory requirement is only 8.4 × 0.6 = 5.04 KB, where 0.6 is the compression ratio. In compare to conventional method, 85 % configuration memory saving can be achieved.

V. CONCLUSIONS

In this paper, an intrinsic evolvable system on dynamic partial reconfigurable platform using bitstream relocation technique was proposed. The efficiency of the proposed method is illustrated by the low configuration memory overhead and high reconfiguration speed. The effectiveness of the proposed method is evident from the experimental results obtained for the adaptive FIR filter application. Moreover, the bitstream relocation feature eliminates the need of creating specific PB for each PRP in conventional methods, which can alleviate the time-consuming job. Although the tool control language (TCL) scripts provided by Xilinx can automate the design flow to some extent, human interventions are still required to create the PRPs in current flow. Future works may be focus on developing custom software to automate this flow.

REFERENCES

[1] P. C. Haddow, A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future", *Genet Program Evolvable Mach.*, vol. 12, pp. 183–215, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10710-011-9141-6

[2] R. F. Demara, K. Zhang, C. A. Sharma, "Autonomic fault-handling and refurbishment using throughput-driven assessment", *Applied Soft Computing*, vol. 11, pp. 1588–1599, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2010.01.020

[3] P. Kaufmann *et al.*, "Classification of electromyographic signals-comparing evolvable hardware to conventional classifiers", *IEEE Trans. Evolutionary Computation*, vol. 17, pp. 46–63, 2013. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2012.2185845

[4] G. He *et al.*, "Evolvable hardware design based on a novel simulated annealing in an embedded system", *Concurrency Computat. Pract. Exper.*, vol. 24, pp. 352–368, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.1604

[5] S. Kim *et al.*, "A hierarchical self-repairing architecture for fast fault recovery of digital systems inspired from paralogous gene regulatory circuits", *IEEE Trans. VLSI. Syst.*, vol. 20, pp. 2315–2328, 2012.

[6] J. Q. Xu, Y. Dou, Q. Lv, "A bio-inspired fault-tolerant hardware system supporting hierarchical self-healing", *Elektronika ir Elektrotechnika*, vol. 18, pp. 103–106, 2012.

[7] X. Yao, T. Higuchi, "Promises and challenges of evolvable hardware", *IEEE Trans. Syst. Man. Cybern.*, vol. 29, pp. 87–97, 1999. [Online]. Available: http://dx.doi.org/10.1109/5326.740672

[8] J. Wang, Q. S. Chen, C. H. Lee, "Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware", *IET Comput. Digit. Tech.*, vol. 2, pp. 386–400, 2008. [Online]. Available: http://dx.doi.org/10.1049/iet-cdt:20070124

[9] T. Higuchi *et al.*, "Evolving hardware with genetic learning: A first Step toward building a Darwin machine," in *Proc. 2rd Int. Conf. Simulation of Adaptive Behavior*, New York, 1992, pp. 417–424.

[10] B. I. Hounsell, T. Arslan, R. Thomson, "Evolutionary design and adaptation of high performance digital filters within an embedded reconfigurable fault tolerant hardware platform", *Soft Computing*, vol. 8, pp. 307–317, 2004. [Online]. Available: http://dx.doi.org/10.1007/s00500-003-0287-x

[11] I. Harvey, A. Thompson, "Through the Labyrinth evolution finds a way: a silicon ridge", in *Int. Conf. on Evolvable Systems*, Tsukuba, 1996, pp. 406–422.

[12] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware", in *Int. Conf. on Evolvable Systems*, Trondheim, 2003, pp. 186–197.

[13] A. Otero *et al*., "A fast reconfigurable 2D HW core architecture on FPGAs for evolvable self-adaptive systems", *NASA/ESA Conf. on Adaptive Hardware and Systems*, California, 2011, pp. 337–343.

[14] P. Layzell, "Reducing hardware evolution's dependency on FPGAs", *Int. Conf. on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, Granada, 1999, pp. 171–178. [Online]. Available: http://dx.doi.org/10.1109/MN.1999.758861

[15] A. J. Greensted, A. M. Tyrrell, "RISA: a hardware platform for evolutionary design", *IEEE Workshop on Evolvable and Adaptive Hardware*, Honolulu, 2007, pp. 1–7.

[16] W. Barker *et al*., "Fault tolerance using dynamic reconfiguration on the POEtic tissue", *IEEE Trans. Evolutionary Computation*, vol. 11, pp. 666–684, 2007. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2007.896690

[17] A. Upegui, E. Sanchez, "Evolving hardware by dynamically reconfiguring Xilinx FPGAs", *Int. Conf. on Evolvable Systems*, Sitges, 2005, pp. 56–65.

[18] C. S. Choi, H. Lee, "A self-reconfigurable adaptive FIR filter system on partial reconfiguration platform", *IEICE Trans. Inf. & Syst*., vol. E90-D, pp. 1932–1938, 2007.

[19] S. Corbetta *et al*., "Internal and external bitstream relocation for partial dynamic reconfiguration", *IEEE Trans. VLSI. Syst*. vol. 17, pp. 1650–1654, 2009.

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

[21] Y. Ichinomiya *et al*., "A bitstream relocation technique to improve flexibility of partial reconfiguration", *Int. Conf. on Algorithms and Architectures for Parallel Processing*, Fukuoka, 2012, pp. 139–152. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33078-0_11

[22] M. Koester *et al*., "Design optimizations for tiled partially reconfigurable systems", *IEEE Trans. VLSI. Syst*. vol. 19, pp. 1048–1061, 2011.

[23] S. Hauck, W. D. Wilson, "Runlength compression techniques for FPGA configurations", *IEEE Symposium on FPGAs for Custom Computing Machines*, California, 1999, pp. 286–287.

[24] PRC/EPRC: Data integrity and security controller for partial reconfiguration, xapp887.pdf Xilinx., 2012. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp887_PRC_EPRC.pdf

[25] E. Kose, *et al*., "Sliding mode control based on genetic algorithm for WSCC systems include of SVC", *Elektronika ir Elektrotechnika*, vol. 19, pp. 19–24, 2013.

[26] A. Sirbu, *et al*., "Improved genetic algorithm for the bandwidth maximization in TDMA-based mobile Ad Hoc networks", *Elektronika ir Elektrotechnika*, vol. 19, pp. 104–109, 2013.

[27] J. C. Gallagher, S. Vigraham, G. Kramer, "A family of compact genetic algorithms for intrinsic evolvable hardware", *IEEE Trans. Evolutionary Computation*, vol. 8, pp. 111–126, 2004. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2003.820662

[28] A. L. Goldberger *et al*., "Components of a new research resource for complex physiologic signals", Circulation 101(23):e215-e220 [Online]. Available: http://circ.ahajournals.org/cgi/cont-ent/full/101/23/e215]