A Functionality-Based Runtime Relocation System for Circuits on Heterogeneous FPGAs

A Functionality-Based Runtime Relocation System for Circuits on Heterogeneous FPGAs

Godwin Enemali, Adewale Adetomi, and Tughrul Arslan, *Senior Member, IEEE*

***Abstract*—Runtime Relocation of circuits on FPGAs have been proposed for achieving many desirable features including fault tolerance, defragmentation and system load balancing. However, the changes in the architectural composition of FPGAs has made relocation more challenging mainly because FPGAs have become more heterogeneous. Previous and state-of-the-art circuit relocation systems on FPGAs have relied only on direct bitstream relocation which requires the source and destination resource layouts to be the same, as well as access to the design bitstream for manipulation. Hence, their efficiency on modern heterogeneous chips greatly reduces, and mostly cannot be applied to encrypted bitstreams of intellectual property (IP) blocks. In this paper, we present a circuit relocator which augments direct bitstream relocation with a functionality-based relocation scheme. We demonstrate the feasibility of the proposed technique using a CORDIC application and show that an average of over 2.6-fold increase in number of relocations can be obtained compared to only direct bitstream relocation at expense of a small memory overhead and manageable relocation time for this case study.**

***Index Terms*—Bitstream Relocation, FPGA, Reconfigurable Hardware, Relocation, Look-up-table, Heterogeneous.**

I. Introduction And Related Works

Field Programmable Gate Arrays (FPGAs) have gained increasing popularity in many domains, including consumer electronics, aerospace and defense, scientific instruments, autonomous vehicles and various video processing applications [1] [2]. In embedded systems, they have continued to be popular because of their unique combination of performance and flexibility. One very key feature of modern FPGAs is that circuits (or a subset of circuit(s)) configured on them could be removed when not needed to make room for other circuits or to modify the functionality of the system, a feature called dynamic partial reconfiguration (DPR). In addition, circuits could be moved from one location to another on the FPGA, a process termed relocation. Relocation of circuits on FPGA chips is beneficial for many reasons. Three important ones are: to circumvent permanent damages on chips and consequently improve fault tolerance of critical applications in hostile environments such as space [3], to achieve defragmentation of the chip area [4] and to maintain a desired thermal distribution on the chip [5]. Reconfigurable Operating Systems (ROS) have been proposed to manage relocation and other activities on FPGA chips in runtime, especially in embedded systems which are often self-contained. An integral part of an ROS is a relocation manager which coordinates the repositioning of circuits on the chip.

A major condition that needs to be satisfied for relocation of a circuit to be possible in runtime is that the resource composition of the original location for which the circuit was synthesized should be the same as the intended destination location. That is, the source and destination are required to have identical chip area, not only in size, but also in the type, number and order of the resources they contain. This condition was easily satisfiable in older versions of FPGAs which were essentially homogeneous. Modern FPGA chips, in a bid to improve performance and lower power consumption, have hard blocks such as memory blocks (BRAMs) and digital signal processors (DSPs) sandwiched between the conventional Configurable Logic Blocks (CLBs) [6]. In addition, these BRAMs, DSPs and CLBs sometimes have different orientations (left and right) which differ in routing types as in the Xilinx 7 series FPGAs. Thus, FPGAs have become increasingly heterogeneous, and hence places greater restrictions on the relocation of circuits. The result of this increase in heterogeneity is that the number of relocations possible for typical circuits has reduced with newer generations of FPGAs. Figure 1 illustrates this point. The figure shows that while on a homogeneous chip the circuit on LOC 1 could be relocated to 2 additional identical locations (LOC 2 and LOC 3), no identical location can be found on the heterogeneous chip.

Most of the circuit relocation systems present in the literature still demand that identical location(s) must exist on chip before any form of circuit relocation is possible [5] [7] [8] [9] [10]. They address only direct bitstream placement and relocation and do not provide any means of relocating circuits to non-identical locations. In [11], the authors reported a technique that successfully relocated a design bitstream synthesized for a location containing a set of CLBs and an unused DSP to another location with same CLB layout but with an unused BRAM replacing the DSP. However, the technique is based on online editing of configuration bitstreams which could be time consuming. In addition, the routing between the DSP and BRAM are required to be identical, which is not the case in recent FPGA architectures like the Xilinx 7 series. In an earlier work [10], we presented a scheme addressing placement and relocation of circuits on heterogeneous chips using only direct bitstream placement. That work focused on optimizations to improve place-ability of circuits in run-time by carefully selecting synthesis locations at design time and minimizing fragmentation in runtime. It does not present any means of dealing with circuit placement on non-identical locations, and thus is completely different from the focus of this paper. Hence, we consider the work presented in this paper as novel, as we are not aware of any other report in the literature, including any of our own previous works, on circuit relocation on FPGAs based on functionality.

In this paper, we propose a relocation manager to improve the number of relocations for a circuit on heterogeneous FPGAs. The proposed relocator augments bitstream relocation with a functionality-based relocation. The functionality-based relocation presented in this work relies on the technique of replicating the functionality of a circuit with a look-up-table or a memory block in runtime for selected circuits. The basic idea is shown in figure 2.

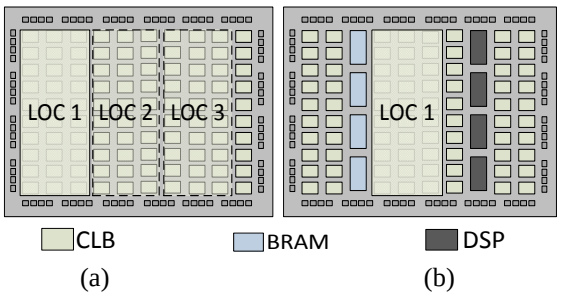


Fig. 1. Number of relocations on homogeneous and heterogeneous FPGAs
a) Upto 2 relocations are possible b) No relocation is possible

Manuscript received February 27, 2018; revised April 6, 2018. Accepted April 7, 2018. Date of current version April 8, 2018. This brief was recommended by Deputy Editor-in-Chief Jose de la Rosa *(Corresponding Author: Godwin Enemali)*

G. Enemali, A. Adetomi, and T. Arslan are with the School of Engineering, University of Edinburgh, Edinburgh, United Kingdom, (e-mail: g.enemali@ed.ac.uk)

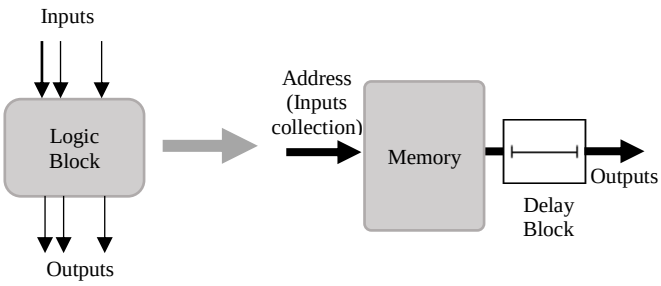


Fig. 2. Transformation of logic block to memory block

In this paper, the main distinct contributions are:

- A technique of relocation for relocating circuits to a location with non-identical resources based on functionality.
- Implementation of runtime circuit relocator, combining bitstream relocation and the proposed functionality-based relocator, with a practical case study.

The rest of this paper is organized as follows: section II discusses the proposed system architecture and presents the details of the relocation flow proposed. Implementation details of the proposed design is presented in section III, in addition to a case study application. Experimental results are presented and discussed in section IV, and section V gives a conclusion.

II. SYETEM OPERATION AND ARCHTECTURE

Our complete relocation flow occurs in two stages: first the possibility of relocating a circuit by direct bitstream is evaluated and in cases when this is not possible or leads to undesirable effects, functionality-based relocation is resorted to. The proposed functionality-based relocation is done when an exact matching position for the circuit's original bitstream is either not available, or would lead to undesirable effects, such as increased fragmentation of the chip area. Our flow for relocating circuits by direct bitstream can be found in [10]. The following description will focus on the functionality-based relocation aspect. A circuit to be relocated using this technique has its computation results memorized during its normal operation. A bitstream of an LUT or memory resource template is pre-synthesized and stored in an off-chip memory at design time. When relocation is required in runtime, a destination location is configured with the bitstream template, and its memory content filled with the outputs of the original circuit previously memorized.

The operational flow of the relocation mechanism is shown in figure 3. When a request is received to relocate a circuit (after attempts to find an exact location for the original bitstream on the chip is found to be infeasible or unprofitable), a duration evaluator carries out a check to see if the timing constraints associated with the relocation request can be met.

Next, an area check is done to find a suitable location for a pre-synthesized memory template. The details of the time required for a relocation procedure is given in section II.B below while Section II.C explains the procedure for the area check. If both checks are successful, then the relocation request is accepted and executed in 3 additional steps:

i) The outputs of the circuit not present in memory are computed and saved
ii) A memory template is configured on the chip
iii) Data is copied from the original circuit's memory unto the already configured template.

These operations are managed by various units of a relocation module discussed below. The architectural composition of the proposed relocation module consists of an Output Memorizer, Duration Evaluator, and Area Finder.

A. *Output Memorizer*

The output memorizer basically saves the results of computations of selected circuits in memory in runtime. Thus, it connects to the circuits whose outputs it memorizes. It has 3 units: task memory, evaluation logic (which we shall memo logic) and output memory. These are shown in figure 4. The task memory saves the list of circuits which are currently configured on the FPGA chip and are potentially relocatable by functionality. The memo logic manages the conversion of the raw inputs to address values, determines if the output for an input has been previously saved and switches mode to save the current output of the application when it has not been saved previously. Each circuit has a unique identifier (Circuit ID) which corresponds to its address in the task memory (Base_Addr). The memo-logic has a fixed 3 clock cycle overhead when operating in CHECK mode where it verifies if an input has been previously saved, and an overhead of 2 clock cycles when in SAVE mode where it saves an output unto its output memory if not already saved. Basically, the fixed number of clock cycles is achieved by concatenating the inputs of a circuit into a unique address value (Base_Addr + offset), with Base_Addr being the start of the memory location assigned to the circuit and offset determined using information on the circuit's input and tolerance. Hence, our proposed technique is based on memory space reservation (since each input translates to a unique address) rather than a greedy search procedure, where a series of values from memory are compared against the current input. In CHECK mode, the memo-logic operates in parallel with the operation of the original circuit, and thus does not add any pre-processing overhead to circuits which take at least 3 clock cycles for their normal operations. In SAVE mode (executed only when an input has not been previously saved in memory), 2 post-processing clock cycles are needed.

The output memory contains the results of computations. An application with multiple outputs have these outputs concatenated and saved at an address. The least significant bit (LSB) of each output memory location is reserved to be checked for validity of the value stored at that address as shown in figure 5. This bit is checked to determine if results of a computation are available in memory or not. A value of '1' at that location indicates that a previous value has been saved and is valid and a '0' means that valid output is missing for this input and the original circuit would have to compute it.

To compute missing outputs in runtime after a request to relocate a circuit is received, the memo logic iterates through the LSBs of the section of its own output memory dedicated to memorizing the circuit's outputs. The LSB of a missing output has a value of '0'. The address indices (which corresponds to inputs) of missing outputs are then each decoupled and fed into the original circuit as inputs for it to compute corresponding outputs. It is worth re-stating that the LSBs of the output memory are used to keep track of valid outputs. This is because in reconfigurable computing, the functionality of a circuit could be changed in runtime, for example, when a part of that circuit is reconfigured with a different functionality in runtime using DPR. Under such conditions, the memo logic refreshes previously computed outputs by resetting the LSBs of the output locations to '0'.

The sizes of the task and output memories of the Output memorizer are determined by the number of relocatable circuits on the chip, the sum of the number of inputs of the constituent circuits and the *tolerance* of the circuits. By tolerance, we mean permissible variation in a circuit's outputs. Since this technique requires that space be reserved for all potential outputs, its memory overhead could be a major bottleneck for large port width applications that require numerous distinct outputs to be saved. Hence, we acknowledge that to keep the memory requirement reasonable, the port width of the circuits which can be relocated using this mechanism must be small, or if the port width is large, then the application tolerance must be large as well. Moreover, the functionality-based relocation proposed in this work is only applicable to circuits which are referentially transparent – that is, circuits implementing systems that *produce the same set of outputs for the same set of inputs*.

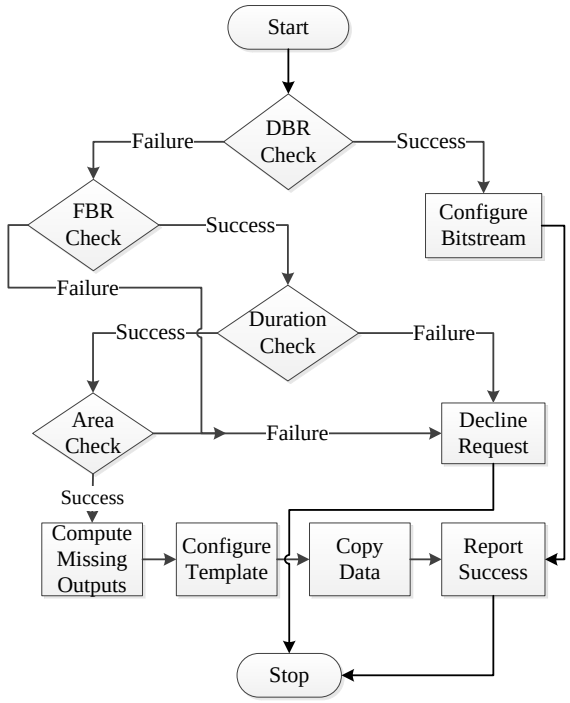


Fig. 3. Operational flow of proposed functionlaity based relocation system

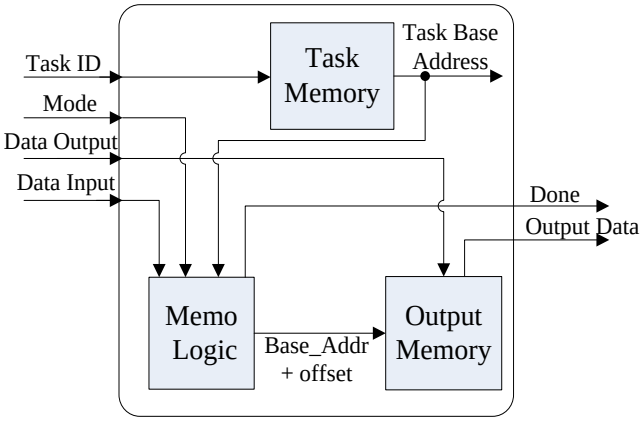


Fig. 4. Architectural overview of the output memorizer

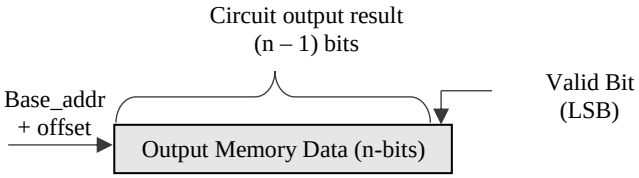


Fig. 5. Data distribution in output memory of output memorizer

Circuits whose current outputs depend on some internal states, or are determined by factors other than the current input(s) are not directly relocatable by the technique proposed in this work. Nevertheless, there are many applications which can profit from the proposed scheme even with these limitations. Three Examples are: an RGB to YCrCb colour conversion circuit which is widely used in computer graphics, CORDIC circuits designed to compute the trigonometry of angular inputs, and multiplier circuits which form basis for many other applications.

B. *Duration Evaluator*

This unit checks if the requested relocation can be completed within the time constraint associated with the request. Its architecture consists of an LUT RAM which contains the essential parameters of the circuits, including the duration associated with the circuit's operations such as configuration time, number of clock cycles for computation of outputs (e) as well as duration of data transfer from the output memorizer's memory to a memory template. The time constraint of a relocation request is evaluated using (1). The term R_t in (1) is the total time required for relocation, C_t is the greater of the time required for the memory template to be configured on the chip and the area finder module to execute; and e is the time required to compute a missing output of the circuit(s) to be relocated, with n being the number of the missing (yet to be saved) outputs. M_t is time required for the memorized memory content to be transferred to the template. It is worth noting that the operation of the area finder and the computation of the missing outputs of a circuit to be relocated are done in parallel, thus C_t in (1) takes the value of the greater of time required to complete these two operations. e is initially measured at design time like the configuration duration of the circuit. However, since e depends on the architecture and functionality of a circuit, when these are changed by DPR, its new value is measured (by observing the duration required by the updated circuit to change a set of inputs into outputs) and updated in runtime.

$$R_t = \sum_{i=0}^{n} e_i + C_t + M_t \quad (1)$$

C. *Area Finder*

The area evaluator basically checks if there is an area on the chip for a template to be placed on. It has access to a RAM containing the state of the chip (State Memory), as well as a memory containing all the potential locations of the template. The State Memory represents the state of all resources on the chip by an M x N Matrix, where M and N are respectively the number of rows and column in the device. An available resource is represented by a '0' and a used or damaged resource by a '1'. Thus, each element in the matrix define the state of a specific reconfigurable resource on the chip. A scan function is used to check the availability of potential locations for the circuit in the light of the current state of the chip. Further details of the scan procedure can be seen in [10].

Finally, it is worth stating that the memory template consists of a generic memory block capable of holding all potentially required output data of the circuit(s) it is designed to replace. It also contain associated logic to manage functionalities such as memory read and delay management. Its memory size is determined like the output memory of the output memorizer discussed in section II.A. The delay management block manages the difference between the timing behavior of the memory template and the original circuit so as to maintain the timing characteristics of the entire system. It does this by delaying the asserting of 'done' by the difference in the number of clock cycles between the operation of the memory template and that of the original circuit.

III. Implementation Details

A. *Case Study Application*

We implemented a CORDIC application using Xilinx IP blocks. The application consists of 3 independent circuits: Square Root, Sin/Cos trigonometric operations and the hyperbolic tangent (Tanh) computing circuits. CORDIC was chosen as it is an important algorithm for various mathematical functions [12]. Details of the circuits' operations as well as their data format can be found in [13]. We created a custom wrapper for the circuits to make it compatible with our relocation model. Each circuit was optimized to take an 8-bit $DataIn$ and produce an 8-bit $DataOut$ and ap_done signal. We synthesized the application and its components, along with a top wrapper module using Xilinx Vivado suite for the Xilinx xc7a35tcpg236-1 FPGA chip. The top wrapper includes a $TaskId$ signal that is used to select a particular circuit. Table I shows the resource utilization of the circuits, while table II shows the number of clock cycles for each operation. The partial bitstream of the application is 140kB in size.

B. *Relocation Module*

The relocation module comprising of an output memorizer, duration checker and area finder described in section II was implemented using Xilinx Vivado 15.1 design tools. Its resource utilization is shown in table III. A total of 66 LUTs, 58 Flip flops and a single 18kb BRAM were used on the xc7a35tcpg236-1 chip. It is

worth noting that the size of memory used is dependent on the application. We chose an 18kb memory because it is sufficient to save all the outputs of our target case study application. The relocation module connects to the inputs and outputs of application(s) to memorize new computations by the application. It is also worth noting that practical relocation techniques require access to the configuration memory of the FPGA, as well as a means of communication between a relocated module and other parts of the chip. Thus, a self-reconfiguration controller [14] with the required access to the configuration memory was instantiated. The controller is used for configuring the chip, as well as copying of data between block memories of the relocation module and the relocated module via the configuration layer. To address the need of a communication technique that supports relocation, we adopted the technique described in [15] which makes use of those clock buffers not used by applications for on-chip communication. Our case study application used a single global clock buffer (BUFG) out of the 32 available on the xc7a35tcpg236-1 for clock network delivery. Thus, the remaining 12 horizontal clock buffers (BUFH) and 2 multi-regional clock buffers (BUFMR) per clock region present on the chip are available for on-chip communication without any conflict with our case study application or relocation management module. The technique is used to maintain communication between the relocated circuits and other circuits on the chip and/or the FPGA ports.

Next, a memory template for relocation was implemented. This template reserves 10kB of memory and manages the delay of the application it replaces. This memory size was determined by the maximum memory requirement of the circuits whose functionality it is intended to replace. The actual resource utilization of the memory template on the target FPGA is 14 LUTs, 21 Flip flops, and 18kb RAM and it has a 2-clock cycle delay. The bitstream size of template is 76.9kB. The delay mechanism is used to ensure that the relocated equivalent does not alter the timing of the relocated application so as not to lose synchronization with the entire system.

TABLE I.
RESOURSE UTILIZATION OF A CORDIC CIRCUIT CASE-STUDY APPLICATION

Circuits	LUTs	Memory LUTs	Flip Flops	BRAM
Square Root	71	1	100	-
Sine/ Cosine	277	4	307	-
Hyperbolic Tangent	1583	4	2218	-
Wrapper + All modules	2226	13	2920	-
Memo Block Template	14	14	21	1

TABLE II.
LATENCY OF CORDIC CIRCUIT CASE-STUDY APPLICATION

Circuits	Clock Cycles (e)
Square Root	15
Sine/ Cosine	19
Hyperbolic Tangent	56
Wrapper + All modules	NA
Memo Block Template	2

TABLE III.
RESOURSE UTILIZATION OF PROPOSED RELOCATION MODULE

Unit	LUT	FF	BRAM
Output Memorizer	10	11	1
Duration Checker	36	30	-
Area Finder	20	17	-
Total	66	58	1

IV. RESULTS AND DISCUSSION

At runtime, we initiated a relocation request when 50% of the outputs of the application have been saved by the output memorization module. The floor-plan of the application required 8 contiguous CLB columns on the xc7a35tcpg236-1 chip, which occur only once on that chip. Hence, only a functionality-based relocation was possible. The timing constraint associated with the relocation request was such that the relocation was required to take a maximum of 1ms. The total time duration for the relocation was measured as 306.80µs at 100MHz, with configuration of memory template taking 82.30µs, computation of missing outputs taking 175.36µs and copying of data from memorization unit memory to the template taking 49.14µs. We also measured the worst-case relocation duration for this module as 361.86µs and best case as 131.44µs. This was done by generating relocation requests when 0% (worst case) and 100% (best case) respectively of the outputs have been saved. The time required for configuration of the memory template and copying of data is constant for an application, irrespective of when a relocation request is received. Figure 6 shows floorplan of the original circuit and relocated equivalent.

We also observed the outputs of both the original circuit and the relocated equivalent for the same inputs. The results were the same for both circuits – in both cases, the value of $Data\_out$ *when* the ap_done signal goes high is the same. This is shown in figure 7. In addition, we evaluated the improvement in the number of possible relocations brought about by incorporating the proposed technique into the state-of-the-art direct bitstream relocation technique. Table IV shows the result for different Xilinx FPGA chips. As shown, the proposed technique leads to a significant improvement in the number of relocations. For the chips compared, an average of about 36 more relocations (an increase of over 260%) of the case study circuits could be obtained using the proposed technique. This is a great advantage in applications which aim to improve reliability by circumventing permanent damage on the chip such as in space applications. It means that augmenting the traditional bitstream relocation with the proposed functionality-based technique would significantly improve the fault tolerance of a design.

As already noted in section II and in the relocation case study used, our relocator resorts to a functionality-based relocation when the bitstream of the original design cannot be placed on a matching location on the chip, leads to undesired effects or where access to the location information of the bitstream is not possible (such as encrypted bitstreams). The technique is especially suitable on modern heterogeneous FPGA chips, such as the Xilinx 7 Series and Ultra-scale FPGAs, which are rich in memory resources, many of which are sometimes unused. We have also noted in section II that relocation by functionality is only applicable to circuits with low port width. This is due to its large memory overhead, which does not scale well with port width. To this end, it is important to re-state that the relocator system which we present is also capable of bitstream relocation for circuits which *cannot* be memorized.

In addition, we compared the time overhead of direct bitstream relocation and our proposed functionality-based relocation. Table V shows the relocation time for both techniques for 3 different circuits: CORDIC [13], RGB to YCrCb colour converter [16] and a multiplier circuit [17]. All the circuits were implemented using Xilinx IPs from Xilinx Vivado 15.1 for the xc7a35tcpg236-1 chip. As shown, functionality-based relocation technique has a larger time overhead than direct bitstream relocation for a majority (2 out of 3) of the cases. For example, direct bitstream relocation duration for a 12-bit RGB to YCrCb colour converter circuit would only require 174.46µs as against a minimum of 326.15µs required for the functionality-based technique. It is worth noting that the relocation time for the functionality-based technique is proportional to the port width of the circuit. Hence, for the CORDIC circuit with 10-bits inputs, its relocation time is smaller than direct bitstream relocation. With increase in port width, the relocation time for direct bitstream relocation has better performance. A major disadvantage of functionality-based relocation technique is that it does not scale well with increase in port width. In fact, the memory requirement doubles for each bit increase in port width. However, since

direct relocation is *impossible* in certain cases such as for encrypted bitstreams and when an identical location is not present on the chip, servicing relocation requests whose time constraint can be satisfied in those cases is always an advantage. Therefore, it is an added layer of advantage to relocate circuits by functionality whenever direct bitstream relocation is *impossible* or leads to undesired effects.

Finally, the size of the additional memory template bitstream required for functionality-based relocation is only 55% of that of the original circuits' in our case study. Hence, in terms of additional memory required, the functionality-based technique would be better compared to having to store multiple bitstreams of the original circuit, not to mention that since it is an empty memory template most of the bits in its bitstreams are '0's and would be much smaller when compressed compared to the original circuit's bitstream.

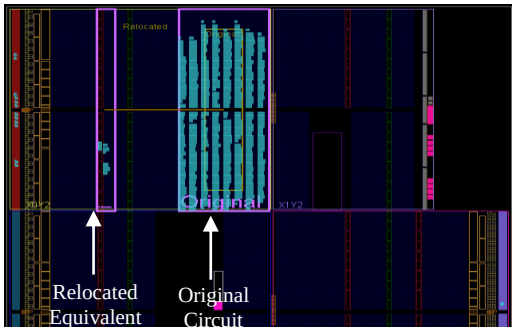
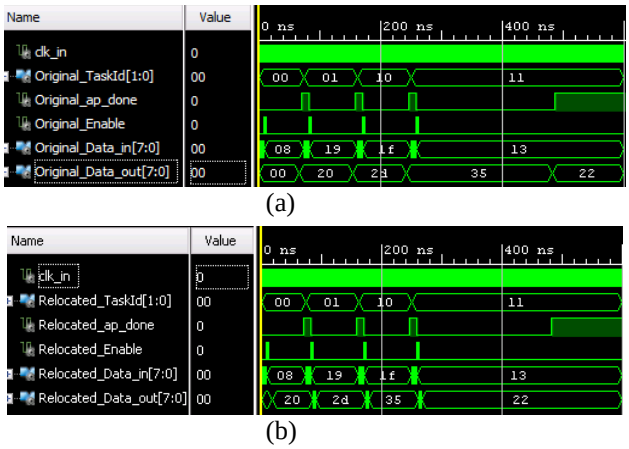


Fig. 6. Floor plan of the CORDIC application and its relocated equivalent circuit on xc7a35tcpg236-1



(a) (b)

Fig. 7. Output waveforms of Original and Funtionality-based relocated circuits. a) Original CORDIC circuit b) Relocated Equivalent

TABLE IV.
COMPARISON OF NUMBER OF POSSIBLE RELOCATIONS OF PROPOSED TECHNIQUE WITH STATE OF THE ART SCHEME

Target Chip	State-of-the-Art	Proposed
Artix-7 (xc7a35tcpg236-1)	1	8
Kintex-7(xc7k325tffg900c-2)	19	64
Virtex-7 (xc7vx485tffg1761c-2)	21	77
Total	41	149

TABLE V.
COMPARISON OF RELOCATION TIME OVERHEAD OF DIFFERENT RELOCATION TECHNIQUES

Circuit	Port Width	Direct Bitstream	Functionality-based (best case)	Functionality-based (worst case)
CORDIC	10	369.02	131.44	361.86
RGB to YCrCb	12	174.46	326.15	367.63
Multiplier	16	92.54	774.88	1430.26

V. CONCLUSION

In this paper, we have presented a runtime circuit relocation system which can relocate circuits on FPGAs based on functionality, in addition to the state-of-the-art direct bitstream relocation. The additional functionality-based relocation capability is based on replicating the functionality of the original circuit by memorizing previous computation results of circuits. The technique is applicable to applications that are referentially transparent and have low port widths. We have demonstrated its feasibility using a set of CORDIC circuits and shown that it has potentials to greatly increase the average number of relocations (over 2.6-times increase for our case study), while incurring only small additional bitstream storage overhead (only 55%) and manageable relocation time. In our future work we hope to explore the possibility of using compression and hashing mechanisms to extend the proposed technique to circuits with larger port width as well as examine test circuits with outputs dependent on internal states.

REFERENCES

[1] P. D. Possa, S. A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, 'A Multi-Resolution FPGA-Based Architecture for Real-Time Edge and Corner Detection', *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2376–2388, Oct. 2014.

[2] P. D. Possa, S. A. Mahmoudi, N. Harb, and C. Valderrama, 'A new self-adapting architecture for feature detection', in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 643–646.

[3] X. Iturbe, A Ebrahim, K. Benkrid, C. Hong, T. Arslan, J. Perez, D. Keymeulen, M. D. Santambrogio, 'R3TOS-Based autonomous fault-tolerant systems', IEEE Micro, vol. 34, no. 6, pp. 20–30, Nov. 2014.

[4] G. Enemali, A. Adetomi, and T. Arslan, 'A placement management circuit for efficient real-time hardware reuse on FPGAs targeting reliable autonomous systems', in Proc. 50th IEEE International Symposium on Circuit and Systems (ISCAS), 2017, pp. 1 – 4.

[5] C. Beckhoff, D. Koch, and J. Torresen, 'Portable module relocation and bitstream compression for Xilinx FPGAs', in proc. International Conf on Field Programmable Logic and Applications (FPL), 2014, pp. 1–8.

[6] I. Kuon and J. Rose, 'Measuring the gap between FPGAs and ASICs', IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 26, no. 2, pp. 203–215, Feb. 2007.

[7] J. A. Clemente, J. Resano, C. Gonzalez, and D. Mozos, 'A hardware implementation of a run-time scheduler for reconfigurable systems', IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 19, no. 7, pp. 1263–1276, Jul. 2011.

[8] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert, 'Design optimizations for tiled partially reconfigurable systems', IEEE Trans. Very Large Scale Integr. VLSI Syst., vol. 19, no. 6, pp. 1048–1061, Jun. 2011.

[9] X. Iturbe, Khaled Benkrid, C. Hong, A. Ebrahim, R. Torrego, I. Martinez, T. Arslan, and J. Perez, 'R3TOS: A novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs', IEEE Trans. Comput., vol. 62, no. 8, pp. 1542–1556, Aug. 2013.

[10] G. Enemali, A. Adetomi, and T. Arslan, 'Expanding the un-usable area strategy for improved utilization of reconfigurable FPGAs', in proc. 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2017, pp. 139–144.

[11] T. Becker, W. Luk, and P. Y. Cheung, 'Enhancing relocatability of partial bitstreams for run-time reconfiguration', in proc. 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM), 2007, pp. 35–44.

[12] H. T. Nguyen, X. T. Nguyen, and C. K. Pham, 'A Low-Power Hybrid Adaptive CORDIC', IEEE Trans. Circuits Syst. II Express Briefs, vol. 65, no. 4, pp. 496–500, Apr. 2018.

[13] Xilinx, 'CORDIC v6. 0 LogiCORE IP Product Guide'. [Online] Available: https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf [Accessed 29-Oct-2017].

[14] A. Adetomi, G. Enemali, and T. Arslan, 'A fault-tolerant ICAP controller with a selective-area soft error mitigation engine', in proc. 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2017, pp. 192-199.

[15] A. Adetomi, G. Enemali, and T. Arslan, 'Relocation-aware communication network for circuits on Xilinx FPGAs', in proc. 2017 International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1-7.

[16] Xilinx, 'RGB to YCrCb Color-Space Converter v7.1 LogiCORE IP Product Guide'. 2015. [Online] Available: https://www.xilinx.com/support/documentation/ip_documentation/v_rgb2ycrcb/v7_1/pg013_v_rgb2ycrcb.pdf [Accessed 20-Oct-2017].

[17] Xilinx, 'Multiplier v12.0 LogiCORE IP Product Guide'. 2015. [Online] Available: https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf [Accessed 29-Oct-2017].