Journal of King Saud University – Computer and Information Sciences 31 (2019) 514–527

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

On the comparison of memristor-transistor hybrid and transistor-only heterogeneous FPGAs☆

Umer Farooq [a,*], M. Hassan Aslam [b], Muhammad Usman [b]

[a] Department of Electrical and Computer Engineering, Dhofar University, Salalah, Oman
[b] COMSATS Institute of Information Technology, Lahore, Pakistan

ARTICLE INFO

Article history:
Received 23 September 2017
Revised 3 March 2018
Accepted 3 March 2018
Available online 6 March 2018

Keywords:
Heterogeneous FPGA
Memristor-transistor hybrid approach
Reconfigurable architectures
Exploration flow

ABSTRACT

Recently, memristor-CMOS based hybrid, homogeneous reconfigurable architectures have gained popularity as they offer better area results compared to their CMOS-only counterparts. But no work has been done on hybrid heterogeneous reconfigurable architectures (i.e. heterogeneous FPGAs). In this work, we design and simulate basic as well as complex memristor-transistor hybrid building blocks of heterogeneous island-style FPGA which are later combined together to construct complete hybrid heterogeneous FPGA architecture. Area comparison between hybrid and conventional building blocks of heterogeneous FPGA shows that hybrid blocks are 25–60% smaller compared to transistor-only blocks. For complete architecture level comparison, we propose an exploration flow which is built by combining several open source and indigenous tools. For experimentation and comparison between two FPGA architectures, two sets of open source heterogeneous benchmarks are used. These benchmarks are passed through our proposed flow to generate the area results for hybrid as well as conventional heterogeneous FPGA architectures. Area comparison results show that, on average, hybrid architecture requires 59% less area for two benchmark sets.

1. Introduction

The practically feasible field programmable gate arrays (FPGAs) were first introduced by Xilinx almost three decades ago. Due to the limited capability of associated design flow tools and large processing technology, FPGAs were homogeneous in nature as they supported only a single type of computation blocks. Furthermore, they had limited computation potential and they were used only as glue logic for different blocks of complex digital systems. However, they have come a long way since their inception and now they have a multi-billion-dollar market with ability to implement almost any complex digital system that requires huge computation power and very high performance. This is largely because of the rapid advancement in processing technology, which is further aided by optimizations and innovations in FPGA architectures as well as their improved design flows. Today, FPGAs are used in a whole range of complex digital systems like data centers, automobiles, compute-intensive digital signal processing (DSP) applications, and many more. For low to medium level production of digital systems, FPGAs are now seen as a viable 'avant-garde' to application specific integrated circuits (ASICs) because compared to ASICs, they are reconfigurable in nature with negligible non-recurring engineering (NRE) cost and very rapid time to market.

As stated above, FPGAs were initially homogeneous, generalized in nature and they could be used for implementation of almost any digital system. Along with that, they offer in principle infinite times the reconfiguration of the architecture. However, pre-fabricated and generalized reconfigurable nature of homogeneous FPGAs makes them much slower, larger, and immensely power hungry as compared to their standard cell-based counterparts (Kuon et al., 2008). As a consequence, they do not remain viable for performance critical applications albeit they offer smaller costs and shorter time to market compared to the ASICs. Thus, in order to make the FPGAs attractive for high performance applications, architectural modifications were made in them and they were made heterogeneous in nature (Luu et al., 2009). Contrary to

* Corresponding author at: Office No. 222 C, Department of Electrical Engineering, Dhofar University, Salalah, Oman.
E-mail address: ufarooq@du.edu.om (U. Farooq).

Peer review under responsibility of King Saud University.

☆ This document is a collaborative effort.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.03.004
1319-1578/

homogeneous FPGAs which contain only single type of generalized, configurable computation blocks (normally termed as configurable logic blocks (CLBs)), heterogeneous FPGAs are comprised of a mixture of configurable and specific-purpose hard blocks (HBs) like adders, multipliers, memory blocks, and even DSP blocks. Heterogeneous FPGAs have shown significant improvement over homogeneous FPGAs (Luu et al., 2009; Farooq et al., 2011; Underwood et al., 2004; Beauchamp et al., 2006) in terms of area, performance, power consumption and they are widely fabricated by commercial vendors as well. Although introduction of heterogeneity in FPGAs has helped in bridging the gap between FPGAs and ASICs, still FPGAs are much larger and slower compared to ASICs (Kuon et al., 2008) and other options are being investigated to make FPGAs more efficient.

With the advancement in processing technology gradually coming to a respite, researchers are looking to other components that could eventually replace metal oxide semiconductor (MOS) transistors (Strukov et al., 2008). Memristor is one such two terminal circuit component that could be used for the design of future digital systems. It was first discovered in 1971 by a missing relationship between flux linkage and charge (Chua, 1971) where it was established that known fundamental elements could not produce the behavior of memristor. It is a device whose resistance changes with magnitude and polarity of the voltage and contrary to the existing elements which are characterized by voltage and current, a memristor is characterized by flux and charge. Memristor also remembers its resistance until the voltage is re-applied (Williams, 2008). Despite promising characteristics, memristor remained largely unappreciated till 2008 when Hewlett-Packard (HP) demonstrated its practical feasibility in nano scale era (Williams, 2008). Since then, it has been shown that memristor excels transistor as a component in terms of area and power consumption (Sarwar et al., 2013). Moreover, it requires no modification in software for fabrication; hence can be built upon current fabrication technology (Vourkas et al., 2014). Memristor can either be used as a replacement of transistor or can also be used in conjunction with transistor for design of basic building blocks like logic gates. These blocks can then be used to construct larger blocks like memories, adders, multipliers which were previously constructed using transistors only (Williams, 2008). Usage of memristor in the construction of larger blocks can eventually lead to very complex circuits having better performance compared to the circuits that are based on transistors only.

In this work, we propose a novel hybrid heterogeneous FPGA architecture. The proposed architecture is based on memristor-transistor hybrid approach. Although some work regarding hybrid architectures exists for homogeneous FPGAs (Cong and Xiao, 2011; Farooq and Aslam, 2015), to the best of our knowledge, no prior work has been done on hybrid heterogeneous FPGA architectures. For the proposed heterogeneous FPGA architecture, we design and simulate basic building blocks of heterogeneous FPGA architecture using hybrid approach. These blocks include CLBs, programmable connection boxes (CBs), programmable switch boxes (SBs), as well as HBs like adders and multipliers. Main motivation behind this approach is to propose an architecture that is more efficient than transistor-only conventional architecture. Quality of an FPGA architecture is normally best assessed using accompanying design flow. For this purpose, we develop a generalized exploration flow (i.e. environment) in this work. The exploration environment is developed using a mixture of locally developed and state-of-the-art open source academic tools. The environment is unique in the sense that it can be used for exploration of conventional as well as proposed FPGA architecture. In order to validate the flow, we use two sets of large open source heterogeneous benchmarks which are placed and routed on the proposed architecture. Results obtained after placement and routing indicate that the proposed architecture far surpasses the conventional heterogeneous FPGA architecture. Further details on the proposed architecture, design of building blocks, and exploration environment are discussed in succeeding sections of the paper.

The main contributions of this work are summarized below:

- Design, simulation of basic building blocks as well as HBs of heterogeneous FPGA using memristor-transistor hybrid approach and their comparison with transistor-only blocks.
- Development of a generalized exploration environment that gives end-to-end experience. Environment is developed through a combination of state-of-the-art and locally developed academic tools.
- Congregation of two sets of large, open source, heterogeneous benchmarks and validation of proposed exploration environment through extensive experimentation.
- Comparison between proposed and transistor-only heterogeneous FPGA architecture using the results that are obtained through exploration environment.

Remainder of the paper is organized as follows. In Section 2, we give a comprehensive overview of related research in the domain of hybrid reconfigurable architectures. Section 3 then briefly describes related aspects of reference heterogeneous FPGA architecture. Section 4 details the design and simulation results of building blocks of reference heterogeneous FPGA architecture that we have designed using hybrid approach. Section 5 describes the generalized flow which is used for the exploration and experimentation of hybrid as well as conventional FPGA architecture. This section also sheds light on the heterogeneous benchmarks that we have used for experimentation. Experimental results are discussed in Section 6 where comprehensive comparison between two architectures is presented and finally this work is concluded in Section 7.

2. Background and related work

Since 1965, when Moore's law was first presented, metal oxide semiconductor (MOS) transistors have been shrunk down at an unprecedented rate. However, scaling the processing technology further has become really intimidating as it is very difficult to contain the leakage current and parasitic capacitance issues (Bi et al., 2012; Scaramuzzo, 2014). Researchers in recent years have scientifically probed many other components to find an alternative to MOS based processing technology and memristor is one such component which has shown many advantages such as non-volatility and no leakage current. The first known fabrication of memristor was accomplished by HP labs and I-V characteristics of the fabricated memristor are shown in Fig. 1 (Williams, 2008). It is a two layer titanium dioxide (TiO2) cube between two crossed nano wires. It can be seen from the figure that it is a voltage regulated device where the value of applied voltage (V) controls the thickness of doped region (w) and turns it on (R_{on}) or off (R_{off}).

Depending upon the last applied voltage value, memristor is capable of storing logic values in terms of high or low impedance. The capability of logic storage without power consumption makes it an ideal candidate for storage systems (Sarwar et al., 2013). The ability of memristor to remember its resistance is used in analog circuits where they act as reconfigurable resistors (Pershin and Di Ventra, 2010). Memristor resistance is used as logic state level in Kvatinsky et al. (2011) where memristors are used to perform logic operations. Memristive logic operations are also used in Memristor Ratioed Logic (MRL) and Memristor-Aided Logic (MAGIC) (Kvatinsky et al., 2012; Kvatinsky et al., 2014) architectures where CMOS compatible hybrid CMOS-memristive logic

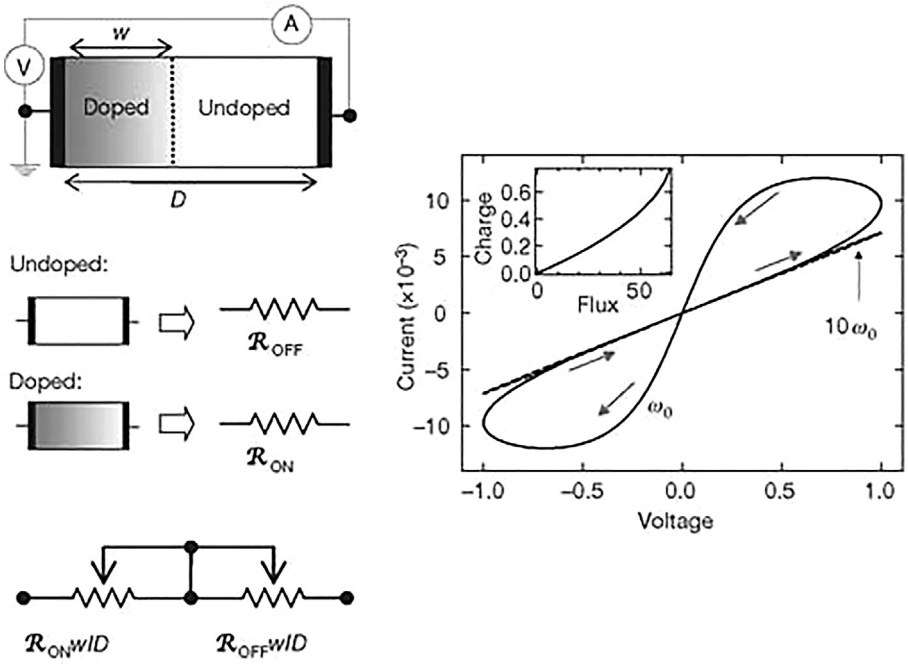

Fig. 1. Model and I-V characteristics of Hewlett-Packard (HP) memristor.

family is used to construct basic logic gates. Another memristor-based logic architecture is presented by Hamdioui et al. (1718). This architecture has the potential of performing computation and storage in the same physical layer; hence making memristor-based devices more capable in terms of handling data-intensive applications which require low leakage current and high communication throughput. Memristors can also be used as reconfigurable switches which can be immensely helpful in making efficient reconfigurable architecture (Cong and Xiao, 2011). In such architecture, CMOS-memristor hybrid approach is used where memristors are used for configuration bits and CMOS-based components are used for logic elements. Similarly, memristors can also be used with transistors to construct look-up table (LUT) for FPGAs (Almurib et al., 2014). In this approach, memristors are used for storage and NMOS transistors are used for selection purposes. Authors in Almurib et al. (2014) propose a novel memristor-based read and write technique which is more efficient in terms of power consumption as compared to conventional techniques. Finally, authors in Strukov and Mishchenko (2010) present a monolithically stacked hybrid FPGA which is based on memristor-transistor hybrid approach. Monolithic FPGA is shown to produce significant improvements in terms of area while having similar power results as compared to conventional CMOS FPGA architectures.

From the work presented in Kvatinsky et al. (2012, 2014), Hamdioui et al. (1718), Almurib et al. (2014), Strukov and Mishchenko (2010), it is evident that work on memristor-transistor hybrid reconfigurable architectures is a prevalent research domain and hybrid architectures have shown significant improvements over conventional architectures in terms of area and power consumption. It is important to note here that work in Kvatinsky et al. (2012, 2014), Hamdioui et al. (1718), Almurib et al. (2014), Strukov and Mishchenko (2010) incorporates memristors either to build solely routing interconnect or as a replacement of some memory elements. Authors in Aslam et al. (2016) extend the idea of hybrid reconfigurable architectures and present design of basic building blocks as well as routing interconnect of hybrid homogeneous FPGA architecture. In addition, they also present complete environment for the exploration of hybrid homogeneous FPGA architecture.

However, in the context of hybrid reconfigurable architectures, all the work cited above considers homogeneous architectures only where homogeneous CLBs are designed using hybrid approach. In this work, we extend the design of hybrid reconfigurable architectures to heterogeneous domain (i.e. heterogeneous FPGAs). For this purpose, we design and simulate basic building blocks like CLBs as well as HBs like adders and multipliers using memristor-transistor hybrid approach. Some examples of the design of individual HBs like adders and multipliers exist in the literature (El-Slehdar et al., 2015; Mane et al., 2014; Fey, 2014; Guckert and Swartzlander, 2017). But, to the best of our knowledge, no detailed work has been done on complete hybrid heterogeneous FPGA architectures. Apart from logic and routing interconnect of hybrid heterogeneous FPGA, we also present a complete exploration environment in this work. The proposed environment is generalized in nature and it is built using a mixture of open source and indigenously developed tools. A preliminary version of this work was presented in Farooq et al. (2016). However, here we present detailed design of proposed building blocks. Additionally, we also present detailed discussion on the proposed exploration environment. For experimentation purposes, we add more benchmarks to our benchmark suite and present a detailed discussion on exploration and comparison results.

3. Reference FPGA architecture

Homogeneous FPGAs were first introduced almost three decades ago and they got instant popularity because of their generalized reconfigurable nature. An abstract level view of a homogeneous mesh-based FPGA is shown in Fig. 2. It can be seen from this figure that CLBs are arranged in a two dimensional mesh and they are surrounded by uniformly distributed SBs and CBs. Furthermore, it can also be seen that routing resources are arranged in the form of horizontal and vertical routing channels where each routing channel contains uniform number of routing

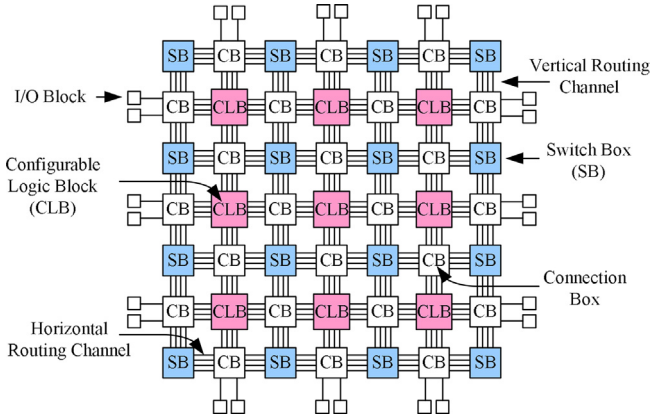

Fig. 2. Abstract level view of mesh-based homogeneous FPGA.

wires. Computation capability of a homogeneous FPGA mainly comes from CLBs which are connected together to implement complex logic functions. Internal architecture of a four-input CLB is shown in Fig. 3. It can be seen from this figure that 4-input CLB contains a 4-input LUT which further contains fifteen 2-input multiplexers.

Programmable nature of the routing interconnect of FPGAs is mainly due to reconfigurable SB and CB of the FPGA. SBs in an FPGA are used to provide programmable interconnect between horizontal and vertical routing channels (see Fig. 2) which are incident on them whereas CBs provide programmable interconnect between I/Os of CLBs and the routing interconnect of the FPGA. Internal structure of sample SB and CB of an FPGA is shown in Fig. 4. Fig. 4a shows the internal structure of a programmable SB. Similarly, Fig. 4b shows the internal structure of a sample connection box where I/Os of CLB are connected to surrounding routing channels.

Contrary to homogeneous FPGAs that contain only a single type of computation blocks (i.e. CLBs), heterogeneous FPGAs contain a mixture of general purpose (i.e. CLBs) and specific purpose (i.e. HBs) computation blocks. The CLBs and HBs in a heterogeneous FPGA are arranged in a mesh-based manner and they are surrounded by evenly distributed programmable routing resources. An abstract level view of the mesh-based heterogeneous FPGA used in this work is shown in Fig. 5. Just like homogeneous FPGA, CLBs, SBs, and CBs in heterogeneous FPGA are programmable in nature while HBs are specific purpose blocks that can perform

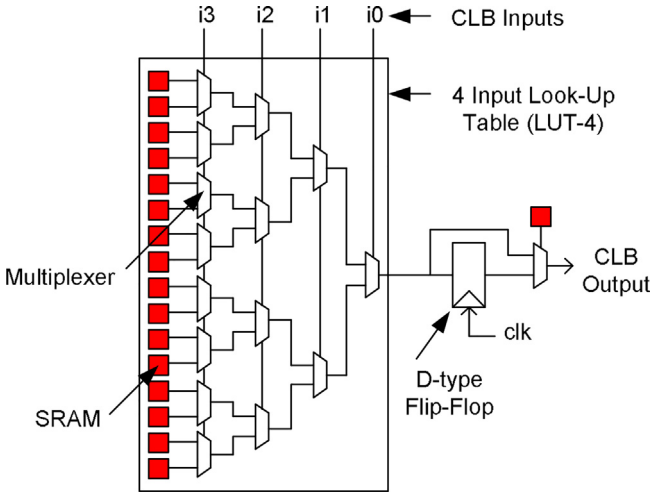

Fig. 3. Internal structure of CLB.

specific computation operations only. Inclusion of HBs in heterogeneous FPGAs has made them less generalized in nature compared to their homogeneous counterparts. In addition, specific nature of HBs can lead to potential wastage of precious computing resources (Jamieson and Rose, 2006). But overall, they give better area results compared to homogeneous FPGAs and are widely regarded as a possible replacement of cell-based, application specific circuits (Pangracious et al., 2013). Normally, in conventional heterogeneous FPGAs, CLBs, SBs, CBs, and HBs are constructed using transistors only; hence they are termed as transistor-only architectures. In this work, we propose the design of these blocks using memristor-transistor hybrid approach. In the next section, we present the memristor-model that we have used for the design of these blocks and then we present the spice simulation results of some sample blocks as well.

4. Proposed hybrid building blocks

Internal structure of sample CLB, SB, and CB shown in Figs. 3 and 4 indicates that homogeneous FPGAs can be constructed using a combination of multiplexers, F/Fs, and SRAMs. Furthermore, we know that HBs like adders and multipliers can be constructed using a combination of basic logic gates. Conventionally, building blocks of FPGAs are created using transistors only, but in this section we present the design of these blocks using memristor-transistor hybrid approach which will lead to a much efficient hybrid FPGA architecture. In the following part of this section, first we present the memristor model that we have used for the design of hybrid blocks and then we present the spice simulation results of the design of these hybrid blocks.

4.1. Spice memristor model

First physical implementation of memristor was announced by HP in 2008 (Williams, 2008) where they proved the feasibility of the element. Since then, a lot of research has been carried out on spice models of memristor as it plays a critical role in the advancement of research. Researchers have developed different models of memristor (Biolek et al., 2009; Batas and Fiedler, 2011; Rák and Cserey, 2010) using different environments like PSPICE, LTSPICE, and Matlab. However, no standard model in HSPICE exists for memristor simulation. In this work, we use HSPICE based current controlled memristor model (Biolek et al., 2013). Selected memristor model is an ideal model and it is based on Eq. (1).

$$Rq(t) = R_{off} + \frac{(R_{on} - R_{off})}{(\alpha e^{-4kq(t)} + 1)} \tag{1}$$

In Eq. (1), R_{on} ($1k\Omega$) is conducting memristor resistance, R_{off} ($100k\Omega$) is non-conducting memristor resistance and k is memristor film thickness (10 nm). Furthermore, the value of α is calculated using Eq. (2) where R_{init} is initial memristor resistance and its value is $5k\Omega$ (Biolek et al., 2013). By using Eqs. (1) and (2), we have modeled a memristor in HSPICE that replicates its behavior shown in Fig. 1.

$$\alpha = \frac{R_{init} - R_{on}}{(R_{off} - R_{init})} \tag{2}$$

4.2. Building block design and simulation

Memristor model described above is combined with transistors to construct basic components like NOT gate, NOR gate, multiplexers, F/Fs, and memory cell etc. These components are then combined together to construct hybrid blocks like CLBs, SBs, and CBs of a homogeneous FPGA. Design and simulation results of these

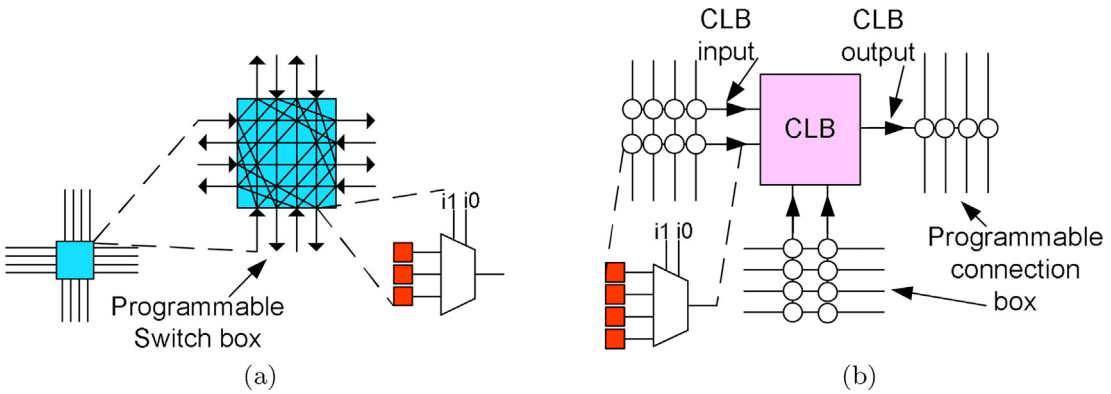

Fig. 4. (a) Internal structure of programmable switch box; (b) Internal structure of programmable connection box.

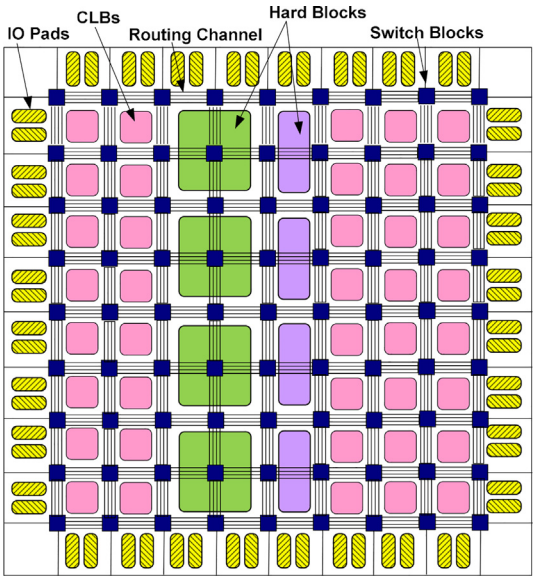

Fig. 5. Abstract level view of mesh-based heterogeneous FPGA.

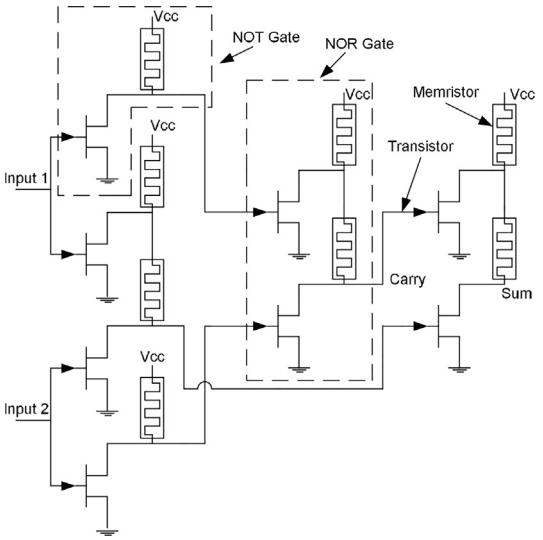

Fig. 6. Schematic circuit diagram of a 1-bit half adder.

hybrid building blocks have already been presented in our previous work (Aslam et al., 2016). Here we extend our hybrid approach to design HBs like adders and multipliers. Hybrid HBs are then combined with hybrid CLBs, CBs, and SBs to construct a hybrid heterogeneous FPGA like the one shown in Fig. 5. For the sake of explanation, we start with the design of basic hybrid HB like half-adder which can be then used to construct large HBs like multi-bit full adder and multiplier. Two NOT gates and three NOR gates are normally required to construct a half adder. Schematic circuit diagram of a half adder constructed using hybrid approach is shown in Fig. 6. Schematics of hybrid NOT and NOR gate are shown inside the dashed boxes of Fig. 6. Value of Vcc is set to be 1.8 volts in this circuit. Spice simulation results of hybrid 1-bit half adder are shown in Fig. 7.

Combination of single bit hybrid half adders and hybrid logic gates is used to construct single bit full adder. Logic diagram of single bit full adder constructed using two half adders and an OR gate is shown in Fig. 8. For the sake of simplicity, we do not show schematic circuit diagram of hybrid full adder here but spice simulation results of designed hybrid full adder are shown in Fig. 9. We know that heterogeneous FPGAs contain complex HBs like multi-bit adders and multipliers. In this work, we use heterogeneous FPGAs that contain 20 + 20 bit adders. So, multiple instances of single bit hybrid full adder described above are combined together to construct large ripple carry adders which form the HB of our target hybrid heterogeneous FPGA architecture.

Apart from a large adder as a HB, we have also designed a multi bit hybrid multiplier in this work. Here, we present the design and simulation results of only a 2-bit hybrid multiplier as a sample. Block diagram of a sample 2-bit multiplier is shown in Fig. 10. It can be seen from this figure that a 2-bit array multiplier is composed of a combination of two half adders and four AND gates. Deign of hybrid half adder has already been discussed before and schematic circuit diagram of hybrid AND gate is shown in Fig. 11. So, we combine hybrid half adders and AND gates to construct a hybrid 2-bit multiplier. Simulation results of hybrid multiplier are shown in Fig. 12. Here, we have shown design and simulation results of 2-bit array multiplier only. But for our experimentation, we have designed multiplier HBs which have large sizes. For example, in this work, we design 18 × 18-, 16 × 16-bit multipliers as per the requirements of our target heterogeneous benchmarks. These large hybrid multipliers are designed as array multipliers and they are constructed by combining together multiple AND gates, half adders, and full adders.

Specifications of hybrid HBs presented in this section are combined with hybrid CLBs, CBs, and SBs to generate the hybrid heterogeneous FPGA architecture. Multiple heterogeneous benchmarks are then placed and routed on the proposed FPGA architecture through a generalized exploration flow and their area results are compared with those of transistor-only heterogeneous FPGA.

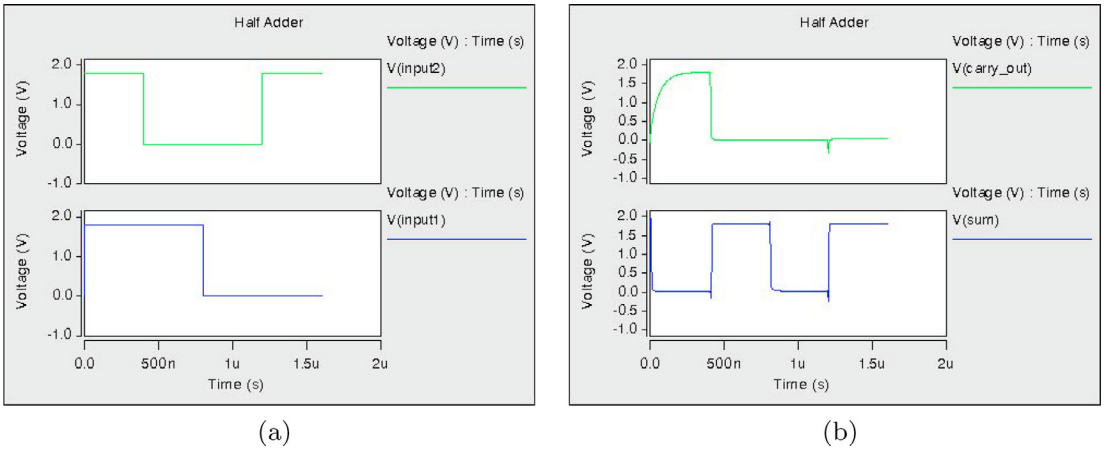

Fig. 7. (a) Half adder input simulation waveform; (b) Half adder output simulation waveform.

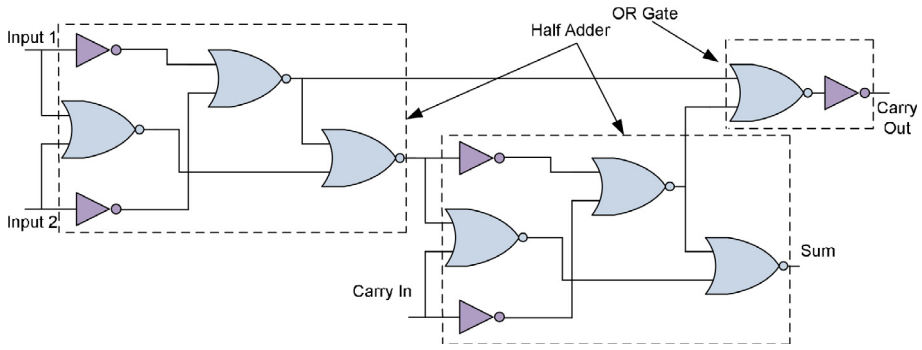

Fig. 8. Gate level logic diagram of 1-bit full adder.

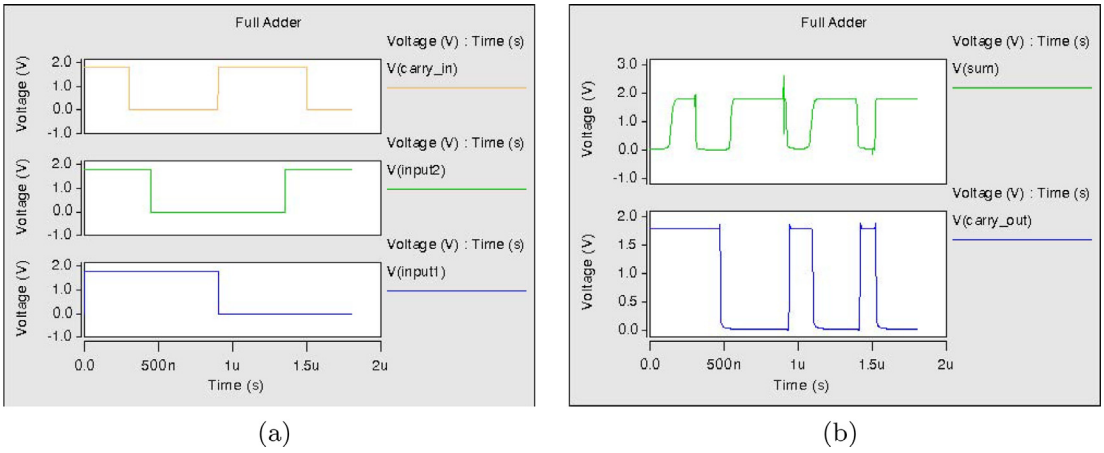

Fig. 9. (a) Full adder input simulation waveform; (b) Full adder output simulation waveform.

Detailed discussion on different steps involved in the exploration flow and relevant tools are discussed in the next section of the paper.

5. Exploration flow

In this work, we present area comparison between two heterogeneous FPGA architectures. In order to have a fair comparison between the two architectures, we develop a generalized exploration flow that can cater the needs of both architectures. An overview of our exploration flow is given in Fig. 13. It can be seen from this figure that our exploration flow is mainly divided into two parts. First part takes hardware description of the benchmark as input and after passing through various steps, these benchmarks are converted into netlists (.net format). Second part takes netlists as inputs which are then placed and routed on the target FPGA architecture. Target FPGA architecture can either be a hybrid heterogeneous FPGA architecture or a conventional heterogeneous FPGA architecture and our architecture independent, general purpose tools serve well for both architectures.

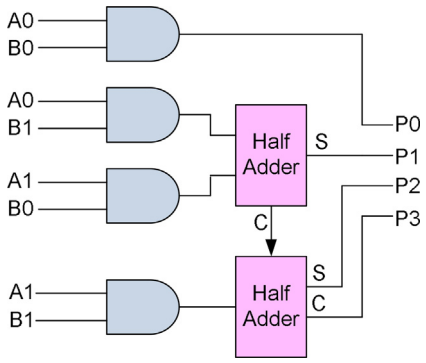

Fig. 10. Block diagram of 2-bit multiplier.

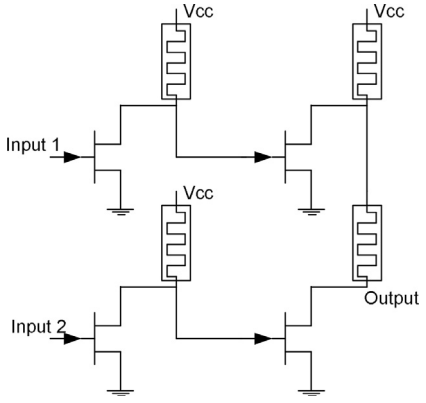

Fig. 11. Schematic circuit diagram of hybrid AND gate.

5.1. Benchmark selection and netlist conversion

Fig. 13 shows that first part of exploration takes structural hardware description (.vst) of the benchmark under consideration. This description is then passed through VST TO BLIF tool (B. University of California, 1992) that converts standard hardware description into I/O port, gate, and HB instances. The output of this tool is a.blif file which is then supposed to pass through a synthesis tool so that it may be converted into LUT format, since, in this work, we use benchmarks having HBs and we want to preserve them. So, the benchmark under consideration is first passed through a parser that we term as Parser-I. The main objective of this parser is to remove HBs from blif file in such a way that all the dependence between HBs and rest of the blif file is preserved. Parser-I performs a number of modifications in the blif file. Some of the modification performed by Parser-I are illustrated in Fig. 14. For example Fig. 14a shows a scenario where outputs of HB are connected to inputs of other gates. In order to remove the HB and preserve the dependence, main inputs are added to the netlist and they are connected to the inputs of the gates eventually resulting in netlist of Fig. 14b. Similarly, Fig. 14c shows a scenario where output of HB is connected to main output of the netlist. In order to preserve dependence, main outputs are connected to zero gates (see Fig. 14d). This is done because when HBs are removed, the main outputs do not remain unconnected. Similarly, two other possible modification cases are shown in Fig. 14e, f where inputs of removed HBs are added as the main outputs of the netlist. Once the requisite modifications are performed by Parser-I, all the HBs are removed from blif file and it is then passed to SIS tool (Sentovich et al., 1992) for logic optimization. SIS performs logic optimization of blif file and maps the gate level presentation into LUT and F/F format. For mapping, FlowMap mapper is used which is integrated in SIS tool. Fig. 15a and b show the sample benchmark before and after mapping. Once mapping is performed, the LUTs and F/F in the netlist are packed through T-VPACK (Marquart et al., 1999). Transformation of a netlist from mapping to packing is shown in Fig. 15b and c respectively. After packing, the netlist is finally passed through Parser-II that removes previously added temporary dependencies and also adds previously removed HBs. After passing through Parser-II, benchmark is converted into.net format and it is now ready to be placed and routed on the target FPGA architecture.

In this work, we use two sets of open core benchmarks. Hardware description of these benchmarks is obtained from their respective sources and they are then converted to.net format using series of steps detailed above. Details of two benchmark sets obtained after conversion to.net format are given in Tables 1 and 2 respectively. It can be seen from these tables that division of these benchmarks in two sets is mainly based on the types of hardware blocks supported by each set. Furthermore, it can also be noted from these tables that we use benchmarks having CLB size equal to 4 as it is reported to produce best area results (Aslam et al., 2016).

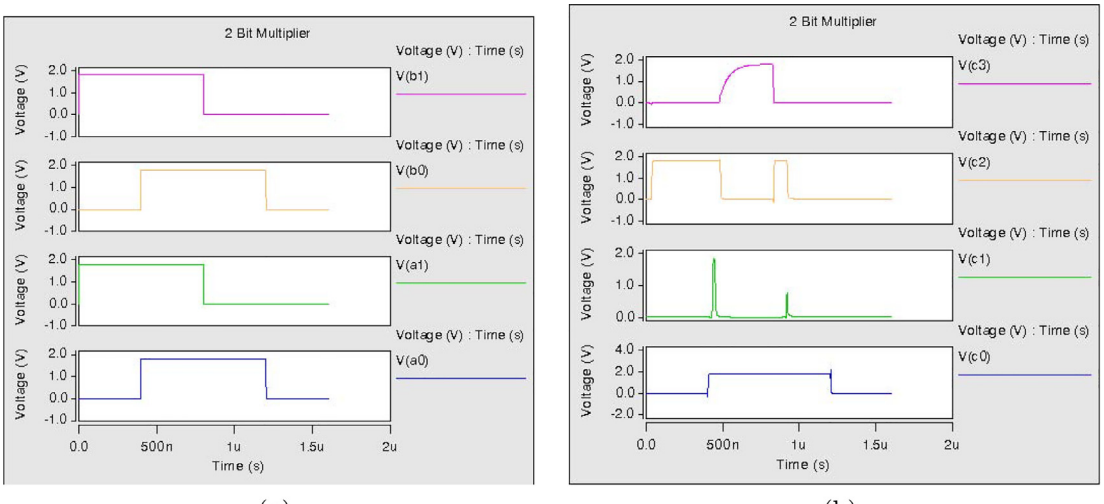

Fig. 12. (a) Multiplier input simulation; (b) Multiplier output simulation.

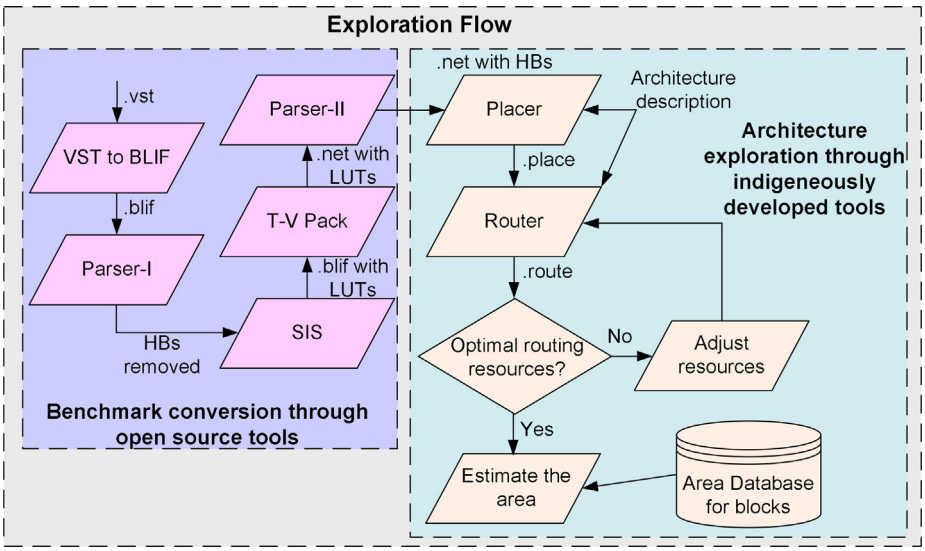

Fig. 13. Flow diagram of exploration environment.

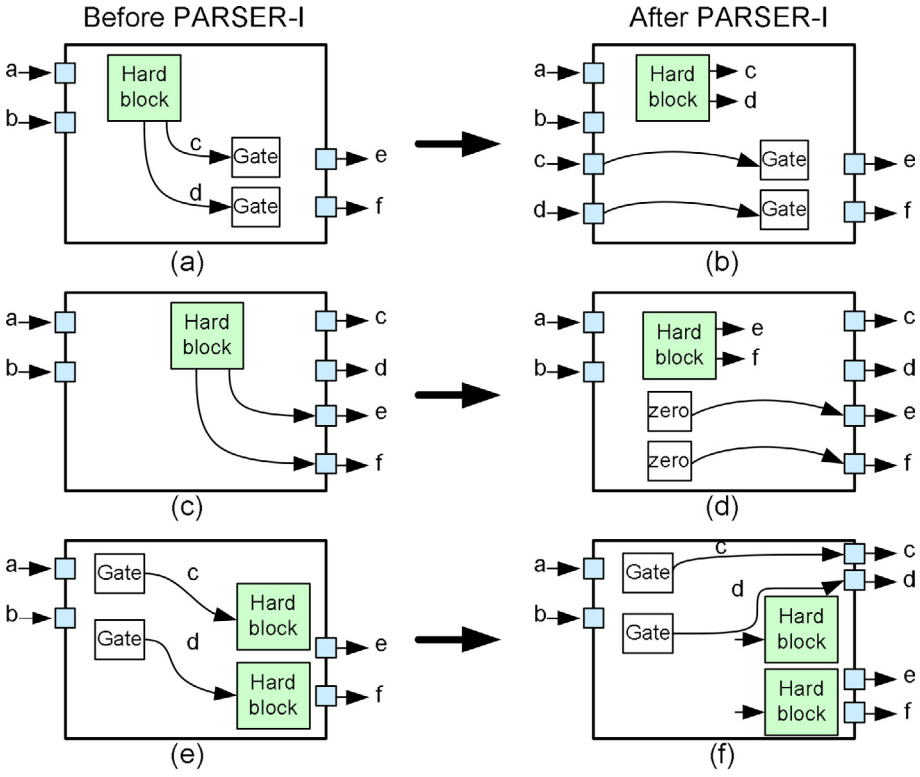

Fig. 14. Illustration of modifications performed by Parser-I.

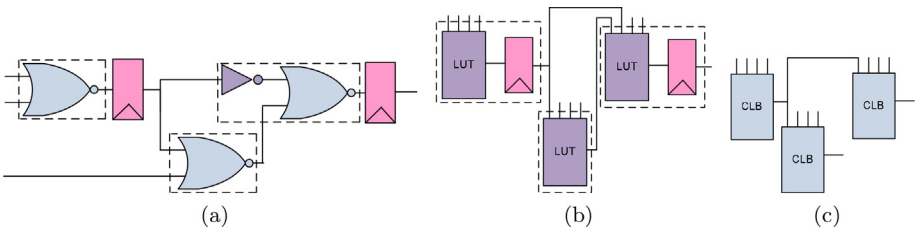

Fig. 15. Illustrations of (a) Sample BLIF file; (b) Sample mapped file; (c) Sample packed file.

Table 1
Details of open core benchmarks SET I.

Circuit name	No of inputs	No of outputs	No of 4-input CLBs	No of multipliers (18 × 18)	Function
cf_fir_3_8_8	42	22	224	4	Third order FIR filter
cf_fir_24_16_16	496	250	3348	30	24 order FIR filter
diffeq_f_systemC	66	99	1560	4	Differential equation solver
diffeq_paj_convert	12	101	756	7	Differential equation converter
fir_scu	10	27	1400	20	16 order FIR filter
iir	33	30	400	5	Infinite impulse response (8 bit)
iir1	28	15	648	6	Infinite impulse response (16 bit)
rs_decoder_1	13	20	1560	20	Reed Solomon decoder (4 bit)
rs_decoder_2	21	20	2968	14	Reed Solomon decoder (8 bit)

Table 2
Details of open core benchmarks SET II.

Circuit name	No of inputs	No of outputs	No of 4-input CLBs	No of multipliers (16 × 16)	No of Adders (20 + 20)	Function
cf_fir_3_8_8	42	18	159	4	3	Third order FIR filter
cf_fir_7_16_16	146	35	638	8	14	Seven order FIR filter
cf_fir_24_16_16	520	230	2275	26	52	24 order FIR filter
cfft16x8	20	40	1511	-	26	Complex fast fourier transform
cordic_p2r	18	32	803	-	43	Polar to rectangular
cordi_r2p	34	40	1328	-	52	Rectangular to polar
fm	9	12	1308	1	19	Frequency modulation transmitter
fm_receiver	10	12	910	1	20	Frequency modulation receiver
lms	18	16	940	10	11	Adaptive equalizer routine
reed_solomon	138	128	537	16	16	High speed reed solomon encoder

5.2. Architecture exploration

It can be seen from Fig. 13 that once the benchmark is converted into.net format, it can then be placed and routed on the target FPGA architecture for exploration and final resource estimation.

5.2.1. Placer

For placement, in this work, we develop a generalized placement tool that takes architecture description and.net files as input and generates the placement file as output. The placement tool reads specific information about interconnection of CLBs, HBs, and I/Os from.net file and then places them on the target architecture through heuristic based simulated annealing placement algorithm (Kirkpatrick et al., 1983). The objective of the simulated annealing algorithm is to place connected blocks close to each other and minimize the total wire length requirement of a placement solution. The total wire length of a placement solution is estimated using Eq. (3) where N is the total number of nets in the netlist, $bbx(i)$ is the horizontal span of net i, $bby(i)$ is its vertical span, and $q(i)$ is a correction factor. Simulated annealing algorithm uses an iterative approach to gradually reduce the wire cost of placement solution over multiple iterations. This algorithm is architecture independent and it is applicable to both conventional as well as hybrid FPGA architecture.

$$WireCost = \sum_{i=1}^{N} q(i) \times (bb_x(i) + bb_y(i)) \tag{3}$$

5.2.2. Router

After the successful placement of netlist, routing between source and destination pins of different blocks of the netlist is performed. Routing problem in FPGAs consists of finding dedicated routing resources for individual nets. In order to perform routing, in this work, we use Pathfinder algorithm (McMurchie et al., 1995) which is an architecture independent, commonly used routing algorithm for FPGAs. Due to generalized nature of pathfinder, the routing resources of FPGA are first modeled as directed graph $G(V, E)$ and pathfinder then routes individual nets of the netlist using an iterative approach. In this graph, the I/O pins of different blocks are represented as nodes V and these nodes are connected to each other through edges E which represent wires of routing interconnect of FPGA architecture. Fig. 16a shows sample portion of the routing interconnect of an FPGA architecture and Fig. 16b shows the corresponding directed graph abstraction. Given this graph abstraction, the goal of the routing algorithm is to find unique, non-intersecting paths for all the nets of the netlist under consideration which is a difficult problem. In order to resolve this problem, pathfinder uses an iterative, negotiation based, congestion driven routing approach. In this approach, multiple iterations of routing are performed until all congestion is removed. In each routing iteration, individual nets are routed using Dijkstra's shortest path algorithm (Dijkstra, 1959) and congestion of a node (c_n) is controlled using cost function of Eq. (4) where b_n is the base cost of using the resource n, h_n is the history of congestion of node n, and p_n is the present number of nets sharing the node n in the current iteration.

$$c_n = (b_n + h_n) \times p_n \tag{4}$$

5.2.3. Resource optimization

It can be seen from Fig. 13 that when one routing round is complete, routing file is generated that contains all the information about different paths taken by different nets of the netlist under consideration. In order to optimize the routing resource requirement of the netlist, we perform multiple rounds of routing. The resource optimization of the FPGA architecture is performed through binary search algorithm where for each round of routing, resources of the architecture are adjusted and this process continues until the optimal resources are found. After the completion of binary search flow, we know the logic and routing resource requirement of the netlist under consideration and area of the FPGA architecture is calculated. Discussion on the used area model and the area estimation is described next.

5.2.4. Area model

Once the resource optimization process is completed, we know the total resource requirement and area of FPGA architecture is

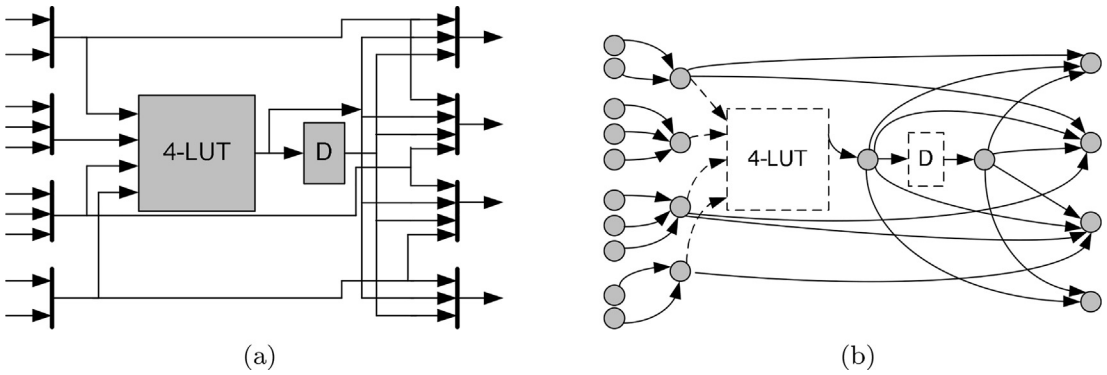

Fig. 16. (a) Sample interconnect portion of FPGA; (b) Directed routing graph of sample interconnect.

then calculated using our generalized area model. It is discussed in Betz et al. (1999), DeHon (2001) that cell area dominates the wiring density issue for digital systems and we have to care about number of switches rather than wiring density of a network. For this reason, we have devised an area model based on the area of cells used by the FPGA architecture. Area of a heterogeneous FPGA can be mainly divided into two parts: logic area and routing area.

Logic area of an FPGA is composed of sum of area of all the CLBs and HBs in the architecture. From discussion presented in Section 3 and from structure shown in Figs. 3, 8, we know that CLBs and HBs are further composed of multiplexers, F/Fs, SRAMs, and basic logic gates. We calculate area of aforementioned components, determine their total number in FPGA, and then sum them up to calculate total logic area of architecture. Routing area, on the other hand, comprises of sum of area of all the CBs, SBs in the routing interconnect of the architecture. From Fig. 4, we know that internal structure of above mentioned blocks is further comprised of multiplexers, buffers, SRAMs etc. In this work, we calculate the area of individual cells (both conventional and hybrid) and then by combining the area of individual cells we calculate the routing area of the architecture. Once the logic area and routing area of architecture under consideration are calculated, we combine them to calculate the total area of architecture. Further discussion on the individual cell area calculation and area comparison between two architectures is given in the next section of paper.

6. Experimental results and analysis

Experimental results presented in this section are mainly divided into two parts. In the first part, we present and compare area results of basic building blocks of FPGA. In this part, we present results of both hybrid and conventional building blocks. In the second part, we present comprehensive area comparison between conventional and hybrid heterogeneous FPGA architectures.

6.1. Component level area comparison

As discussed in Section 4, the fundamental component of basic building blocks of FPGA is transistor for conventional architectures and memristor, transistor for hybrid architectures. In this work, we have used HSPICE metal oxide semi-conductor field effect transistor (MOSFET) model. The values of channel length (65 nm) and width (100 nm) are taken from the data manual for 65 nm processing technology. Based on these values, area of a single transistor is calculated to be $Area_{Transistor} = 0.0065\ um^2$. As demonstrated by HP labs and also discussed in Strukov et al. (2008), Williams (2008), as a rough first order estimation, the area of memristor is considered to be half of area of MOSFET of equivalent technology.So, the area of memristor is assumed to be $Area_{Memristor} = 0.00325\ um^2$ for our experimentation.

Given the above values of memristor and transistor areas and by using the design of different blocks discussed in Section 4, we calculate their areas and present them in Table 3. In this table we present number of I/Os, basic component requirement, and area estimation results of both hybrid as well as conventional building blocks of heterogeneous FPGA. It can be seen from this table that hybrid building blocks in general give better area results as compared to conventional building blocks. However, the area gain of hybrid blocks varies as compared to conventional blocks. Reason for the variation in this area gain is varying component requirement for hybrid building blocks as compared to their conventional counterparts. For example, in case of hybrid multiplexer, we need three transistors and one memristor to achieve the desired multiplexer behavior (Aslam et al., 2016). But for conventional multiplexer, we need six transistors to achieve the requisite behavior. This variation in component requirement leads to an area gain of 42% for hybrid multiplexer as compared to conventional multiplexer. Similarly, for NOR gate, the number of components required for conventional block (4 transistors) is same as hybrid block (2 memristors + 2 transistors). But smaller area of memristors gives 25% area gain for hybrid NOR gate as compared to conventional NOR gate. Similar trend is observed for NOT gate, AND gate, buffer, and F/F etc. As far as CLB is concerned, the difference in component requirement is quite significant and it ultimately leads to an area gain of 60% for hybrid CLB as compared to conventional CLB. The reason behind this component requirement gap can be understood by considering the internal structure of CLB. For example, inside a 4-input, 1-output CLB (see line 7 of Table 3), there are 16 memory cells, 16 2 ×;1 MUXs, and 1 D-flip flop. Although hybrid MUX and D-flip flop give significant area gain as compared to their conventional counterparts, the main gain for hybrid CLB comes from the replacement of 16 memory cells with memristors. In conventional CLB, each memory cell is comprised of 6 transistors, whereas in hybrid cells they are directly replaced with memristors; hence leading to 60% area gain of hybrid CLB over conventional CLB. As far as multi bit hybrid adder and multiplier are concerned, we have designed them using a combination of single bit half adders, full adders, and AND gates. Since hybrid half adder, full adder, AND gate give smaller area gain (i.e. 25%) compared to CLB, so, the area gain of hybrid multi bit adder and multipliers is also comparatively small.

Area results of different blocks presented in Table 3 are used in the area estimation model discussed in Section 5.2.4. These area estimation results are then used for comparison between hybrid and conventional heterogeneous FPGA architectures. Although we have used metrics of 65 nm processing technology for area

Table 3
Area comparison of memristor-transistor and pure transistor based basic building blocks of heterogeneous FPGA.

Component	Input	Output	Hybrid approach		Transistor approach		Area gain (%)
			No. of comps	Area (um^2)	No. of comps	Area (um^2)	
2 × 1 MUX	2	1	4	0.0225	6	0.039	42
NOR gate	2	1	4	0.0195	4	0.026	25
NOT gate	1	1	2	0.0097	2	0.013	25
AND gate	2	1	8	0.039	8	0.052	25
Buffer	1	1	4	0.0195	4	0.026	25
D flip-flop	1	1	24	0.166	24	0.221	25
CLB	4	1	104	0.578	216	1.469	60
Half adder	2	2	16	0.078	16	0.104	25
Full Adder (1-bit)	3	2	38	0.185	38	0.247	25
Mult (16 × 16)	32	32	1856	9.048	1856	12.06	25
Adder (20 + 20)	41	21	760	3.705	760	4.94	25
Mult (18 × 18)	36	36	2320	11.31	2320	15.08	25

calculation of basic building blocks of FPGA, similar trends hold for smaller processing technologies and memristors show even more encouraging results with shrinking processing technologies (Sarwar et al., 2013; Vourkas et al., 2014).

6.2. Architecture level area comparison

For area comparison between two heterogeneous FPGA architectures, we use two sets of open core benchmarks in this work. Benchmark specifications of two sets are already given in Tables 1 and 2 respectively. These benchmarks are placed and routed on two heterogeneous FPGA architectures using the flow of Section 5 and respective area estimation is performed using area values of individual cells in the area model of Section 5.2.4. Fig. 17 shows logic area comparison between conventional and hybrid heterogeneous FPGAs for benchmarks of SET I. It can be seen from this figure that hybrid architecture gives better logic area results as compared to conventional FPGA architecture for all the benchmarks of SET I. In this figure, first nine columns give individual logic area comparison while last column gives average area comparison which shows that, on average, hybrid FPGA requires 58% less logic area as compared to conventional FPGA architecture for SET I benchmarks. Although individual logic area gain of hybrid CLB is 60% compared to conventional CLB, the smaller area gain of hybrid multiplier compared to conventional multiplier leads to overall smaller logic area gain for hybrid architecture.

Routing area of FPGA is normally further divided into buffer area and switch area. Switch area comprises the area of all the configurable switches in the routing interconnect whereas buffer area comprises the area of all the buffers in the architecture. Buffers in the architecture are inserted at regular intervals to maintain the signal strength that traverse long distances in the architecture. Fig. 18 shows the buffer area comparison between hybrid and conventional FPGA architecture. Since individual hybrid buffer cell area gain is 25% over conventional buffer cell, the overall buffer area gain for hybrid architecture is also 25% compared to the buffer area of conventional FPGA architecture.

Switch area in an FPGA normally comprises the sum of area of configurable switches and configuration memory cells. In a conventional FPGA, configuration memory cells are comprised of SRAMs whereas memristors act as configuration memory cells in hybrid FPGAs (Wang et al., 2010). A single memristor is around 90% smaller in area as compared to a single bit SRAM cell. However, when coupled with configuration overhead, this area gain might be reduced (Sarwar et al., 2013). Furthermore, configuration memory cells make 35% of total routing area of an FPGA architecture. Although multiplexers form the major part of switch area of FPGA and their area gain is small (i.e. 42%), huge area gain of configuration memory leads to a significant switch area gain. Fig. 19 shows switch area comparison between conventional and hybrid FPGA and it can be seen from this figure that hybrid FPGAs require, on average, 60% smaller switch area compared to conventional FPGAs.

Although buffer area gain of hybrid FPGA is smaller (i.e. 25%) as compared to switch area gain (i.e. 60%), but switch area makes 94% of routing area of FPGA; hence leading to an overall routing area gain of 59% for hybrid FPGA as compared to conventional FPGA. Routing area comparison results are shown in Fig. 20. Finally, the total area comparison results of conventional and hybrid FPGA for set I benchmarks are shown in Fig. 21. It can be seen from this figure that hybrid FPGA requires, on average, 59% smaller area compared to conventional FPGA.

Area comparison results for SET II benchmarks are shown in Figs. 22–26. It can be seen from Fig. 22 that just like SET I benchmarks, hybrid FPGA gives on average 56% logic area gain for SET II benchmarks. The reason for slightly smaller logic area gain

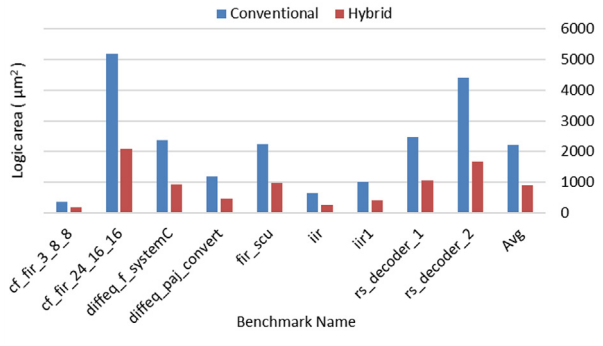

Fig. 17. Logic area comparison between conventional and hybrid heterogeneous FPGAs for SET I benchmarks.

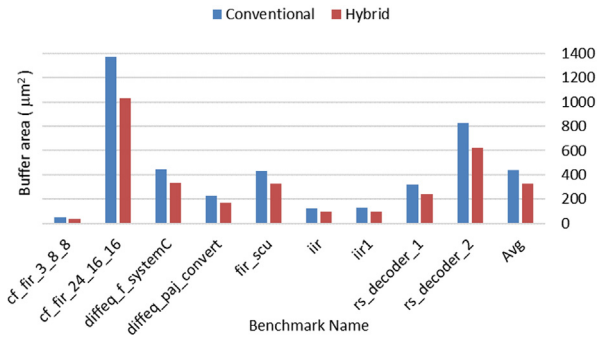

Fig. 18. Buffer area comparison between conventional and hybrid heterogeneous FPGAs for SET I benchmarks.

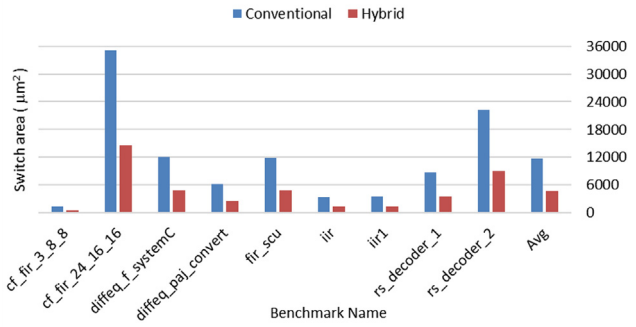

Fig. 19. Switch area comparison between conventional and hybrid heterogeneous FPGAs for SET I benchmarks.

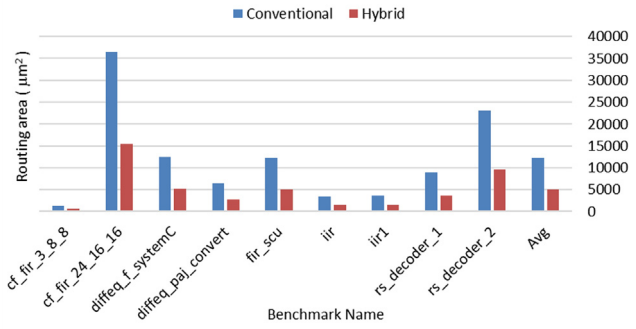

Fig. 20. Routing area comparison between conventional and hybrid heterogeneous FPGAs for SET I benchmarks.

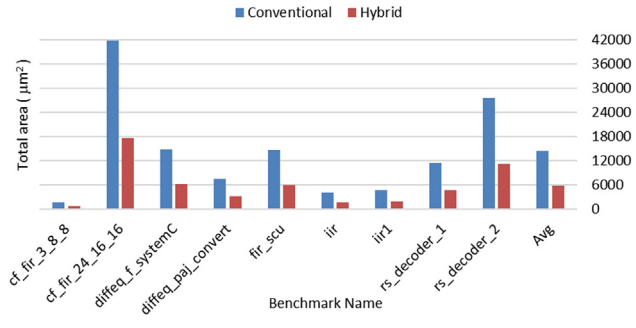

Fig. 21. Total area comparison between conventional and hybrid heterogeneous FPGAs for SET I benchmarks.

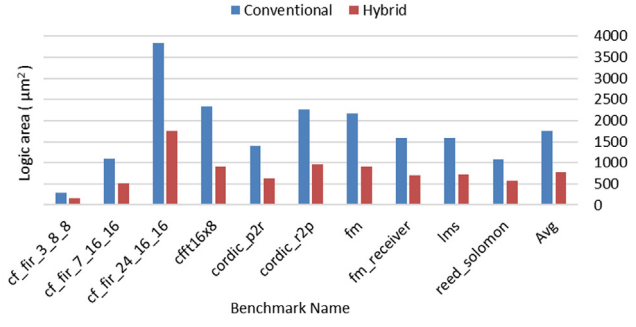

Fig. 22. Logic area comparison between conventional and hybrid heterogeneous FPGAs for SET II benchmarks.

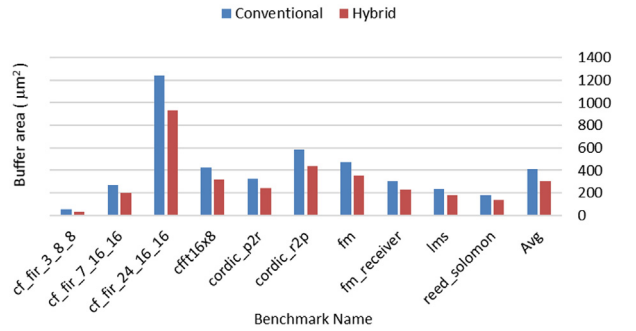

Fig. 23. Buffer area comparison between conventional and hybrid heterogeneous FPGAs for SET II benchmarks.

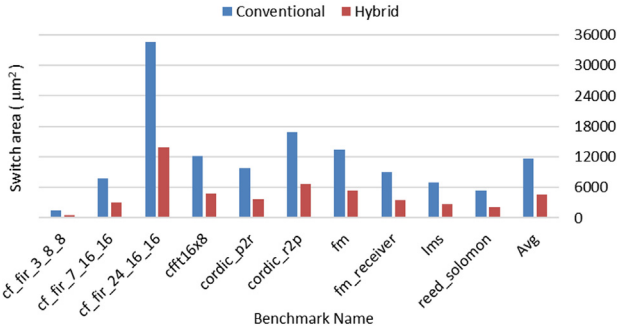

Fig. 24. Switch area comparison between conventional and hybrid heterogeneous FPGAs for SET II benchmarks.

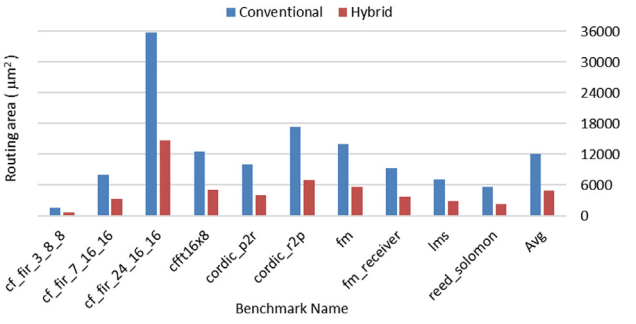

Fig. 25. Routing area comparison between conventional and hybrid heterogeneous FPGAs for SET II benchmarks.

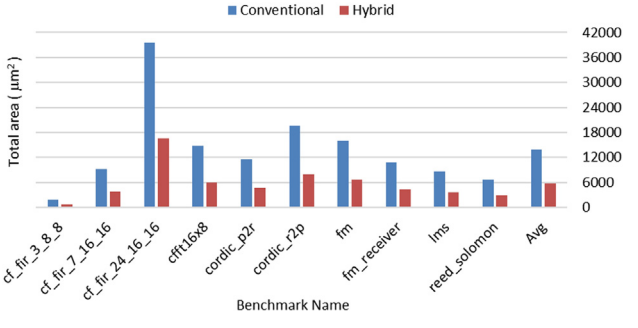

Fig. 26. Total area comparison between conventional and hybrid heterogeneous FPGAs for SET II benchmarks.

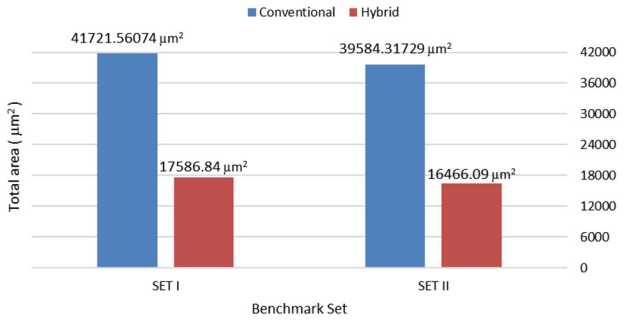

Fig. 27. Generic area comparison between conventional and hybrid heterogeneous FPGAs for SET I and SET II benchmarks.

compared to SET I benchmarks is increased number of HBs in SET II benchmarks. As we can see from Table 3 that hybrid HBs have smaller gain (i.e. 25%) as compared to hybrid CLBs (i.e. 60.4%), so an increased number of HBs result in smaller logic area gain of Fig. 22. Furthermore, it can be observed from Figs. 23, 24 that buffer area and switch area comparison give similar results for SET II benchmarks as observed for SET I benchmarks. It can be seen from these figures that average buffer area gain, switch area gain for hybrid FPGA is 25%, 60% respectively. These trends eventually lead to routing area, total area results of Figs. 25, 26 respectively where hybrid FPGA requires, on average, 60% smaller routing area and 59% smaller total area as compared to conventional FPGA.

Different area comparison results for individual netlists of SET I and SET II are shown in Figs. 17–26. Results shown in these figures indicate that hybrid FPGA significantly outperforms conventional FPGA. Apart from individual netlist results, we present combined area comparison results for SET I and SET II also. We know that FPGAs are pre-fabricated devices and if an FPGA satisfies the requirements of largest benchmark of a set then it can satisfy the requirements of other benchmarks of the suite too. Keeping this in mind, we define generic FPGA architecture for SET I and SET II benchmarks respectively. These generic FPGAs can place and route all the benchmarks of their respective sets and their corresponding area results are shown in Fig. 27. It can be seen from this figure that similar to previous results, hybrid FPGA gives 58% area gain for SET I and 59% area gain for SET II benchmarks respectively.

7. Conclusion

In this work, a comprehensive area comparison between a novel memristor-transistor hybrid heterogeneous FPGA and conventional heterogeneous FPGA is presented. Basic building blocks of hybrid heterogeneous FPGA are designed using HSPICE and their area is estimated using 180 nm processing technology. Area comparison between hybrid and conventional transistor-only blocks shows that hybrid blocks require significantly less area as compared to conventional blocks and their area gain ranges from 25% to 60%. Apart from individual block level comparison, we also perform complete architecture level comparison between hybrid and conventional heterogeneous FPGA architectures. For this purpose, we implement a complete exploration flow which is generalized in nature and gives complete experience both for conventional as well as hybrid FPGA architecture. This flow starts with the hardware description of the benchmark under consideration, transforms it into netlist, performs place&route on the target architecture, and eventually terminates with the area estimation of the architecture. The uniqueness of the proposed flow lies in the fact that it is completely built using state-of-the-art, open source tools and offers huge potential for future research on CAD flow for reconfigurable architectures. For experimental purpose, we use two sets of open core benchmarks. These benchmarks are placed and routed on the two architectures under consideration using the proposed exploration flow. Experimental results show that hybrid FPGA architecture produces much better area results as compared to conventional FPGA architecture. Comparison analysis shows that hybrid FPGA requires, on average, 58%, 59%, 59% less logic area, routing area, total area for SET I benchmarks respectively and these gain are 56%, 60%, and 59% for SET II benchamrks.

References

Almurib, H., Kumar, T., Lombardi, F., 2014. A memristor-based lut for FPGAs. In: 2014 9th IEEE International Conference on Nano/Micro Engineered and Molecular Systems (NEMS), pp. 448–453.
Aslam, M.H., Farooq, U., Awais, M.N., Bhatti, M.K., Shehzad, N., 2016. Exploring the effect of lut size on the area and power consumption of a novel memristor-transistor hybrid FPGA architecture. Arab. J. Sci. Eng. 41, 3035–3049.
Batas, D., Fiedler, H., 2011. A memristor spice implementation and a new approach for magnetic flux-controlled memristor modeling. IEEE Trans. Nanotechnol. 10, 250–255.
Beauchamp, M., Hauck, S., Underwood, K., Hemmert, K., 2006. Embedded floating-point units in FPGAs. In: Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays. ACM, New York, NY, USA, pp. 12–20.
Betz, V., Rose, J., Marquardt, A. (Eds.), 1999. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers, Norwell, MA, USA.
Bi, Z., Zhang, Y., Xu, Y., 2012. Memristor Working Condition Analysis Based on SPICE Model. Springer, Berlin Heidelberg, pp. 242–252.
Biolek, D., Biolkova, V., Biolek, Z., 2009. Spice model of memristor with nonlinear dopant drift. Radioengineering.
Biolek, D., Ventra, M.D., Pershin, Y.V., 2013. Reliable spice simulations of memristors, memcapacitors and meminductors. Radioengineering 22, 945–968.
B. University of California, 1992. Berkeley logic synthesis and verification group, berkeley logic interchange format (blif). http://vlsi.colorado.edu/vis/blif.ps.
Chua, L., 1971. Memristor-the missing circuit element. IEEE Trans. Circuit Theory 18, 507–519.
Cong, J., Xiao, B., 2011. mrFPGA: a novel FPGA architecture with memristor-based reconfiguration. In: 2011 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), pp. 1–8.
DeHon, A., 2001. Rent's Rule Based Switching Requirements. System Level Interconnect Prediction Workshop.
Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271.
El-Slehdar, A., Fouad, A., Radwan, A., 2015. Memristor based n-bits redundant binary adder. Microelectron. J. 46, 207–213.
Farooq, U., Aslam, M., 2015. Design and implementation of basic building blocks of FPGA using memristor-transistor hybrid approach. In: Fifth International Conference on Innovative Computing Technology (INTECH), pp. 142–147.
Farooq, U., Parvez, H., Mehrez, H., Marrakchi, Z., 2011. Exploration of heterogeneous FPGA architectures. Int. J. Reconfig. Comput. 2011. 2:1–2:18.
Farooq, U., Bhatti, M.K., Aslam, M.H., 2016. A novel heterogeneous FPGA architecture based on memristor-transistor hybrid approach. In: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS). IEEE, pp. 1–6.
Fey, D., 2014. Using the multi-bit feature of memristors for register files in signed-digit arithmetic units. Semicond. Sci. Technol. 29, 104008.
Guckert, L., Swartzlander, E.E., 2017. Mad gates memristor logic design using driver circuitry. IEEE Trans. Circuits Syst. II: Express Briefs 64, 171–175.
Hamdioui, S., Xie, L., Nguyen, H.A.D., Taouil, M., Bertels, K., Corporaal, H., Jiao, H., 1718. Memristor based computation-in-memory architecture for data-intensive applications. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1718–1725.
Jamieson, P., Rose, J., 2006. Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters. IEEE International Conference on Field-Programmable Technology (ICFPT), 1–8.
Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220, 671–680 .
Kuon, I., Tessier, R., Rose, J., 2008. FPGA architecture: survey and challenges. Found. Trends Electron. Des. Autom. 2, 135–253.
Kvatinsky, S., Kolodny, A., Weiser, U., Friedman, E., 2011. Memristor-based imply logic design procedure. In: 2011 IEEE 29th International Conference on Computer Design (ICCD), pp. 142–147.
Kvatinsky, S., Wald, N., Satat, G., Kolodny, A., Weiser, U., Friedman E., 2012. Mrl memristor ratioed logic. In: 2012 13th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA). IEEE, pp. 1–6.
Kvatinsky, S., Belousov, D., Liman, S., Satat, G., Wald, N., Friedman, E., Kolodny, A., Weiser, U., 2014. Magic memristor aided logic. IEEE Trans. Circuits Syst. II: Express Briefs 61, 895–899.

Luu, J., Kuon, I., Jamieson, P., Campbell, T., Ye, A., Fang, W.M., Rose, J., 2009. VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling. FPGA, 133–142.

Mane, P.S., Paul, N., Behera, N., Sampath, M., Ramesha, C.K., 2014. Hybrid cmos – memristor based configurable logic block design. in: 2014 International Conference on Electronics and Communication Systems (ICECS), pp. 1–5.

Marquart, A., Betz, V., Rose, J., 1999. Using cluster-based logic block and timing-driven packing to improve FPGA speed and density. In: Seventh International Symposium on FPGA, Monterey, pp. 37–46.

McMurchie, L., Ebeling, C., 1995. Pathfinder: a negotiation-based performance-driven router for FPGAs. In: ACM International Symposium on Field-Programmable Gate Arrays. ACM Press, New York, NY, USA, pp. 111–117.

Pangracious, V., Mehrez, H., Beltaief, N., Marrakchi, Z., Farooq, U., 2013. Exploration environment for 3d heterogeneous tree-based FPGA architectures (3d ht-FPGA). In: 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 1–6.

Pershin, Y., Di Ventra, M., 2010. Practical approach to programmable analog circuits with memristors. IEEE Trans. Circuits Syst. I: Reg. Papers 57, 1857–1864.

Rák, Á., Cserey, G., 2010. Macromodeling of the memristor in spice. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29, 632–636.

Sarwar, S., Saqueb, S., Quaiyum, F., Rashid, A.-U., 2013. Memristor-based nonvolatile random access memory: hybrid architecture for low power compact memory design. IEEE Access 1, 29–34.

Scaramuzzo, J., 2014. The flash transformed data center. In: Fifth Annual Non-Volatile Memories Workshop.

Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K., Sangiovanni-Vincentelli, A.L., 1992. SIS: a system for sequential circuit analysis. Technical Report UCB/ ERLM92/41. University of California, Berkeley.

Strukov, D., Mishchenko, A., 2010. Monolithically stackable hybrid FPGA. In: 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010), pp. 661–666.

Strukov, D.B., Snider, G.S., Stewart, D.R., Williams, R.S., 2008. The missing memristor found. Nature 453, 80–83.

Underwood, K., Hemmert, K., 2004. Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance. In: 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004. FCCM 2004, pp. 219–228.

Vourkas, I., Sirakoulis, G.C., 2014. Incorporating memristors in currently established logic circuit architectures: design considerations and challenges. In: workshop on Memristor Technology, Design, Automation and Computing (MemTDAC) affiliated with the HiPEAC 2014 conference.

Wang, W., Jing, T., Butcher, B., 2010. FPGA based on integration of memristors and cmos devices. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1963–1966.

Williams, R., 2008. How we found the missing memristor. IEEE Spectrum 45, 28–35.