

# Technology-Optimized Fixed-Point Bit-Parallel Multipliers for FPGAs

Burhan Khurshid<sup>1</sup>

Received: 25 January 2016 / Revised: 31 August 2016 / Accepted: 14 October 2016 / Published online: 24 October 2016  
© Springer Science+Business Media New York 2016

**Abstract** Modern day field programmable gate arrays have carry-chains and look-up tables as the basic logic elements. Efficient realization of different arithmetic circuits on FPGAs demands an improved mapping of different functionalities onto these logic elements. A majority of the work related to the implementation of different arithmetic circuits on FPGAs focusses on the architectural optimization that can be carried out at the top level. While this works well for ASICs, the performance on FPGAs may be degrading due to the poor mapping of the architecture onto the underlying FPGA resources. In this paper, we present technology-optimized fixed-point bit-parallel multiplier structures. The multipliers are technology-optimized by re-structuring the initial Boolean networks and transforming them into an optimized circuit net-list that utilizes the target elements efficiently. A detailed theoretical and experimental analysis of our implementation using Xilinx FPGAs shows a subsequent speed-up in performance when compared to the conventional realizations. We have also compared our implementation against various technology-independent realizations reported in prior literature. The idea is to provide a clear cut analysis about the performance speed-up that is achievable through technology-dependent approaches.

**Keywords** Circuit optimization · Field programmable gate arrays · Fixed-point bit-parallel multipliers · Technology-dependent optimizations · Technology mapping

---

✉ Burhan Khurshid  
burhan\_07phd12@nitsri.net

<sup>1</sup> Department of Computer Science and Engineering, NIT, Srinagar, India

## 1 Introduction

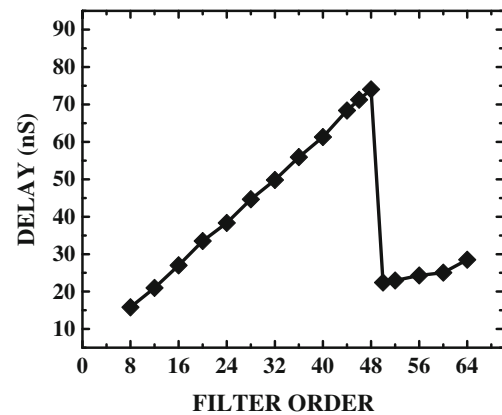
Fixed-point multiplication is frequently used in many digital signal processing (DSP) applications like filtering, Fourier transformations, convolution etc. [1–3]. Bit-parallel implementation styles are often preferred because of their ability to provide high throughput. Traditional implementations, however, have mostly been processor oriented and the design process focusses on developing the high-level code for a fixed processor architecture. This severely limits the evolution of the underlying architecture. For increased performance various platform oriented solutions like application specific integrated circuits (ASICs) and structural ASICs have been used [3]. These permit the evolution of the architecture to specifically meet the performance requirement of the intended application [4]. However, the huge non-recurring engineering (NRE) costs associated with ASICs has typically cornered their use for special application domains only [4].

Very recently FPGAs have been used as an alternative to ASICs. FPGAs avoid high NRE costs by giving designers the flexibility of configuring the device in field [5, 6]. With features like large scale integration [5, 7], lower energy requirement using same process technology, availability of several on-board macros and intellectual property (IP) cores [8] etc., FPGAs are fast moving into low and medium volume productions [1, 9, 10]. Look-up tables (LUT) and carry-chains form the basic logic elements in modern FPGAs [11–13]. An LUT is a  $k$ -input block RAM that can implement any Boolean function of  $k$  variables. State-of-art FPGAs support 6-input LUTs with the capability of implementing a 6-input Boolean function or two 5-input Boolean functions with shared inputs [14–16]. Carry-chains enable fast and efficient realization of arithmetic circuits by avoiding the general FPGA routing [17, 18]. Modern FPGAs also support full custom processing elements like multipliers, DSP blocks, high speed clocking and

I/O resources etc. [19]. The use of these optimized blocks can lead to efficient implementation of arithmetic functions [20, 21].

Whilst the pre-fabricated aspects of FPGAs have many advantages, it limits the optimized synthesis of different circuits on these platforms. Traditional approaches mainly focus on architectural modifications that can be carried out at the top level. The effectiveness of these approaches, which are generally well suited for ASICs is limited in FPGAs. The main reason for this difference is the pre-fabricated nature of the FPGAs and the fixed routing structures, which does not allow, otherwise architecturally optimized systems to map efficiently on them. The use of specialized custom blocks enhances the performance, but these are always limited in number and once that limit is reached the performance degrades rapidly. For example, Xilinx Virtex-5 xc5vlx50 has 48 DSP48E slices which can be used to perform multiply-accumulate (MAC) operation. A direct Finite Impulse Response (FIR) filter implementation based on the use of Generic Multiplier (GM) will utilize these DSP slices and an optimum performance is observed. However, as the filter order increases beyond 48, the DSP48E slices are used up and there is a rapid dip in the performance. This is shown in Fig. 1 where number of slices (a) and power dissipation (b) are plotted against the filter order. Another issue with hard IP cores and macro blocks is that they are fixed in the FPGA fabric. This increases the cost of routing data to and from these blocks [22]. Figure 2 shows a plot of the critical path delay against the filter order for a GM based direct FIR filter. For filter orders greater than 48, general LUT fabric is used whose position can be easily altered during the Placement & Route (PAR) phase of the FPGA design cycle, resulting in the reduction of the overall critical path delay. For filter orders less than 48, DSP48E blocks are used and owing to their fixed position longer critical paths are experienced.

The aim is, thus, to utilize the huge processing resources in the most efficient manner, so that the solution can out-perform other available platforms. This not only requires the development of a suitable architecture at the top



**Figure 2** Variation in critical path delay with filter order.

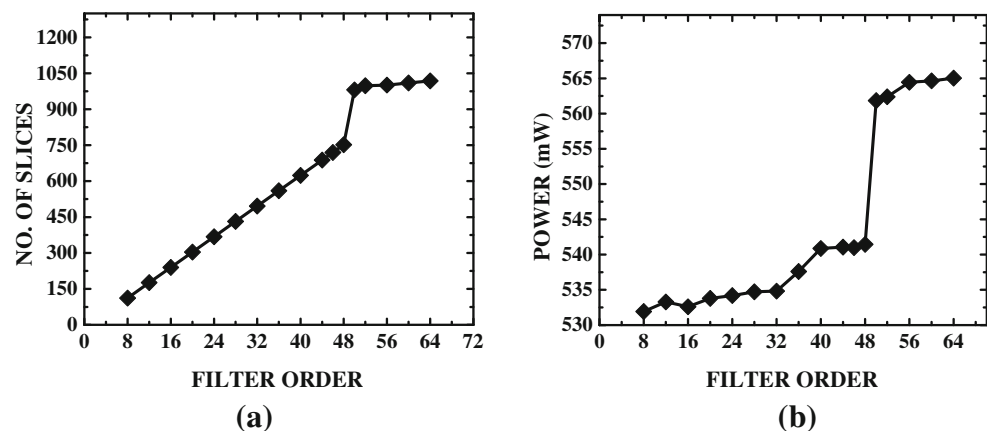
level, but also how efficiently the architecture is mapped on to the underlying FPGA fabric. In this paper, we consider technology-dependent approaches for efficient realization of multiplier circuits. We propose a general heuristic for efficiently mapping the Boolean networks onto 6-input LUTs. Based on this heuristic technology-dependent optimization of fixed-point bit-parallel multipliers is carried out.

The rest of the paper is organized as follows: Section 2 briefly discusses fixed-point bit-parallel multipliers used in this paper. Section 3 discusses some basic terminology used in this paper. Section 4 discusses the proposed heuristic for efficiently mapping Boolean networks onto 6-input LUTs. Based on this heuristic, technology-dependent optimization of three different bit-parallel multipliers is discussed in section 5. Synthesis and implementation is carried out in section 6. Conclusions are drawn in section 7 and references are listed at the end.

## 2 Fixed-Point Bit-Parallel Multipliers

A variety of architectures have been proposed to improve the performance of multiplication operation in DSP systems

**Figure 1** Variation in **a** Number of slices and **b** Power dissipation with filter order.



[23–28]. These mainly follow three implementation styles viz. bit-parallel, bit-serial and digit-serial. Bit-parallel systems are ideal for high-speed applications. Since modern day FPGAs are mainly speed oriented [29, 30], multiplier architectures falling in this category have been considered in this work.

Three widely used bit-parallel multipliers have been considered for technology-dependent optimizations. These include parallel Ripple Carry Array (RCA) multipliers, Parallel Carry-Save Array (CSA) multipliers and Carry-Save (CS) based Baugh-Wooley (BW) multipliers. The operands in each case are assumed to be in fixed-point 2’s complement representation. Therefore,  $N$ -bit operands  $X$  and  $Y$  may be represented as under:

$$X = x_{N-1} \cdot x_{N-2} x_{N-3} \dots x_1 x_0 \tag{1}$$

$$Y = y_{N-1} \cdot y_{N-2} y_{N-3} \dots y_1 y_0 \tag{2}$$

The most significant bit in each case is the sign bit with ‘0’ denoting a positive number and ‘1’ a negative number. The magnitude of these numbers lies in the range  $[-1, 1)$  and is given by:

$$X = -x_{N-1} + \sum_{i=1}^{N-1} x_{N-1-i} 2^{-i} \tag{3}$$

$$Y = -y_{N-1} + \sum_{i=1}^{N-1} y_{N-1-i} 2^{-i} \tag{4}$$

This type of representation is apt for DSP operations like filtering where the filter coefficients that model the frequency behavior of the signal have values in the range  $-1$  to  $1$ . The value of the product  $P = X \times Y$  is given by:

$$P = -p_{2N-2} + \sum_{i=1}^{2N-2} p_{2N-2-i} 2^{-i} \tag{5}$$

The product  $P$  may be represented as:

$$P = p_{2N-2} \cdot p_{2N-3} p_{2N-4} \dots p_1 p_0 \tag{6}$$

In constant word-length multiplication, the  $N-1$  lower order bits in the product  $P$  are truncated, and the product is given by:

$$Z = -z_{N-1} + \sum_{i=1}^{N-1} z_{N-1-i} 2^{-i} \tag{7}$$

The constant word length product  $Z$  may be represented as:

$$Z = z_{N-1} \cdot z_{N-2} z_{N-3} \dots z_1 z_0 \tag{8}$$

The product  $Z$ , therefore, is not a full-precision product [31]. Complete details about truncated multiplication can be found in [32–34]. Based on the concept of constant word-length multiplication, three traditional approaches to bit-

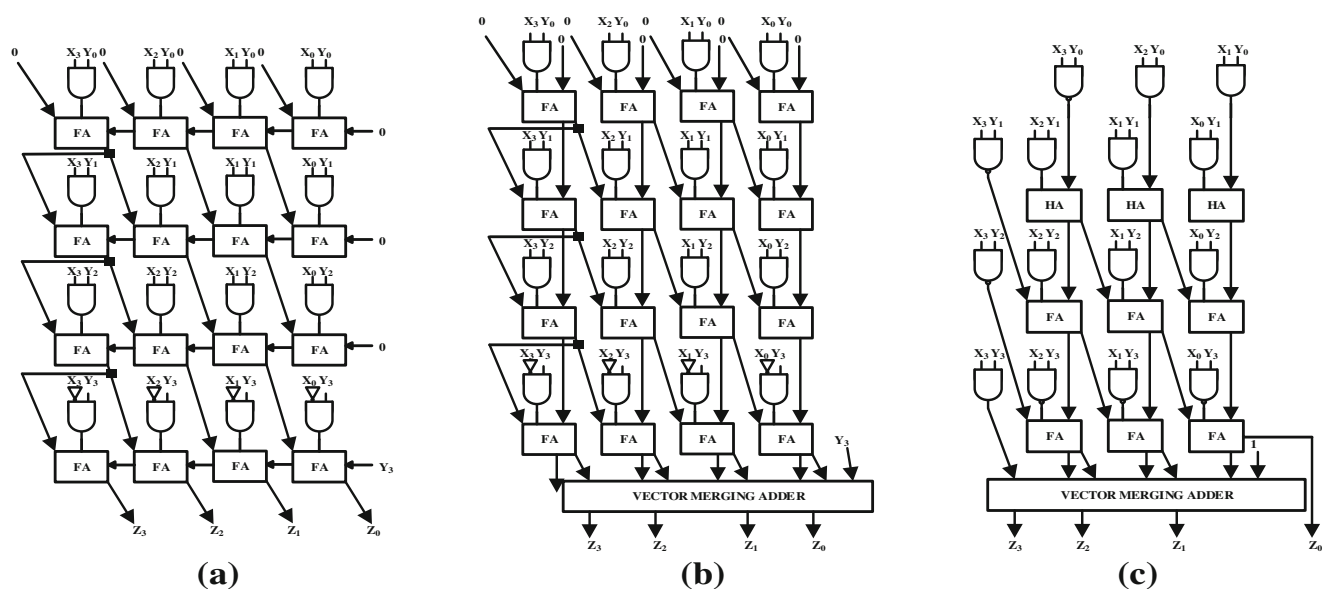


Figure 3 Bit-parallel fixed-point multipliers. a RCA multiplier. b CSA multiplier. c BW multiplier.

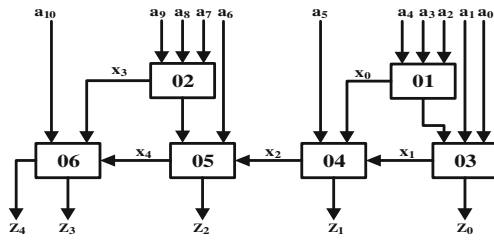


Figure 4 A generalized Boolean Network.

parallel fixed-point multiplication have been used. The complete details about these multipliers are given in [31, 35]. Brief description about these multipliers is provided below:

### 2.1 RCA Multiplier

RCA multiplier operates by rippling the carry to the adder to the left in the same row. Thus, within a row each adder has to wait for the carry input to perform its computation. In other words, there exists an intra-iteration constraint between any two adjacent adder nodes within a row, assuming there is no pipelining involved. Owing to this ripple-carry nature the critical paths involved are quite large which limits the speed of multiplication. Figure 3(a) shows the schematic for a 4-bit RCA fixed-point multiplier.

### 2.2 CSA Multiplier

In CSA multiplier, the carry outputs are saved and used in the adder in the next row. In this case, the partial product is replaced by a partial sum and a partial carry, which are saved

and passed on to the next row. The advantage of carry-save multiplication is that the additions at different bit positions in the same row are now independent of each other and can be carried out in parallel, which essentially speeds up the addition phase of each cycle, and hence speeds up the multiplication. The addition of the partial sum and the partial carry at the last step is performed by a Vector Merging Adder (VMA), which may be implemented either as a ripple-carry adder or a carry-save adder. Figure 3(b) shows the schematic for a 4-bit CSA fixed-point multiplier. The VMA details are not shown in the figure.

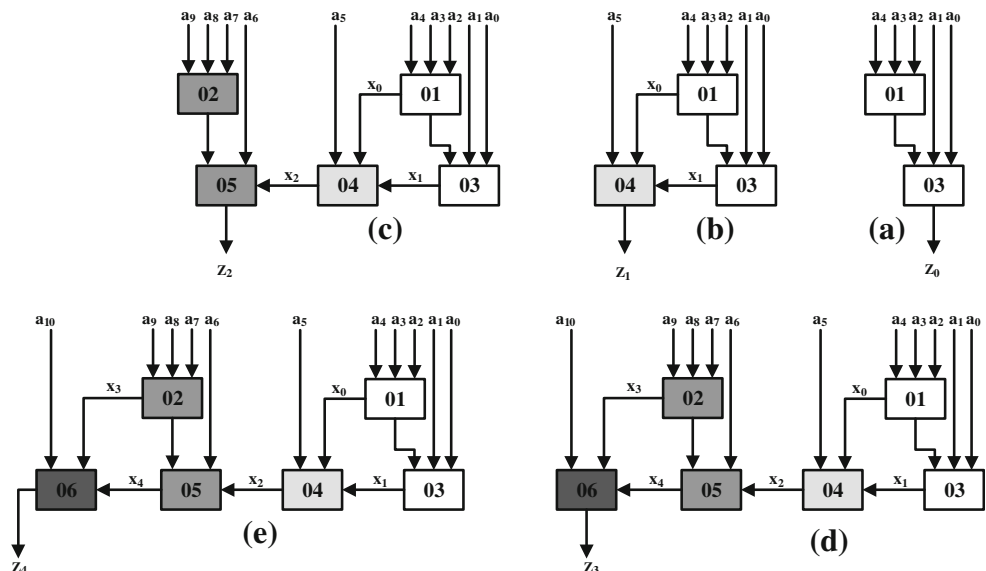
### 2.3 BW Multiplier

The difficulty of 2’s complement multiplication lies in handling the sign bits of the multiplicand and multiplier. An efficient way to overcome this problem is provided by the BW multiplication algorithm. The algorithm is an efficient way to handle the sign bits and helps in designing regular multipliers using 2’s complement operands. The BW multiplication may be implemented as either a carry-ripple array or a carry-save array. Figure 3(c) shows the schematic for a 4-bit BW multiplier implemented as a carry-save array.

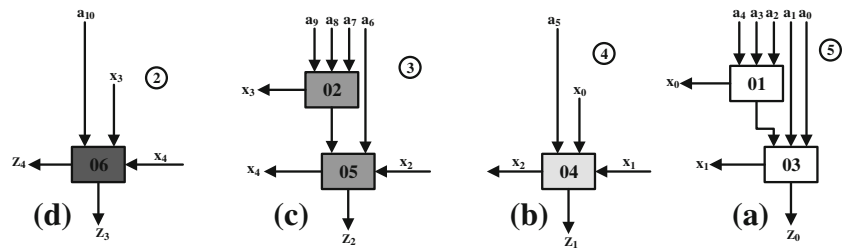
## 3 Preliminary Terminologies

Logic synthesis is concerned with realizing a desired functionality with minimum possible cost. In the context of general digital design the *cost* of a circuit is a metric which provides an idea about the speed, area consumption or power dissipation of a circuit. In some cases, the

Figure 5 Boolean networks for individual outputs. Repeated nodes are shown in same shades.



**Figure 6** Boolean networks for repeated nodes. The number in the circle represents the priority of each network.



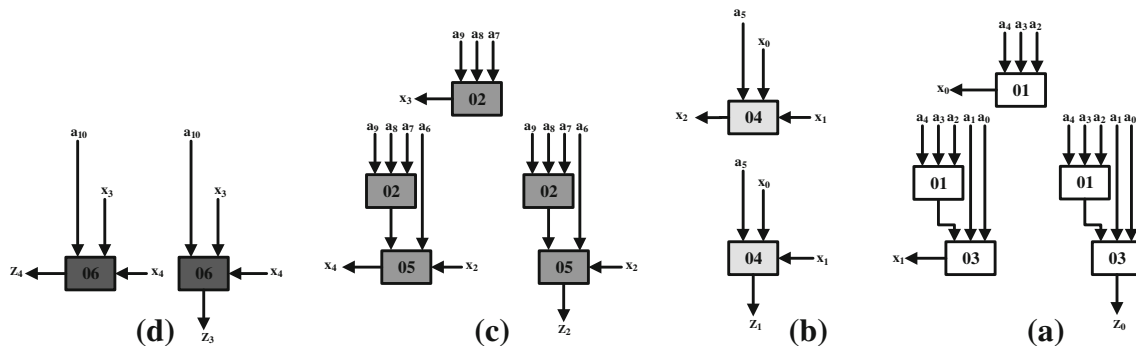
cost may be expressed as a combination of these parameters. A combinational function may be represented graphically as a directed acyclic graph (DAG) called the *Boolean network*. Individual *nodes* within this network may represent logic gates, primary inputs (PIs) or primary outputs (POs). The Boolean networks considered in this work are very simple, therefore, actual gates have been used for nodes in most of the cases. Each node implements some *local* function and together with its predecessor nodes implements a *global* function. A node in a Boolean network may be driven by one or more predecessor nodes known as *fan-in* nodes. Similarly a node may drive one or more successor nodes known as *fan-out* nodes. The PIs of a Boolean network are the nodes without any fan-in. Similarly the POs are the nodes without any fan-out.

A *cone* of a node  $v$ ,  $C_v$ , is a sub-graph that includes the node  $v$  and some of its non-PI predecessor nodes. Any node  $u$ , within this sub-graph has a path to the node  $v$ , which lies entirely in  $C_v$ . The node  $v$  is called the root of the cone  $C_v$ . The *level* of the node  $v$  is the length of the longest path from any PI node to node  $v$ . If node  $v$  is a PO node then the level will give the *depth* of the Boolean network. Thus network-depth is the largest level of a node in the network. A net-list of connected LUTs and carry-chains obtained after transformation of the Boolean network is called a *circuit*. The critical path and resource utilization of a circuit is measured by the depth and number of utilized LUTs, respectively.

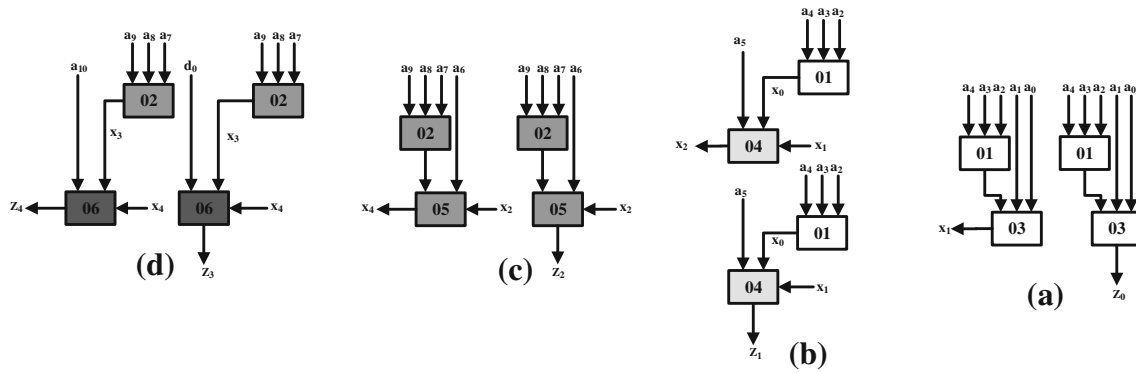
### 4 General Heuristic for Technology-Dependent Optimization

Technology-dependent optimization transforms the initial Boolean network into a circuit netlist that utilizes the target logic elements efficiently. The transformation should be optimal i.e. the logic distribution among the targeted elements should ensure minimum possible depth and minimum resource utilization. The target element in majority of FPGAs is  $k$ -input LUT [11–13]. It is a block RAM that can implement any Boolean function of  $k$  variables by directly storing its truth table. State-of-art FPGAs support 6-input LUTs with the capacity of implementing a single 6-input Boolean function or two 5-input Boolean functions that share inputs [14–16]. An efficient utilization of this circuit element could lead to higher logic densities resulting in a reduced fan-out of the logic nets and thus a minimal-depth circuit. Apart from the general LUT fabric, most of the FPGAs also support a dedicated carry-chain in each of the logic slices [17, 18]. The inclusion of this primitive makes the carry propagation fast as the general FPGA routing is avoided.

Technology-dependent optimization using LUTs is a two-step process. First, the parent network is partitioned into suitable Boolean networks. Individual nodes within each network are then covered with suitable cones. The local function implemented by each cone is then mapped onto a separate LUT. In the second step, the entire parent network is constructed by assembling the individually optimized Boolean networks.



**Figure 7** Sub-networks for different eliminated networks.



**Figure 8** Re-structuring of eliminated networks for efficient utilization of LUTs.

The performance speed-up achieved from technology-dependent optimization may vary from network to network. Similarly, the optimization process may be different for different networks. We propose a general heuristic that may be followed towards the optimization process. The heuristic involves *construction* of different Boolean networks from the parent network; *identification* of repeated nodes from the constructed networks; *prioritization* of the identified networks; *covering* and *re-structuring* of the identified networks and finally *re-construction* of the parent network. We explain the different steps by considering the Boolean network of Fig. 4. Note that Fig. 4 presents a more generalized network having multiple inputs and multiple outputs. In actual arithmetic circuits, the networks are much simpler and some of the steps may not be required at all.

**4.1 Construction**

The first step constructs separate Boolean networks for each output from the parent network. This is done by traversing the parent network in post-order depth-first fashion and dividing it at the output nodes. For the parent network of Fig. 4, there are five output nodes. The network is traversed in a post-order depth first fashion and five separate networks corresponding to outputs  $Z_0, Z_1, Z_2, Z_3$  and  $Z_4$  are constructed. This is shown in Fig. 5.

**4.2 Identification**

After constructing individual networks, the search for repeated nodes begins in each of the networks. Different repeated nodes existing in networks of Fig. 5 are shown in the same shade. The network for repeated nodes is then drawn separately as shown in Fig. 6.

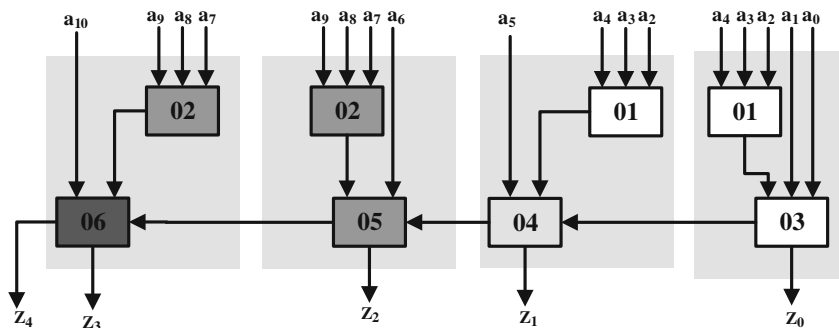
**4.3 Prioritization**

In this step each identified network is assigned a priority based on the number of appearances in the original networks. For example, the network in Fig. 6(a) is assigned a priority of 5, because it appears in five different networks. Similarly, the network in Fig. 6(b) is assigned a priority of 4 because it appears in four different networks and so on. Note that the entire parent network can be constructed by interconnecting these identified networks.

**4.4 Covering and Re-Structuring**

In this step, the individual nodes within the identified networks are covered with suitable cones which are later optimally mapped onto LUTs. Mapping is done as per the priority, as it results in the maximum logic

**Figure 9** Optimally mapped Boolean network.

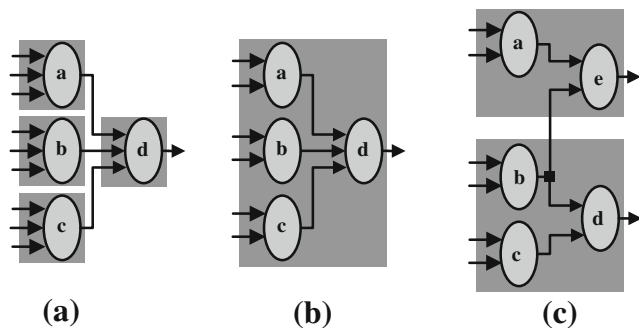




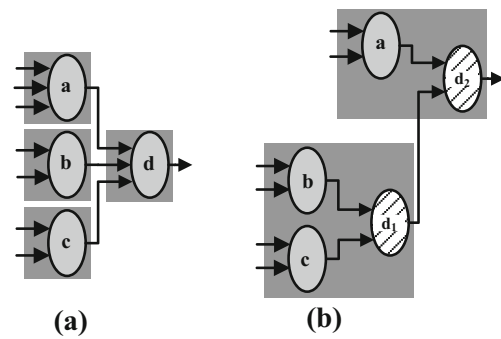
density. For example, the network in Fig. 6(a) has a priority of 5 and, if mapped optimally, will result in an improved logic density in all the networks it is a part of. For efficient mapping each network in Fig. 6 is divided into sub-networks. This is again done by traversing the network in post-order depth-first fashion and dividing it at output nodes. Thus the network of Fig. 6(a) is divided into three sub-networks corresponding to outputs  $X_0$ ,  $X_1$  and  $Z_0$ . Similarly, networks in 6(b), 6(c) and 6(d) are divided into different sub-network, as per their fan-out. This is shown in Fig. 7. A straight forward approach to mapping would be to assign the logic implemented by each sub-network to a separate cone and map it onto a separate LUT. This, however, leads to under-utilization of the resources. For efficient mapping, therefore, the entire assembly of sub-networks is re-structured. This requires transferring some sub-networks from their original networks and combining them with sub-networks that belong to different networks. For example, sub-network  $X_0$ , that originally belonged to 7(a) is now transferred to 7(b) and included with sub-networks  $X_2$  and  $Z_1$ . Similarly, the sub-network  $X_3$  is now included with sub-networks  $Z_3$  and  $Z_4$  that belongs to 7(d). This re-structuring of sub-networks ensures a proper utilization of the LUT fabric. The re-structured sub-networks are shown in Fig. 8. The re-structured sub-networks are then efficiently mapped onto 6-input LUTs by directly mapping their functionalities onto these target elements.

### 4.5 Re-Construction

The parent network is then re-constructed by connecting the mapped networks from step iv. The overall structure is a simple feed-forward structure having a unidirectional dataflow. The final mapped structure is shown in Fig. 9.



**Figure 10** Covering of nodes for networks with different number of inputs and outputs.

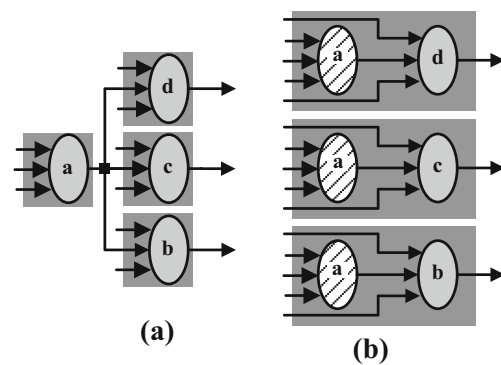


**Figure 11** Node Decomposition. **a** Normal covering. **b** Covering with node decomposition.

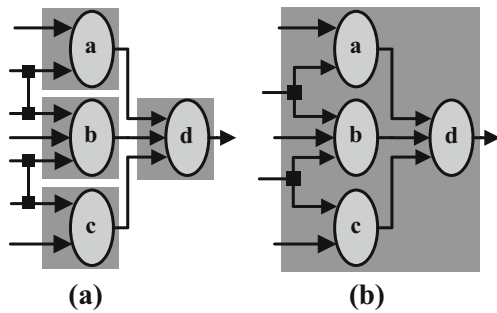
The covering and re-structuring process will determine the effectiveness of the technology-dependent optimization in improving the performance of the mapped circuit. The process is limited by the number of inputs and outputs in a network. For example, if we are targeting 6-input LUTs then the covering of the network in Fig. 10(a) will require separate LUTs for each node, as the total number of input exceeds six. However, for Fig. 10(b) the total number of inputs is six and the entire network can be mapped onto a single LUT. Similarly, in Fig. 10(c), although the number of inputs is six, the number of outputs exceeds one. In this case the number of outputs will determine the number of required LUTs. For effective covering and re-structuring the redundancies within the networks need to be exploited. In arithmetic circuits three types of redundancies may exist. These may be exploited to design high density minimum-depth circuits. We first discuss these redundancies and how they can be exploited to design efficient circuits.

### 4.6 Node Decomposition

Decomposition of a node into two or more nodes may lead to an effective combination of the nodes within a network. The decomposed nodes can be combined with other nodes and



**Figure 12** Node duplication. **a** Normal covering. **b** Covering with duplicated nodes.



**Figure 13** Re-convergent paths. **a** Normal covering. **b** Covering exploiting re-convergent paths.

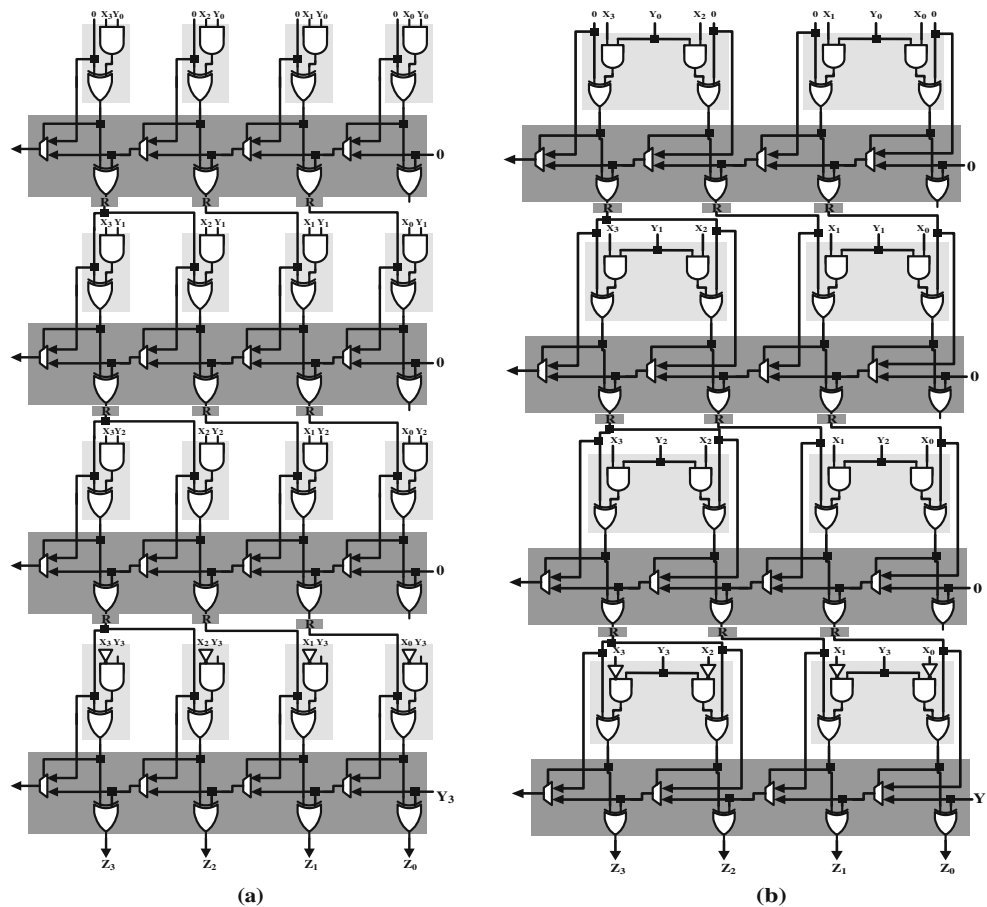
covered with fewer cones. For example, consider the network of Fig. 11(a). It consists of three input nodes  $a$ ,  $b$  and  $c$  and a single output node  $d$ . The input nodes drive the output node  $d$ . Normal covering will require a cone for each of the input and output nodes resulting in a network depth of two and a total LUT count of four. However, if the output node  $d$  is decomposed into two nodes  $d_1$  and  $d_2$ , and then combined with input nodes, a better covering is possible. This is shown in Fig. 11(b) where the node  $d_1$  is combined with nodes  $a$  and

$b$  and node  $d_2$  is combined with node  $c$ . With this realization the total LUT count is reduced to two. Node decomposition is usually employed for the output nodes.

### 4.7 Node Duplication

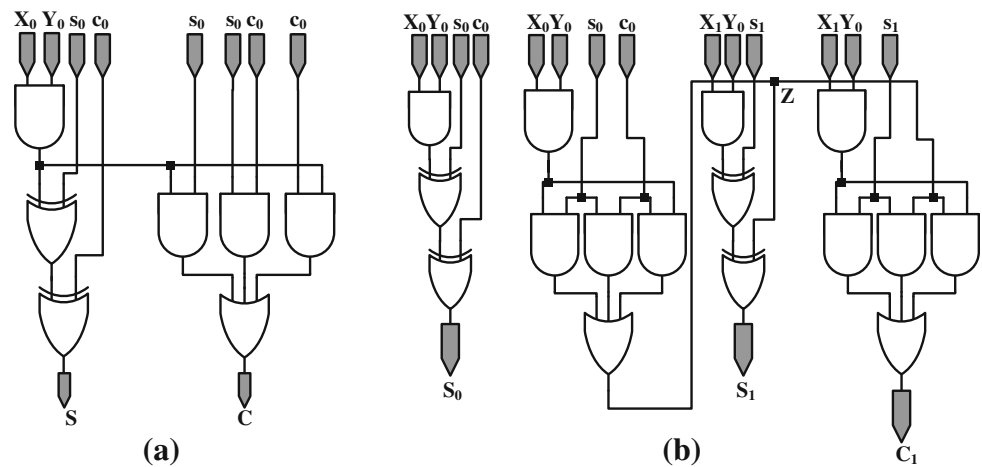
In some cases the output of a single node may be serving as input to multiple nodes within a network. In such cases the input node may be duplicated and covered separately along with the different nodes it serves. For example, in Fig. 12(a), input node  $a$  drives three output nodes  $b$ ,  $c$  and  $d$ . Normal covering will require a cone for each node resulting in an LUT count of four and a network depth of two. However, if the input node is duplicated and included separately with the output nodes  $b$ ,  $c$  and  $d$ , an efficient covering will result. This is shown in Fig. 12(b) where the input node  $a$  is included separately with output nodes  $b$ ,  $c$  and  $d$ . The total number of LUTs is reduced to three and the network depth is reduced to one. Node duplication is usually employed at the input nodes.

**Figure 14** LUT-Carry-chain based 4-bit RCA multiplier. **a** Traditional realization. **b** Modified realization.





**Figure 15** Boolean network for RCA multiplier. **a** Basic cell. **b** Two-bit cell.



**4.8 Re-Convergent Paths**

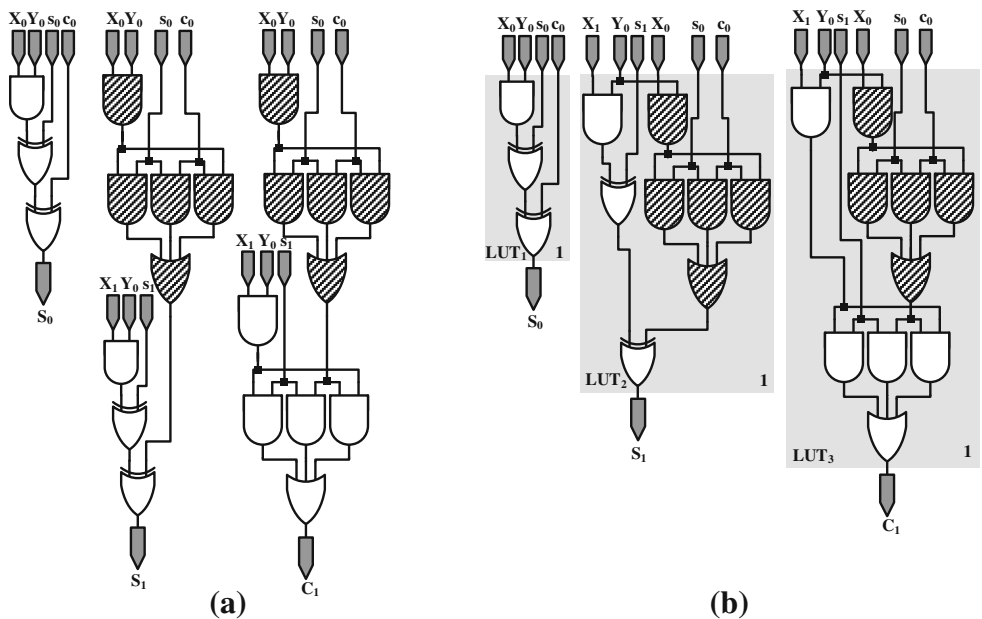
A node in the network with a fan-out greater than one, which terminates at other nodes within the same network, is a source of re-convergent path. Re-convergent paths can be realized within the LUT and the total number of inputs to a network can be reduced. For example the network of Fig. 13(a) has two re-convergent paths that can be included within a LUT. This will reduce the number of inputs from seven to five. A reduction in the number of inputs ensures that the entire network is covered using a single cone that can be mapped onto a single LUT. This is shown in Fig. 13(b) where the LUT count and network depth has been reduced to one.

**5 Technology-Dependent Optimization of Fixed-Point Bit-Parallel Multipliers**

**5.1 Bit-parallel Fixed-Point RCA Multiplier**

As mentioned previously an RCA multiplier has rows of ripple carry chains. Traditionally, these are implemented on FPGAs using a combination of LUTs and carry-chains. The carry-chain handles the rippling of carry in each row and the LUTs implement the associated logic. Such a realization is shown in Fig. 14(a) where a 4-bit fixed-point RCA multiplier is implemented using a combination of LUTs and carry-chains. Bit-parallel structures can be easily pipelined by inserting registers

**Figure 16** Mapping of two-bit cell. **a** Logic replication. **b** Optimized circuit using 6-input LUTs.



between different rows. This works well for FPGAs which have abundant unused registers. Pipelining the structure also ensures that a synchronous design practice is followed, which is always desirable for FPGA implementation.

The realization of Fig. 14(a) requires 16 LUTs and has a critical path limited by the delay of a single LUT and carry-chain. For an  $N \times N$  multiplier, therefore, the resource utilization would be  $N^2$  and the overall critical path delay for a pipelined implementation would be:

$$T_{CP} = T_L + NT_C \tag{9}$$

Where,

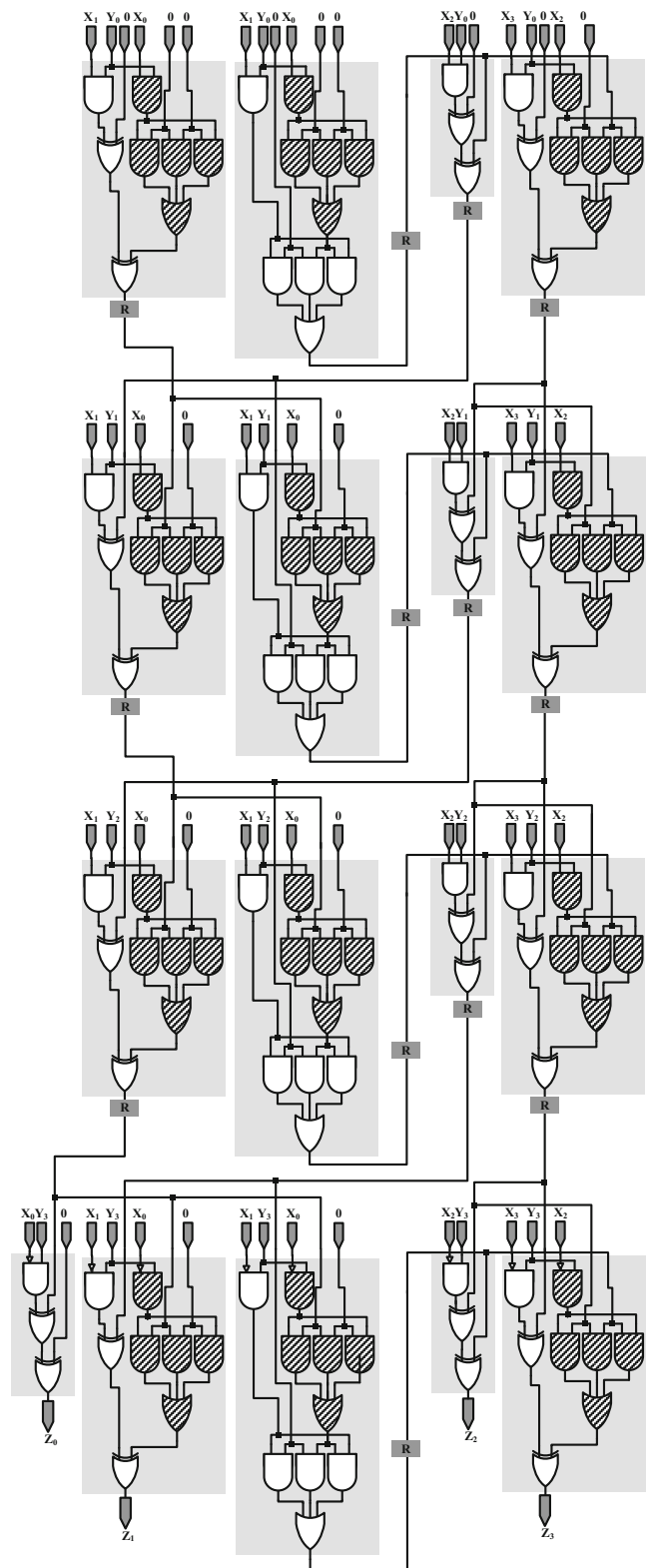
$T_L$  is the delay associated with a single LUT.

$T_C$  is the delay associated with a single carry.

In this section two technology-optimized bit-parallel RCA multipliers will be discussed. The first one is a modification of the traditional LUT-Carry-chain structure and results in reduced logic utilization and power dissipation. The second one is an LUT-Register based novel structure that results in a reduced critical path, at the expense of extra LUT cost. The structures are discussed in detail below:

### 5.1.1 LUT-Carry-Chain Based Bit-Parallel RCA Multiplier

The LUT-Carry-chain based bit-parallel RCA multiplier is similar to the traditional RCA multiplier. The carry-chain is again used to handle the rippling of carry in each row. The associated logic is re-structured to take advantage of the dual output capability of 6-input LUTs. Figure 14(b) shows the schematic of the modified 4-bit RCA multiplier. The total LUT count that implements the combinational logic is eight and the critical path delay is limited by the delay associated with a single LUT and carry-chain. An  $N \times N$  multiplier will require  $N^2/2$  LUTs to implement the combinational logic, which is 50 % less than the traditional realization. A reduction in the utilized logic also results in the reduction in dynamic power dissipation. This is indicated in the experimental results discussed in section 6. The critical path delay for a pipelined implementation will be the same as given by Eq. 9.



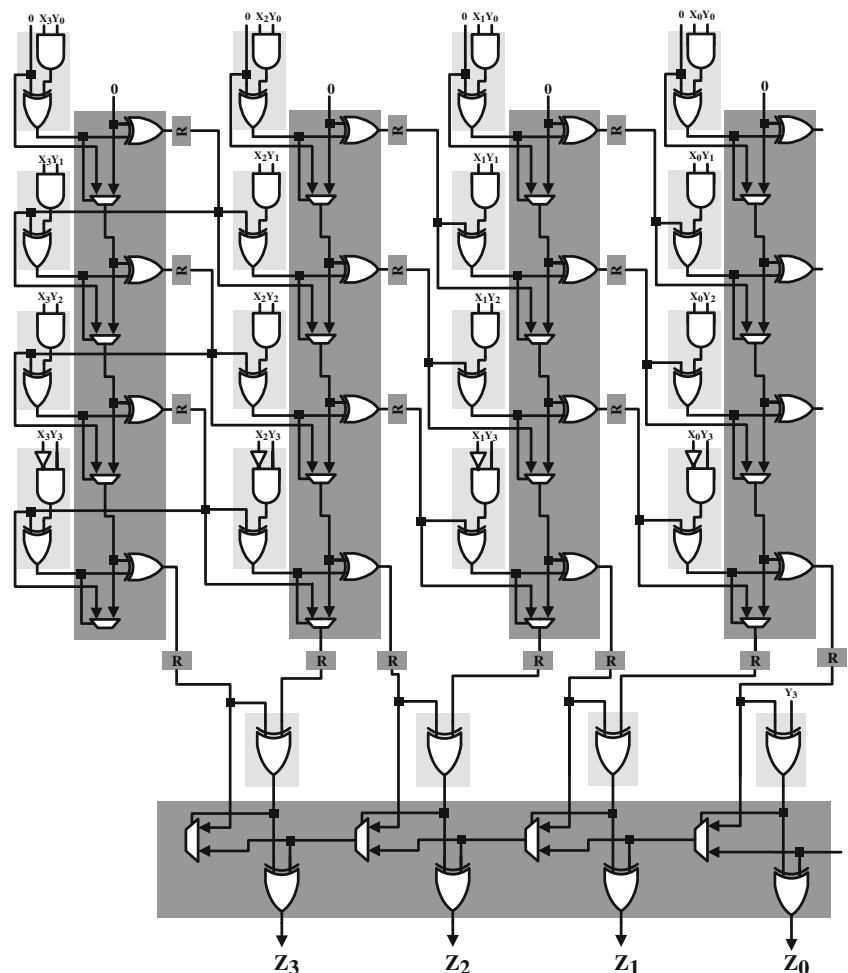
**Figure 17** Technology-optimized 4-bit RCA multiplier based on LUT-Register combination.

5.1.2 LUT-Register Based Bit-Parallel RCA Multiplier

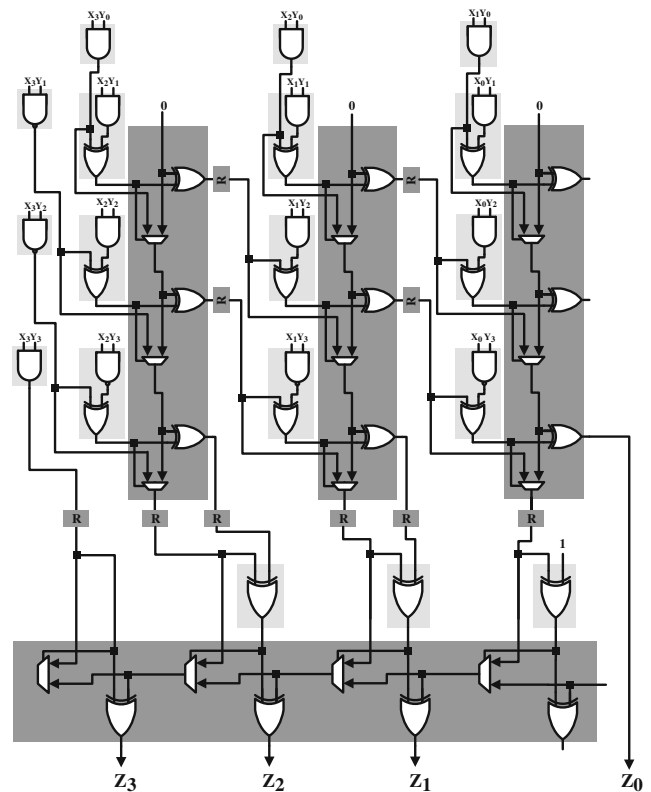
The LUT-Register based multiplier is a novel structure that utilizes a combination of LUTs and registers instead of the LUT-Carry-chain combination. Figure 15(a) shows the parent network for the basic cell used in an RCA multiplier. Two separate Boolean networks corresponding to the sum (S) and carry (C) outputs may be constructed from the parent network. Identification and prioritization steps may then be used to obtain separate networks. However, owing to the simplicity of the parent network, the re-structuring of the identified networks will lead back to the parent network of Fig. 15(a). Therefore, identification and prioritization steps are completely omitted and the covering of the parent Boolean network is directly considered. A straight forward approach would be to cover each node with a separate cone and then map the local function implemented by each cone onto a separate LUT. This, however, will lead to under-utilization of the available resources, thereby resulting in low density and increased in-

terconnect length in the final circuit implementation. Since interconnects in FPGAs are configurable switches, an increased interconnect length not only contributes to the overall combinational delay but also increases the switching activity along the paths. This may be hazardous for power-critical applications as the power dissipation in FPGAs is a strong function of the switching activity. It is thus desirable to have an implementation with high logic density and minimum possible routing. This issue is countered by considering the Boolean network for two multiplier cells simultaneously. Figure 15(b) shows the parent network corresponding to a two-bit multiplier cell. The network is partitioned into three Boolean networks corresponding to two sum bits  $S_0$  and  $S_1$  and a carry bit  $C_1$ . This is done by replicating the logic at the fan-out node Z. The replicated logic is shown as shaded portion and appears in both  $S_1$  and  $C_1$  networks. This is shown in Fig. 16(a). The individual networks are now mapped optimally onto suitable LUTs. Sub-network  $S_0$  computes the sum bit and utilizes one LUT. Sub-networks  $S_1$  and  $C_1$  are implement-

**Figure 18** Conventional LUT-Carry-chain based realization of a 4-bit CSA multiplier.

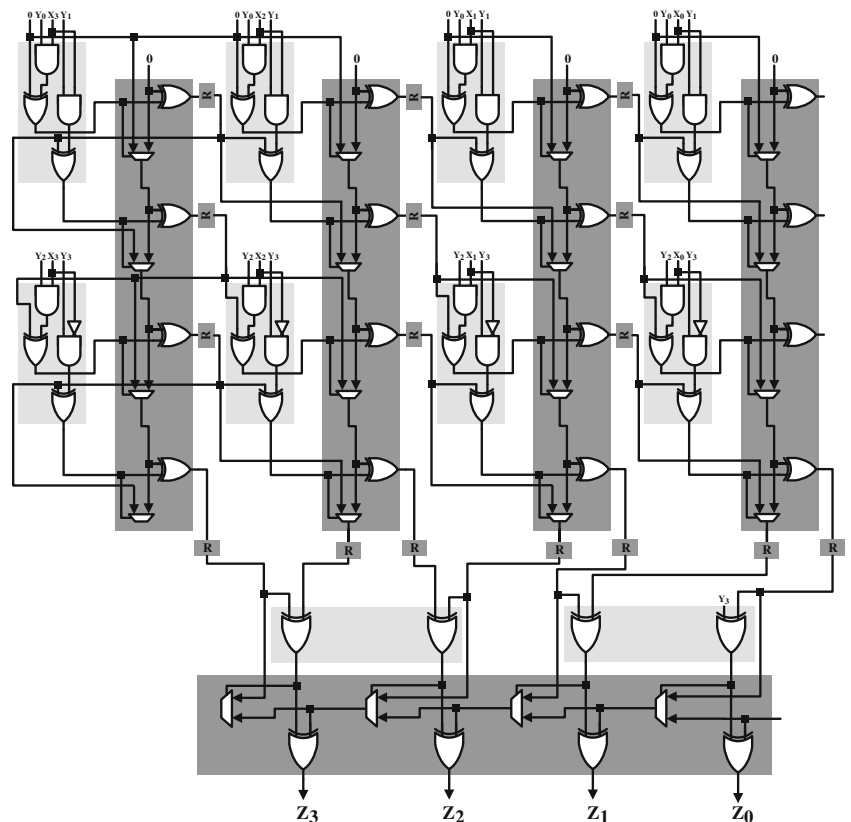


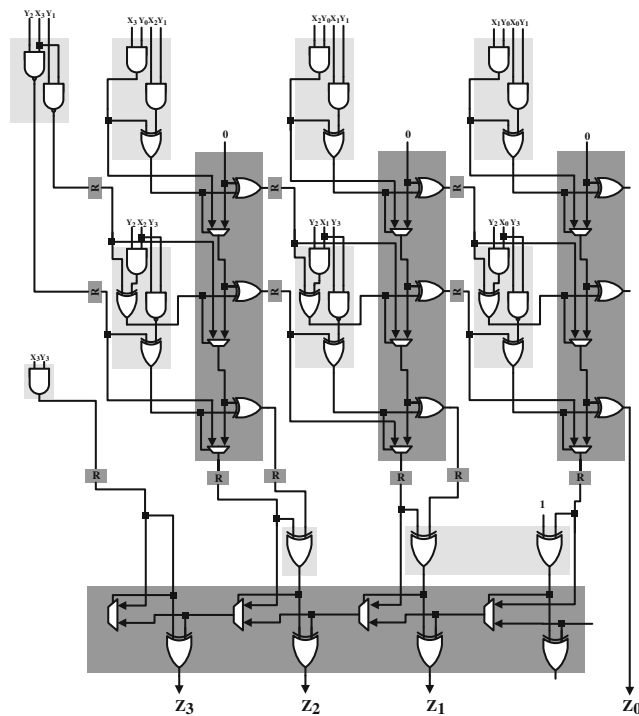
ed as 6-input functions and are mapped separately onto 6-input LUTs. The entire distribution is shown in Fig. 16(b). The overall multiplier unit is then realized by cascading the binary cells together. Such a realization is purely combinational in nature. It is observed that in each row the LUTs computing the first sum bit ( $S_0$ ) and last carry bit ( $C_{N-1}$ ) are not actually required, except for the last row where the sum bit is required. Thus, there is a reduction of two LUTs in each intermediate row and a reduction of one LUT in the last row. An  $N \times N$  multiplier based on the realization of Fig. 16(b) will thus require  $(3N^2/2) - 2N + 1$ , LUTs and will have a critical path delay that varies linearly with the operand word-length. The combinational realization can be easily pipelined by placing registers at the output of each binary cell. Such a realization is shown in Fig. 17. The pipelined realization breaks the data path from input to output which enables the multiplier to be clocked at higher frequencies. For correct operation, however, the inputs must remain constant during the period of operation. While this works fine for constant-coefficient multiplication, additional registers are required at the input if the data is varying in real-time. This is not a problem for FPGAs as there are abundant unused registers in each logic slice. Further, the use of input/output buffers also ensures that the data is held sufficiently long enough for correct operation.



**Figure 20** Conventional LUT-Carry-chain based realization of a 4-bit BW multiplier.

**Figure 19** Modified LUT-Carry-chain based realization of a 4-bit CSA multiplier.





**Figure 21** Modified LUT-Carry-chain based realization of a 4-bit BW multiplier.

### 5.2 Bit-parallel Fixed-Point Carry-Save Based Multipliers

Ripple-carry based architectures map well on FPGAs because of their inherent ability to utilize the fast carry-chains. Carry-save based architectures are not generally suited for FPGAs because there is no rippling of carry in these structures and the fast carry-chains remain unutilized. In this section, efficient mapping of fixed-point bit-parallel carry-save multipliers on FPGAs is discussed. It is shown that by proper restructuring and covering, carry-save based multipliers can be implemented on FPGAs with almost same performance as that of RCA multipliers. Two carry-save based multipliers have been considered; CSA multiplier and CS based BW multiplier. An important part of these multipliers is the VMA that

combines the partial sum and partial carry to generate the final result. Like with RCA multipliers, LUT-Carry-chain and LUT-Register realizations of these multipliers will be discussed.

#### 5.2.1 LUT-Carry-Chain Based Bit-Parallel Carry-Save Multipliers

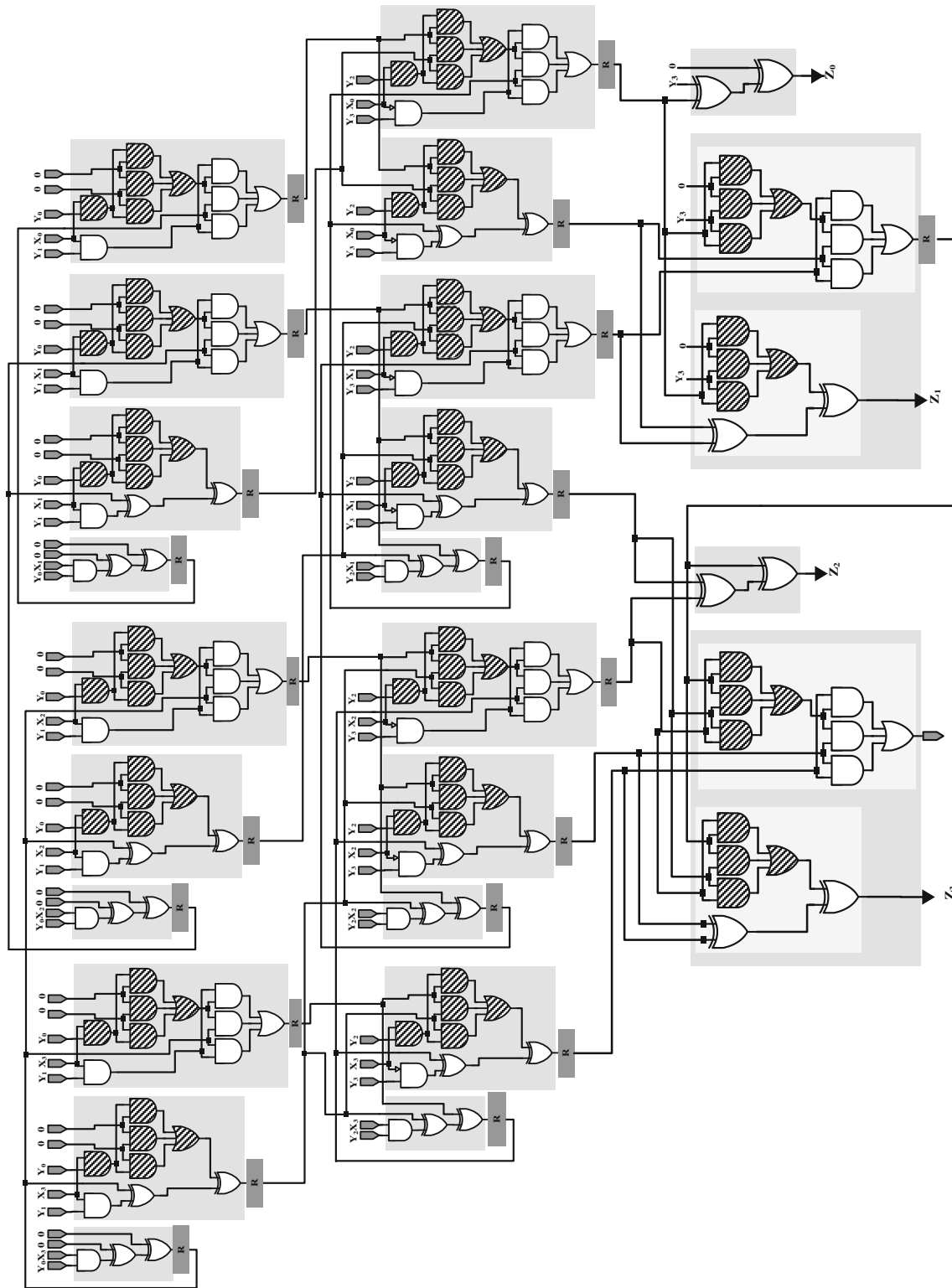
From the top level schematics of Fig. 3(b) and 3(c) it is observed that there is no rippling of carry within a row of carry-save multipliers. However, the carry does ripple along each column from one row to the next. Thus instead of placing the carry-chains horizontally along a row, they can be placed vertically along a column. The associated logic is again implemented using 6-input dual-output LUTs. Such an arrangement is shown in Figs. 18 and 19 where conventional and modified 4-bit CSA multipliers are implemented using LUTs and carry-chains. Figures 20 and 21 show the same realization for a CS based BW multiplier. The structures have approximately the same hardware cost as that of an RCA multiplier. The only overhead is that of the terminating VMA unit, which requires an additional carry-chain and LUTs to implement the associated logic. Resource utilization and critical path are listed in Table 1 for an  $N$ -bit operand word-length. Actual synthesis and implementation is discussed separately in section 6.

#### 5.2.2 LUT-Register Based Bit-Parallel Carry-Save Multipliers

The basic cell in carry-save based multipliers remains the same as in RCA multiplier. However, as mentioned previously, the carry ripples along a column from one row to the next. This requires a vertical arrangement of binary cells along a column. The final circuit net-list that combines the individual binary cells is thus different. This is shown in Figs. 22 and 23, where CSA and CS based BW multipliers are implemented, respectively, for an operand word-length of four bits. Since the basic

**Table 1** Performance comparison of different fixed-point bit-parallel multiplier schemes.

Multiplier Scheme	Design	No. of LUTs that implement logic	Critical path
RCA	Conventional LUT-Carry-chain	$N^2$	$T_L + NT_C$
	Modified LUT-Carry-chain	$N^2/2$	$T_L + NT_C$
	Proposed LUT-Register	$(3N^2/2)-2N + 1$	$T_L$
CSA	Conventional LUT-Carry-chain	$N(N + 1)$	$T_L + NT_C$
	Modified LUT-Carry-chain	$N(N + 1)/2$	$T_L + NT_C$
	Proposed LUT-Register	$3N^2/2$	$T_L$
BW	Conventional LUT-Carry-chain	$N(N + 1)-2$	$2T_L + NT_C$
	Modified LUT-Carry-chain	$N(N + 1)/2$	$T_L + NT_C$
	Proposed LUT-Register	$[3N(N-2)/2] + 5$	$T_L$

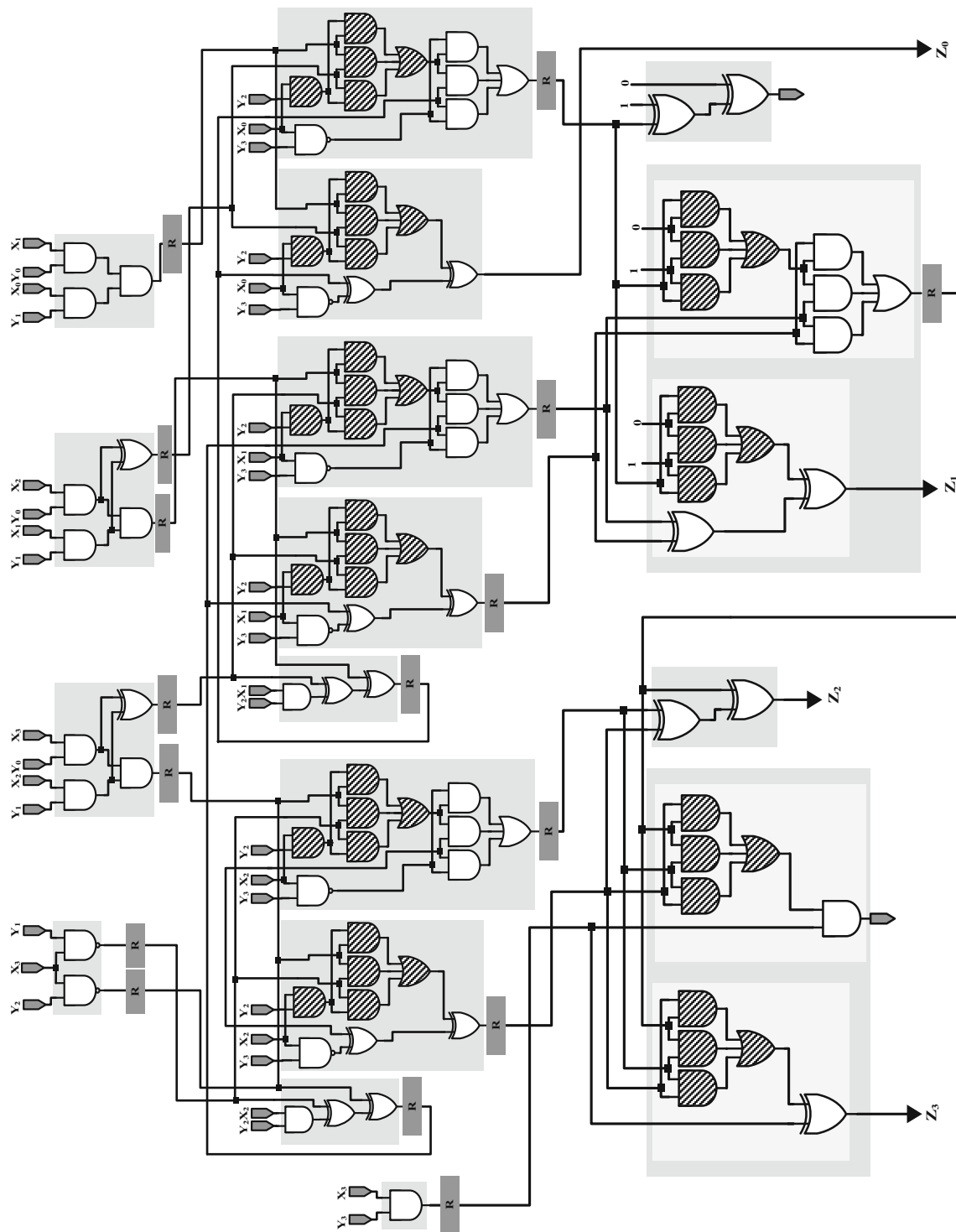


**Figure 22** Technology-optimized 4-bit CSA multiplier based on LUT-Register combination.

cell is the same as the one used in an RCA multiplier, the optimizations done at the lower level are preserved. However, there is some additional overhead owing to the VMA that generates the final product. The VMA is a simple binary adder

and can be realized using the same general heuristic as discussed in section 4. Figure 24 shows the parent network and the corresponding LUT net-list for the basic cell that is used to implement the VMA. Note that some modification is

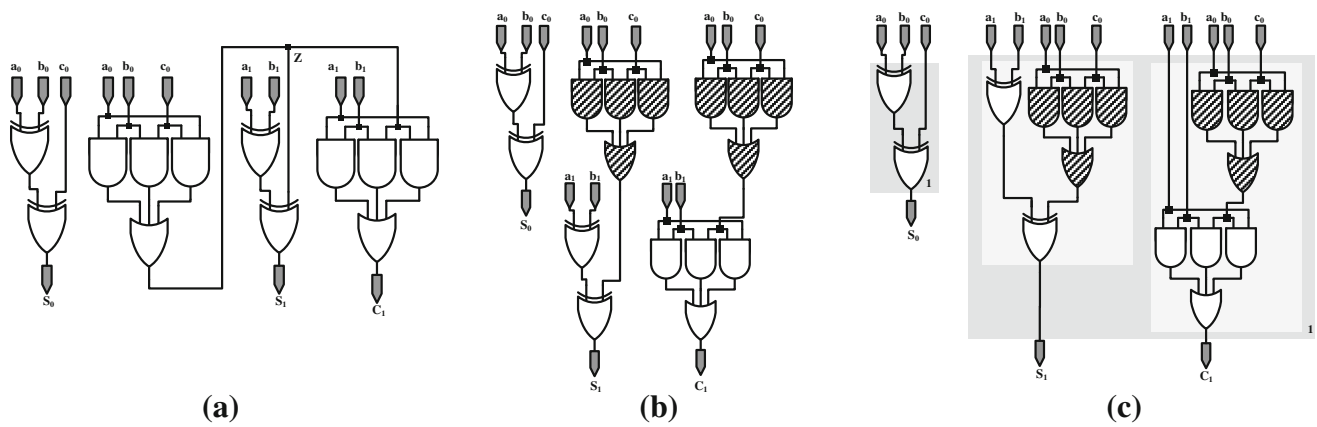




**Figure 23** Technology-optimized 4-bit BW multiplier based on LUT-Register combination.

required in the first and the final row of the BW multiplier. Like with RCA multipliers, the final net-list is further optimized by eliminating some redundant LUTs that are not required. This involves the elimination of the sum bits in the first column except for the terminating cell of the column where the sum bit has to be retained. The

schematic showing the arrangement of binary cells in three different multipliers is shown in Fig. 25. A theoretical comparison of the resource utilization and critical path for different multiplier schemes is listed in Table 1. Actual synthesis and implementation is discussed separately in section 6.



**Figure 24** a Parent network for VMA. b Re-structured Boolean networks. b Optimized LUT net-list.

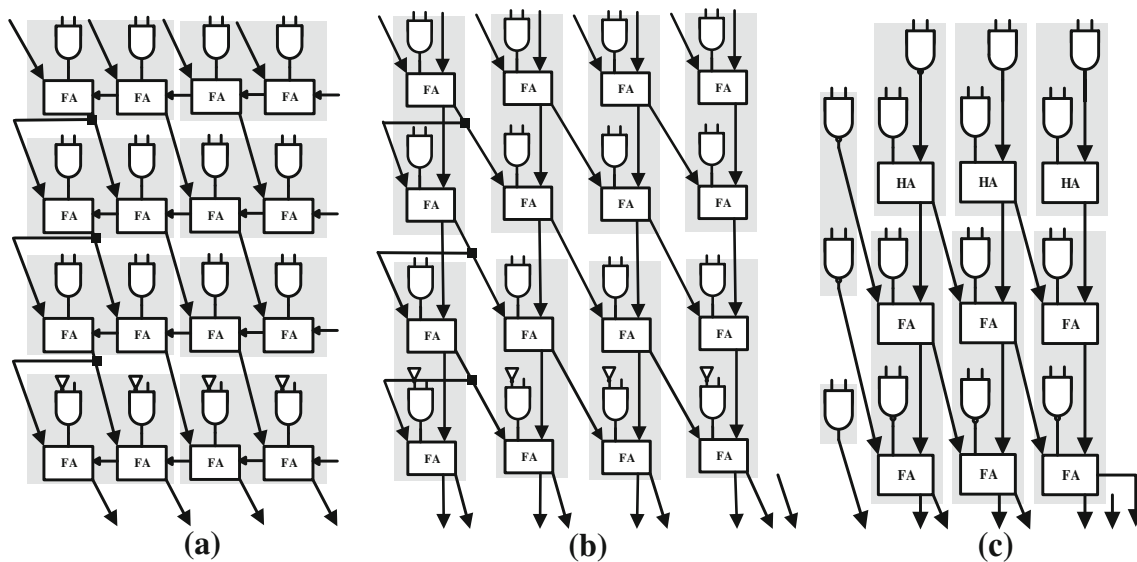
## 6 Synthesis, Implementation and Discussions

### 6.1 General Considerations and Methodology

The implementation in this work targets FPGAs that have 6-input LUTs as the basic logic element. This includes state-of-art Xilinx 5th, 6th and 7th generation FPGAs. In particular, devices from Virtex-5 family have been considered for implementation. Each logic slice in these FPGAs supports combinational and synchronous resources like 6-input LUTs, fast carry-chains, storage elements, function generators etc. [12]. The amount of supported resources varies from family to family, and within a family from device to device. The 6-input LUTs supported by these devices can be used in dual

output mode to implement two 5-input Boolean functions that share inputs. Each carry-chain supports four-bit operands and performs fast arithmetic addition based operations within a slice. Registers are synchronous elements that are used for pipelined implementations. Each logic slice can provide a combinational output or a registered output.

The implementation in this work is carried out for different operand word-lengths. The parameters considered are area, timing and power dissipation. Area is measured in terms of on-chip resources utilized. These include LUTs, flip-flops and slices. Timing is concerned with the clock speed of a design and is limited by the setup time of the input/output registers, propagation and routing delays associated with the critical path, clock to



**Figure 25** Cell arrangement in different multipliers. a RCA multiplier. b CSA multiplier. c BW multiplier.

output time associated with the flip-flops and the skew between the input register and the output register. For pure combinational circuits, timing is mainly concerned with the propagation and routing delays associated with the critical path. Throughout this work, inherent Digital Clock Managers (DCMs) have been used to control the mapping of the clocking resources onto the underlying FPGA fabric. This helps in keeping the skew to a minimal.

Timing analysis may be either static or dynamic. Static timing analysis gives information about the speed of the implemented design. This is usually expressed in terms of the critical path delay and operating frequency of the design. Static timing analysis is done post synthesis and post PAR. However, the metrics obtained after synthesis are often not accurate enough due to the programmability of the FPGA which allows for interconnect delays to change significantly between iterations. Therefore, the metrics presented in this work are post PAR, i.e. once the design has been translated and mapped and all the components have been placed and routed to meet the requisite design constraints. Dynamic timing analysis verifies the functionality of the design by applying test vectors and checking for correct output vectors. An important result from dynamic timing analysis is the switching activity information captured in the value change dump (VCD) file.

Power dissipation is given by the sum of static power dissipation and dynamic power dissipation. Static power dissipation consists of device static power dissipation and design static power dissipation. Device static power dissipation represents the power dissipated by the device to operate and be available for programming. It forms the major portion of the FPGA static power dissipation and varies with family. Design static power dissipation

is the additional continuous power drawn from the source when the device is configured and there is no activity. Dynamic power dissipation is related to the charging and discharging of capacitances along different logic nodes and interconnects. The software device database models the capacitance of each FPGA resource according to its configuration and connectivity. The software algorithms then predict the activity of each node in the netlist before calculating and adding up the power of all components. Alternatively, the designer may supply the activity file obtained during dynamic timing analysis for power calculations.

In this work, similar test-benches have been used for simulation and metrics generation and are typically designed to represent the worst case scenario (in terms of switching activity) for data entering into the synthesized blocks. Design entry is done using VHDL. The constraints relating to synthesis and implementation are duly provided and a complete timing closure is ensured. Synthesis and implementation is done using Xilinx ISE 12.1 [36]. Power analysis is done using Xpower analyzer [36].

### 6.2 Coding Strategy

Design entry is one of the most important steps in FPGA design flow. It is the only manual step in the entire design flow. The rest of the steps are either fully automated or semi-automated. For getting maximum performance from an FPGA design, a proper coding of its functionality is important. HDL coding styles can have a significant impact on the quality of results that are achieved once the design is implemented. It is thus desirable to write HDL codes that map efficiently onto the target FPGA. It is usually preferable to describe the behavior of the design and let the synthesis tool do the

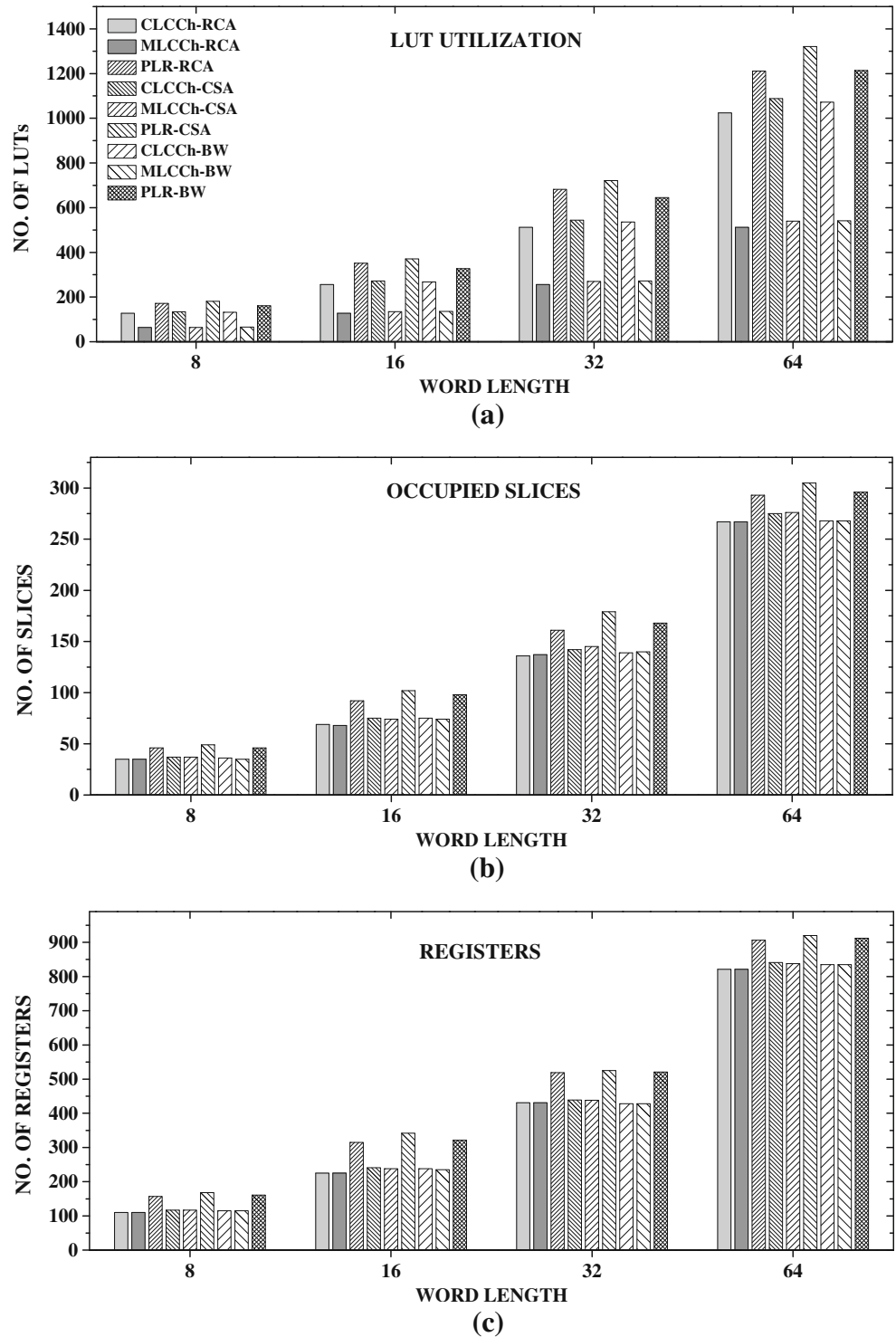
**Table 2** Resource utilization for different technology-optimized bit-parallel multiplier schemes.

Scheme	Design	No. of LUTs that implement logic	No. of slices	No. of registers
RCA	Conventional LUT-Carry-chain (CLCCh)	256	69	225
	Modified LUT-Carry-chain (MLCCh)	128	68	225
	Proposed LUT-Register (PLR)	352	92	315
	Conventional LUT-Carry-chain (CLCCh)	272	75	241
CSA	Modified LUT-Carry-chain (MLCCh)	135	74	238
	Proposed LUT-Register (PLR)	371	102	342
	Conventional LUT-Carry-chain (CLCCh)	268	75	238
BW	Modified LUT-Carry-chain (MLCCh)	136	74	235
	Proposed LUT-Register (PLR)	327	98	321

mapping of the code. Such a coding strategy is inferential, in the sense that the HDL code is inferred by the synthesizer and the distribution of logic to the underlying resources is done by the synthesizer itself.

Inferential coding styles are simple and have many advantages like readability and portability. However, the synthesis tool does not have any information about the purpose or intent of the design. As such the optimiza-

**Figure 26** Resource utilization versus word-length for different technology-optimized realizations. **a** LUT utilization. **b** Occupied slices. **c** Registers.



tions performed by the synthesis tool may not be optimum for the intended application.

Instantiation based coding strategy may be used as an alternative for mapping a desired functionality onto the FPGA device. With instantiation based coding styles the designers can control the exact placement of the individual blocks. Although instantiation based codes are complex and suffer from poor readability, the notion that the designer has control over the mapping process makes it desirable. In fact, instantiation may sometimes be the only way to make use of the complex resources available in the device.

The logic blocks synthesized in this work are technology-optimized. It is, therefore, important that the optimizations done prior to design entry remain preserved throughout the design flow. For this reason, instantiation based coding strategy has been used throughout this work.

## 7 Results and Discussions

In this section, actual synthesis and implementation of different multiplier realizations discussed in section 5 is carried out. The performance of these multipliers is compared against the Xilinx IP multiplier v.11.2 and some technology-independent realizations reported in [38]. The analysis is done for an operand word-length varying from 8 to 64 bits and the targeted device is xc5vlx30-3ff324 from Virtex-5.

### 7.1 Area Analysis

Table 2 provides a comparison of the resources utilized for logic implementation by different multiplier realizations for an operand word-length of 16 bits. In each multiplier scheme the modified LUT-Carry-chain realization requires least number of LUTs to implement the combinational logic. This is due to the efficient utilization of the underlying FPGA fabric as discussed in section 5. Amongst different multiplier schemes, the RCA based multiplier structures map efficiently onto the FPGA device. Evidently, for a given realization, the RCA scheme has the minimum LUT and slice count. Further analysis is carried out by plotting the resources utilized as a function of operand word-length. The results are shown in Fig. 26.

Next, different multipliers reported in [37] are compared against the technology-optimized realizations. Comparison is also provided against the Xilinx IP multiplier v.11.2. Table 3 shows the resources utilized for an operand word-length of 16 bits. Note that the Xilinx IP multiplier v.11.2 can be implemented using LUTs or DSP cores. For area analysis, the realization utilizing LUTs is considered. The results again indicate that technology-optimized multipliers utilize the underlying fabric more efficiently. Further analysis is done by plotting the resources utilized against the operand word-length. The results are shown in Fig. 27.

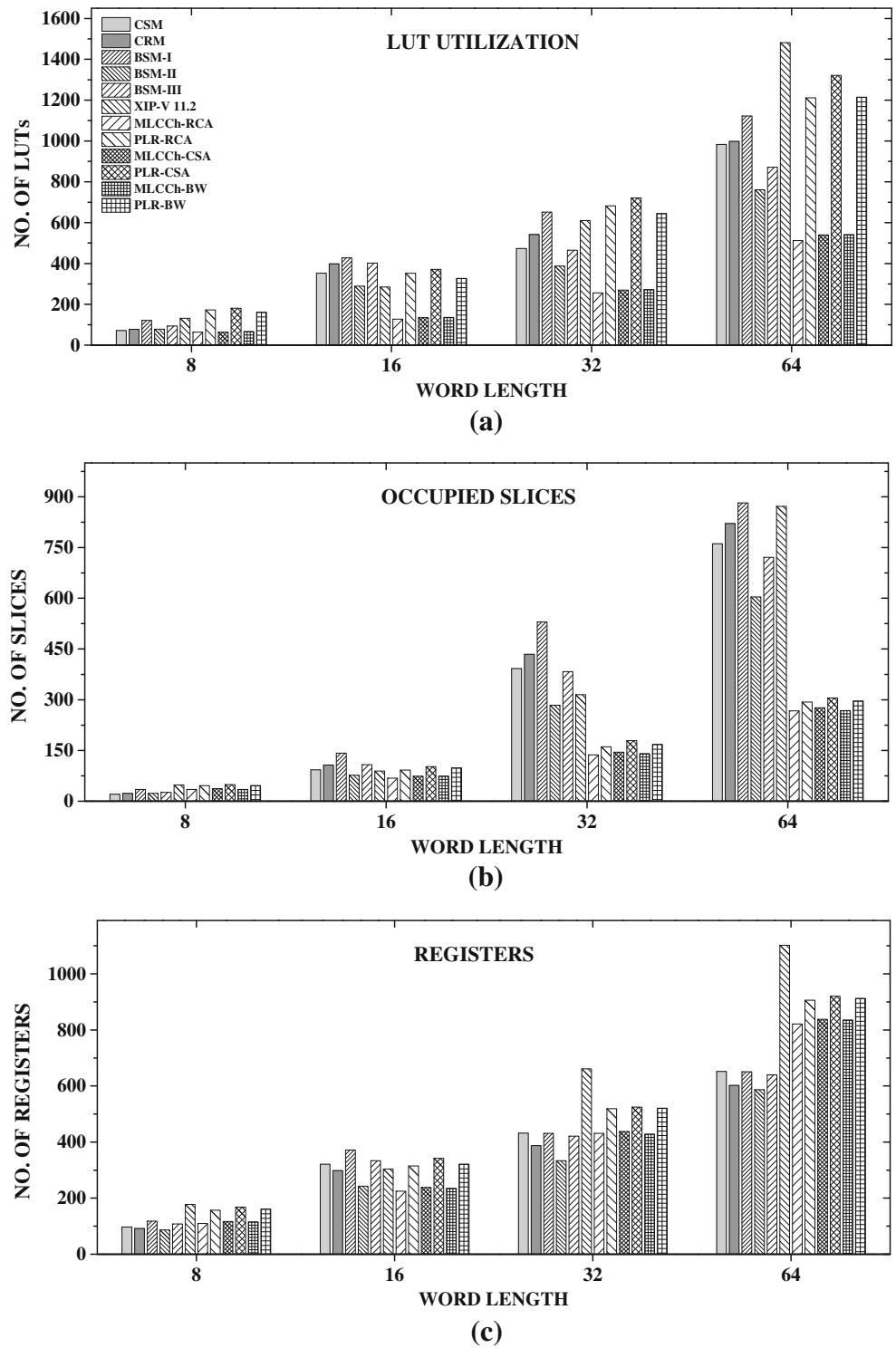
### 7.2 Timing Analysis

Table 4 provides a comparison of the critical path delay and maximum achievable clock frequency for different

**Table 3** Resource utilization for proposed technology-optimized realizations and different technology-independent realizations.

Design	No. of LUTs that implement logic	No. of Slice	No. of Registers
RCA based Modified LUT-Carry-chain (MLCCh-RCA)	128	68	225
RCA based Proposed LUT-Register (PLR-RCA)	352	92	315
CSA based Modified LUT-Carry-chain (MLCCh-CSA)	135	74	238
CSA based Proposed LUT-Register (PLR-CSA)	371	102	342
BW based Modified LUT-Carry-chain (MLCCh-BW)	136	74	235
BW based Proposed LUT-Register (PLR-BW)	327	98	321
Xilinx IP Multiplier v.11.2 (XIP-V 11.2)	285	89	304
Multipliers reported in [37]			
Carry Save Multiplier (CSM)	353	93	321
Carry Ripple Multiplier (CRM)	398	107	298
Booth Signed Multiplier-I (BSM-I)	428	142	372
Booth Signed Multiplier-I (BSM-I)	289	77	243
Booth Signed Multiplier-I (BSM-I)	402	108	334

**Figure 27** Resource utilization versus word-length for different multiplier realizations. **a** LUT utilization. **b** Occupied slices. **c** Registers.



multiplier realizations discussed in section 5. The operand word-length is 16-bits. In each multiplier scheme, the LUT-Register based realization shows an improved

timing behavior. This is because the LUT-Register based structure is implemented with a reduced circuit depth and the critical path in these structures is effectively



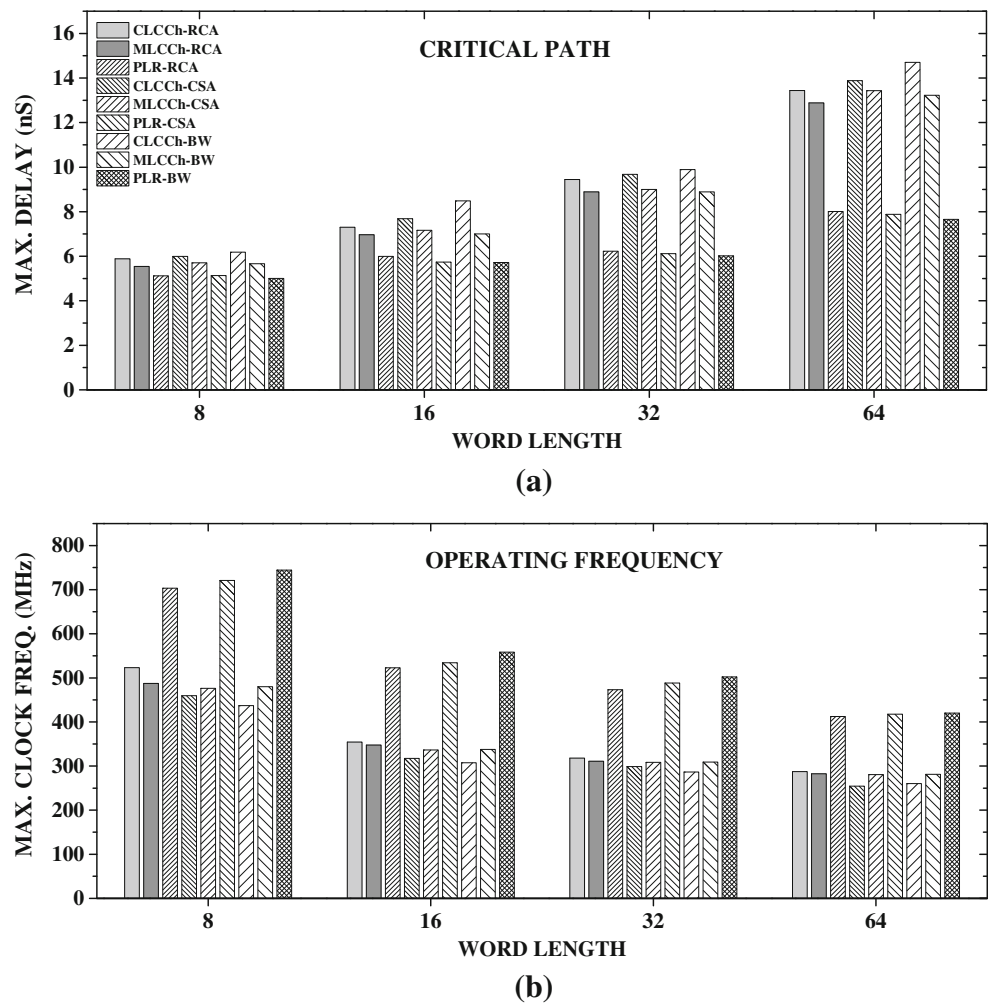
**Table 4** Timing analysis for different technology-optimized bit-parallel multiplier schemes.

Scheme	Design	Critical path (nS)	Max. clock frequency (MHz)
RCA	Conventional LUT-Carry-chain (CLCCh)	7.298	354.2
	Modified LUT-Carry-chain (MLCCh)	6.964	347.58
	Proposed LUT-Register (PLR)	5.994	523.01
	Conventional LUT-Carry-chain (CLCCh)	7.689	317.3
CSA	Modified LUT-Carry-chain (MLCCh)	7.172	336.8
	Proposed LUT-Register (PLR)	5.736	534.18
	Conventional LUT-Carry-chain (CLCCh)	8.481	307.4
BW	Modified LUT-Carry-chain (MLCCh)	7.002	338.002
	Proposed LUT-Register (PLR)	5.724	548.24

re-timed. Amongst different multiplier schemes, the BW based LUT-Register structures show an improved timing behavior. Further analysis is carried out by plotting the

critical path delay and maximum clock frequency as a function of operand word-length. The results are shown in Fig. 28.

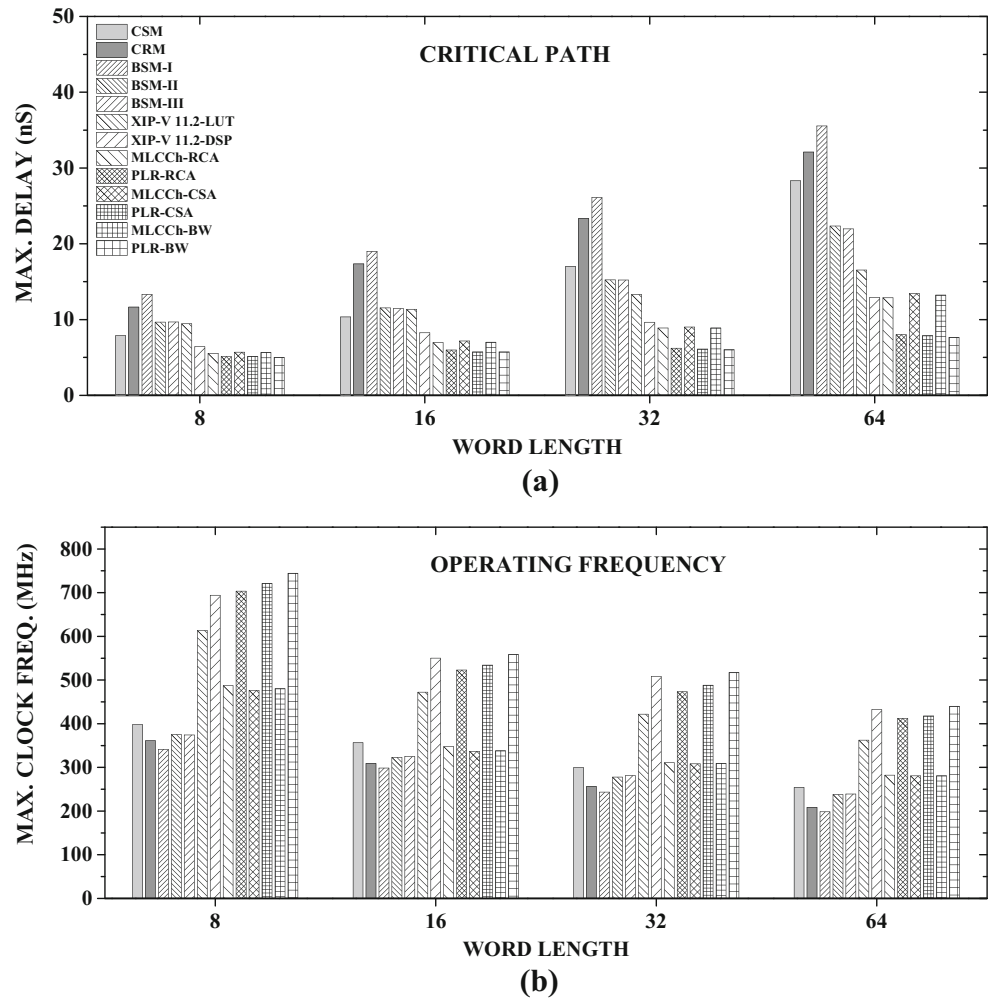
**Figure 28** Timing parameters versus word-length for different technology-optimized realizations. **a** Critical path. **b** Operating frequency.



**Table 5** Timing analysis for proposed technology-optimized realizations and different technology-independent realizations.

Design	Critical path (nS)	Max. clock frequency (MHz)
RCA based Modified LUT-Carry-chain (MLCCh-RCA)	6.964	347.58
RCA based Proposed LUT-Register (PLR-RCA)	5.994	523.01
CSA based Modified LUT-Carry-chain (MLCCh-CSA)	7.172	336.8
CSA based Proposed LUT-Register (PLR-CSA)	5.736	534.18
BW based Modified LUT-Carry-chain (MLCCh-BW)	7.002	338.002
BW based Proposed LUT-Register (PLR-BW)	5.724	548.24
LUT based Xilinx IP Multiplier (XIP-V 11.2_LUT)	11.363	472.3
DSP based Xilinx IP Multiplier (XIP-V 11.2_DSP)	8.26	550.05
Multipliers reported in [37]		
Carry Save Multiplier (CSM)	10.34	356.21
Carry Ripple Multiplier (CRM)	17.342	308.71
Booth Signed Multiplier-I (BSM-I)	19.008	298.71
Booth Signed Multiplier-I (BSM-I)	11.543	323.002
Booth Signed Multiplier-I (BSM-I)	11.45	325

**Figure 29** Timing parameters versus word-length for different multiplier realizations. **a** Critical path. **b** Operating frequency.



**Table 6** Dynamic power dissipation for different technology-optimized bit-parallel multiplier schemes.

Scheme	Design	Logic power (Watt)	Dynamic power dissipation (Watt)
RCA	Conventional LUT-Carry-chain (CLCCh)	0.01008	0.0336
	Modified LUT-Carry-chain (MLCCh)	0.00747	0.0252
	Proposed LUT-Register (PLR)	0.01027	0.0321
	Conventional LUT-Carry-chain (CLCCh)	0.0098	0.0338
CSA	Modified LUT-Carry-chain (MLCCh)	0.00783	0.0261
	Proposed LUT-Register (PLR)	0.0098	0.0331
	Conventional LUT-Carry-chain (CLCCh)	0.00983	0.0339
BW	Modified LUT-Carry-chain (MLCCh)	0.00774	0.0258
	Proposed LUT-Register (PLR)	0.01056	0.033

The performance of technology-optimized multiplier structures is also compared against the Xilinx IP multiplier v.11.2 and some technology-independent realizations reported in [37]. With respect to Xilinx IP multiplier v.11.2, it should be noted that both LUT and DSP based realizations have been considered. Also the multipliers reported in [37] have been re-implemented using synchronous design practices by effectively pipelining these structures. The underlying platform and timing constraints are kept same for all implementations. Table 5 provides a comparison of different timing parameters for an operand word-length of 16 bits. The modified LUT-Carry-chain structures and the LUT-Register based structures again show an enhanced timing performance. Further analysis is carried out by plotting different timing parameters as a function of operand word-length. The results are shown in Fig. 29.

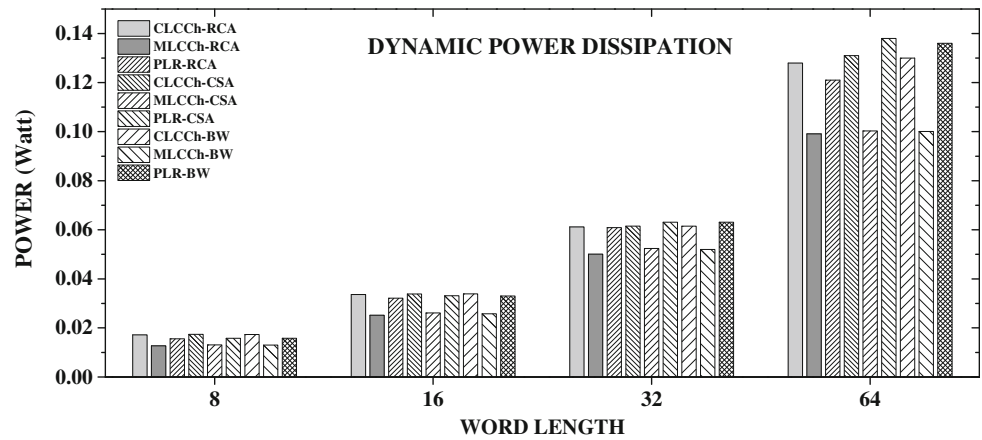
### 7.3 Power Analysis

Table 6 gives the dynamic power dissipation for different multiplier realizations discussed in section 5. The analysis is done for a constant clock frequency of 300 MHz and an operand word-length of 16 bits. In each multiplier scheme, the modified LUT-Carry-chain structure has minimum dynamic power dissipation. This is due to the reduced logic utilization in these structures. Amongst different multiplier schemes, the RCA based multipliers have slightly lesser power dissipation. This may again be attributed to their ability to use the underlying carry-chains efficiently. Since all the realizations are pipelined, there is a relaxed switching activity in all the multiplier structures. Evidently, technology-optimized realizations have considerably

**Table 7** Power dissipation for proposed technology-optimized realizations and different technology-independent realizations.

Design	Dynamic power dissipation (Watt)
RCA based Modified LUT-Carry-chain (MLCCh-RCA)	0.0252
RCA based Proposed LUT-Register (PLR-RCA)	0.0321
CSA based Modified LUT-Carry-chain (MLCCh-CSA)	0.0261
CSA based Proposed LUT-Register (PLR-CSA)	0.0331
BW based Modified LUT-Carry-chain (MLCCh-BW)	0.0258
BW based Proposed LUT-Register (PLR-BW)	0.033
LUT based Xilinx IP Multiplier (XIP-V 11.2_LUT)	0.0242
DSP based Xilinx IP Multiplier (XIP-V 11.2_DSP)	0.0182
Multipliers reported in [37]	
Carry Save Multiplier (CSM)	0.03569
Carry Ripple Multiplier (CRM)	0.06713
Booth Signed Multiplier-I (BSM-I)	0.06974
Booth Signed Multiplier-I (BSM-I)	0.06889
Booth Signed Multiplier-I (BSM-I)	0.05652

**Figure 30** Dynamic power dissipation versus word-length for different technology-optimized realizations.



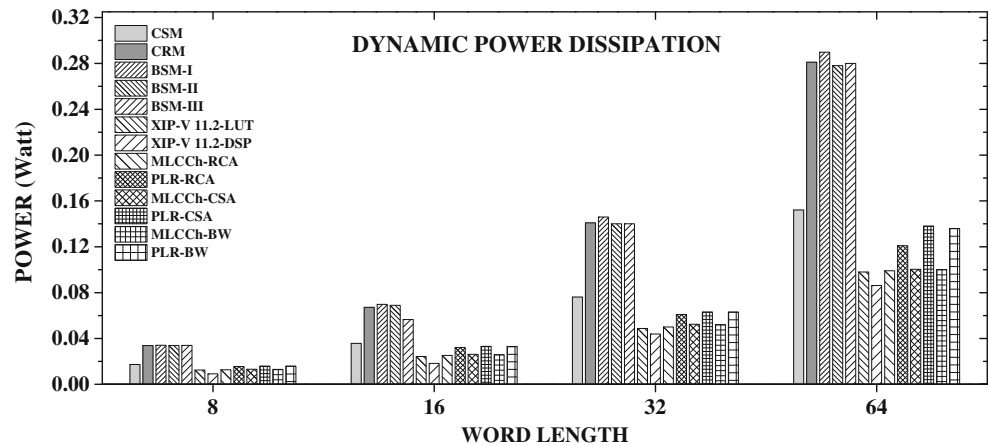
lower power dissipation when compared to the technology independent realizations reported in [37]. This is shown in Table 7, where power dissipation of technology-optimized multipliers is compared against different realization reported in [37]. However, LUT and DSP based Xilinx IP multiplier v.11.2 are the most power efficient realizations. This is due to the inherently optimized structure of these IPs. Further analysis plots the variation in dynamic power dissipation with word-length for different multiplier realizations. The results are shown in Figs. 30 and 31.

### 8 Conclusions

This paper discussed the optimum realization of three different fixed-point bit-parallel multipliers on FPGAs.

The optimizations presented in this paper were technology dependent and aimed at achieving improved integration levels in FPGAs by considering the synthesis process at a lower technology-dependent level. Optimized structures utilizing the basic FPGA fabric viz. LUTs, carry-chains and registers were discussed and their performance was theoretically compared against the conventional realizations. Performance comparisons, based on generated synthesizer metrics against various existing approaches were also provided. The proposed realizations showed a subsequent improvement in performance. This was in accordance with the theoretical analysis done. The results presented in this paper showed that technology dependent optimizations have a direct impact on area, delay and power dissipation of the design. The idea was to present a clear cut analysis that would provide an insight about the performance speed-up that is achievable through technology dependent optimizations.

**Figure 31** Dynamic power dissipation versus word-length for different multiplier realizations.



## References

1. Compton, K., & Hauck, S. (June 2002). Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, 34(2), 171–210.
2. Tessier, R., & Burlison, W. (2002). Reconfigurable computing and digital signal processing: past, present and future. *Programmable Digital Signal Processors*, 147–186.
3. Ashour, M. A., & Saleh, H. I. (2000). An FPGA implementation guide for some different types of serial-parallel multiplier structures. *Microelectronics Journal*, 31, 161–168.
4. Woods, R., McAllister, J., Lightbody, G., & Yi, Y. (2008). *FPGA-based implementation of signal processing systems*. Wiley.
5. Tessier, R., & Burlison, W. (2001). Reconfigurable computing for DSP: A Survey. *Journal of VLSI Signal Processing*, 28, 7–27.
6. Todman, T. J., Constantinides, G. A., Wilton, S. J. E., Mencer, O., Luk, W., & Cheung, P. Y. K. (2005). Reconfigurable computing: architecture and design methods. *IEEE Proceedings. Computer Digital Technology*, 152(2).
7. Naseer, R., Balakrishnan, M., & Kumar, A. (1998). Direct mapping of RTL structures onto LUT-Based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(7).
8. Jaiswal, M. K., & Cheung, R. C. C. (2013). Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support. *Microelectronics Journal*, 44, 21–430.
9. Hauck, S., & Dehon, A. (2008). *Reconfigurable computing: the theory and practice of FPGA-based computing*. Morgan Kaufmann series.
10. Brown, S. D., Rose, J., Francis, R. J., & Vranesic, Z. G. (1992). *Field programmable gate arrays*. Kluwer Academic Publisher.
11. Ling, A., Singh, D. P., & Brown, S. D. (2005). FPGA technology mapping: a study of optimality. *IEEE Proceedings Design Automation Conference*, 427–432.
12. Chen, D., & Cong, J. (2004). *DAO map: a depth-optimal area optimization mapping algorithm for FPGA designs*. IEEE/ACM International Conference on Computer Aided Design.
13. Anderson, J. H., & Wang, Q. (2011). Area-efficient fpga logic elements: architecture and synthesis, 16th Asia and South Pacific Design Automation Conference (ASP-DAC).
14. Xilinx (2009). Virtex-5 family overview, DS100 (v 5.0) Feb. 6, 2009. [www.xilinx.com](http://www.xilinx.com).
15. Xilinx (2010). Virtex-6 libraries guide for HDL designs, UG623 (v 12.3) September 21, 2010. [www.xilinx.com](http://www.xilinx.com).
16. Xilinx (2011). Spartan-6 family overview, DS160 (v 2.0) October 25, 2011. [www.xilinx.com](http://www.xilinx.com).
17. Cardarilli, G. C., Pontarelli, S., Re, M., & Salsano, A. (2008). On the use of Signed Digit Arithmetic for the new 6-Inputs LUT based FPGAs, 15th IEEE International Conference on Electronics, Circuits and Systems, ICECS.
18. Zhou, G., Li, L., & Michalik, H. (2008). Area optimization of bit parallel finite field multipliers with fast carry logic on FPGAs, International Conference on Field Programmable Logic and Applications, FPL.
19. Gao, S., Khalili, D. A., & Chabbini, N. (2007). Optimized realization of large-size two's complement multipliers on FPGAs, IEEE Northeast Workshop on Circuits and Systems, NEWCAS.
20. Altera Corporation (2006). "Stratix III device handbook," V.1.
21. Xilinx DSP Design Considerations (2006). Xtreme DSP for Virtex-4 FPGAs, UG073, (v 2.2).
22. Kuon, I., & Rose, J. (2007). Measuring the gap between FPGAs and ASICs. *IEEE Transactions on CAD*, 26(2), 203–215.
23. Abedelgwad, A., & Bayoumi, M. (2007). High speed and area efficient multiply accumulate (MAC) unit for digital signal processing applications," in Proceedings of IEEE International Symposium on Circuits and Systems, ISCAS.
24. Fayed, A. A., & Bayoumi, M. (2002). A merged multiplier accumulator for high speed signal processing applications, in proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).
25. Wei, G., & Ying, W. F. (2013). The implementation of FIR low-pass filter based on FPGA and DA, in Proceedings of the 4th International Conference on Intelligent Control and Information Processing, Beijing, China, 9–11 June 2013.
26. Yuan, X., Ying, T., & Chunpeng, G. (2010). Improved design of multiplier in the digital filter. International Conference on Computer and Communication Technologies in Agriculture Engineering.
27. Wirthlin, M. J., & McMurtrey, B. (2001). Efficient constant coefficient multiplication using advanced FPGA architectures. International Conference on Field Programmable Logic and Applications.
28. Wirthlin, M. J. (2004). Constant coefficient multiplication using look-up tables. *Journal of VLSI Signal Processing*, 36, 7–15.
29. Virtex-6 FPGA Data Sheet (2010). Xilinx, Inc., San Jose, CA.
30. Stratix-IV FPGA Family Data Sheet (2010). Altera, Corp., San Jose, CA.
31. Parhi, K. (1999). *VLSI digital signal processing systems design and implementation*. Wiley.
32. Chang, H., & Satzoda, R. K. (2010). A low error and high performance multiplexer based truncated multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 18, No. 12, December 2010.
33. Kidambi, S. S., Guibaly, F. E., & Antoniou, A. (1996). Area-efficient multiplier for digital signal processing applications. *IEEE Transactions on Circuits and Systems –II: Analog & Digital Signal Processing*, Vol. 43, No. 2.
34. Stine, E., & Duverne, O. M. (2003). *Variations on truncated multiplication*. Proceedings of the Euromicro Symposium on Digital System Design.
35. Baugh, R., & Wooley, B. (1973). A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22(12), 1045–1047.
36. <http://www.xilinx.com>.
37. Bhattacharjee, S., Sil, S., Basak, B., & Chakarbarti, A. (2011). Evaluation of power efficient adder and multiplier circuits for FPGA based DSP applications, International Conference on Communication and Industrial Application (ICCIA).



**Burhan Khurshid** received the B.E. degree in Electronics and Communications Engineering from the Kashmir University, India, in 2008, the M. Tech degree in Communications and IT from National Institute of Technology, Srinagar, India in 2011. Currently he is pursuing his PhD in "System design for DSP applications" through the department of Computer Science and Engineering, National Institute of Technology Srinagar. He is expected to receive his doctoral degree by 2017. His research interests include Reconfigurable architectures, Platform oriented solutions for arithmetic and DSP algorithms, Architectural and technology-dependent optimizations targeted for FPGA platforms, etc. He has many publications in the related field and is a student member of IEEE. He is also a lifetime member of IETE.