A Feasibility Study on Realizing General-purpose Technology Mapper for DSPs of FPGAs Using Exhaustive Search

Koyo Shibata[†1] Takashi Imagawa[†2] Hiroyuki Ochi[†1,2]

†1 Graduate School of Information Science and Engineering, Ritsumeikan University
†2 College of Information Science and Engineering, Ritsumeikan University
1-1-1 Noji-higashi, Kusatsu, Shiga, 525-8577 Japan
is0358ir@ed.ritsumei.ac.jp, takac-i@fc.ritsumei.ac.jp, ochi@cs.ritsumei.ac.jp

Abstract— This paper proposes a technology mapping algorithm applicable for arbitrary single-output tree-structure DSP block whose operation nodes have up to two fan-ins of FPGAs, based on an exhaustive search to find an optimal implementation of the given application circuit description. DSP blocks consisting of hard macro multipliers, etc. have become essential in FPGAs to achieve high performance and area efficiency. For the effective use of DSP blocks, a technology mapping algorithm is indispensable to find the optimal implementation of a given circuit using DSP blocks. Ronak *et al.* have proposed a greedy algorithm to search for a mapping that maximizes throughput, targeting the Xilinx DSP48E1. Our proposed algorithm applies to a broader range of DSP blocks since it automatically generates a database of valid configurations from a structural description of the target DSP block. Replication of the operators allows us to find solutions with a smaller number of DSP blocks than those by the conventional algorithm while reducing global nets. To reduce runtime, we also introduce pruning techniques and graph partitioning that do not affect the optimality. From experiments using DFGs with 33, 58, and 100 nodes, the proposed method reduces the number of DSP blocks by 7.94–10.81% compared with the conventional algorithms.

I. Introduction

Digital signal processing (DSP), which is one of the main target applications of FPGAs, requires a lot of arithmetic operations such as multiply-and-accumulate operation. The LUT-based implementation of arithmetic operation degrades the area, power consumption, and performance of application circuits because it requires a large number of logic blocks and their interconnections. On the other hand, the arithmetic operations can be implemented efficiently by using the DSP block integrated into FPGAs [1].

Since a DSP block consists of multiple arithmetic units such as multipliers and adders, it is necessary to assign as many operations as possible to one DSP block in order to further improve the implementation efficiency of arithmetic operations. The assignment is called as *mapping* and the algorithm which explores an appropriate set of assignments is called as *technology mapping* in this paper. Ronak *et al.* [2] have proposed a greedy algorithm to explore for a mapping that maximizes throughput, and the target DSP block is Xilinx DSP48E1.

The technology mapping proposed in this paper exhaustively explores a mapping that minimizes the number of DSP blocks and that of interconnection between the blocks. Note that, in the proposed algorithm, a target application circuit is represented as a data flow graph (DFG) whose node is arithmetic operation. The algorithm replicates nodes to reduce the number of DSP blocks. This replication is inspired by the replication of logic gates with multiple fanouts in technology mapping for LUT. On the other hand, an exhaustive search takes a very long time, so for mappings that are clearly inefficient, the search is terminated without losing the exhaustiveness. Different from the conventional method, the target DSP architecture of the proposed algorithm is not limited to those of specific models and vendors.

The remainder of the paper is organized as follows. In section 2, we describe the DSP blocks and conventional mapping methods. In section 3, we describe the proposed mapping method, and in section 4, we show experimental results using application circuits to compare the conventional and proposed methods. Finally, section 5 conclusion and future works are described.

II. Background

A. DSP block

A DSP block is a hardware block in an FPGA. It consists mainly of dedicated circuits for arithmetic operations to improve performance and energy efficiency, with a certain level of programmability to enhance applicability to various complex operations[1]. Since multiply-add operations frequently appear in the signal processing domain, a

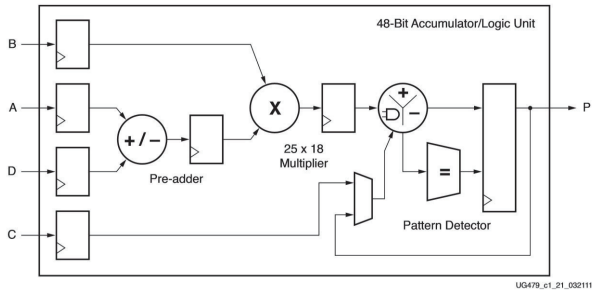


Fig. 1. Xilinx's DSP48E1 architecture[3]

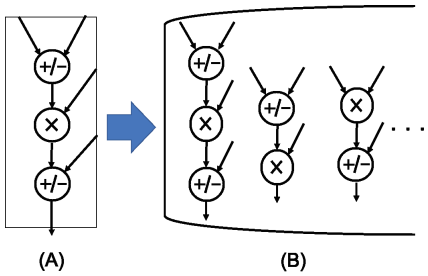


Fig. 2. Generation of template database

DSP block typically consists of multiplier(s) and adder(s) so that a single DSP block can implement a multiply-add operation. The architectures of DSP blocks can vary due to the difference in each FPGA vendor's design concept and the difference in the target application domain. Fig. 1 shows the block diagram of Xilinx's DSP block DSP48E1. It consists of a pre-adder, a multiplier, and an accumulator, which are essential to implement a multiply-add operation in a single DSP block. Additionally, it has many registers, and so on. Note that the search algorithm proposed in this paper does not consider the registers inside the DSP block.

B. *Conventional technology mapping algorithms for DSP blocks*

Ronak *et al.* proposed technology mapping algorithms that generate a Verilog-HDL netlist of DSP blocks from a given arithmetic assignment statement list.

First, a template database is generated in preparation for mapping. The DSP48E1 targeted in [2] is modeled as three arithmetic units connected in series, as shown in Fig. 2 (A). There are seven possible valid configurations for it since there are seven possible ways to choose arithmetic units to enable; these are manually stored in the template database. Mapping is performed based on this database, and two types of algorithms, Greedy Segmentation and Improved Segmentation, have been proposed.

The flow of Greedy segmentation is as follows.

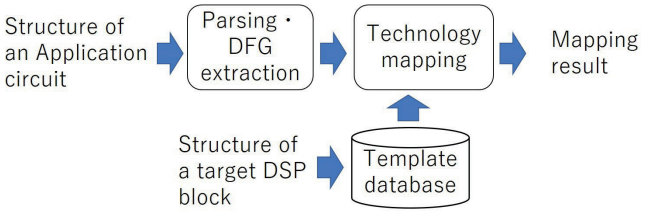


Fig. 3. Overall Flow of Proposed Mapping Method

1. Select an unchecked node n randomly in the DFG.
2. Find a fanout-free subgraph of up-to-three consecutive unchecked nodes starting from n that matches a template in the database and mark them as checked.
3. Repeat above until all nodes in the DFG are checked.

While the time complexity of the algorithm is linear to the DFG size, the DSP count depends on node n in Step 1. To cope with the above problem, they also proposed an improved algorithm, in which matchings to three nodes are searched first, followed by matchings to two nodes and one node are searched.

Since the above algorithm is specialized to DSP48E1, it is not applicable to other FPGAs containing DSPs of different architecture.

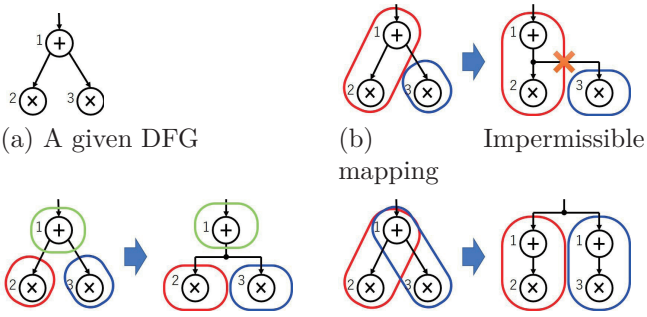
Furthermore, the above algorithm searches for only fanout-free subgraphs to match the template, failing to find an optimal solution. In the technology mapping for LUTs, it has been proven that replicating multiple-fanout gates enables us to find better solutions. Inspired by the LUT mapping, we will develop a new mapping algorithm for DSP blocks in which multiple-fanout arithmetic operations are replicated similarly.

III. Proposed Method

A. *Overall Flow of Proposed Mapping Method*

The overall flow of the proposed mapping method is shown in Fig. 3. It uses a description of the target DSP block and the application circuit to be mapped as input. Using the former, subgraphs of the DSP block are automatically extracted and stored in the template database. The subgraph of a DSP block is a graph consisting of all or some of the arithmetic operations that make up a DSP block and corresponds to the arithmetic functions that can be realized in a single DSP block by configuration. By parsing the latter, the corresponding DFG is generated. Using the DFG and the template database, it performs technology mapping to search for optimal mapping based on an exhaustive search.

In the following, we will focus on our technology mapping algorithm.



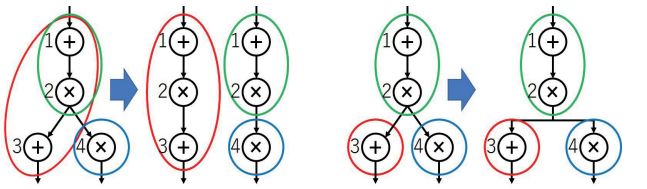
(a) A given DFG

(b) Impermissible mapping

(c) Mapping without replicating nodes (DSP:3, nodes:3)

(d) Mapping to replicate nodes (DSP:2, nodes:4)

Fig. 4. Node Replication



(a) In the case of redundant node replication (DSP:3, nodes:6)

(b) In the case of no node replication (DSP:3, nodes:6)

Fig. 5. Redundant node replication

B. Node Replication

The proposed mapping algorithm targets an FPGA architecture with a single type of DSP block from which no intermediate output can be extracted. It is assumed to map to a netlist consisting of DSP blocks only. For example, given a DFG, as shown in Fig. 4(a), the mapping, as shown in Fig. 4(b), is not allowed. The conventional methods demand three DSP blocks for mapping, as shown in Fig. 4(c), where every multiple-fanout node should be an output of a DSP block. The proposed method uses node replication to reduce the number of DSP blocks to achieve efficient mapping. The proposed method can map the DFG with two DSP blocks by replicating node 1 as shown in Fig.4(d) at the cost of increased arithmetic operations.

We must note that node replication is not always the best choice. Although both (a) and (b) in Fig. 5 require three DSP blocks, Fig. 5(a) requires two more total operations than (b), suggesting that such replication is redundant.

C. Mapping algorithm

The proposed technology mapping algorithm traverses the DFG from the output nodes to the input nodes in a depth-first manner, and for each node, considers each subgraph corresponding to the DSP that covers the node. Since multiple candidate subgraphs cover a node, we use a depth-first search to explore the solution space exhaustively to find the optimal solution. An overview of the proposed algorithm is shown in Fig. 6.

```
01 // TDB : Template Database
02 // DFG : Data Flow Graph
03 mapper_main() {
04       T = All end nodes of the DFG;
05       dfs(T, φ); // Output optimal solution
06 }
07 dfs(T, M) {
08       if (T == φ) return φ;
09       t = pop(T);
10       S = A set of subgraphs containing t ∩ TDB;
11       for (i=0; i < |S|; i++) {
12             M_new = Nodes mapped by S_i;
13             T_new = Parent nodes of M_new uncontained in (M_new ∪ M);
14             R_i = dfs(T_new ∪ T, M_new ∪ M)×{S_i};
15       }
16       return ∪_{i=0}^{|S|} Ri;
17 }
```

Fig. 6. Overview of the proposed algorithm

First, the main routine *mapper_main*() creates a list of target nodes that will be the starting point of the mapping (hereafter the target list) T and calls the search function *dfs*(). It takes two arguments, T and M. These values for the first call are all the output nodes of the DFG and a null set ϕ, respectively. The recursive function *dfs*() picks out one node t from T and extracts a subgraph ending with t. Let the maximum number of nodes in the subgraph be the number of nodes in the DSP block, extract all valid subgraphs, and leave those that match the template database (TDB) as candidate subgraphs S that cover t. Let the set of candidate subgraph S be the set of subgraphs in the DFG that cover t and that match TDB. The number of subgraphs $|S|$ is the number of branches in the search.

In practice, we set a restriction on the extraction of subgraphs containing t. We exclude subgraphs containing a node that has been mapped as the output node of a DSP. For example, consider extracting a subgraph from the DFG in Fig.7(1) with a target node D. Since the output of node B has been obtained, there is no need to consider node B and its ancestors. Thus, if the target node is D, the extracted subgraph is node D itself only.

We also exclude the mapped nodes whose parents are also mapped. For example, consider extracting a subgraph from the DFG in Fig.7(2) with a target node D. Since the input of node B has been obtained, it falls under the above restriction of subgraph extraction. Hence, if the target node is D, the subgraph to be extracted is a subgraph up to node B, where the input is obtained.

To increase pruning effectiveness, which will be de-

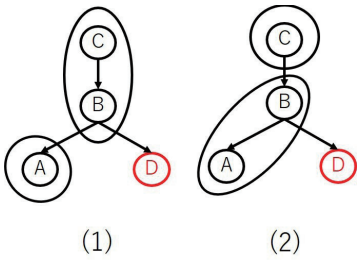


Fig. 7. Subgraph extraction

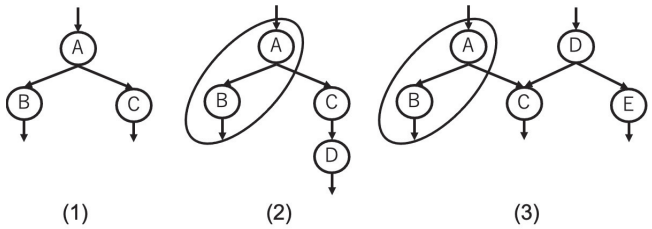


Fig. 8. Pruning redundant node replication

scribed later, we preferentially map from subgraphs with a high number of nodes. At the branch where subgraph S_i is selected from S, we remove t from the first argument T and, instead, add unmapped parents of the nodes covered by S_i, and add the nodes covered by S_i to the second argument M, then call *dfs*() recursively.

The completion of mapping is determined whether the target list is empty. Then, backtracking is performed as appropriate. When the search returns to the main routine, the exhaustive search is complete.

D. Pruning Techniques

Since the number of possible mapping is enormous and the time required for an exhaustive search is extremely long, an exhaustive search of all subgraphs is not realistic. Therefore, it is necessary to prune the subgraphs without compromising their optimality. We introduce the following pruning in the proposed algorithm.

1. Pruning redundant node replication
2. Pruning with the minimum number of DSPs needed to map the remaining nodes
3. Pruning inefficient mapping

Pruning in this paper implies ignoring the recurrence calls of the *dfs*() (line 14 in Fig.6).

D.1. Pruning redundant node replication

We prune the subgraphs that necessitate redundant node replication, as demonstrated in Fig. 5. For example, assume that node A and B in Fig. 8(1) are mappable to a single DSP block, and node A and C are not (e.g., A and C are multipliers, while the target DSP block has only one multiplier). In this case, the replication of node A is necessary to generate input to node C. Since such replication is redundant, we prune a subgraph containing a node like A unless it is the subgraph's output.

Consider another example in Fig. 8(2), where Node A and B have already been mapped to a DSP block, and thus node C should be mapped to a DSP block together with replicated node A. This implies that a subgraph that contains node C and not node A should be pruned. In other words, when the target node is D, only two subgraphs, ACD and D, should be considered.

Fig.8(3) shows another example where nodes A and B have been mapped, and node C must be mapped with the same DSP block as replicated node A. This implies that we cannot map nodes C and D together. This also implies that node C requires node D's output. This further implies that we cannot map nodes E and D together. Thus, we can prune subgraphs that violate the above restriction.

These techniques enable us to prune the subgraphs in the search tree's shallow level, and thus efficient search is expected.

D.2. Pruning with the minimum number of DSPs needed to map the remaining nodes

We calculate the minimum number of DSP blocks required to map the remaining unmapped nodes and prune when the sum of it and those already mapped exceeds those in the optimal solution at that time.

The proposed method calculates the nodes, considering the types of operators. For each type of operator, we divide the number of remaining nodes in the DFG by the number of nodes of the corresponding type in a DSP block. The maximum value among all types gives the lower bound of DSP blocks needed. Pruning is performed based on this value.

D.3. Pruning inefficient mapping

We leave only subgraphs that cover as many nodes as possible without compromising optimality. First, we find a subgraph, say s, that covers as many nodes as possible. If s satisfies the conditions described below, we can safely prune the subgraphs that are included by s without losing chance to find better solutions.

The conditions for s to allow it to prune other subgraphs that are included by s without loss of optimality are as follows.

1. A subgraph whose parent of its head node is a single-output node.

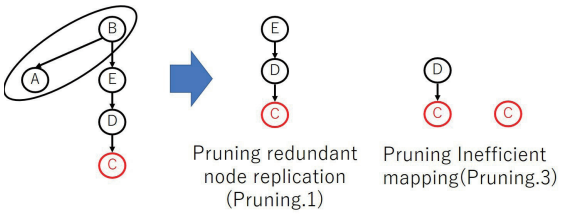


Fig. 9. Inefficient mapping pruning conditions (1)

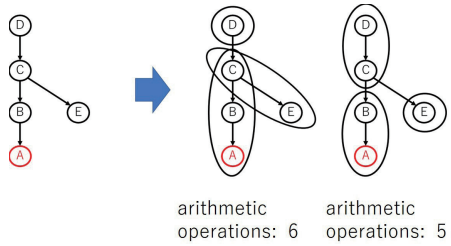


Fig. 10. Inefficient mapping pruning conditions (2)

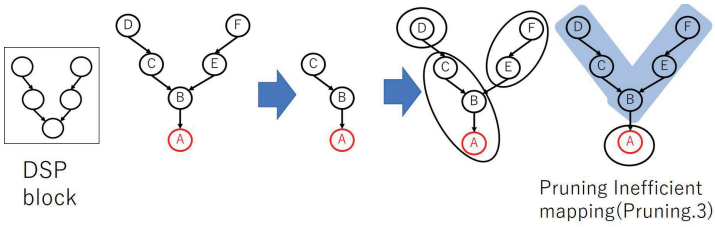


Fig. 11. Inefficient mapping pruning conditions (3)

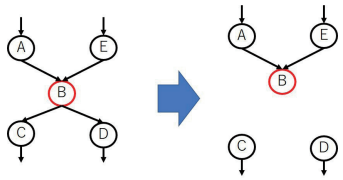


Fig. 12. Partitioning the graph by deleting edges

2. A subgraph that consists only of single-output nodes.

3. A subgraph that consists only of single-input nodes.

The condition (1) is necessary to prohibit pruning the essential subgraphs when the parent node has multiple outputs. Consider the case where s is a subgraph headed by a node that cannot be the head of the subgraph due to Pruning 1 (Pruning redundant node replication). Since s will be pruned by Pruning 1, subgraphs included by s should be preserved; otherwise, the necessary subgraphs are pruned by this pruning technique and no candidate subgraphs will be left and the mapping will be terminated midway (Fig.9).

The condition in (2) is necessary to prohibit pruning for more efficient subgraphs, since the same number of DSP blocks are used but may be mapped without node replication (Fig.10).

The condition in (3) is unnecessary if the target DSP block consists of arithmetic units connected linearly, but it is necessary for pruning without loss of optimality when the DSP blocks have a joining structure (Fig.11).

E. Graph Partitioning

The total number of subgraphs in the DFG is up to M^N, where M is the number of nodes in the DFG, and N is the number of nodes in the DSP block, resulting in a huge number of subgraph combinations. Thus, the proposed algorithm reduces the number of searches by dividing the graph and reducing the number of nodes in the DFG mapped at a time.

As an example, let us partition the graph in Fig.12. Assume that node B is selected as the target node. When its output destination nodes C and D cannot be mapped with the same DSP block as node B, the output of node B is required. Therefore, node B can be the end node, nodes C and D can be the beginning node. That is why we delete all the edges between nodes , C and D. If the deleted edge is a bridge, the graph is partitioned.

IV. Performance Evaluation

This section presents the mapping results of the two conventional methods, Greedy Segmentation and Improved Segmentation, and the proposed method. We compare them in terms of the number of DSP blocks, the number of nets, the total number of arithmetic operations, and the runtime. The number of nets refers to the number of inter-DSP-block connections, which influences the routability of the FPGA. Since the result of conventional methods depends on the order of node selection, we executed them 100,000 trials while changing the selection order randomly and obtained the number of DSP blocks in the best case and total runtime. We used three DFGs with 33, 58, and 100 operation nodes, respectively, for the evaluation. The operations at each node are randomly set up from addition, subtraction, and multiplication. Each node has one or two inputs and its output is connected to other nodes and/or external inputs/outputs. Similar to the previous studies, we used the Xilinx DSP48E1 structure for the target DSP block. We implemented both conventional and proposed algorithms with Python (ver.3.6.9) and PyPy (ver.7.3.1). We used Networkx (ver. 2.2) for graph manipulation.

Table I shows the mapping results by the conventional

TABLE I
MAPPING RESULTS OF PROPOSED AND CONVENTIONAL METHODS

(a) DFG with 33 nodes

Mapping method	#DSP blocks	#Net	#Arith ops
Greedy	22	23	33
Improved	22	23	33
Proposed	20	21	36

(b) DFG with 58 nodes

Mapping method	#DSP blocks	#Net	#Arith ops
Greedy	37	39	58
Improved	37	39	58
Proposed	33	35	63

(c) DFG with 100 nodes

Mapping method	#DSP blocks	#Net	#Arith ops
Greedy	63	70	100
Improved	61	68	100
Proposed	56	63	107

TABLE II
RUNTIME FOR MAPPING IN SECONDS

(a) Average runtime for 100,000 trials with conventional methods

Mapping method	33 nodes	58 nodes	100 nodes
Greedy	20.40	35.85	68.04
Improved	20.48	35.22	64.66

(b) Runtime with the proposed method

Mapping method	33 nodes	58 nodes	100 nodes
no pruning	151.78	NA	NA
Pruning 1	16.6	NA	NA
Pruning 1–2	1.75	304.83	NA
Pruning 1–3^V	0.98	56.53	NA
Pruning 1–3^S	0.69	7.17	3513.34
Pruning 1–3 + Partitioning^V	0.87	16.97	247.32
Pruning 1–3 + Partitioning^S	0.68	4.16	35.81

V: Applicable to any single-output DSP block architecture.
S: Applicable only to DSP blocks with linearly connected internal nodes like Add-Mult-Add.

and proposed methods. These results show that the proposed method successfully reduces the number of DSP blocks by 7.94-10.81%. Moreover, the number of nets also decreases, which will improve the routability of the FPGA.

Table II(a) shows the average runtime for 100,000 trials for the conventional methods, and Table II(b) summarizes the runtime for the proposed method. "NA" in Table II(b) means that the mapping was not completed within the time limit (86400 sec). Table II(b) presents results with various combinations of techniques proposed in this paper to clarify each method's contribution. Pruning 1 to 3 here corresponds to the methods described in D.1 to D.3 in Section III, respectively. The results in Table II(b) show that the proposed pruning method contributes significantly to reducing the runtime. Table III summarizes the graph partitioning results of the target DFGs. Table III shows that graph partitioning enables us to break up the 100-node DFG to DFGs of at most 54 nodes, thus reducing the time required.

Comparing Tables II(a) and II(b), the proposed method's runtime seems significantly longer than the conventional ones. However, it is noteworthy that the conventional methods are a heuristic search and require 100,000 trials to obtain high-quality mapping results shown in Table I. Thanks to the pruning and graph partitioning techniques developed in this study, the time required by the proposed method is comparable to that of the conventional methods for DFGs of the sizes tested in this study.

TABLE III
GRAPH PARTITIONING RESULTS

Number of nodes	Number of graph partitions	Maximum size of partitioned graphs
33 nodes	4	28
58 nodes	6	49
100 nodes	13	54

V. Conclusion

This paper proposed an exact-optimal technology mapping method based on an exhaustive search with applicability to arbitrary single-output tree-structure DSP block whose operation nodes have up to two fan-ins of FPGAs. It detects valid configurations for a given complex DSP block and automatically generates a template database. In the exhaustive-search-based mapping algorithm, the node replication technique reduces the DSP block usage, especially when there are multi-fanout arithmetic nodes in the given DFG. Furthermore, the pruning and graph partitioning techniques reduce the runtime without losing the optimality. However, it is necessary to further reduce the time required when the number of nodes in the DFG is more than 100. Comparisons of the mapping results between the proposed and conventional methods for DFGs with 33, 58, and 100 nodes show that the proposed method reduces the number of DSP blocks by 7.94–10.81%, at the cost of increased runtime.

References

[1] Amano, H.: *Principles and Structures of FPGAs*, Springer (2018).

[2] Ronak, B. and Fahmy, S. A.: Mapping for Maximum Performance on FPGA DSP Blocks, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 35, No. 4, pp. 573–585 (2016).

[3] Xilinx Corporation: *7 Series DSP48E1 Slice User Guide*, `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf` (2011).