Performance Estimation of FPGA Modules for Modular Design Methodology using Artificial Neural Network

Kalindu Herath, Alok Prakash, and Thambipillai Srikanthan

Nanyang Technological University, Singapore
kalindub001@e.ntu.edu.sg,{alok,astsrikan}@ntu.edu.sg

Abstract. Modern FPGAs consist of millions of logic resources allowing hardware designers to map increasingly large designs. However, the design productivity of mapping large designs is greatly affected by the long runtime of FPGA CAD flow. To mitigate it, modular design methodology has been introduced in the past that allows designers to partition large designs into smaller modules and compile & test the modules individually before assembling them together to complete the compilation process. Automated decision making on placing these modules on FPGA, however, is a slow and tedious process that requires large database of pre-compiled modules, which are compiled on a large number of placement positions. To accelerate this placement process during modular designing, in this paper we propose an ANN based performance estimation technique that can rapidly suggest the best shape and location for a given module. Experimental results on legacy as well as state-of-the-art FPGA devices show that the proposed technique can accurately estimate the F_{max} of modules with an average error of less than 4%.

Keywords: FPGA, floorplaning, modular design methodology, computer-aided designing

1 Introduction

The rapid scaling of transistors over the past decade has allowed commercial Field Programmable Gate Array (FPGA) giants like Altera[1] and Xilinx[7] to realize FPGAs with tens of millions of logic gates alongside useful ready-to-use hard Intellectual Property (IP) cores such as Block-RAMs (BRAMs) of different size and Digital Signal Processors (DSPs). Such resource-rich modern FPGAs permit system designers to use FPGAs for application specific implementation as well as to map increasingly complex applications. At the same time, the increasingly strict non-recurring engineering (NRE) costs and time-to-market (TTM) constraints are also pushing FPGAs favourably when compared to ASICs [24].

However, while FPGA devices are becoming more sophisticated, the computer-aided design (CAD) tools used for FPGA-based designs have not yet matured sufficiently to efficiently map large-scale applications into such FPGAs [26]. It is observed that typical CAD flow can take tens of minutes to hours or even days [8] for such applications, thereby significantly limiting design productivity.

Such mapping with existing CAD can also result in poor placement and routing decisions, which can lead to degradation in the quality of result (QoR), i.e. performance and power consumption of the final design. Placement step of current CAD flow has also been identified as the most time consuming step, contributing to almost 50% of total CAD runtime [20]. Traditional Simulated Annealing (SA) based placement algorithms are known to produce superior quality placement results for small and medium scale designs. However, SA does not scale well for larger designs, and hence resulting in longer runtime [9].

There have been attempts in developing scalable CAD to reduce the runtime. Altera improves their proprietary CAD tool, Quartus, by introducing parallelism in their SA based Q2P placer [18]. Similarly, Xilinx introduces an analytical placement strategy in their CAD [11], inspired by ASIC placement strategies.

However, the runtime of these CAD tools are still considerably high, especially for large and complex designs. To address the design productivity inefficiencies while compiling large applications [16], existing research work in FPGA CAD has proposed a new CAD flow, called *modular design methodology (MDM)*. This technique breaks a large system into smaller modules. Some of the modules, for example, board support packages or external memory interfaces, do not change frequently in a development cycle. Hence, these modules can be placed and routed individually and stored in a library and therefore can be excluded from subsequent compilations. The reuse of pre-compiled library modules significantly decrease the runtime of the flow [14]. Commercial FPGA vendors have also introduced augmented modular compile flow support to their CAD flow. Altera uses Quartus Incremental Compilation Flow [3] and Xilinx has integrated Hierarchical Design Suite [6] in PlanAhead Design and Analysis Tool [5]. However, this design technique is not sensitive to the connection information between modules. It avoids optimization of those connections that is possible in usual compilation flow. As a result, QoR of modular design methodology is typically degraded [12].

Modular design workflows use a large database of pre-compiled modules. Modules are placed and routed at large number of possible locations on FPGA during offline *module creation phase*. During *design assembly phase*, best set of pre-compiled modules are searched and mapped to get the final design. Large heterogeneous FPGAs offer greater possible ways a module to be placed on FPGA space, making large solution space for the module assembly stage. However, due to availability of different types of resources on these FPGAs, selecting a place and a shape for a given module from the library becomes more critical. Wrong selection either could lead to resource wastage [16] or could affect the performance of the module [15]. In addition, pre-compiling large number of variations per each module becoming increasingly time consuming. During our experiments, each module took about 3 hours to compile all possible placement variations. Therefore, it is apparent that module creation phase has become less feasible, especially if the module library changes frequently. [19].

In this paper, we propose an artificial neural network (ANN) based approach to estimate the shape and placement of a given module in order to achieve the best performance. Aim of this estimation is to make swift placement decisions during module based development, eliminating the need of having frequently changing pre-compiled module database. In the next section we discuss the existing modular design workflows, followed by a motivational example in section 3. In section 4, we explain the proposed methodology that estimates performance of modules in detail. Section 5 evaluates the proposed method and discusses the experimental results. Section 6 concludes this paper.

2 Related Workflows

MDM is an extensively explored design paradigm. Frontier [23] is a module based hierarchical placement framework. It uses a library of pre-placed modules to identify similar patterns in a given application. Identified patterns are clustered, and placed on FPGA by using library information. However, Frontier does not preserve routing information of the modules. HMFlow [14], on the other hand, maintains a library with pre-routed modules. It also creates new library modules for application logic which does not match with any library Module. BPR[8] uses coarser pre-compiled modules as compared to HMFlow to improve compilation time. In addition, its module library keeps different variations of each module by placing and routing for all possible locations on FPGA. During full compilation, only one version of each module is selected. qFlow [9] divides a given design into evolving (frequently changes) and invariant portions. Evolving portion is compiled as modules and is combined with invariant section. TFlow [17] extends qFlow methodology and uses bitstream level modules during design assembly phase. However, placement of bitstream level modules is less flexible due to restriction in bitstream configuration boundaries.

The effect of module size on QoR has been explored in [10]. Authors analysed QoR in their module based placement approach by varying the granularity of the modules. Analysis has been done in terms of runtime of the placement tool, and the Half Perimeter Wire Length of the placement solution. The impact of shapes of modules to overall FPGA resource utilization and placement flexibility is greatly discussed in [16]. They have shown multi shape pre-compiled modules can lead to better resource utilization and additional placement flexibility, which ultimately results more packing of modules on FPGA. Module shape exploration has been used in FPGA floorplanning techniques such as [22] and [25] but they considered shapes with single resource type. On the other hand, [19] argues pre-synthesising each module for different shape ratios is a tedious process. They suggest a method to change the shapes of modules during the placement stage using a pre-placed library of modules with a single size.

Most of the modular design workflows aim on turnaround time of a design cycle, rather than achieving both high performance while improving runtime. Large pre-compiled libraries involve in most workflows, while having a single shape for each pre-compiled module. Shape exploration for library modules with heterogeneous resources has been done, but it targets resource utilization on FPGA as a measurement of QoR. Previous work lacks a shape exploration framework to achieve better performance in each module.

3 Motivation

Modern FPGA architectures, including Altera and Xilinx FPGAs, are typically categorized as island-style FPGA architecture where resources such as BRAMs and DSPs are arranged in columns and interleaved between a sea of Configurable Logic Blocks (CLBs) columns. Interconnect wires are provided in copious amounts to provide for high bandwidth connectivity between these resources. It has also been observed that these resources are arranged homogeneously along the vertical axis such that most of the rows look identical in an FPGA.

However, as FPGAs continue to become more heterogeneous, incorporating different types and location of resources, it is important to identify the optimum placement and shape for a design module implemented in such FPGAs. The shape of a module refers to the rectangular space in the FPGA within which the module can be placed and routed. The optimum placement and shape ensures that the most widely used resources are available at a physically closer distance, thereby potentially reducing length of the interconnecting wires, ultimately leading to better performance and lower power consumption [13].

Figure 1 shows two different shapes and placement locations for a single module. The module is isolated from the rest of the design, so that it is independent from the connections for module to the rest of the design. We have observed that the shape given in Figure 1 (b) produces 12% greater maximum allowable clock frequency (F_{max}) as compared to (a). While similar number of resources are consumed in both cases, the main difference is the arrangement of resource columns. Depending on how the CLBs BRAMs and DSPs are connected within a module, an arrangement of columns in a rectangular region can be more preferable for the module. Hence, in this paper we propose a methodology, which is able to predict the F_{max} of given module on a FPGA device, when shape and application parameters given. As discussed in the previous section, existing module based designing techniques have not explored the relationship between module performance and its shape as well as the location in the FPGA fabric.

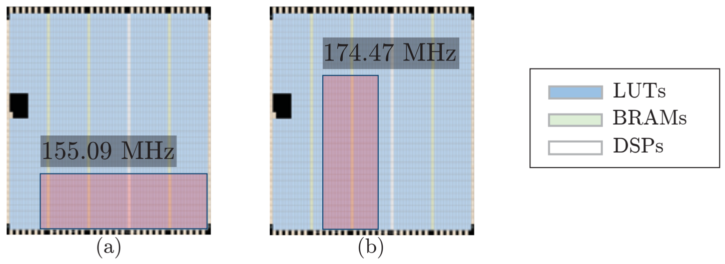

Fig. 1: Performance (F_{max}) of a module in two location and shapes

4 Methodology

In this section, we first provide some background information before explaining our approach. We use the Altera's Quartus CAD tools to explain the proposed methodology as well as for our experiments. It should be noted that the proposed methodology does not depend on vendor specific CAD tools. It is also noteworthy that Xilinx offers equivalent functionalities in their PlanAhead tool flow [5].

4.1 Background

Quartus Incremental Compilation (QIC) In the default Quartus compilation flow, also referred to as 'flat compilation', all the RTL code files are processed simultaneously in the Analysis and Synthesis step and the entire post-synthesis netlist is placed and routed subsequently by the fitter. If any changes need to be made after a flat compilation, one needs to perform a fresh re-compilation.

Altera includes a toolset for modular design methodology (MDM) in Quartus, called Incremental Compilation Design Flow. It allows hardware designers to follow a divide and conquer strategy, by partitioning a large design into relatively smaller modules and develop each module separately. In the rest of the paper, we use the term *module* and *partition* interchangeably. Some modules in a design might require frequent revisions than some other modules. For such situations, incremental compilation technique allows designers to change the code segments of only some modules and compile them separately without having to compile the whole project or alter the rest of the design.

Partitioning a large design into relatively smaller modules is typically performed using a bottom-up approach, where one or more VHDL entities or Verilog modules in the RTL code are defined as a design partition by the designer. Performing an incremental compilation after defining partitions creates separate netlist for each partition after each stage. For instance, after the 'Analysis and Synthesis' step a *post-synthesis netlist* or after 'Fitter' step *post-fit netlist* is generated. A single full post-fit netlist is created by merging netlists from each partition at the end of the incremental compilation flow. Quartus allows us to preserve the generated netlist of each partition at any stage by setting a parameter called *preservation level*. A partition with post-fit netlist preservation will not be processed during both synthesis and fitter steps, and its post-fit netlist generated in previous compilation is used for producing the full post-fit netlist.

LogicLock The atomic components (CLBs, BRAMs and DSPs) of a partition might be placed anywhere in the FPGA space during the fitter step. However, Quartus LogicLock feature allows designer to restrict the placement of these components to a rectangular region in the FPGA space. Designers can define such regions by providing the shape information (width and height), placement information (bottom left coordinate) and VHDL entities/Verilog modules belonging to the partition which needs to be in the region. Such rectangular regions with shape and placement information are called *footprint* of a partition.

4.2 Footprint generation for Design Partitions

Footprint of a Design Partition A design partition P_i is characterized by a resource requirement, which we express as a tuple $A_i = \{l, m, d\}$ where l, m, d

are the minimum requirement of CLBs, BRAMs and DSPs on the FPGA space in order to successfully map the partition into FPGA. During the incremental compilation flow, designers have to ensure that the final compilation of the overall design at least provides for A_i resources for each partition P_i. Therefore, during the early stages, we reserve a FPGA region with A_i resources for P_i by defining a LogicLock region LL_i. This LogicLock region is referred to as the footprint of the design partition. The shape and location properties of footprint LL_i can be expressed as a tuple $S_i = \{x, y, w, h\}$ where x and y are the bottom left corner of the rectangle in FPGA space, and w, h is width and height of the rectangle. In a typical island style FPGA, the bottom left corner block is marked as the origin of the coordinate plane which is $\{1, 1\}$.

For example, consider two different footprints $LL_{1,1}$, $LL_{1,2}$ for partition P_1 with $A_1 = \{421, 20, 0\}$ on Altera EP2C35 FPGA [2] as shown in figure 2. The shape parameters of two footprints are $S_{1,1} = \{1, 1, 56, 8\}$ and $S_{1,2} = \{1, 1, 52, 9\}$. Here, the second variable (k) in the notations, $LL_{i,k}$ & $S_{i,k}$, denotes the different footprints and shape parameters respectively of a single design partition (i). Both footprints consist of 3 BRAM columns and 1 DSP column but contain 52 CLB columns in the former case and 48 CLB columns in the latter. $LL_{1,1}$ offers FPGA resources $\{432, 24, 8\}$ which satisfies the requirement A_1. If we reduce the width of $LL_{1,1}$ to 52 without adjusting the height, the resultant resource coverage on FPGA will be only $\{384, 24, 8\}$ which does not satisfy the CLB requirement hence it is an invalid footprint. However, by increasing the height to 9 while reducing width to 52, we can achieve $LL_{1,2}$ which offers FPGA resources $\{432, 27, 9\}$. Note that both $LL_{1,1}$ and $LL_{1,2}$ cause resource wastage. However, if we select a region with height of 7 for P_1, the CLB requirements cannot be satisfied even if its width is set to the entire device width of this FPGA (i.e. 64). The subsequent sections uses the proposed ANN based performance estimation technique to identify the best footprint for each module in a modular design methodology.

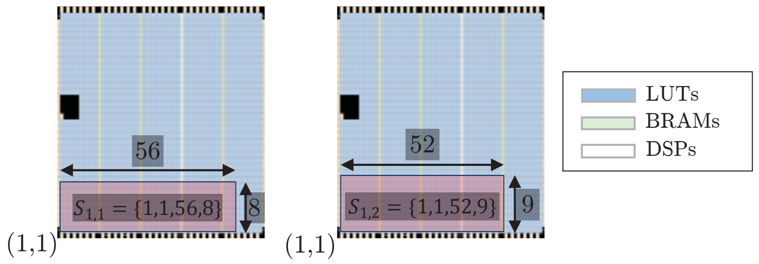

Fig. 2: Footprint variation for a given subsystem

4.3 Methodology for training an Artificial Neural-Network

Now, we present our artificial neural-network (ANN) based approach to estimate the performance of a given design partition. We train the ANN using application and architecture parameters through supervised training technique by providing the expected output performance (F_{max}) for the respective input data.

Training dataset In order to generate a large dataset for a better ANN training, we use the RTL design of a benchmark application and treat it as a design partition p_1. For this p_1, we exhaustively generate many footprints $LL_{1,x}$, where each $LL_{1,x}$ has different values for the parameters as explained below, and is compiled using Altera Incremental Compilation to get the respective F_{max}. We generated the footprints in this exhaustive manner for several benchmark as well as handwritten applications in order to create a large training data for the ANN.

Modelling parameters The following are the input parameters we use to model the performance of a given footprint. These parameters are analogous to independent variables in typical regression.

1. Architecture features: Starting horizontal coordinate (x), width (w) and height (h) of the rectangular region on FPGA space.
   Note that we ignore the vertical coordinate (y), as a feature due to uniformity of the FPGA architecture along vertical direction. In other words, if we move any rectangular region along vertical axis on FPGA space, it does not change the number of resources within that region. The FPGA architectural features in a region, such as the number and position of resource columns of each resource type, are captured by the (x), (w) and (h) parameters.
2. Application Area: Area requirement A_i, as discussed in Section 4.2, signifies the minimum resources needed to implement the application partition p_i.
3. Application internal links (c_i): Performance (F_{max}) of a design partition depends on number of available routing resources in the respective region. High routing resource utilization within the region typically results in longer routing paths for some routes, which could lower the performance.
4. Application default critical path (cp_i): Critical path of a design is the longest path between two registers. Some applications may contain critical paths along the combinational logic while others may have paths along memory and DSP I/O. To have this differentiation in our dataset, we include a parameter as an estimation of the critical path for each application. However, this estimated critical path is same for all the footprints of a single application, and should not be mistaken with the output of the model.

Problem statement Given a design/application partition p_i with an area requirement A_i, with c_i internal links and a default critical path cp_i, what is the expected F_{max} if it is compiled to a footprint of $S_i = \{x, w, h\}$.

Training procedure As shown in figure 3, we use a benchmark or handwritten application in entirety for the training step. The application is treated as an isolated module and is defined as a design partition in Quartus tool. We first extract the application specific parameters 2,3 and 4 defined above by performing a one-time incremental compilation for the defined designed partition. This compilation is done without footprints information.

Next, we generate a series of footprints $LL_{i,k}$ by setting different shape properties $S_{i,k}$ using LogicLock for set of i applications. Figure 4 shows the way we set $S_{i,k}$. For $LL_{i,k}$ we set $x = 1$ and w to the device maximum width (i.e. 64 for our case). The height h of the footprint is calculated using A_i, which is known from onetime-compilation discussed above. For instance, if $A_i = \{421, 20, 0\}$ then for a rectangular region on an EP2C35 FPGA device, with starting coordinate $x = 1$ and $w = 64$, the height should be at least 8 to satisfy the area constraint

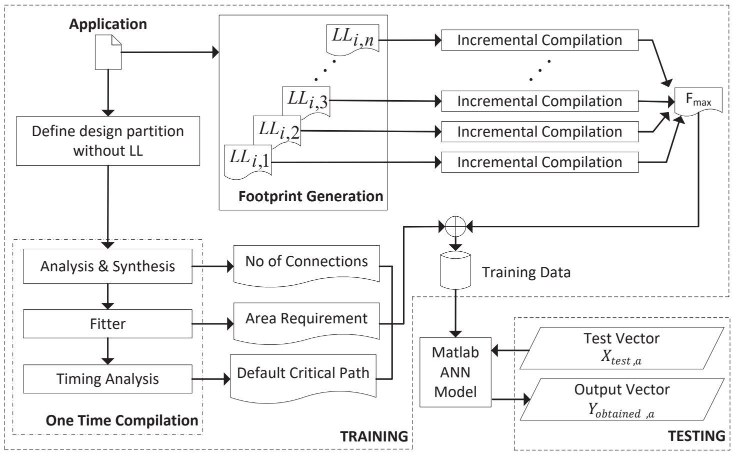

Fig. 3: ANN Training Methodology

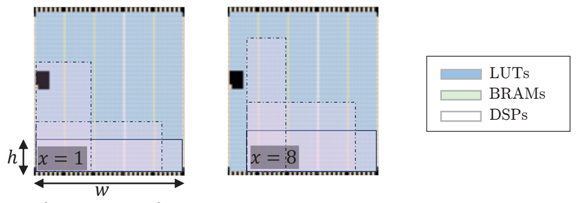

Fig. 4: Generating footprints for training

A_i. Therefore, we set $S_{i,1} = \{1, 64, 8\}$. For the next footprint $LL_{i,2}$, w is set to 63, reduced by one compared to $LL_{i,1}$, while keeping $x = 1$. Similar to previous case, h is calculated according to A_i, resulting $S_{i,2} = \{1, 63, 8\}$. Likewise, set of footprints are generated by reducing w and calculating respective h while $x = 1$, until w cannot be reduced further to generate a valid footprint. This footprint generation process can be repeated by shifting the starting position $x = 2, 3, ...$ until there are no more valid footprints. We obtained the training data by changing x and w with an interval of 2.

We used separate Quartus project to compile each footprint using Quartus Incremental Compilation. The application code is included into a single VHDL entity and kept within project's top level entity. During the incremental compilation, we set the netlist preservation level of the application to *Source Code Level* while the top level entity is kept as *Empty* preservation level in order to treat the application as an isolated module. After each compilation, we note the performance (F_{max}) of the application located at the respective footprint. This (F_{max}) is used as the output of training data set. Note that, all these footprints carry same application specific modelling parameters. An example for a training dataset entry k of application i can be notified as $\{S_{i,k}, A_i, c_i, cp_i, F_{max_{i,k}}\}$. Once the training dataset is created using all the benchmark and handwritten applications, we train a standard MatLab deep learning ANN model [4]. ANN model is trained with 8 hidden layers and using standard Resilient Backpropagation training algorithm. Appropriate number of hidden layers and the training function were found empirically.

5 Results and Discussion

5.1 Benchmarks applications and target platforms

We use a total of 26 applications, 16 from Polybench benchmark suite [21] and 10 handwritten, in this work. Out of these 26 applications, 23 were used during the training phase, while the remaining 3 were used to evaluate the proposed methods. Application parameters of these applications are shown in Table 1. As discussed in section 4.3, Table 1 presents the modelling parameters for each application, i.e. Application area ($A_i(l, m, d)$), Application internal connections (c_i) and Application default critical path (cp_i).

For an extensive evaluation, we experimented using four FPGA devices; (a) Cyclone II, EP2C35 - a smaller device with homogeneous floorplan (b) Cyclone V, 5CGXBC4 - one of the latest device with heterogeneous floorplan which includes number of BRAM and DSP columns (c) Cyclone V, 5CGTFD9 - largest FPGA of Altera Cyclone FPGAs (d) Cyclone 10LP, 10C025 - very recently released FPGA. Characteristics of these devices are listed in Table 2.

Table 1: Benchmark applications used for modelling and testing

	Application	CLB(l)	BRAM(m)	DSP(d)	Connections(c_i)	Default Critical Path(cp_i)
1	2mm	36	40	3	128	98.01
2	3mm	51	56	9	384	234.96
3	bicg	25	12	3	256	235.07
4	conv2d	19	16	2	64	108.31
5	conv3d	25	64	2	64	101.55
6	doitgen	22	72	3	160	167.95
7	fdtd2d	56	25	2	384	91.93
8	gemm	26	24	3	128	197.75
9	gemver	72	16	9	544	163.23
10	gesummv	34	19	6	224	189.54
11	mvt	29	12	6	256	209.47
12	symm	26	24	3	128	212.76
13	syrk	33	16	6	160	194.59
14	syr2k	37	24	6	192	211.73
15	atax	25	11	3	224	239.29
16	trmm	21	16	3	96	234.58
17	subsys1	421	20	0	3200	163.91
18	subsys2	360	18	0	2688	167.5
19	subsys3	349	20	0	2688	165.7
20	subsys4	357	22	0	3008	168.92
21	subsys5	325	21	0	2496	170.39
22	subsys6	121	16	0	960	183.82
23	subsys7	61	14	0	480	191.75
24	subsys8	91	21	0	800	168.63
25	subsys9	523	30	0	4512	157.8
26	subsys10	523	36	0	4704	164.8

Table 2: Target FPGA architecture characteristics

Family	Device	Logic elements	BRAM columns	DSP columns
Cyclone II	EP2C35	35k	3	2
Cyclone V	5CGXBC4	50k	8	5
Cyclone V	5CGTFD9	301k	11	6
Cyclone 10LP	10CL025	25k	2	2

5.2 Training Error and Testing Error

Training the ANN based model The training data set contains data points from 23 applications as described above. These applications are compiled at various footprints as described in Section 4.3 to generate the large training dataset.

We calculate the training error for each application for every footprint as a percentage difference between the actual performance (F_{max}) and the predicted performance from the model. The average training error across all applications at every footprint for each device is shown in Table 3.

Table 3: Training Error

Device	No.of data points	Average training error(%)
EP2C35	6818	3.5
5CGXBC4	7059	3.27
5CGTFD9	19738	2.65
10CL025	3471	2.58

Testing the model To evaluate the effectiveness of the model, we use three applications, where the expected output (ground truth) was first obtained by compiling these applications using Quartus at every footprint. We use the *atax* and *2mm* applications from Polybench [21] and a handwritten *subsys10* application for the testing phase.

We calculate the test error similar to the training error above. Table 4 shows the individual test error for the 3 test applications, average across all footprint, for each device. Overall, the maximum average error across all applications and devices is less than **4%**.

To evaluate the error values in a greater detail, we also plot the histogram of errors at the various footprints for each of the test applications on the EP2C35 device. Figure 5 shows these histogram plots. It can be clearly observed that at the majority of footprints, the error stays at the lower end of the histogram, showing the overall effectiveness in predicting the performance (F_{max}) using the proposed ANN based technique. Error histograms for the other devices are similar to figure 5 hence are not shown in the paper due to the space constraints.

5.3 Discussion

While the above results confirm the accurate prediction of F_{max} using the proposed ANN based technique for a given design partition in a footprint, a more interesting application of our approach is to identify the best footprint that provides the highest F_{max} for a given design partition.

It is indeed possible to obtain this using the proposed methodology by rapidly predicting the performance at all footprints using our model and then identifying

Table 4: Test error percentages

Device	atax	2mm	subsys10
	Average Error(%)	Average Error(%)	Average Error(%)
EP2C35	2.56	1.36	2.30
5CGXBC4	3.76	2.43	2.99
5CGTFD9	3.79	1.76	3.63
10CL025	3.35	1.68	3.08

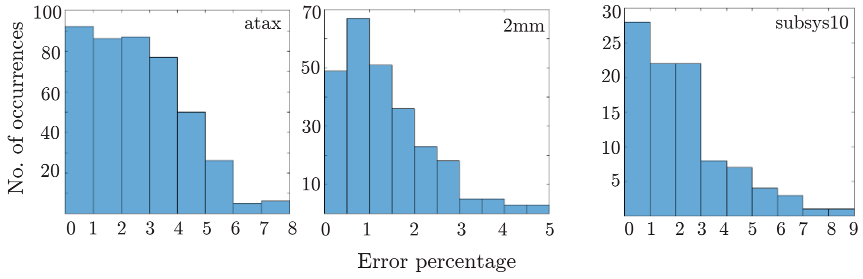

Fig. 5: Test error percentage histogram

the footprint with the highest F_{max}. This is immensely useful in a modular design methodology, where, instead of storing a pre-placed and routed database for all the partitions for all possible locations, a smaller database could be maintained during full design incremental compilation by setting preservation level of the best footprint set of *post-fit*. Though we explain the proposed work using Quartus specific terminology, similar toolset is available in Xilinx environments as well [5]. Hence the proposed methodology is also applicable Xilinx based designs.

It should also be noted that, the model is architecture specific and needs to be trained for each device. However, this requirement is common for all the module based FPGA designing approaches since the post-fit netlists are indeed architecture specific.

6 Conclusion

In this paper, we presented our artificial neural-network based methodology to estimate F_{max} of a design partition at any footprint on a FPGA device. We have used Altera Quartus incremental compilation tools for implementing the design partitions and used MatLab to train the ANN model. The proposed methodology accurately estimates F_{max} with less than 4% average error across the test applications and 4 widely used latest state-of-the-art FPGA devices. The proposed technique can be of immense benefit in a modular design methodology to reduce module library sizes by keeping only the best footprints set for a given partition while discarding the other footprints.

While in this work, we can accurately estimate the best footprint for a partition in isolation, in future, we propose to find the best footprints for all the partitions in a design in order to obtain a globally optimal footprint for the entire design that achieves the highest F_{max} performance.

References

1. Altera. https://www.altera.com/
2. CycloneII FPGAs. https://www.altera.com/products/fpga/cyclone-series/cyclone-ii/support.html
3. Increasing Productivity with Quartus II Incremental Compilation. https://goo.gl/uy225f
4. Neural Network Toolbox. https://www.mathworks.com/products/neural-network.html
5. PlanAhead Design and Analysis Tool. https://www.xilinx.com/products/design-tools/planahead.html
6. Vivado Design Suite User Guide-Hierarchical Design. https://goo.gl/6bUqqD
7. Xilinx. https://www.xilinx.com/
8. Coole, J., et al.: BPR: Fast FPGA Placement and Routing Using Macroblocks. In: CODES+ISSS (2012)
9. Frangieh, T., et al.: A design assembly framework for FPGA back-end acceleration. Microprocessors and Microsystems (2014)
10. Gort, M., et al.: Design re-use for compile time reduction in FPGA high-level synthesis flows. In: FPT (2014)
11. Gupta, S., et al.: CAD Techniques for Power Optimization in Virtex-5 FPGAs. In: Custom Integrated Circuits Conference, 2007. CICC'07. IEEE (2007)
12. Haroldsen, T., et al.: Rapid FPGA design prototyping through preservation of system logic: A case study. In: FPL (2013)
13. Herath, K., et al.: Communication-aware Partitioning for Energy Optimization of Large FPGA Designs. In: GLSVLSI (2017)
14. Lavin, C., et al.: HMFlow: Accelerating FPGA Compilation with Hard Macros for Rapid Prototyping. In: FCCM (2011)
15. Lavin, C., et al.: Impact of hard macro size on FPGA clock rate and place/route time. In: FPL (2013)
16. Lee, K., et al.: Shape Exploration for Modules in Rapid Assembly Workflows. In: ReConFig (2015)
17. Love, A., et al.: In pursuit of instant gratification for FPGA design. In: FPL (2013)
18. Ludwin, A., et al.: Efficient and deterministic parallel placement for FPGAs. ACM Transactions on Design Automation of Electronic Systems (TODAES) (2011)
19. Mao, F., et al.: Dynamic Module Partitioning for Library based Placement on Heterogeneous FPGAs . In: RTCSA (2017)
20. Murray, K.E., et al.: Titan: Enabling large and complex benchmarks in academic CAD. In: Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on (2013)
21. Pouchet, L.N.: Polybench: The Polyhedral Benchmark Suite. http://web.cs.ucla.edu/ pouchet/software/polybench/ (2012)
22. Rabozzi, M., et al.: Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming. In: FCCM (2014)
23. Tessier, R.: Fast Placement Approaches for FPGAs. TODAES (2002)
24. Trimberger, S.M.: Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. Proceedings of the IEEE (2015)
25. Vipin, K., et al.: Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration. Reconfigurable Computing: Architectures, Tools and Applications (2012)
26. Wirthlin, M., et al.: Future Field Programmable Gate Array (FPGA) Design Methodologies and Tool Flows. Tech. rep. (2008)