Communication-aware Partitioning for Energy Optimization of Large FPGA Designs

Kalindu Herath, Alok Prakash, Jiang Guiyuan, Thambipillai Srikanthan
Nanyang Technological University, Singapore
kalindub001@e.ntu.edu.sg, {alok, gyjiang, astsrikan}@ntu.edu.sg

ABSTRACT

Modern FPGAs integrate multi-million logic resources that allow the realization of increasingly large designs. However, state-of-the-art simulated annealing based CAD tools for FPGA suffer from long runtime, poor performance and sub-optimal routing and placement decisions, especially for large applications, leading to less energy efficient designs. In this paper, we present a partitioning methodology that divides large application into smaller subsystems based on the communication frequency between these subsystems. We leverage the existing CAD tools to compile the large design, which is now annotated with their subsystems, to obtain the final bitstream. Experiments show that the proposed strategy can lead to a performance gain of over 60% while still achieving more than 20% reduction in energy consumption.

Keywords

CAD; communication-aware partitioning; energy-efficient

1. INTRODUCTION

The continuous scaling of transistor has enabled Field Programmable Gate Array (FPGA) manufacturers such as Xilinx [4] and Altera [1] to integrate millions of logic resources into modern FPGAs, providing opportunities for designers to develop increasingly complex FPGA designs. At the same time, the incorporation of FPGA in the latest iPhone 7 smartphone from Apple™ [15] bears true testament to the increasing popularity and usefulness of FPGAs. However, FPGAs are still less energy-efficient than their ASIC counterparts and hence their integration in battery powered devices can prove to be detrimental. This issue can be alleviated, to some extent, by extensive power-conscious application-specific customization of FPGA designs [6]. The current Computer-Aided Design (CAD) tools, however, are not sophisticated enough to cater for such customizations. Specially, compiling large applications typically results in very long compilation times, poor placement and routing decisions that inevitably leads to degradation in circuit performance, area and energy consumption [5].

Among CAD steps, scalability of placement step has been discussed recently [10] [5]. An extensive search in literature shows that the widely popular Simulated Annealing (SA) based placement tools poorly scale for larger applications [5] [9] despite being capable of producing high-quality placement for smaller designs [10]. Partitioning based placement tools [12] show faster runtime, but, this comes at the cost of degraded quality of result. Analytical placement tools [10] have been known to achieve better balance between scalability and quality of results. There have also been attempts to lower the runtime of placement by introducing pre-compiled macro block based design flow [8] [7]. But these techniques introduce large degradation in placement quality. The commercial FPGA vendors have also improved the scalability of their placement tools. For instance, Altera uses parallel SA techniques [11] whereas Xilinx relies on scalable analytical placement techniques [10]. These research endeavors, however, mostly focus on optimizing performance, leaving an inadequacy for an energy efficient CAD tools for large applications. In addition, these approaches have not considered runtime characteristics of the application during compilation. Existing work has also identified that the FPGA interconnect resources consumes most of its dynamic power[14][16]. Hence, reducing the usage of these interconnect resources can effectively lead to power and energy-efficient FPGA designs.

In this paper, we propose a communication-aware design mapping technique that is especially targeted towards large and complex FPGA designs. *The proposed technique partitions such FPGA designs into smaller subsystems such that the intra-subsystem communication frequency is maximized, while the inter-subsystem communication frequency is minimized.* This strategy minimizes the energy consumption in the interconnects by reducing the frequency of long distance communication. Additionally, the partitioning of the large design into smaller subsystems also aids the CAD tools during compilation since it already excels in placement for smaller designs [10]. We rely on the existing CAD tools for the entire compilation process and only annotate the input design with the partitioning information using pragmas.

2. MOTIVATION

The main modes of power dissipation in FPGAs are transistor leakage current (static power), short circuit current and the charging/discharging of capacitive loads. Amongst these modes of power dissipation, charging/discharging of capacitive loads contributes the most to the overall power dissipation, and is modelled as [14];

$$P = \frac{1}{2}\Sigma_{i \in nets} C_i.\alpha_i.v^2 \quad (1)$$

where $nets$ are all the connections and respective components in the design, P is the average power consumption, C_i is the capacitance of net i, α_i is the average toggle rate of net i and v is the supply voltage. In a typical application, some code sections are executed more frequently than the rest. Hence, the data flow rate of the interconnect links between such frequently executed computation units and the corresponding memory blocks is relatively higher, leading to higher toggle rate α and consequently higher intercon-

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada

DOI: http://dx.doi.org/10.1145/3060403.3060441

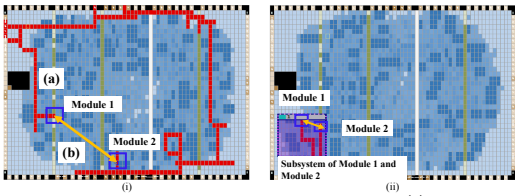

Figure 1: Floorplan for sample application (i) Without Subsystems (ii) With Subsystems

nect power consumption. Additionally, as discussed in the previous section, dynamic power dissipation of FPGAs are dominated by the power consumed by the interconnect resources. This is due to its higher capacitive loads which typically increases with increase in the length of the interconnect[14]. Hence, *it is essential to have shorter routing paths and even more so for the nets with high toggle rate to have overall lower power consumption.*

In order to observe the extent of degradation in power and performance of a large application on an FPGA, we performed a case study using the commercial Quartus CAD flow. An application is considered to be 'large' if it consumes above 80% device logic, memory and routing resources. We developed a hand-coded application, which contains N number of computation units (CU) and memory units (MU). Each CU is connected to n number of MUs. We tuned the parameter N and n to satisfy the above requirement of a large application. Figure 1 shows the floorplan of a Quartus mapping of a sample large application. The high demand for routing resources for the application in Figure 1 has exhausted the routing resources in the central region of the FPGA floorplan. Hence, 2 modules as shown in the floorplan have been connected via an unusually long interconnect path 'a', instead of a far more preferable interconnect path, 'b'. Additionally, since the tool does not consider the communication frequency between the modules during the routing phase, the problem is exacerbated if these two modules communicate frequently, thereby resulting in high power consumption. We observe the maximum allowable frequency (F.Max) for the resulting design, as well as its total energy consumption after running the design on the target FPGA.

Next, we partition this large application into a set of subsystems such that highly communicating computation and memory nodes become part of a single subsystem. We will discuss in detail regarding the subsystems and the exact algorithm for the partitioning step later in Section 3. Next, we use pragmas to instruct the Quartus to place the identified subsystem RTL components as a group. After the mapping is done by the Quartus, we again observe the F.Max and the total energy consumption for this partitioned design.

We observed that the F.max increased from 50.61 MHz to 89.17 MHz while the total energy consumption reduced by over 17%, by using the proposed partitioning strategy. This example proves that the Quartus is unable to optimally map large application into the FPGA. It also shows the suitability of the proposed communication-aware partitioning strategy, explained in the next section, for large application to improve the quality of the final mapping.

3. PARTITIONING FRAMEWORK

The proposed algorithm takes an RTL design, represented as a graph, as input and produces a network of subsystems. A subsystem refers to a group of computation and memory modules with high intra-group communication, which we identify to be placed in close proximity to each other.

3.1 Problem Statement

An FPGA RTL level design can be represented as a graph $G(V, E)$ where V is a set of nodes where each node represents a RTL module; i.e. computation unit or a memory units, and E is a set of edges where each edge represents a connection between two RTL modules. Each design unit node has an area requirement on the FPGA floorplan. The minimum area needed by the unit (vertex) v_i is noted as a_i (l_i, m_i) where l_i and m_i represent number of CLBs and number of Block RAMs (BRAMs) respectively. The communication frequency between two modules (nodes) is analogous to the toggle rate α in the equation 1 and is represented by the cost value of each connection (edge) e_j. The total available number of CLBs and BRAMs in the device can be noted as A_l and A_m respectively. Maximum allowed wires per unit FPGA area (also referred to as connection density) is represented by C and is assumed to be a constant for the entire FPGA area. Given the design graph $G(V, E)$, area requirement of design units $a(l, m)$, communication frequency of each connection between design units α and the FPGA parameters A_l,A_m, and C, the communication-aware design partitioning problem can be thought of as dividing the graph into n ($\leq N$) number of subgraphs (referred to as subsystems), where N is the upper bound of number of subgraphs. The optimization goal for the partitioning problem is to minimize the total inter-subsystem communication activity, and is subjected to the following constraints.

1. Area Constraint: Total number of CLBs and BRAMs required by each subgraph must be less than a predefined value $A_{l\prime}$ and $A_{m\prime}$ respectively
2. Congestion Constraint: Ratio of Area and the corresponding connections in each subgraph must be less than C

3.2 Algorithm

Algorithm 1 presents our greedy clustering strategy for the partitioning step. We start the clustering process by treating each computational module and memory module as a subsystem with only one element. The algorithm maintains a list *edge_list* of all the edges and respective communication frequency of the connection in Line 1 of Algorithm 1. In Line 3 through 6, each iteration of the *while* loop, an edge is first selected from this list using the *get_valid_edge*(*edge_list*) procedure. Then, the associated two subsystems of the selected edge are merged to form one (larger) subsystem using the *merge*(*edge*) procedure. This selection of edges and their merging process is repeated until all the edges are exhausted.

3.2.1 Valid Edge Identification

An edge can be identified as a valid candidate for *merging operation* if the associated subsystems (nodes) of the edge satisfy the Area and Congestion constraints, as identified above, upon merging. Hence, in Line 8 through 18 of Algorithm 1, if there exists any valid edge, we choose the most profitable edge among them for the *merging operation.*

3.2.2 Merge Operation

Line 20 through 31 of Algorithm 1, present the *merge*(*edge*) procedure. Nodes a and b, associated to the edge e are merged to form a new subsystem a', and the edge is removed from *edge_list*. There can be other nodes, e.g. c, adjacent to both a and b that should now maintain a single edge from a'. Other edges associated with nodes a and b are added to the subsystem without modifying their communication frequency. The changes of the graph due to new merged node a' are handled by *update_edge_list*() procedure on Line 30.

Algorithm 1 Communication-aware partitioning algorithm

```
1: edge_list ← communication activity α for all e ∈ E
2:
3: while has_valid_edge do
4:     edge ← get_valid_edge(edge_list)
5:     merge(edge)
6: end while
7:
8: procedure GET_VALID_EDGE(edge_list)
9:     e_max_profit ← first_edge(edge_list)
10:    for each e ∈ edge_list where e ∉ marked_edges do
11:        if meet_constraints(e) then
12:            if profit(e) > profit(e_max_profit) then
13:                e_max_profit ← e
14:            end if
15:        end if
16:    end for
17:    return e
18: end procedure
19:
20: procedure MERGE(edge)
21:    {a, b} ← get_connecting_nodes(edge)
22:    a′ ← merge of {a, b}
23:    remove(edge)
24:    for each node c where c ∼ a and c ∼ b do
25:        e_c1 ← get_associated_edge(c, a)
26:        e_c2 ← get_associated_edge(c, b)
27:        x ← merge of e_c1, e_c2
28:        α_x ← α_e_c1 + α_e_c2
29:    end for
30:    update_edge_list()
31: end procedure
```

3.2.3 Calculating the profit

We consider two factors to evaluate profit of merging two nodes (subsystems) a and b, connected by edge e, into $a\prime$. First is the communication between all the RTL modules in a and b. The higher the communication between modules, the better the quality of the resultant subsystem $a\prime$. We term this communication as *intra-subsystem communication*. The other factor is the communication between resultant subsystem $a\prime$ and the nodes outside, which we call as *inter-subsystem communication*. The clustering process strives to increase the *intra-subsystem communication* and decrease the *inter-subsystem communication*. Hence, the profit of merging subsystems (a, b) which is connected by edge e is:

$$profit_{e(a,b)} = \frac{intra\text{-}communication}{inter\text{-}communication} \quad (2)$$

3.2.4 Constraints

The partitioning strategy, discussed above, can lead to a trivial solution in extreme cases, where all the nodes are clustered together into a single subsystem so that the inter-communication is zero and intra-communication is maximum. Even in a less than extreme situation, it is not always beneficial to have lesser number of partitions that only satisfy the intra- and inter-subsystem communication goals and maximize the profit term expressed in Equation 2. To avoid such trivial solution, we control the maximum area of a subsystem during merging RTL components by setting area constraint $A_{l\prime}$ and $A_{m\prime}$ for LUTs and BRAMs respectively.

Regarding the congestion constraint, it is noteworthy that while restricting high communication links in a relatively small area improves the intra-communication, merging more nodes into subsystems also involves having more internal edges in the subsystem. These edges ultimately map to physical routing wire of the FPGA. Hence, many connections within a subsystem may cause high routing congestion regions after getting mapped to FPGA. Therefore, to avoid such regions, we control number of connection per unit area by introducing a term called *connection density*.

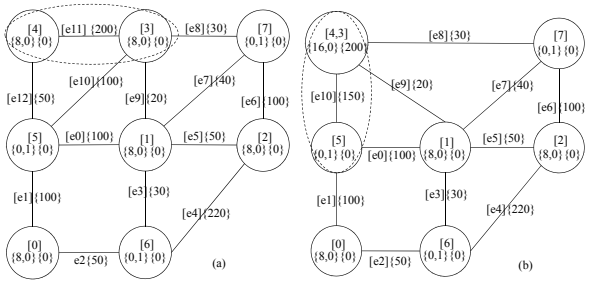

Figure 2: Execution of the algorithm a) Initial condition b) After the first execution

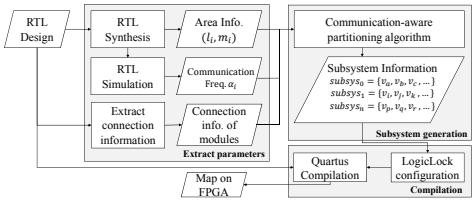

Figure 3: Communication-aware Partitioning

We explain our algorithm using a sample application shown in Figure 2(a). Each circle represents nodes within the subsystem, area consumption of the subsystem(LUTs,BRAMs) and intra-communication in the notation $[node\_list]$ $\{l, m\}\{\alpha_{intra}\}$. The edges in the graph are annotated with edge-id and inter-communication value as a tuple $[edge\_no]$ $\{\alpha_{inter}\}$. From Figure 2 (a), $e11$ is selected for merging as it shows a greater profit than $e4$, although $\alpha_{e11} < \alpha_{e4}$. Merging respective connecting subsystems of $e11$, subsystems [4] and [3], make a new subsystem $[4, 3]$ of area $(l, m) = (16, 0)$, as shown in Figure 2 (b). Since the edge $e11$ is now internal to the subsystem $[4, 3]$, the α_{intra} of the subsystem is now 200. In the next iteration, merging $e10$ is profitable, as it makes $\alpha_{intra} = 350$ and $\alpha_{inter} = 250$, resulting in a profit of 1.4 according to the Equation 2.

3.3 Communication-aware Partitioning

Figure 3 shows our communication-aware FPGA design partitioning methodology. We take an RTL design as the input. From a given RTL design code, the following parameters are extracted to execute the partitioning algorithm:

i) Area of modules: Typical RTL synthesis step of CAD flow estimates area of design modules. Since the runtime of RTL synthesis is minimal as compared to the runtime of placement and routing steps, we obtain area $a(l, m)$ of each node in $G(V, E)$ by performing a quick RTL Synthesis step.

ii) Communication Frequency of Edges: Next, we perform a RTL simulation where each edge $e \in E$ is treated as signal. Simulation signal dump files are then used to obtain communication frequency of each edge e, α_e.

iii) Graph Connectivity: We need a netlist of a given RTL design in order to build the graph $G(V, E)$. However, since commercial CAD tools like Quartus do not allow users to read generated netlist, we perform text processing on design code to extract the relevant information.

Table 1: Performance ($F.Max$) Improvement

App.	Without subsys. F.Max(MHz)	With subsys. F.Max(MHz)	F.Max Imp. (%)
Synth1	50.61	89.17	76.19
Synth2	50.61	69.26	36.85
Synth3	52.43	93.9	79.10
Atax′	51.91	90.23	73.82
Bicg′	57.8	81.01	40.16

Table 2: Power and Energy reduction using subsystems

App.	Total Power Reduction(%)	Routing Power Reduction(%)	Total Energy Reduction(%)
Synth1	6.51	19.80	41.77
Synth2	2.05	9.47	18.00
Synth3	0.56	1.31	18.67
Atax′	3.12	8.47	16.06
Bicg′	0.76	1.64	7.37

After the extraction, these parameters are input to our communication-aware partitioning algorithm. The output of the algorithm is subsystem pragmas for the LogicLock feature in Quartus tool. Each LogicLock is set of RTL modules belongs to a subsystem to be placed in a square region on FPGA. However, we do not instruct the size of the LogicLock and its physical placement of FPGA, and let Quartus compiler decide them. An application with its subsystems identified is then compiled using Quartus.

4. RESULTS AND DISCUSSION

We used the proposed communication-aware partitioning algorithm on various applications and observed their performance in terms of F_{max} and energy consumption. Altera's Quartus toolchain[2] was used for our experiments. For both, traditional and the proposed approaches, Quartus compiler optimizes for the performance of the mapping solution (rather than optimize on area).

After the Quartus compilation finishes for both cases, with and without subsystems, the performance (maximum running frequency - F_{max}) of each case is obtained from the compilation report. We also use the Quartus PowerPlay feature [3] by performing a gate level simulation to obtain an accurate power estimation in each case. Total Energy consumption is also derived from these parameters.

To evaluate our approach, we created three hand-coded applications, as described in section 2, to exhaust the logic and memory resources on a Altera EP2C35 FPGA device. We also modified and experimented with *Atax* and *Bicg* applications from Polybench [13] benchmark suite. Since, these application kernels are relatively small-scale design, we duplicated them multiple times with minor modifications to the kernel itself, to fully utilize the logic and memory resources of a Altera EP2C70 series FPGA device with 70K logic elements. We refer to them as *Atax′* and *Bicg′*.

The three hand-coded applications are mapped to the Altera EP2C35 FPGA device while *Atax′* and *Bicg′* are mapped to the Altera EP2C70 FPGA device, without and with subsystems generated from the proposed technique. Table 1 shows performance of these five applications in terms of maximum achievable frequency, F_{max}. As evident from this table, partitioning the large applications into subsystems, has significantly increased the performance of mapped solution with an average of over 61% in their F_{max}.

Next, Table 2 shows the percentage reduction in overall power consumption as well as for the interconnect resources only. It is evident that the proposed approach can lead to a significant reduction in the interconnect power consumption. Even the total power consumption is reduced while achieving higher performance, when compared to the traditional compilation method without communication-aware design partitioning. Applications such as Synth3 and *Bicg′* do not show significant reduction in power. However, they still show a significant improvement in their performance as seen in Table 1. After careful investigation of these applications, we observed that the CAD tools kept the communicating modules in close proximity for these applications even without partitioning the application into subsystems. Hence, these application do not show a significant reduction in power after the proposed communication-aware partitioning. However, this behavior of the CAD tools is not guaranteed for every application as evident by the significant improvement in power consumption for other applications.

Finally, as shown in Table 2, there is a significant reduction in the overall energy consumption when using the proposed partitioning strategy, especially for applications where both the performance and power consumption improve significantly when compared to the existing tools.

5. CONCLUSION

In this paper, we presented our communication-aware partitioning approach for large FPGA designs, that not only reduces the energy consumption, but also improves the maximum achievable operating frequency (F.Max) of the resulting design significantly. Experimental results on 2 FPGA devices show an average of over 61% improvement in F.Max and more than 20% reduction in energy consumption when compared to the traditional design flow.

6. REFERENCES

[1] Altera. https://www.altera.com/.

[2] Intel™FPGA Development Tools - Quartus Prime. https://goo.gl/3kMRQN.

[3] PowerPlay Power Analyzer Support Resources. https://goo.gl/1fE7NI.

[4] Xilinx. https://www.xilinx.com/.

[5] H. Bian et al. Towards scalable placement for FPGAs. In *FPGA*, 2010.

[6] D. Chen et al. Low-power high-level synthesis for FPGA architectures. In *ISLPED*, 2003.

[7] J. Coole et al. BPR: fast FPGA placement and routing using macroblocks. In *CODES+ISSS*, 2012.

[8] C. Lavin et al. HMFlow: accelerating FPGA compilation with hard macros for rapid prototyping. In *FCCM*, 2011.

[9] M. Lin et al. Improving FPGA Placement with Dynamically Adaptive Stochastic Tunneling. *IEEE TCAD*, 2010.

[10] T.-H. Lin et al. An efficient and effective analytical placer for FPGAs. In *DAC*, 2013.

[11] A. Ludwin et al. Efficient and deterministic parallel placement for FPGAs. *ACM TODAES*, 2011.

[12] P. Maidee et al. Timing-driven partitioning-based placement for island style FPGAs. *IEEE TCAD*, 2005.

[13] L.-N. Pouchet. Polybench: The polyhedral benchmark suite. https://goo.gl/t4dm3U, 2012.

[14] L. Shang et al. Dynamic power consumption in Virtex-II FPGA family. In *FPGA*, 2002.

[15] A. Tilley. This Mysterious Chip In The iPhone 7 Could Be Key To Apple's AI Push. https://goo.gl/ZVBgUH, 2016.

[16] T. Tuan et al. A 90Nm Low-power FPGA for Battery-powered Applications. In *FPGA*, 2006.