Power-Efficient Mapping of Large Applications on Modern Heterogeneous FPGAs

Kalindu Herath, Alok Prakash, Suhaib A. Fahmy, *Senior Member, IEEE*, and Thambipillai Srikanthan, *Senior Member, IEEE*

***Abstract*—The increasing size of modern FPGAs allows for ever more complex applications to be mapped onto them. However, long design implementation times for large designs can severely affect design productivity. A modular design methodology can improve design productivity in a divide and conqueror fashion but at the expense of degraded performance and power consumption of the resulting implementation. To reduce the dominant power dissipation component in FPGAs, the routing power, methodologies have been proposed that consider data communication between modules during module formation and placement on the FPGA. Selecting proper mapping region on target FPGAs, on the other hand, is becoming a critical process because of the heterogeneous resources and column arrangements in modern FPGAs. Selecting inappropriate FPGA regions for mapping could lead to degraded performance. Hence, we propose a methodology that uses communication-aware module placement, such that modules are mapped by selecting the best shape and region on the FPGA factoring the columnar resource arrangements. Additionally, techniques for module locking and splitting have been proposed for deterministic convergence of the algorithm and for improved module placement. This methodology exhibits nearly 19% routing power reduction with respect to commercial CAD flows without any degradation in achievable performance.**

***Index Terms*—FPGAs, CAD, floorplanning, modular design methodology, routing power.**

I. Introduction

Modern Field Programmable Gate Arrays (FPGAs) comprise millions of logic resources, as well as different types of hard modules such as Digital Signal Processing (DSP) Blocks and Block Memories (BRAMs) in various sizes. The growth in resource capacity and falling costs of FPGAs have enabled the implementation of larger applications onto them [1]. Though FPGAs offer a promising target platform for large applications from domains such as artificial intelligence and deep neural networks [2], cloud services [3] and autonomous vehicles [4] [5], [6], the design process is cumbersome and time consuming. Compilation of large applications using low-level Hardware Description Language (HDL) based design tools is cumbersome and verification is challenging. There have been a range of efforts to improve design productivity in commercial FPGA CAD tools recently. However, compilation times remain long, hampering the design cycle. Studies have shown that compiling large applications can take hours or even days [7]. Moreover, tools tend to generate implementations with lower quality of results in terms of performance and power compared to the implementation of smaller designs.

Modular Design Methodology (MDM) has been proposed to improve the design productivity of large FPGA designs. It divides large designs into smaller modules, compiles them individually and assembles them to obtain a complete implementation. MDM provides better quality of results and helps designers preserve satisfactory compilation of modules. However, the division of large designs into smaller modules can result in a loss of inter-module connectivity information, resulting in degradation of performance and power.

The charging and discharging of capacitive loads on the interconnect fabric that consumes up to 80% of total FPGA area, has been identified as the dominant power dissipation factor on FPGAs, contributing to as much as 50% of total dynamic power dissipation [8]. Since longer wire length increases effective capacitance, switching activity on longer interconnect wires should be minimized in order to reduce power dissipation.

On FPGAs, architecture data-flow is analogous to the switching activity of all the nets routed on interconnect fabric. In order to minimize switching on longer interconnect wires, we previously proposed an MDM-based technique in [9], where design entities with data-flow intensive links are grouped into a single module. The CAD tool is then instructed to allocate these modules into a rectangular region (size and location determined by the CAD tool). Hence the entities are included in the same module, and the data-flow intensive links are expected to be routed with shorter interconnect wires.

However, arbitrarily increasing the size of such modules, i.e. grouping several entities into a single module, inversely affects wire-length as entities with intensive data-flow links may no longer be placed in close proximity. Hence, the module creation step requires an upper bound for module size. As a result, however, all high data-flow intensive links may not be encompassed within modules, resulting in significant inter-module data-flow. Therefore, an inter-module data-flow aware module placement strategy is required in order to minimize the overall power dissipation.

Modern heterogeneous FPGA architecture floorplans incorporate 2 types of heterogeneity. 1) Different resource types like DSPs and BRAMs, with potential variations. For instance, Intel FPGAs offer BRAMs with different sizes, such as M9K (9 Kb) BRAMs and M144K (144 Kb) BRAMs, and modern Xilinx devices include 36 Kb BRAMs and 288 Kb UltraRAMs (URAMs). 2) Resources arrangements with variably spaced columns. In such FPGA floorplans, the physical shape and

K. Herath, A. Prakash, and T. Srikanthan are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. e-mail: (kalindu.herat, alok, astsrikan)@ntu.edu.sg.

S. A. Fahmy is with King Abdullah University of Science and Technology (KAUST), Saudi Arabia. e-mail: suhaib.fahmy@kaust.edu.sa

placement of modules on the FPGA determine the availability of different resource types and arrangements. Further, certain column arrangements better suit modules with different resource mixes. Thus, module placement as well as the shape of FPGA region where the module is mapped are both critical. Sub-optimal assignment of shapes and locations to modules incurs resource wastage and degrades performance and power.

In this work we propose a module placement methodology which:

- minimizes the distance between modules with higher inter-module data-flow, hence reducing routing power dissipation
- estimates the performance of each module for different FPGA regions and shapes, and hence selects the FPGA region that gives the highest performance

The proposed technique leverages the Artificial Neural Network based module performance technique proposed in [10] in order to obtain estimated performance for different FPGA regions. Preliminary investigation of this methodology was presented in [11]. In this paper we extend the approach to further reduce power consumption and improve performance (frequency) while minimizing the runtime of the proposed algorithm. The rest of the paper is organized as follows. In Section II, we discuss the background and existing literature followed by the motivation in Section III. The proposed methodology is discussed in Sections IV through VI. The methodology is evaluated in Section VII and we conclude in Section VIII.

II. Background and Related Work

Modular Design Methodology (MDM) was developed to increase design productivity by allowing designers to compile designs partially and preserve satisfactory partial compilations. A MDM flow constitutes two phases: *module creation* and *module assembly*.

In the module creation phase, identified modules of a design are compiled for every possible location on FPGA and stored in a module library. Module creation in Frontier [12] and HMFlow [13] are done at a finer level where RTL components such as multiplexers and adders are considered as modules. BPR [7], on the other hand, creates modules with much coarser components such as FFT and FIR filters. Module creation in QFlow [14] is based on the rate of modifications required in each module creating invariant and evolving sets rather than ones based on granularity. However, all these techniques focus on faster CAD flow runtime, but do not consider performance or power. Our previous work in [9] suggested a methodology to partition a large design into modules such that entities with higher data-flow between them are grouped together, resulting in closer proximity. Consequently, links with higher data-flow within the modules are routed with shorter interconnect wires leading to a reduction in routing power. However, module size constraints could still mean some high data-flow links were not suitably grouped.

In the module assembly phase, individual modules are placed and routed to form the complete design and hence finalize the compilation process. Here, the pre-compiled or pre-synthesized modules are selected from the large module database for each module, with the objective of reducing a placement cost function. In Frontier [12], the placement cost is a combination of timing and wiring cost, whereas BPR [7] also considers expected routing congestion. In the module assembly phase of HMFlow [13], its heuristic placer introduces more parameters such as the size of modules, amount of connectivity, and also the presence of BRAMs and DSPs in addition to the conventional wire-lenth and timing based cost functions. A hetereogeneous resource column based approach to floorplanning was proposed in [15], with a focus on partial reconfiguration. A standard cost function was altered in [16] by adding priority levels to modules during module placement, and in [17] by including area requirements and aspect ratios of modules to the cost function. In our prevous work [18], the module placement cost function was altered to perform power-aware module placement by incorporating data-flow between modules as a placement cost. However, most MDM workflows including [18], consider only one shape per pre-compiled module during module creation. In [19], a block based compilation process is proposed in order to reduce compilation time. Modules are assigned to regions without taking the the underlying architecture into consideration. The floorplanning technique proposed in [20] considers the heterogeneous resource column arrangement in modern FPGAs. However, the objective of that work is to find regions for each module while optimizing the wire-length. The shape of modules is the deciding factor in the utilization of the heterogeneous column arrangement of FPGAs and therefore, state-of-the-art module placement could cause suboptimal placement and resource wastage. As shown later in Section VII, this leads to excessive routing power consumption.

The authors in [21] extensively analyzed module shape and have shown that multi-shaped pre-compiled modules lead to better resource utilization. One of the reasons that multi-shape modules have not been widely used is the increase in module library size. To mitigate this the authors in [22] proposed a methodology to derive different module shape versions by partitioning a given version of a module. As a result, they could eliminate the need for pre-compiling all the module shape versions on every possible location on FPGA. However, none of these attempts have eliminated the need for a pre-compiled module library. The work in [10], proposed an Artificial Neural Network (ANN) based methodology to estimate performance for a given module on FPGA. The ANN model takes module application characteristics, intended module shape and placement information as inputs and estimates performance of the module. Using this technique, one can rapidly estimate the performance of a module for any shape and placement on FPGA. However, the ANN model is specific to the target FPGA which requires a one-time training process. In [11] we presented a methodology to use the ANN based performance estimator to obtain candidate shapes and placements for all the modules in the design and hence select the best module shape and placement for each module in order to complete the compilation process. The placement decision in that work aims to lower routing power dissipation by reducing the distance between modules with higher inter-

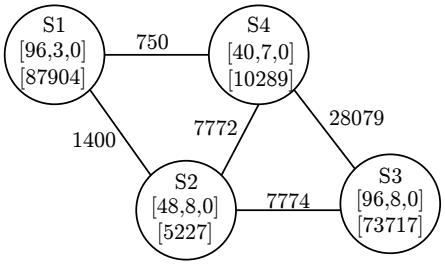


Fig. 1: A graph of subsystems.

module data-flow.

Commercial FPGA vendors provide tools to perform MDM in their CAD software. For instance, Intel Quartus Prime and Xilinx Vivado include incremental compilation tools for floorplanning. However, these CAD tools do not perform automatic and informed module creation and placement and require developer expertise. The academic and commercial state-of-the-art show the need for a power-aware MDM which considers the heterogeneous column arrangements in modern FPGAs.

III. Motivation and Proposed Method

A. Motivational Example

There are two modes of power dissipation on FPGAs; static power dissipation and dynamic power dissipation. The former occurs due to leakage current and the latter occurs due to signal transitions on capacitive loads. Although static power dissipation is considerable on modern FPGAs, dynamic power still dominates total power dissipation [23], [24]. Dynamic power is characterized in equation 1:

$$P_{dyn} = \frac{1}{2} \sum_{i \in nets} C_i.\alpha_i.v_{dd}^2 \quad (1)$$

where, $nets$ represents all physical connections on the FPGA, C_i is the capacitance of net i, α_i is the signal toggle rate of net i, and v_{dd} is the operating voltage of the FPGA. According to equation 1, high toggle rates (due to data-flow) with the higher capacitance of longer interconnect will drastically increase dynamic power dissipation. It should be noted that the term *communication* is used as an approximation to signal toggling (α_i) due to data-flow. Communication is defined as the total number of toggles of an RTL signal. The term *subsystem* refers to a collection of RTL entities that have high data-flow between them. Subsystems are analogous to modules that have been created using a communication-aware module creation methodology in state-of-the-art MDM, similar to [9].

In Figure 1, an example design is represented as a graph, where a node in the graph represents a subsystem S_i and an edge between S_i and S_j represents an inter-subsystem connection $S_{i,j}$. Nodes are annotated with a resource requirement tuple (CLBs, BRAMs, DSPs) $[l_i, m_i, d_i]$ and intra-subsystem communication α_i. Intra-subsystem communication refers to the total number of RTL signal toggles that occur between all the entities within a given subsystem. Similarly, inter-subsystem communication refers to the total number of RTL signal toggles that occur between corresponding subsystems.

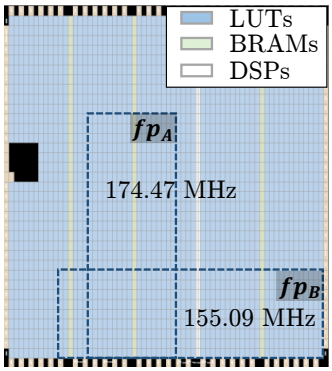


Fig. 2: Effect of footprint shape on performance.

In order to minimize dynamic power consumption, subsystem S_3 and S_4 must be placed in close proximity and subsystem S_2 should be closer to subsystems S_3 and S_4.

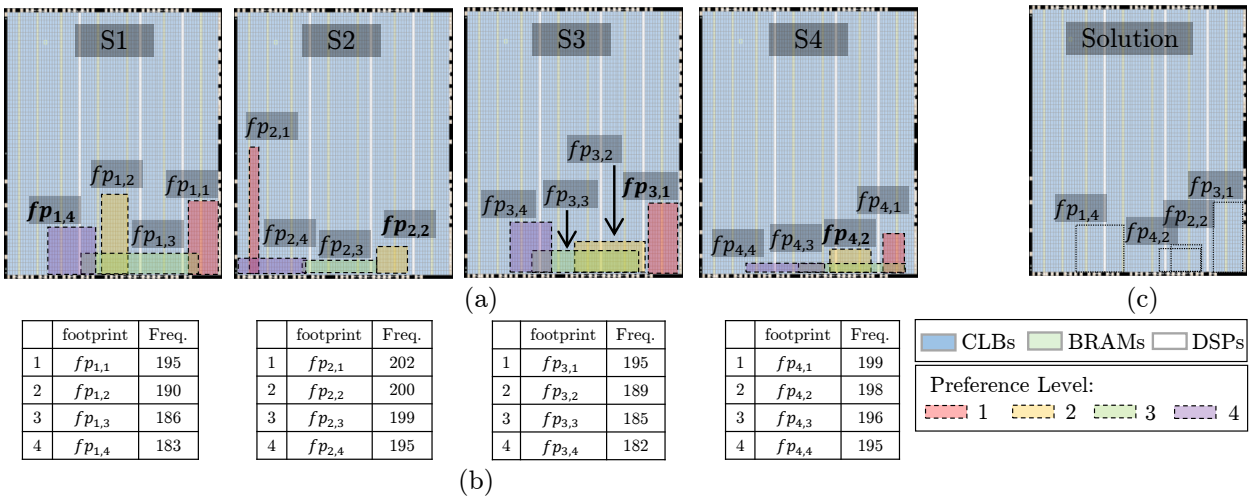
A subsystem (for instance S_1) can be mapped into a rectangular shape on the FPGA fabric. In this paper, the physical shape and placement of a subsystem on the FPGA is referred to as a *footprint*. In Figure 2, two footprints (fp_A and fp_B) are shown where a particular subsystem has been mapped. However, the resource availability and column arrangements within the footprints are different. For instance, in fp_A, the BRAMs within the footprint are arranged in a single column, whereas in fp_B, BRAMs are arranged in three columns. As a result, the two footprints achieve different frequencies as shown in Figure 2.

Some footprint versions are preferable for subsystem mappings in terms of performance due to their resource column arrangement [10], [21], depending on the characteristics of the subsystem and the underlying FPGA architecture (column arrangement). In this example, footprint fp_A is preferable for mapping subsystem S_1. In addition, as discussed earlier, subsystems with higher inter-subsystem communication must be mapped to suitable footprints at close proximity in order to minimize dynamic power dissipation. Therefore, this motivates development of a subsystem mapping methodology that considers both inter-subsystem communication and the resource column arrangement on modern heterogeneous FPGAs.

B. Proposed Methodology

The following example illustrates the proposed methodology. For each subsystem S_i shown in Figure 1, a set of footprints is generated for an Altera EP4CGX50 device, as shown in Figure 3(a). Each footprint of S_i is generated such that the resource requirement for the subsystem is satisfied. The expected performance for each footprint is estimated using the performance estimation technique in [10], and the estimated performance for each footprint is tabulated in Figure 3(b). The preference for selecting a particular footprint for each subsystem is based on its performance (preference level 1 for the highest performance), and is marked in a different color for clarity.

Given a graph of subsystems and a set of footprints for each subsystem as explained above, the proposed technique aims to achieve the following:



(b)

	footprint	Freq.
1	$fp_{1,1}$	195
2	$fp_{1,2}$	190
3	$fp_{1,3}$	186
4	$fp_{1,4}$	183

	footprint	Freq.
1	$fp_{2,1}$	202
2	$fp_{2,2}$	200
3	$fp_{2,3}$	199
4	$fp_{2,4}$	195

	footprint	Freq.
1	$fp_{3,1}$	195
2	$fp_{3,2}$	189
3	$fp_{3,3}$	185
4	$fp_{3,4}$	182

	footprint	Freq.
1	$fp_{4,1}$	199
2	$fp_{4,2}$	198
3	$fp_{4,3}$	196
4	$fp_{4,4}$	195

Fig. 3: a) Footprint variations on FPGA. b) Performance of footprints. c) Selecting a footprint for each subsystem.

- For each subsystem, select a footprint with the highest possible preference level
- Footprints belonging to subsystems with higher inter-subsystem communication should be closer to each other on FPGA
- Selected footprints for each subsystem must not overlap in the final solution
- The final mapping solution must not over-consume routing resources

It is noteworthy that the performance of each footprint is obtained using the Artificial Neural Network (ANN) based module estimation technique proposed in [10]. This technique uses a *device-specific ANN* to estimate the performance for each footprint. The ANN model takes the architectural information such as resource column arrangement and delay into consideration, so that it can accurately estimate the performance without pre-compiling footprints on all possible locations on the target FPGA. The footprint generation process is further discussed in Section VI.

A potential solution for the example in Figure 3 is given in Figure 3(c). Here, footprint versions $fp_{1,4}$, $fp_{2,2}$, $fp_{3,1}$ and $fp_{4,2}$ are chosen for subsystems S_1 to S_4. Selecting footprints with a higher preference level for all subsystems may result in overlapping areas. For instance, footprints $f_{4,2}$ and $fp_{2,2}$ in the solution overlap. Such overlaps are resolved by shifting vertically, due to the uniformity of FPGA resource columns in the vertical direction. However, vertical shifting affects the distance between subsystems, so a careful selection between shifting and choosing non-overlapping footprints must be made. In addition, subsystems with higher inter-subsystem communication (for instance S_3 and S_4) might obtain footprints with lower preference level in order to satisfy the closer placement requirement.

The proposed methodology consists of 2 phases:

1) *Subsystem Grouping:* A pre-processing step finds the associativity of subsystems in a given subsystem graph by analyzing inter-subsystem connections and links. For instance, subsystems S_2, S_3, and S_4 in Figure 1 have relatively high inter-subsystem communication, and are hence considered as a group. The footprint information is not considered in this phase to reduce problem size.
2) *Subsystem Mapping:* The footprints for each subsystem are analyzed to select the best shape and location to map each subsystem to on the target device. The subsystem grouping information obtained in the first phase is used to reduce the search space. For instance, subsystems S_2, S_3, and S_4 in Figure 1 are identified as a group and hence should be placed in closed proximity. This is achieved by promoting footprints $fp_{2,2}$, $fp_{3,1}$ and $fp_{4,2}$ whose horizontal locations, x, are relatively close.

IV. Inter-Subsystem Communication-Aware Subsystem Grouping

A. Problem Formulation

Subsystem grouping identifies subsystems that should ideally be placed in close proximity. The groups are identified using a rapid placement technique on the target device, requiring the following inputs:

(i) *Information about the application* : $G = \{V, E\}$, where vertices, V, and edges, E, represent subsystems $V = \{S_0, S_1, ..., S_N\}$ and connections between them respectively. Each vertex S_i has a node value with the resource requirement of the subsystem $A_{S_i} = \{l_{S_i}, m_{S_i}, d_{S_i}\}$ and each edge $e_{i,j}$ has an edge cost with inter-subsystem communication $\alpha\prime_{i,j}$ between subsystems S_i and S_j.

(ii) *Parameters of the Target FPGA* : The number of columns, W, and number of rows, H, in the target FPGA along with FPGA column arrangement, $F = f_1, f_2, ..., f_W$, where $f_i = \{l|m|d\}$, represents the resource type of column i on the target device (l, m, and d represent CLBs, BRAMs and DSPs respectively).

First the subsystem graph G is rearranged to $G\prime$ such that the subsystems require only one FPGA resource type. For instance, subsystem S_1 in Figure 1 is divided into two subsystems as shown in Figure 4(a) such that new subsystem S_1 requires only CLBs ($l_{S_1} = 96$) and subsystem $S_{1.1}$ requires only BRAMs. Single resource subsystems that belongs to one subsystem in the subsystem graph G, for example S_1 and $S_{1.1}$,

are maintained as one subsystem by adding a very high inter-subsystem communication N to the respective edges between such subsystems.

Given the above inputs, and the rearranged subsystem graph $G\prime$, the subsystem grouping problem is to find a rectangular region $L_v(x_v, y_v, w_v, h_v)$ for each subsystem v in $G\prime$, where (x_v, y_v) are the bottom left coordinates on the FPGA device and (w_v, h_v) are the width and the height of the region. A rectangular floorplanning region also allows the proposed flow to be further extended for use in partially reconfigurable designs with additional architectural constraints (i.e. height of rectangular regions must be multiples of clock regions), wherein the rectangular shape is a requirement for the generation of valid partial bitstreams [25]. Rectangular region L_v should meet the following requirements:

- Resource requirement of the respective subsystem A_{S_i} must be fulfilled
- Regions are non-overlapping and must fit within the FPGA limits W and H

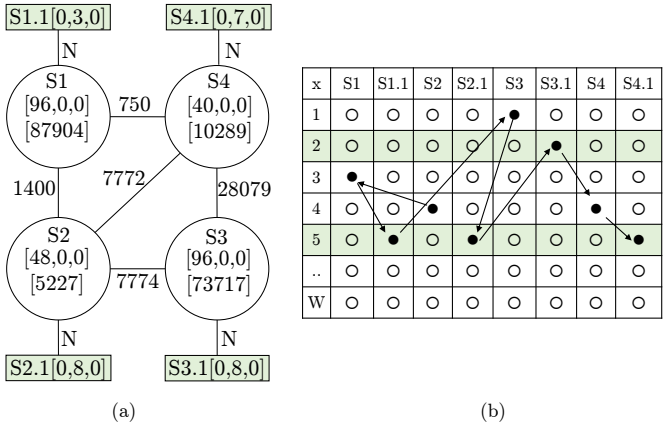


Fig. 4: (a) Divide subsystems based on resource type ($G\prime$). (b) Traversal on point map.

The subsystem grouping problem is modeled as a Traveling Salesman Problem (TSP) as shown in Figure 4. The main objective is to find the horizontal coordinates of rectangular regions, x_v, for each subsystem. Figure 4(b) shows a map for node traversal. Each column in the map has W number of points (rows), and each point (i, j) in the map represents the event of placing the lower left corner of the subsystem that represented by column j at the horizontal coordinate, x, represented by row i. A point is selected from each column during the traversal.The points also represent a resource type. For instance, let the FPGA column arrangement be $F = \{l, m, l, l, m, ..., l\}$. Therefore, the points in row $x = 2$ are BRAM type. Hence, the subsystems that requires only BRAMs (ex. $S_{1.1}$, $S_{2.1}$) are allowed to choose points at $x = 2$ and $x = 5$.

B. Subsystem Grouping using Ant Colony Optimization

The TSP modeled above is solved using an Ant Colony Optimization (ACO) based algorithm. The proposed technique uses a pheromone table similar to Figure 4(b). A cell in the pheromone table, $\tau_{S_i,x}$ represents the likelihood of selecting column x to place subsystem S_i. The pseudo-code of the proposed ACO based placement methodology is given in algorithm 1. In each iteration, a number of solutions are generated by traversing the point map (lines 8–9). The best solution in each iteration Q_{curr_best} can update the pheromone table by increasing the pheromone entries relevant to the solution (lines 14–17). In addition, if Q_{curr_best} is better than the overall best solution Q_{hist_best}, the pheromone table is updated with the solution again (lines 18–21). Pheromone updates increase the likelihood of choosing a better traversal in subsequent iterations. Once the $stopping\_criteria$ is met (maximum number of iterations or very low changes in Q_{hist_best}), the algorithm outputs the best solution, Q_{hist_best}.

Algorithm 1 Communication-aware subsystem grouping.

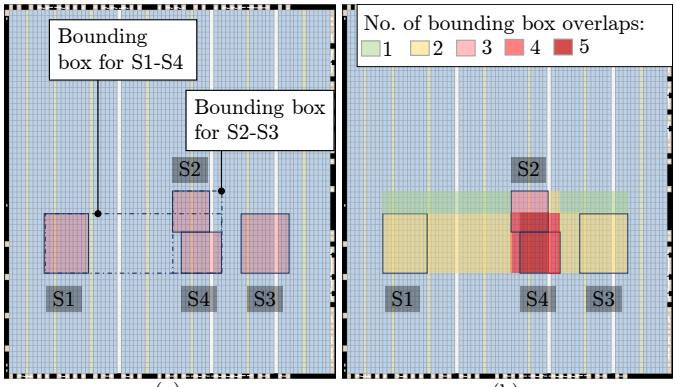
```
1: procedure RUN_ACO1(G,F)
2:     INIT()
3:     Q_hist_best ← Q_init
4:     hist_best ← MAX
5:     while stopping_criteria == FALSE do
6:         Q_curr_best ← Q_init
7:         curr_best ← MAX
8:         for each ant ∈ ANTS do
9:             (seq_ant, x_ant) ← TRAVERSE_ANT(G)
10:            y_ant ← SELECT_Y(x_ant, G)
11:            if IS_VALID(y_ant) == FALSE then
12:                continue
13:            cost_ant ← COST(G, x_ant, y_ant)
14:            if cost_ant < curr_best then
15:                curr_best ← cost_ant
16:                Q_curr_best ← (seq_ant, x_ant)
17:                UPDATE_PHEROMONE(seq_ant, x_ant)
18:            if cost_ant < hist_best then
19:                hist_best ← cost_ant
20:                Q_hist_best ← (seq_ant, x_ant)
21:                UPDATE_PHEROMONE(seq_ant, x_ant)
22:        DECAY_PHEROMONE
23:    return Q_hist_best
```

a) Pheromone Table Initialization: At the beginning, all subsystems have equal probability to select any x point that matches their resource type. Therefore, we initialize all the entries in the pheromone table to a small value in $INIT()$ (line 2). This value is empirically set to 0.2.

b) Traversing: In each iteration, a set of computing agents (artificial ants) traverse through the point map (line 9). Each ant traverses through all the subsystems exactly once by selecting a point from the point map. We use a random number to select the subsystem to traverse. Point selection is done based on the pheromone values $\tau_{S_i,x}$. Each ant produces a solution with a sequence of subsystems (seq_{ant}) and respective x coordinates for each subsystem x_{ant}. Sequence x_{ant} only provides the horizontal coordinate of each subsystem (x_v), and the vertical coordinate (y_v) of all the subsystems is assumed to be 1.

In the subsystem grouping phase, the shape of a subsystem is not prioritized. We use the following technique to infer the rectangular region (w_v, h_v) each subsystem.

- For subsystems with only CLBs (ex. S_3): set height(h) to width(w) ratio close to 1 such that the resource requirement is just satisfied
- Subsystems with other resources (ex. $S_{3.1}$): considered as a single column shape

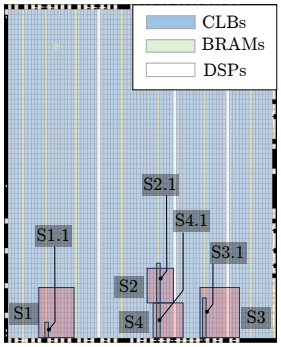


(a) (b)

Fig. 5: Congestion model.

seq_{ant}	x_{ant}	w	h	y
S4	**45**	9	7	1
S3	**59**	12	10	1
S3.1	**60**	1	8	1
S4.1	**46**	1	7	1
S1	**10**	11	10	1
S2.2	**46**	1	8	8
S2	**43**	8	7	8
S1.1	**12**	1	3	1

(a) Solution creation using the best traversed path.



(b) Display the solution on the target FPGA.

Fig. 6: Example of a solution from the subsystem grouping technique.

Defining a rectangular region shape as discussed above may result in overlapping subsystems. In order to resolve these overlaps, subsystems are shifted vertically. For instance, let two subsystems S_a and S_b be placed at $x_a = 10$ and $x_b = 15$, and their rectangular regions (overlapped) are inferred to be $w = 10$ and $h = 5$. If S_b appears in the seq_{ant} before S_a, we shift S_a by 5 (equivalent to h_b), resulting $y_a = 6$ and $y_b = 1$. In this manner, the vertical position is obtained for all subsystems (line 10). The following cost function is used to evaluate solutions:

$$COST(s) = \sum_{e \in E} l_e.\alpha\prime_e \tag{2}$$

where l_e is the half perimeter wire length for the bounding box enclosing the subsystems relevant to edge e and $\alpha\prime_e$ is inter-subsystem communication.

c) Congestion Model: Routability evaluation is an important step in a placement algorithm. Placing a large number of components in a smaller area could lead to excessive usage of routing resources. Typical routing algorithms avoid highly congested areas and therefore may use longer wires to connect components, leading to degraded quality of results (performance and power). In the proposed approach, a bounding box based congestion analysis is incorporated to estimate the routing congestion similar to [26]. In this model, the FPGA area is considered as a 2D array known as a 'congestion map'. All congestion map elements are initialized to zero. For each inter-subsystem connection in Figure 1, a bounding box can be formed as in Figure 5(a) (note that all bounding boxes are not shown for ease of explanation). Next, congestion map elements relevant to each bounding box are incremented. Therefore overlapping bounding boxes reflect high congestion regions. The congestion map relevant to the subsystems in Figure 5(a) is shown in Figure 5(b). The congestion map is populated for each solution. Solutions that exceed a predefined congestion value (referred to as maximum connection density) are discarded to avoid possible routing congestion. Maximum connection density is an architecture-specific approximation for the number of routing resources available per unit area. It is experimentally obtained and defined as follows:

$$max\ connection\ density = \frac{r_{max}}{w \times h} \tag{3}$$

Here, r_{max} is the maximum number of RTL signals that fit into a rectangular FPGA region with w columns wide and h rows tall.

d) Pheromone Update and Decay: Pheromone table entries ($\tau_{S_i,x}$) related Q_{curr_best} and Q_{hist_best} are incremented by 10% in order to promote better solutions in subsequent iterations. It is also noteworthy that ACO mimics the evaporation of natural pheromones such that after each iteration, all the pheromone values in the pheromone table are reduced by a very small amount.

C. Output of Subsystem Grouping

At the end of the ACO algorithm, each subsystem S_i is assigned a coordinate (x_i, y_i), indicating its position on the target FPGA. A standard data clustering technique such as Mean Shift Clustering [27] is used next to identify subsystem groups. For example, seq_{ant} and x_{ant} in Figure 6a shows a traversal solution for the subsystem graph in Figure 1. The traversal seq_{ant} and x_{ant} are used in extracting the shape (w, h) and the vertical coordinate(y) of each subsystem using the methods explained earlier. Figure 6b shows the created solution on the target FPGA. Using the Mean Shift Clustering algorithm, the (x, y) coordinates of each subsystem in the solution are used to identify the subsystem groups i.e. Group $\{S_1\}$ and Group $\{S_4, S_3, S_2\}$ for this sample application. The pre-processing step discussed in this section is utilized in the next section for the final subsystem mapping phase.

V. Subsystem Mapping

The second phase of the proposed methodology finds a specific rectangular FPGA region (shape and location) for each subsystem as described in the following.

A. Problem Formulation

There are four inputs required by this phase:

(i) *Information about the application*: Subsystem graph $G = \{V, E\}$, as previously discussed.

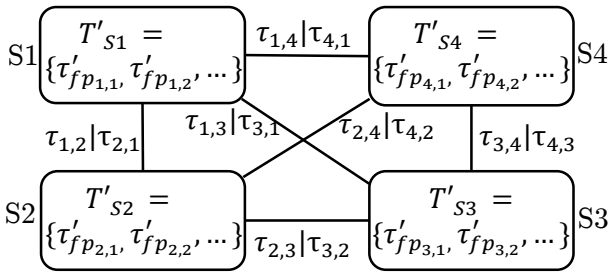


Fig. 7: Complete graph G_c respective to graph G.

(ii) *Subsystem groups*: $GP = \{gp_0, gp_1, ..., gp_m\}$ (identified in Section IV), where each group gp_i is subset of nodes $S_i \in G$

(iii) *Set of footprint lists*: $L = \{l_{S_0}, l_{S_1}, ..., l_{S_N}\}$, where each list l_{S_i} contains n variations of footprints $l_{S_i} = \{fp_{i,0}, fp_{i,1}, ..., fp_{i,n}\}$ of subsystem S_i. A footprint (rectangular region on FPGA) can be represented as a tuple $fp_{S_{i,j}} = \{x_{i,j}, w_{i,l}, h_{i,j}\}$, where x is the bottom left horizontal coordinate of the footprint, w and h are width and height of the footprint.

(iv) *Set of lists with estimated performance of each footprint*: $\hat{F} = \{\hat{f}_{S_0}, \hat{f}_{S_1}, ..., \hat{f}_{S_N}\}$, where each list $\hat{f}_{S_i} = \{f_{i,0}, f_{i,1}, ..., f_{i,n}\}$ consists estimated performance of respective footprints of subsystem S_i, where $f_{i,j}$ represents the estimated performance of the footprint $fp_{i,j}$.The estimated performance of the footprint can be considered as the preference level of the footprint.
Each footprint list l_{S_i} is sorted in descending order based on respective estimated performance $f_{i,j}$. The footprint lists are generated using a target FPGA device characterization model, as proposed in [10].

Given the above inputs, the subsystem mapping problem is to select a footprint $fp_{i,j}$ from each footprint list l_{S_i}. The footprint selection should minimize a cost function while resolving footprint overlaps. Similar to the subsystem grouping in Section IV, subsystem mapping is modeled as a TSP as follows. The subsystem graph G is transformed into a *complete graph* $G_c = \{V\prime, E\prime\}$ as shown in Figure 7. Each node in $V\prime$ is the same as V in G and has a list of footprints that are associated with the respective subsystem. An edge $e\prime_{i,j}$ in $E\prime$ represents an event of choosing subsystem S_j after S_i. Therefore, there is an edge from each node to all the others nodes in the complete graph. An edge $e\prime_{i,j}$ (between nodes S_i and S_j) is associated with a probabilistic value, where a higher value indicates higher likelihood of choosing S_j after S_i. In addition, footprints in all the nodes carry a probabilistic value, where a higher value represents a higher preference of selecting the footprint $fp_{i,j}$ from the respective footprint list l_{S_i}. The aim of the TSP is the following;

- find a route to visit each node in G_c
- select one footprint from each node to form a set of footprints with minimum cost

The order in which the footprints are mapped onto the FPGA is determined by the sequence of node traversal.

B. Subsystem Mapping using Ant Colony Optimization

The TSP model discussed above is solved using Ant Colony Optimization (ACO). However, the TSP variant differs from typical TSP because of the values associated with the nodes in addition to the edges. Therefore, a modified version of ACO is applied to the subsystem mapping problem. In the proposed ACO technique, two types of pheromones (probabilistic values) have been used to represent the edge and node costs.

Algorithm 2 describes the proposed subsystem mapping algorithm. $RUN\_ACO()$ is similar to $RUN\_ACO1()$ in Algorithm 1. In each iteration, a new population of artificial ants produce a number of solutions by traversing G_c and selecting a footprint for each node in $V\prime$. The probabilistic values in the nodes and the edges are represented as pheromones. The best solution in the current ant population, Q_{curr_best}, and the overall best solution, Q_{hist_best}, increase the respective pheromones on the corresponding nodes and edges along the solution path. Once the termination condition is satisfied, Q_{hist_best} is returned as the solution to the footprint placement problem (line 24).

a) Pheromone Initialization: To promote solutions with lower cost, pheromones are introduced to the edges and nodes as shown in Figure 7. An edge in G_c, $e\prime_{i,j}$ (between nodes S_i and S_j), carries a pheromone value $\tau_{i,j}$ which shows the likelihood of choosing S_j after S_i. In addition to the pheromone representation in a typical ACO, a second degree pheromone list for each node in $G\prime$, which is represented as $T\prime_{S_i} = \{\tau\prime_{fp_{i,1}}, \tau\prime_{fp_{i,2}}, ..., \tau\prime_{fp_{i,n}}\}$, is included. The second degree pheromone, $\tau\prime_{fp_{i,j}}$, represents the likelihood of choosing the respective footprint, $fp_{i,j}$, from the list l_{S_i} for the subsystem S_i.

In $INIT()$ (line 2), the edge pheromones $\tau_{i,j}$ are initialized to have the same value, whereas, the second degree pheromone values $\tau\prime_{i,j}$ are interpolated based on the performance of the respective footprint. A footprint variation with higher performance (close to preference level 1) is promoted to be selected. The following interpolation relation is used for initializing second order pheromones.

$$\tau\prime_{i,j} = 0.2 + 0.6 \times \frac{f_{i,j} - f_{i,n}}{f_{i,1} - f_{i,n}} \tag{4}$$

It should be noted that l_{S_i} is an ordered list where $fp_{i,1}$ and $fp_{i,n}$ are the footprint with highest and the lowest performance, respectively.

b) Traversing: In procedure $TRAVERSE\_ANT()$, each ant in the current population (iteration) traverses through each node of G_c exactly once and the order of the nodes is referred to as a sequence. The first node of the sequence is randomly selected from G_c (lines 27–28). Grouping information has been used in selection the next node in the sequence. A temporary complete graph, $G_c\prime$ is updated at every node selection in $MODIFY\_PHEROMONES()$ in order to promote node selection from the same subsystem group (line 32). It is achieved by increasing the edge pheromone values of the edges from the last selected node to the nodes in the same group by 50%. For instance selecting S_3 in Fig. 1 increases edge pheromones $\tau_{3,2}$, $\tau_{3,4}$ in $G_c\prime$ as subsystems S_2 and S_4 are in

Algorithm 2 Ant Colony Optimization based algorithm for subsystem mapping.

```
1: procedure RUN_ACO( G, G_c, L, GP )
2:     INIT()
3:     Q_hist_best ← Q_init
4:     hist_best ← MAX
5:     while stopping_criteria == FALSE do
6:         Q_curr_best ← Q_init
7:         curr_best ← MAX
8:         for each ant ∈ ANTS do
9:             seq_ant ← TRAVERSE_ANT(G′, GP)
10:            fps_ant ← SELECT_FOOTPRINTS(G′, L, GP, seq_ant)
11:            fps_ant ← RESOLVE_OVERLAPS(fps_ant)
12:            if fps_ant.isValid() == FALSE then
13:                continue
14:            cost_ant ← COST(G, fps_ant)
15:            if cost_ant < curr_best then
16:                curr_best ← cost_ant
17:                Q_curr_best ← fps_ant
18:                UPDATE_PHEROMONE(G_c, seq_ant, fps_ant)
19:            if cost_ant < hist_best then
20:                hist_best ← cost_ant
21:                Q_hist_best ← fps_ant
22:                UPDATE_PHEROMONE(G_c, seq_ant, fps_ant)
23:        DECAY_PHEROMONE(G_c)
24:    return Q_hist_best
25:
26: procedure TRAVERSE_ANT(G_c, GP)
27:    rand ← RAND(1, N)
28:    seq.append(S_rand)
29:    G_c′ ← G_c
30:    for i = 1 to N-1 do
31:        gp ← GET_GROUP(seq[i], GP)
32:        MODIFY_PHEROMONES(G_c′, gp)
33:        sum_τ ← Σ τ_seq[i],j ∀j∈G_c′,j∉seq
34:        rand ← RAND(0, sum_τ)
35:        for all j ∈ G_c′, j ∉ seq do
36:            cummu_τ ← cummu_τ + τ_seq[i],j
37:            if cummu_τ ≥ rand then
38:                seq.append(S_rand)
39:                break
40:    return seq
41:
42: procedure SELECT_FOOTPRINTS(G_c, L, GP, seq_ant)
43:    L′ ← L
44:    for all s ∈ seq_ant do
45:        gp ← GET_GROUP(s, GP)
46:        sum_τ′ ← Σ τ′_s,j ,1≤j≤n, j∈l′_s
47:        rand ← RAND(0, sum_τ′)
48:        for j = 1 to n do
49:            cummu_τ′ ← cummu_τ′ + τ′_s,j j∈l′_s
50:            if cummu_τ′ ≥ rand then
51:                fps_ant.append(fp_s,j)
52:                MODIFY_SECOND_PHEROMONE(L′, gp, fp_s,j)
53:                break
54:    return fps_ant
55:
56: procedure RESOLVE_OVERLAPS(fps_ant)
57:    for all fp_j ∈ fps_ant do
58:        for all fp_i ∈ fps_ant | i < j do
59:            if fp_i.overlaps(fp_j) == TRUE then
60:                y_j ← y_j + get_vertical_shift()
61:                i = 1
62:    return fps_ant
63:
64: procedure UPDATE_PHEROMONE(G_c, seq_ant, fps_ant)
65:    for i = 1 to N-1 do
66:        τ_i,i+1 ← τ_i,i+1 + ε_τ
67:    for all fp ∈ fps_ant do
68:        τ′_fp ← τ′_fp + ε_τ′_fp
```

the same group as S_3. Next, a cumulative random distribution is constructed using the edge pheromone values of the edges from the last node in seq to nodes which are not in seq. A random number ($rand$) between 0 and the maximum of the cumulative random distribution (sum_r) is generated, and then an edge is determined using the random number as a cutoff point in the distribution (lines 33-38). This process is continued until seq is build with all the nodes in G_c.

After traversing all the nodes in the graph G_c, a footprint from each footprint list l_{S_i} in seq is selected in $SELECT\_FOOTPRINTS()$. Footprint selection is based on the cumulative spectrum of the secondary pheromone $\tau\prime_{i,j}$ (lines 46-51). We use a footprint promoting technique to promote nearly footprints for subsystems in the same subsystem group. For instance, if footprint $fp_{3,10}$ is chosen for subsystem S_3, the second degree pheromone levels in footprint lists for S_2 and S_4 are updated such that, pheromones associated with the footprints that are closer to $fp_{3,10}$ are increased by 50%. This selection is repeated until a footprint from every node in seq is chosen and appended to the footprint list of the current ant fps_{ant}. The footprint sequence fps generated by the above process may include overlapping footprints. Therefore, we use a simple technique in $RESOLVE\_OVERLAPS()$ to resolve such overlaps based on the priority level of footprints. The same cost function used in the subsystem grouping phase, equation 2, is used in this phase as well. It is noteworthy that vertical shifting of the footprints in $RESOLVE\_OVERLAPS()$ may shift footprints beyond the device boundaries. Therefore, such invalid mappings are discarded (lines 12-13).

c) Pheromone Update and Decay: In the $UPDATE\_PHEROMONES()$ process, the pheromone values $\tau_{i,j}$ of the solution path edges of seq_{ant} and respective secondary pheromones of the footprint entries in fps_{ant} are incremented by ϵ_τ and $\epsilon_{\tau\prime}$ respectively. The increment values are given in equations 5 and 6. The first value is equal to the half of the average edge pheromone values in G_c and the latter is equal to half of the average secondary pheromone values in the pheromone list respective to the footprint.

$$\epsilon_\tau = \frac{1}{2} \times \sum_{e\prime \in E\prime} \tau_{e\prime}/(N \times (N-1)) \tag{5}$$

$$\epsilon_{\tau\prime_{fp}} = \frac{1}{2} \times \sum_{\forall fp_{i,j} \in T\prime_{S_i}} \tau\prime_{i,j}/N \ \textit{where} \ fp \in l_{S_i} \tag{6}$$

After all the ants in the current population finish their traversal, all the edge pheromone values are reduced by a small constant to mimic the evaporation of pheromones (line 23). In the next iteration, a new ant population is created and all the aforementioned steps are repeated. This process is continued until the stopping criteria, i.e. maximum number of iterations, is satisfied.

C. Footprint Locking for Rapid Convergence

As the application size grows, the number of subsystems also increases, causing the ACO algorithm to consider a large number of footprint list entries. Typically, ACO requires a long

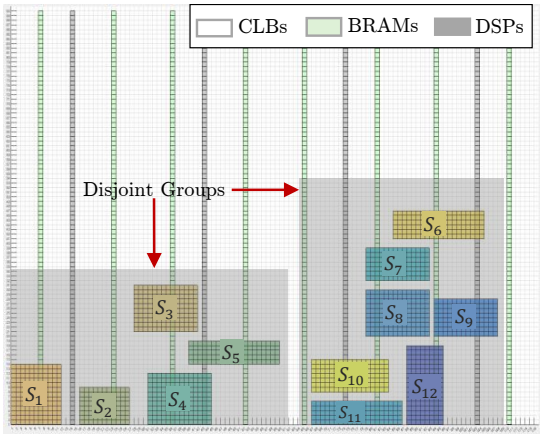


Fig. 8: Suboptimal mapping solution example.

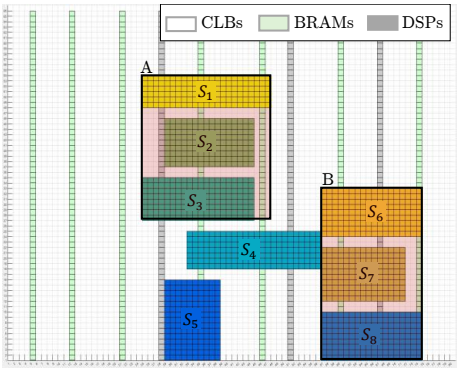


Fig. 9: Subsystem locking example.

runtime to converge to an optimal solution and the probability of converging to a non-optimal solution increases with the problem size [28]. Figure 8 shows an example of converging to a sub-optimal mapping solution with a floorplan with mapped subsystem footprints. The footprints of subsystems S_6 to S_{12} are placed tightly as a group while the rest of the footprints are placed away from this group. This limits the opportunity for global optimization to reduce overall communication for the entire application. Therefore, in this section we discuss techniques to improve the quality of the solution provided by the ACO based subsystem mapping methodology.

After the first run of the ACO algorithm, we observe the footprint mapping solution on the target FPGA and identify the footprints which are placed in close proximity. Such footprints are locked in order to preserve close placement, and the subsystem mapping technique (phase 2) is re-run. These steps are repeated until there are no subsystems to be locked.

Candidate subsystems, which are suitable for footprint locking, are identified using the following criteria. First, the subsystems that are placed at a close proximity are identified. A rectangular bounding box (BB) is then defined around such footprints as shown in Fig. 9. However, the area covered by a bounding box should not be much larger than the total area covered by the individual footprints. In other words, locked footprint area should not include large empty areas. We use the following relationship to evaluate candidate subsystems:

$$\sum_{fp_i \in BB} h_i \times w_i \geq (\delta_{lock} \times Area\ of\ the\ BB) \qquad (7)$$

Here the constant $\delta_{lock}(< 1)$ is found empirically and is close to 1. Identification of subsystems for locking is shown in $LOCK()$ in Algorithm 3 (line 4), along with the overall flow of the mapping algorithm. We start with subsystem grouping (line 1) in order to obtain grouping information. Then, the ACO is run iteratively, as long as subsystem locking produces locked footprints, L_{new} (Lines 3 through 9). Once the candidate subsystems for locking are identified, we update the respective subsystems in the subsystem graph G such that the respective subsystems for locked footprints are merged, and the edges in G are updated with new connections from/to locked subsystems and with updated inter-subsystem communication. The updated subsystem graph is stored as G_{new} and accordingly a new complete graph G_{cnew} is also created.

Algorithm 3 Overall execution of mapping algorithm (with subsystem locking).

```
1: Q_initial ← GROUPING(G)
2: while true do
3:     Q ← RUN_ACO(G, G_c, L, Q_initial)
4:     (G_new, G_cnew, L_new) ← LOCK(Q)
5:     if L_new == L then
6:         break
7:     G ← G_new
8:     G_c ← G_cnew
9:     L ← L_new
10: return Q
```

VI. Implementation

A. Integrating Subsystem Mapping into the CAD Flow

In this section, we discuss the integration of the proposed subsystem methodology for large FPGA designs into a commercial CAD flow. We have chosen Altera Quartus and the Altera Incremental Compilation flow [29]. However, we have also verified that it is implementable using the Xilinx flow [30] too. We have also integrated the subsystem generation technique [9] and subsystem performance estimator [10] in order to have a seamless flow. The integrated framework is shown in Fig. 10 and consists of 5 major steps.

a) Extract Parameters: The input to the framework is a Register Transfer Level (RTL) representation of a design. The parameters such as area of design entities and communication frequencies between entities are extracted by performing RTL synthesis followed by RTL Simulation. RTL code is processed to obtain a graph representation of entity connections.

b) Subsystem Generation: RTL entities are partitioned to form subsystems based on the communication between RTL entities as described in [9]. The size of subsystems is limited to have profitable subsystems which accumulate entities to maximize intra-subsystem communication while inter-subsystem communication is minimized. However, exploring different subsystem sizes is out of the scope of this work. The output of the subsystem generation process is a list of subsystems (G) and the enclosed RTL entities of each subsystem.

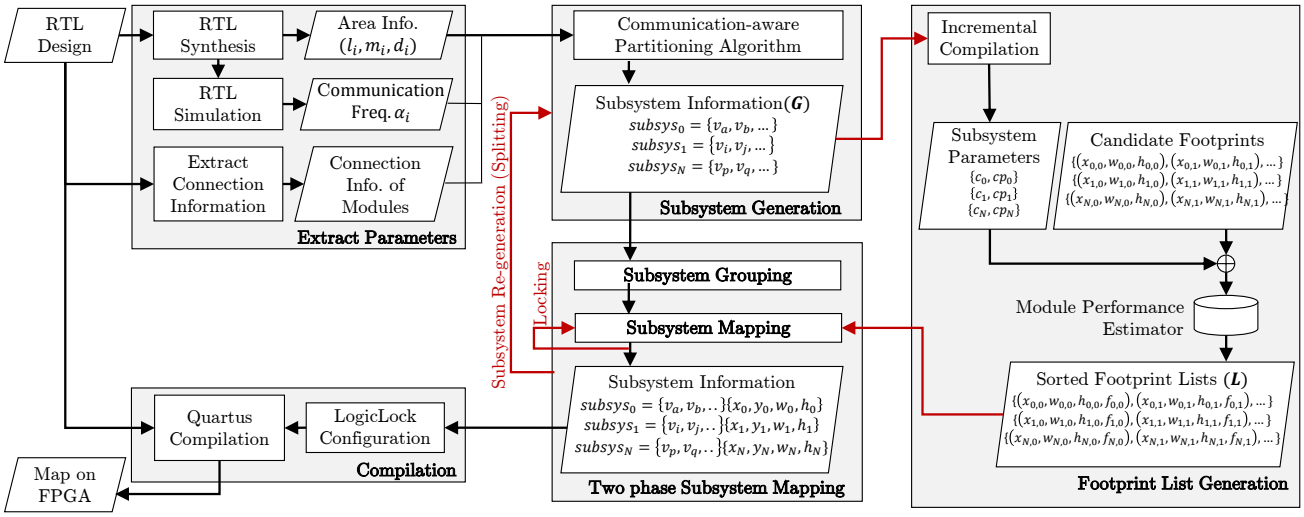


Fig. 10: Implementation of proposed subsystem mapping technique.

c) Footprint List Generation: The subsystems are then isolated and separately compiled in order to obtain subsystem specific parameters (required by the estimator in [10]); number of connections (c_i) and default critical path (cp_i) for each subsystem S_i. These parameters are constant for all the footprints of a given subsystem.

Next, the candidate footprints are inferred using the resource requirement (l_i, m_i, d_i) of each subsystem. A candidate footprint has 3 parameters; bottom left horizontal position of the rectangular shape on target FPGA (x), width (w) and height (h) of the rectangular region. For an arbitrary location x_i on the target FPGA, we can derive a list of rectangular shapes on the target FPGA which satisfies the are requirement (l_i, m_i, d_i). We can extend the same idea at many x_i to obtain candidate footprints for all subsystems S_i as $\{(x_{i,0}, w_{i,0}, h_{i,0}), (x_{i,1}, w_{i,1}, h_{i,1})...\}$. To avoid generating a large number of candidate footprints, we select x_i with intervals instead of considering all possible x_i, especially for larger FPGAs. Typically the intervals in horizontal coordinate x_i are small (i.e. less than 3) and are applicable only if there are no large differences in estimated performance.

Extracted subsystem parameters and corresponding candidate footprint lists are provided to the module performance estimation engine proposed in [10]. The module performance estimator outputs the estimated performance (F_{max}) for each candidate footprint. Footprint list (L) for each subsystem is then obtained by annotating estimated performance (f) to respective candidate footprint parameters. Footprint lists are also sorted in descending order of estimated performance. It is noteworthy that use of module performance estimator obsoletes the large module database in typical MDM flows, which consists of modules pre-compiled at many different locations.

d) Two Phase Subsystem Mapping: The proposed methodology then takes subsystem information (G) from the subsystem generation step and corresponding footprint lists (L) from the previous step, to select the best set of footprints. The integration illustrated in Fig. 10 also shows a mechanism for subsystem re-generation after the two phase mapping process is done, and is discussed in the next sub-section.

e) Compilation: The output of the previous step is the subsystem lists and their footprint on FPGA. These parameters are then fed into the LogicLock feature in the Quartus tool. LogicLocks implement the actual subsystems and footprints on the target FPGA. After that, the Quartus compiler is invoked to generate the bitstream.

B. Splitting Subsystem by Subsystem Regeneration

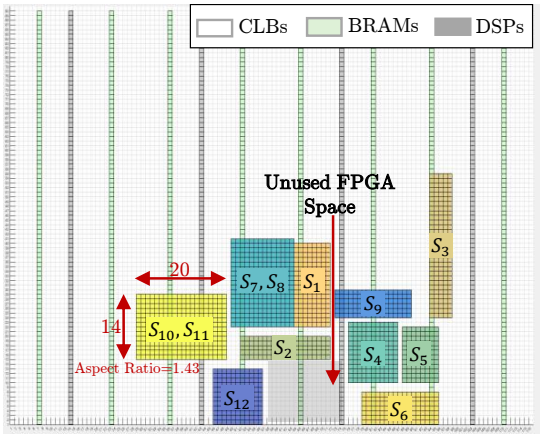


Fig. 11: Unused FPGA resources during subsystem mapping.

Despite the effectiveness of the proposed subsystem mapping technique, we have observed that some of the footprints obstruct better mappings of subsystems, leading to un-utilized space between subsystems. For instance, consider the solution in Fig. 11 which is obtained by the proposed technique for the test application *ataxl*. The footprint S_2 in the mapping solution is wider than the empty space below the footprint, leaving the FPGA area below the subsystem unused. It is important to note that the proposed two phase subsystem mapping approach is a deterministic approach that converges to the best solution for given subsystems. Hence, to overcome

this issue, mapping subsystems to different footprints (with different size and placement) does not guarantee a performance and power optimized mapping.

A subsystem splitting technique that changes the composition of the subsystems is followed in order to overcome such footprints. As shown in Fig. 11, footprint obstruction typically happens due to odd-shaped footprints. We identify these footprints by considering their aspect ratios. First we calculate the average aspect ratio (AR) of all the footprints as follows.

$$AR_{S_i} = \frac{max(w_{S_i}, h_{S_i})}{min(w_{S_i}, h_{S_i})} \tag{8}$$

where w_{S_i} and h_{S_i} represent the width and the height of the footprint S_i respectively. If the average and standard deviation of aspect ratios of the footprints are found $\overline{AR}$ and σ_{AR} respectively, the odd footprints are then identified as follows:

$$oddfootprint = \begin{cases} 1, & \text{if } AR_{S_i} > \overline{AR} + \sigma_{AR} \\ 0, & \text{otherwise} \end{cases}$$

Next we calculate the whitespace area below each odd footprint. The odd footprint with largest whitespace area below it is identified as the an eligible footprint for the splitting process. For instance, footprint S_2 in Fig. 11 is identified as an eligible footprint. Eligible subsystems are first re-iterated through the subsystem generation step with a tighter area constraint in order to re-generate smaller subsystems. The candidate footprint list for the new re-generated subsystems are then obtained using the module performance estimator as discussed in Section VI. The new footprint list and the subsystem information is then provided to the subsystem generation and mapping (two phase subsystem mapping approach) techniques in order to select the best set of footprints for the regenerated subsystems. It should be noted that the footprints of non-eligible subsystems are not changed.

VII. Results and Discussion

In this section, we evaluate the performance of the design methodology we have proposed. The evaluation is done using Intel FPGAs and the Intel Quartus Prime 17.1 CAD flow. The methodology can be adapted to work with other vendor tools. The effectiveness of the methodology is assessed for two parameters: (i) routing power dissipation and (ii) achievable frequency of the final compiled design on the FPGA. Power values are obtained using Quartus PowerPlay power estimation, using signal interaction data from a gate level simulation for accuracy. The achievable frequency for each design is extracted from the Quartus timing report after a successful placement and routing.

We used 8 benchmark applications for the evaluation step. The test applications are created to demonstrate the properties of large applications, using existing benchmark suites such as Polybench [31]. We have also tested on three different FPGAs from two families: the EP4CGX50 and EP4CGX110 from the Intel Cyclone 4 family, and the EP4SGX70 from the Intel Stratix 4 family. The resource requirements of the benchmark applications are show in Table I.

TABLE I: Resource Requirement of Benchmark Applications

Application	Logic Cells	BRAMs	DSPs
Synth1	22,438	80	80
Synth2	22,427	80	80
Synth3	23,566	84	84
atax'	64,169	160	160
bicg'	61,833	200	200
gesummv'	50,509	160	160
cholesky'	75,004	120	120
symm'	68,504	120	120

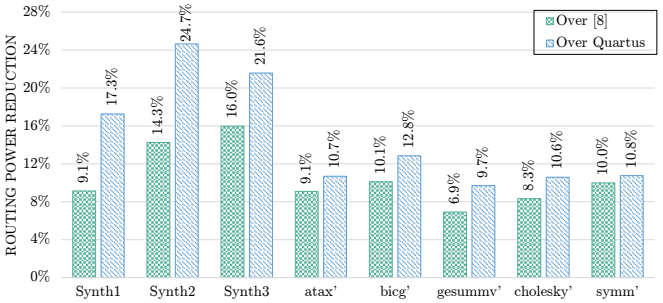


Fig. 12: Routing power reduction over [9] and Quartus.

A. Evaluation of Subsystem Grouping and Mapping Technique

First, the routing power dissipation of benchmark applications, as achieved by following proposed methodology, is compared with routing power obtained by the following techniques:

1) Quartus: The first comparison is made against the commercially available Quartus tool version 17.1. Here, we compile the design using the Quartus CAD flow that does not consider any communication information of the design.
2) A subsystem generation technique in [9]: In this technique, subsystems are created using a communication-aware design partitioning technique to minimize power dissipation through minimization of intra-subsystem communication. This methodology does not produce placement constraints, but rather, placement is decided by Quartus flow.

Fig. 12 shows the percentage power reduction using the proposed technique. The proposed methodology outperforms Quartus by a significant margin, reducing routing power on average by over 14%, ranging from 9.7% to 24.7%. This has been achieved by effectively ensuring shorter interconnect wires for high communication links as well as considering the heterogeneity of the FPGA. Additionally, an average improvement of about 11.0% (ranging from 6.9% to 16.0%) is also achieved over the communication-aware subsystem generation technique in [9]. The subsystem generation technique, *only* considers the intra-subsystem communication. However, depending on the application, there can still be links with high inter-subsystem communication. Hence, inter-subsystem communication-aware subsystem placement, as proposed in this paper, is critical to achieve further power savings.

TABLE II: Runtime Analysis of Subsystem Mapping

Application	Runtime of the Technique (sec)		Reduction (%)
	w/o Grouping	with Grouping	
Synth1	1007	254	74.8
Synth2	969	235	75.7
Synth3	658	193	70.7
atax'	491	170	65.4
bicg'	669	204	69.5
gesummv'	795	212	73.3
cholesky'	693	218	68.5
symm'	824	221	73.2

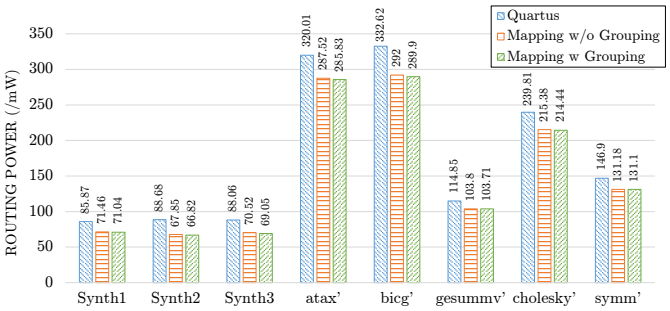


Fig. 13: Effect of grouping technique on routing power.

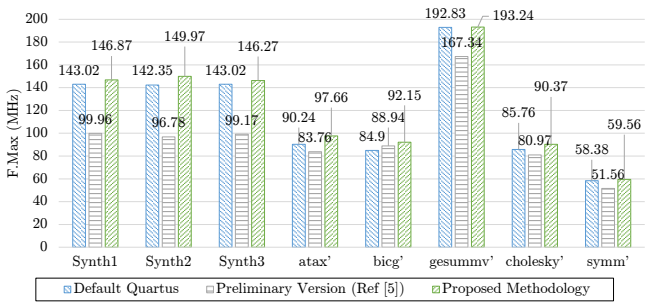


Fig. 14: Effect of grouping technique on F_{max}.

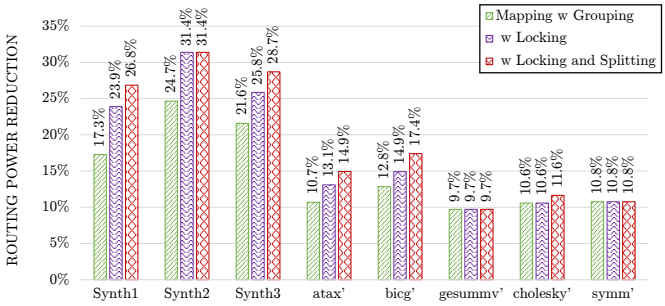


Fig. 15: Routing power reduction due to footprint locking and subsystem splitting.

It is important to note that the proposed methodology does not change application characteristics or the compilation flow. Hence, the FPGA resource consumption as well as static power consumption remain unchanged. It has been observed that the routing power is a dominant component in total power dissipation for most applications. For instance, routing power dissipation of all but one benchmark applications tested here, ranges from 33% to 45% of total power dissipation. Only the *gesummv'* application has a relatively low routing power dissipation of about 15% of its total power.

Secondly, the effectiveness of using the subsystem grouping technique discussed in Section IV during pre-processing is discussed. We compare the runtime of subsystem mapping methodology with and without incorporating the subsystem grouping. In Table II, the runtime of the proposed methodology with and without integrated subsystem grouping technique is shown. The average runtime of the two phase methodology (with grouping) is considerably decreased by over 70% when the solutions obtained from subsystem grouping technique is used to reduce the subsystem mapping problem size. As shown in Fig. 13, integrating grouping information in subsystem mapping has not changed the effectiveness of the proposed footprint mapping in reducing routing power, despite the considerably lower runtime.

Fig. 14 shows the performance of benchmark applications achieved using default Quartus compilation, our preliminary subsystem mapping technique in [11] and using our subsystem mapping methodology with proposed modifications. It is evident that our methodology marginally improves frequency while significantly reducing power dissipation. Test applications we have chosen for evaluation demonstrate the qualities of large application, so the default Quartus compilation produces solutions with degraded power. On the other hand, typical MDM techniques we discussed in Section II improve the design productivity but are inferior in terms of performance compared to non-MDM flows. The reason for low performance is because creating subsystems removes global information and optimization across subsystems cannot performed. However, the proposed methodology uses inter-subsystem connection details to preserve overall performance, while optimizing power dissipation.

B. Evaluation of Footprint Locking and Subsystem Splitting.

Fig. 15 shows the percentage routing power reduction over Quartus using (i) the subsystem mapping technique with grouping, (ii) the subsystem mapping technique with footprint locking, and (iii) the complete proposed methodology with

TABLE III: Compilation Time of Subsystem Mapping

Application	Quartus (/hours)	Proposed Technique		
		Processing (/min)	Compilation (/hours)	Total (/hours)
Synth1	1.43	8.0	1.60	1.73
Synth2	1.20	7.7	1.80	1.93
Synth3	1.20	7.0	1.75	1.87
atax'	2.23	11.0	2.30	2.48
bicg'	2.58	11.1	2.88	3.07
gesummv'	2.18	11.3	2.40	2.59
cholesky'	2.42	11.5	2.75	2.94
symm'	2.38	11.5	2.62	2.81

subsystem splitting. Introducing the footprint locking technique to the subsystem mapping methodology has resulted in further improvement of the average routing power reduction over Quartus from 14.8% to 17.5%. It should be noted that some applications may not produce footprints that are eligible for subsystem locking. For instance, subsystem locking is not possible with the applications *gesummv'*, *cholesky'* and *symm'*. Splitting further improves the average routing power reduction to 18.9%. However, the splitting technique has not found eligible subsystems for subsystem regeneration for applications *Synth2*, *gesummv'* and *symm'*. Therefore, the power dissipation for these applications remains the same. The maximum reduction reported is 31.4% compared to the Quartus flow.

Finally, a comparison of Quartus compilation time (without subsystems) and the total runtime of the proposed subsystem mapping technique is shown in Table III. There are two contributing factors to the runtime of proposed technique: (i) Processing time overhead due to footprint list generation and running subsystem mapping algorithm – (shown in column 3) and (ii) Compilation with subsystem information using standard CAD tools – (shown in column 4). As is evident from Table III, total runtime of the proposed technique is only marginally higher than the compilation time of Quartus without subsystems.

VIII. Conclusion

In this paper we have presented a methodology for placing modules in large FPGA designs. It uses a modified version of Ant Colony Optimization to place module footprints minimizing long distance communication. Footprint placement is done by selecting the most appropriate set of footprint versions for each module from different variations of footprints in order to utilize the heterogeneous resource column arrangement in modern FPGAs. Selecting better footprints ensures the resource column arrangement within the footprint is efficient for better allocation of intra-module communication. Footprint location minimizes the distance between modules with high inter-module communication. We have introduced a footprint merging process and footprint splitting by re-iterating module creation (design partitioning). Further improvements to the methodology reduced routing power by 6.5% on average. As compared to standard Intel Quartus compilation, the new methodology shows average routing power reduction of nearly 19% without a degradation in achievable frequency.

References

[1] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, 2015.

[2] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[3] Microsoft. (2020) Project Catapult. https://www.microsoft.com/en-us/research/project/project-catapult/.

[4] Avnet Inc. (2020) Xilinx Automotive Solutions. https://www.avnet.com/wps/portal/apac/products/automotive-solutions/xilinx/.

[5] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Systems Letters*, vol. 5, no. 1, pp. 12–15, 2013.

[6] G. Zhong, S. Niar, A. Prakash, and T. Mitra, "Design of Multiple-target Tracking System on Heterogeneous System-on-chip Devices," *IEEE Transactions on Vehicular Technology*, 2016.

[7] J. Coole and G. Stitt, "BPR: Fast FPGA Placement and Routing Using Macroblocks," in *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2012, pp. 275–284.

[8] T. Tuan, S. Kao, A. Rahman, S. Das, and S. Trimberger, "A 90nm Low-Power FPGA for Battery-Powered Applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2006, pp. 3–11.

[9] K. Herath, A. Prakash, J. Guiyuan, and T. Srikanthan, "Communication-aware Partitioning for Energy Optimization of Large FPGA Designs," in *Proceedings of the on Great Lakes Symposium on VLSI*, 2017, pp. 407–410.

[10] K. Herath, A. Prakash, and T. Srikanthan, "Performance Estimation of FPGA Modules for Modular Design Methodology using Artificial Neural Network," in *Proceedings of the International Symposium on Applied Reconfigurable Computing*, 2018, pp. 105–118.

[11] K. Herath, A. Prakash, J. Guiyuan, and T. Srikanthan, "Ant Colony Optimization based Module Footprint Selection and Placement for Lowering Power in Large FPGA Designs," in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2018.

[12] R. Tessier, "Fast Placement Approaches for FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 7, no. 2, pp. 284–305, 2002.

[13] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "HMFlow: Accelerating FPGA compilation with hard macros for rapid prototyping," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines*, 2011, pp. 117–124.

[14] T. Frangieh and P. Athanas, "A Design Assembly Framework for FPGA Back-end Acceleration," vol. 38, no. 8. Elsevier, 2014, pp. 889–898.

[15] K. Vipin and S. A. Fahmy, "Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration," in *Proceedings of the International Symposium on Applied Reconfigurable Computing*, 2012, pp. 13–25.

[16] F. Gharibian, L. Shannon, and P. Jamieson, "Identifying and Placing Heterogeneously-sized Cluster Groupings based on FPGA Placement Data," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–6.

[17] M. Rabozzi, J. Lillis, and M. D. Santambrogio, "Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 186–193.

[18] K. Herath, A. Prakash, U. Kanewala, and T. Srikanthan, "Communication-aware Module Placement for Power Reduction in Large FPGA Designs," in *Proeedings of the IEEE Computer Society Annual Symposium on VLSI*, 2018, pp. 209–214.

[19] Y. Xiao, D. Park, A. Butt, H. Giesen, Z. Han, R. Ding, N. Magnezi, R. Rubin, and A. DeHon, "Reducing FPGA Compile Time with Separate Compilation for FPGA Building Blocks," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 153–161.

[20] N. Deak, O. Cret, and H. Hedesiu, "Efficient FPGA Floorplanning for Partial Reconfiguration-Based Applications," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 309–309.

[21] K. Lee and P. Athanas, "Shape exploration for modules in rapid assembly workflows," in *Prodeeings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2015, pp. 1–7.

[22] F. Mao, W. Zhang, B. He, and S.-K. Lam, "Dynamic module partitioning for library based placement on heterogeneous FPGAs," in *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[23] N. Mehta. (2011) Xilinx 7 Series FPGAs: User Guide Lite. https://www.eetimes.com/xilinx-7-series-fpgas-user-guide-lite/.

[24] I. Corporation. Cyclone® V FPGAs. https://intel.ly/3fs2kKI.

[25] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: a survey of architectures, methods, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–39, 2018.

[26] Y. Zhuo, H. Li, and S. P. Mohanty, "A Congestion Driven Placement Algorithm for FPGA Synthesis," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2006.

[27] Y. Cheng, "Mean Shift, Mode Seeking, and Clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, 1995.
[28] H. H. Mukhairez and A. Y. Maghari, "Performance comparison of simulated annealing, GA and ACO applied to TSP," *International Journal of Intelligent Computing Research*, vol. 6, no. 4, 2015.
[29] "Increasing Productivity with Quartus II Incremental Compilation," https://goo.gl/uy225f.
[30] "Vivado Design Suite User Guide-Hierarchical Design," https://goo.gl/6bUqqD.
[31] L.-N. Pouchet, "Polybench: The Polyhedral Benchmark Suite," http://web.cs.ucla.edu/˜pouchet/software/polybench/.

Kalindu Herath received his B.S (Honours) degree in Computer Engineering from the University of Peradeniya, Sri Lanka, in 2011 and his PhD degree from Nanyang Technological University (NTU), Singapore, in 2020. Currently he works as a Research Fellow at Cyber Security Research Centre @ NTU. His research interests include developing techniques for mapping large applications on to modern heterogeneous FPGAs and identifying anomalies and faults in systems using hardware assisted techniques.

Alok Prakash received his PhD degree from Nanyang Technological University (NTU), Singapore, in 2014. He is currently a senior research fellow with the School of Computer Science and Engineering, NTU, where he leads a team of researchers and PhD students in developing low cost camera-based traffic law enforcement sensors under the TUMCreate project. His research interests include intelligent transportation systems, especially focused on solving the first mile last mile issues as well as low-cost and low-power embedded systems design with particular emphasis in mapping complex computer vision applications on modern heterogeneous mobile SoCs. His papers have been nominated for best paper award in DAC 2016 and HEART 2018. He is a member of the IEEE.

Suhaib A. Fahmy (M'01, SM'13) received the M.Eng. degree in information systems engineering and the Ph.D. degree in electrical and electronic engineering from Imperial College London, UK, in 2003 and 2007, respectively.

In 2007 he became Research Fellow at Trinity College Dublin and Visiting Research Engineer at Xilinx Research Labs, Dublin. He joined Nanyang Technological University, Singapore in 2009 as Assistant Professor in the School of Computer Engineering. In 2015, he moved to the University of Warwick, UK, where he was Associate Professor then Reader in the School of Engineering. He joined King Abdullah University of Science and Technology (KAUST) in 2020 as Associate Professor in Computer Science. His research interests include reconfigurable computing, compute acceleration in a connected context, and high-level heterogeneous systems design.

Dr Fahmy was a recipient of the Best Paper Award at the IEEE Conference on Field Programmable Technology in 2012, the IBM Faculty Award in 2013 and 2017, and the ACM TODAES Best Paper Award in 2019. He is a Senior Member of the ACM and Chartered Engineer and Member of the IET.

Thambipillai Srikanthan joined Nanyang Technological University (NTU), Singapore in 1991. At present, he is a Professor and the Executive Director of the Cyber Security Research Centre @ NTU (CYSREN). Prior to this, he was the Chair of the School of Computer Engineering at NTU. He founded CHiPES in 1998 and elevated it to a university level research centre in February 2000. He has published more than 250 technical papers. His research interests include design methodologies for complex embedded systems, architectural translations of compute intensive algorithms, computer arithmetic, and high-speed techniques for image processing and dynamic routing.