Bus Matrix Synthesis based on Steiner Graphs for Power Efficient System-on-Chip Communications

Renshen Wang, Yulei Zhang, Nan-Chi Chou, Evangeline F. Y. Young,
Chung-Kuan Cheng, *Fellow, IEEE* and Ronald Graham

***Abstract*—Power consumption and the thermal wall have become the major factors limiting the speed of VLSI circuits, while interconnect is becoming a primary power consumer. These factors bring new demands on the communication architecture of system-on-chips (SoC). High bandwidth is desired to enhance parallelism for better performance, and the power efficiency on this bandwidth is critical to the overall SoC power consumption. Current bus architectures such as AMBA, Coreconnect and Avalon are convenient for designers but not efficient on power. This paper proposes a physical synthesis scheme for on-chip buses and bus matrices to minimize the power consumption, without changing the interface or arbitration protocols. By using a bus gating technique, data transactions can take shortest paths on chip, reducing the power consumption of bus wires to minimal. Routing resource and bandwidth capacity are also optimized by the construction of a shortest-path Steiner graph, wire sharing among multiple data transactions, and wire reduction heuristics on the Steiner graph. Experiments indicate that the gated bus from our synthesis flow can save more than 90% dynamic power on average data transactions in current AMBA bus systems, which is about 5%∼10% of total SoC power consumption, based on comparable amount of chip area and routing resources.**

***Index Terms*—Algorithm, physical synthesis, communication graph, Steiner graph, power efficiency, data throughput.**

I. Introduction

As the feature size of process technology scales down, system-on-chips (SoC) are capable of integrating more components and gaining higher complexity. Since clock frequency on single components are reaching a limit due to power and thermal limitations, better performance will be mostly exploited through parallelism [1] [3]. As a result, two factors determine that on-chip communication architectures are becoming a critical aspect in future systems. First, the communication latency and bandwidth among system components may become a bottleneck of performance. Second, the percentage of power consumed on inter-component communications in the whole system power has scaled up to a significant level [9] [13] [15].

Industrial on-chip bus standards include AMBA [29] [31], CoreConnect [30], Avalon [32], *etc*. These existing standards can provide an interface for IP developers and a communication solution for system designers. Compared to the network-on-chip [10] type of communications, buses are small on silicon footprint, fast in terms of latency, and easy to implement. Moreover, the implementations can be reconfigured according to specific applications, enabling designers to apply various optimizations for best performance on available resources.

The advantages of simplicity make buses popular in industrial SoC designs. However, current bus architectures are not power efficient on transferring data through bus lines. And since this part of power is scaling up as technology advances [13], it becomes a necessity to introduce physical level optimization on bus synthesis to minimize the power consumed by inter-component communication on bus lines. When high bandwidth is required on these buses, wire efficiency may also become low, which ultimately limits the system bandwidth capacity and performance.

We propose a physical synthesis scheme for on-chip buses to eliminate the disadvantages in existing bus architectures, but not to change the existing protocols and component interfaces. Based on shortest-path Steiner graphs, efficiency on bus lines are maximized without the need to redesign system components and IP modules. Routing resource is also reduced without compromising low power. The cost on our new scheme is the additional silicon resource consumed by distributed controls and switches, which is scaling down by Moore's law. Under technology trends, this physical synthesis scheme is capable of bringing a large improvement on power and performance based on current state-of-the-art on-chip buses and bus matrices.

A. Related work

There is a large body of work on system level communications and related power analysis and power saving techniques. For instance, [16] [17] [18] explored the design space of on-chip buses on tree topologies and/or system floorplan. To evaluate the system performance, usually a "communication constraint graph" [16] [19] is extracted from a specific application, on which each topology/floorplan configuration has an estimated performance by analysis [16] or simulation [17] [18]. An elaborate power analysis on AMBA on-chip bus is performed in [15], where the detailed decomposition of power

Manuscript received April, 2010. This research is partially supported by NSF CCF-0811794, CCF-1017864 and California MICRO Program.

Renshen Wang, Chung-Kuan Cheng and Ronald Graham are with Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0404, USA. e-mail: {rewang, rgraham, ckcheng}@cs.ucsd.edu.

Yulei Zhang is with Department of Electrical and Computer Engineering, University of California, San Diego, La Jolla, CA 92093-0407. e-mail: y1zhang@ucsd.edu.

Nan-Chi Chou is with Mentor Graphics Corporation, San Jose, CA 95131, USA. e-mail: nanchi_chou@mentor.com.

Evangeline F. Y. Young is with Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong. e-mail: fyyoung@cse.cuhk.edu.hk.

consumed by system components are obtained by simulation on NEC's gate-level power estimator.

Power saving techniques have been explored and applied extensively to break through the "power wall" of VLSI circuit performance. Clock gating [5] is nowadays widely used to reduce dynamic power, and power gating [20] is used to avoid unnecessary static power. In bus communications, a large part of the power is consumed on the wires of bus lines [15], which is relatively scaling up with technology and applications [9] [13]. Techniques of clock gating can be used on bus lines to achieve a similar goal, which is to mask off signals wherever they are not needed. Bus segmentation in [6] has such effect to help reduce dynamic power, but the effect is largely limited by tree structures topologies. Also in [18], a power-performance tradeoff is analyzed on bus matrices, where a bus matrix is composed of a set of tree structured buses. We extend the structures from trees to graphs, using Steiner graph connections for a thorough optimization of "bus gating" to minimize the communication power.

Topologies have been mostly discussed in bus optimizations, while the physical/geometrical information are not being emphasized. As the physical locations of on-chip components become more and more relevant to both power and performance of SoCs, physical optimization has become a necessity for efficient bus architectures. On the power side, communication power depends on the wire capacitance involved in data transactions. And on the performance side, delay is dominated by signal propagation distance, bandwidth is limited by routing congestion, and ultimately by power or thermal constraints. Physical and geometrical properties of on-chip buses are tightly correlated with topological structures, so that they need to be co-optimized in the synthesis flow.

B. Paper overview

In this paper, we optimize on-chip bus communications on the tradeoffs between minimal power, maximal bandwidth, and minimal total wire length. We use the protocols of AMBA AHB [29] and AXI [31], since they are most popular in industrial designs. Based on AMBA protocols, we modify the bus structure using a "bus gating" technique, and apply optimizations which is biased towards minimal power, but also favors bandwidth and routing resource. Heuristics are devised to construct a minimal shortest-path Steiner graph, and to reduce its scale with a minimal increment on path lengths. The overall optimization flow can be viewed as three major steps.

Step 1: Generating the shortest-path Steiner graph H
(for minimal power)
Step 2: Deciding edge weights on H
(for adequate bandwidth)
Step 3: Applying incremental modifications on H
(for minimal wire length)

Experiments show large reduction of wire power consumption over existing bus architectures.

The rest of this paper is organized as follows. Section II introduces some background information on bus gating. Section III formulates our bus optimization problems. Section IV shows the heuristics for minimizing power and section V for minimizing wire length. Experiments are illustrated in section VI. Finally, section VII gives our conclusions on bus matrix, with comparisons to network-on-chips and analogies to city traffic planning.

II. Bus Architectures and Bus Gating Background

Standard on-chip buses like AMBA were designed to enable fast and convenient integration of system components into the SoC, where simplicity is one of the major objectives. When the bus power consumption comes to a significant level that we cannot afford to ignore [15], power optimization will be desirable. We introduce a "bus gating" technique [23] to minimize the power on bus lines with a small compromise on design simplicity.

A. AMBA on-chip bus and bus matrix architectures

The AMBA AHB on-chip bus [29] and bus matrix [31] are drawn in figure 1 and figure 2. The components connected by these buses can be classified into masters and slaves. Masters are typically microprocessors, each can start a transaction with one slave device at a time, where the slave is selected by giving an address to the decoder. Slave devices respond to masters passively. When conflicting requests come from multiple masters, arbiters will decide the order of services.

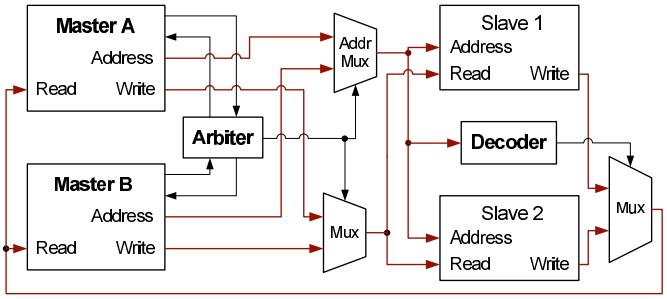

Fig. 1. AMBA AHB bus.

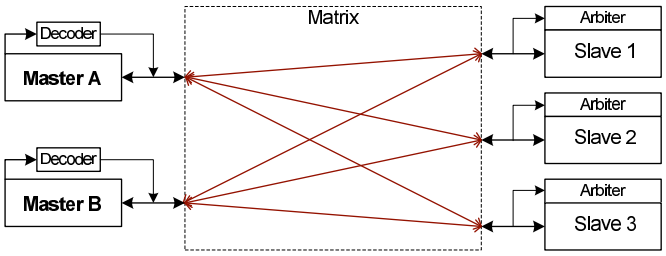

Fig. 2. AMBA AXI full bus matrix (sketch).

The main difference between the bus and bus matrix is on multiple access from masters. The basic bus allows one master access at a time, while the bus matrix may allow multiple accesses. In a full bus matrix like figure 2, the masters and slaves are connected like a bi-clique, and each slave has an arbiter. Full bus matrices have largest bandwidth capacity, typically applied for maximum performance.

B. Power and wire efficiency of gated bus using Steiner graphs

The power efficiency of a bus architecture like figure 1 is low because the bus lines from masters to slaves are connecting all the slave devices by a single large wire net. The same

is on slave-to-master connections. While the communication is one-to-one, the signals are sent to all the receivers regardless whether they are needed, which results in wasted dynamic power on bus wires and component interfaces. Moreover, this low power efficiency is still being worsened by the technical scaling of global wires [13] and the increasing number of components integrated into SoCs.

Gated bus is a solution to save the wasted dynamic power. The simplest way is to add a de-multiplexer after each multiplexer in figure 1, and add a de-multiplexer after each master device in figure 2, so that the signals only propagate to where they are needed. This method works in a similar way as clock gating [5] [11], and can be even more effective because the signal receivers here have much less complex behaviors than in a clock tree.

For tree structured buses, distributing the multiplexer and de-multiplexer into the wire net (figure 3.a) helps to save both power and wires. For wire length, while the single multiplexer needs independent lines from every sender, the lines can be shared with distributed multiplexers and form a Steiner arborescence [7] [21] [22]. An arborescence is a directed tree such that every root-to-leaf path is shortest. On the receivers' side with distributed de-multiplexers, the bus lines change from a rectilinear Steiner minimum tree (RSMT) [12] [14] to a minimum rectilinear Steiner arborescence (MRSA). By the research in [2], this change increases the wire length by only $2\% \sim 4\%$ on average. So the total bus wire length can be reduced by the distributing the multiplexer/de-multiplexers, while the dynamic power can also be reduced at the same time. There is a small control overhead for sending the signals over the arborescence, but compared to the bus width and data throughput, this dynamic power overhead is negligible. Based on the same tree topology, effective bus gating can be applied by distributing the control over the entire tree (arborescence).

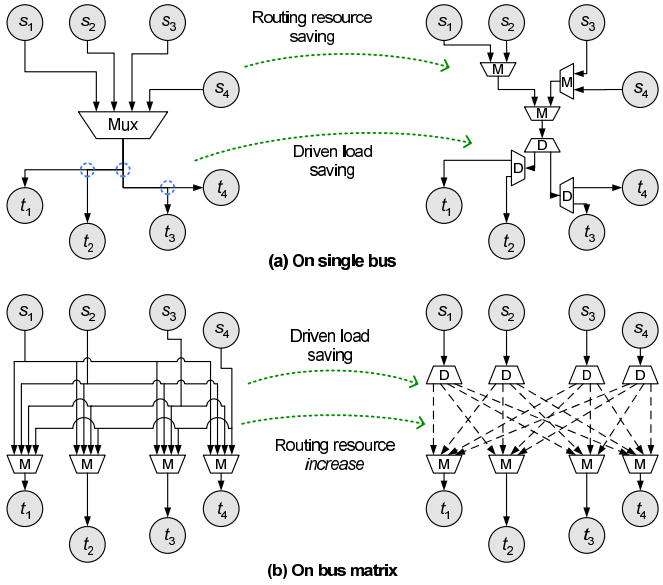

Fig. 3. Bus gating using distributed mux and de-mux.

On bus matrices, however, simply adding de-multiplexers may increase the total wire length, because when the number of master-to-slave paths becomes large, each path will need its own bus wires (as in figure 3.b). To reduce wire length in the bus matrix, also to further reduce power on the basic bus, we adopt the structures of Steiner graphs. A Steiner graph is a generalization of Steiner trees, without the limitation of tree structure that there is only one root placed at a certain point, which cannot be on the shortest path of every connection. By removing the constraint of tree topologies, we gain higher freedom to choose shortest paths for reduced power on data transactions, and to let the paths share wires for reduced routing congestion.

As defined in [4], for an unweighted graph $G = (V, E)$, $G'' = (V'', E'', \omega)$ is a Steiner graph of G if $V \subseteq V''$ and for any pair of vertices $u, w \in V$, the distance between them in G'' is at least the distance between them in G. Figure 4 shows a Steiner graph of G with $V = \{s_1, s_2, t_1, t_2\}$ and $E = \{(s_1, t_1), (s_1, t_2), (s_2, t_1), (s_2, t_2)\}$ with each edge weighted 1, and its implementation as a bus or bus matrix. This graph is minimal in terms of total wire length. Moreover, every edge in E has a path in G'' with minimum length, *i.e.* the path length equals the Manhattan distance between the two vertices. In this way, each data transaction involves minimal wires, leading to minimal dynamic power on bus lines.

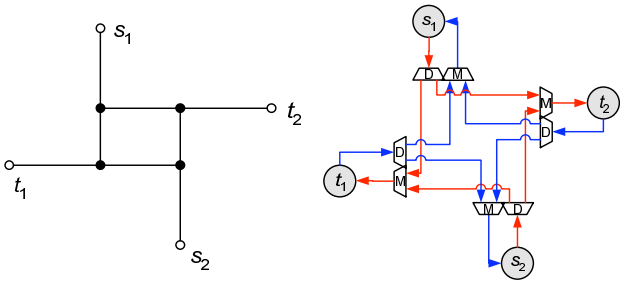

Fig. 4. A shortest-path Steiner graph G'' and its bus implementation.

Shortest-path Steiner graphs have advantage on power efficiency as shown above. Naturally, graph structures also have advantage on communication bandwidth over trees. Our objective of bus gating and bus matrix synthesis is to perform a balanced optimization on power and bandwidth even when available routing resource is limited.

C. Design flow with gated bus synthesis

The bus gating technique may bring some additional complexity in the design flow. Traditionally, physical level design starts from gate level netlist, and goes through placement, routing, timing analysis, verification, *etc*. With bus gating, since the buses are usually included in the system with wires and control units, floorplan and placement depend on the bus connections, while the topology of the gated bus may depend on floorplan and placement.

To resolve this loop of dependency, we need to change the design flow by inserting the bus gating stage into placement and routing. After the initial placement considering the bus as a big wire net, the gated bus synthesis is performed, with a minimal update on the netlists and placement of bus units. Since the updates are limited to the bus or bus matrix part of the system, the process can be controlled in small scale and mostly automated. So provided with appropriate algorithms, the impact of bus gating on design flows can be minimal.

III. Problem Formulations

We require the bus synthesis algorithm to generate a bus matrix based on a given "communication constraint graph" [16] [19] and a placement of master and slave devices. In this way, on-chip bus matrices can be flexibly reconfigured for different system designs and communication patterns. The optimization is on power and wires under the bandwidth requirement given by the graph. Here with AMBA protocols, we can use definition 1 to model the communication graph of bus matrices.

Definition 1: A communication graph $G_C = (V_s, V_t, A)$ is a directed bipartite graph, where V_s is the set of source vertices, V_t is the set of terminal vertices, and A is the set of arcs from V_s to V_t.

We denote the set of master devices by V_s, the set of slave devices by V_t. An arc (v_i, v_j) in G_C means master device i needs to access slave device j. Also given are the fixed on-chip locations of these device.

Definition 2: A placement on a communication graph G_C is a physical location function $P : V_s \bigcup V_t \mapsto R^2$.

A. Power and bandwidth models

To make the objective clear and simple, the power consumption on the bus matrix is estimated by average length of communication paths. Interconnect dominance on power [9] [13] determines that the wires involved in a data transaction consume most of the power, which is proportional to the involved wire capacitance C_{inv} and the square of supply voltage V^2. Assume the voltage is constant, and we use the same type of wires throughout the bus matrix, the power is then proportional to the path lengths of data transactions.

The bandwidth capacity of a bus matrix is basically the number of simultaneous master-slave connections it can hold. Assume no TDMA (time division multiple access) or similar techniques are used, i.e. one set of bus line supports at most one connection at a time. Also, each master or slave device is limited to one connection each time. In this way, we need multiple sets of bus lines between two points u and v to enable the same number of simultaneous connections on (u, v), by which we define the edge weight $\omega((u, v))$. Besides edge weights, bandwidth capacity also depends on other factors, including the locations of all the devices, the communication graph, and the bus matrix topology.

B. Maximum bandwidth bus matrix formulation

To meet the demand of the communication graph G_C, we define the bus matrix graph based on a Steiner graph of G_C. Every connection path should take the shortest rectilinear path for minimal communication power, i.e. the path from a to b has the length of Manhattan distance $\|P(a) - P(b)\|_1$. Path definition is natural (same as in [19]).

Definition 3: For communication graph $G_C = (V_s, V_t, A)$ and placement function $P : V_s \bigcup V_t \mapsto R^2$, a bus matrix graph is a weighted graph $\Theta = (V, E, \omega)$ with placement $P' : V \mapsto R^2$ such that

i) $V_s \subseteq V$
$V_t \subseteq V$

ii) $\forall\ v \in V_s \bigcup V_t,\ P'(v) = P(v)$

iii) *For any $A' \subseteq A$ such that*
$\forall\ (u_i, w_i) \neq (u_j, w_j) \in A',\ u_i \neq u_j \wedge w_i \neq w_j,$
there is a set of paths $\rho : A' \mapsto \Pi$ such that

a) $\forall\ (u, v) \in A',\ \rho((u, v)) \subseteq V \bigcup E$

b) $\forall\ (u, v) \in A',$
$\sum_{(i,j)\in\rho((u,v))} \|P(i) - P(j)\|_1 = \|P(u) - P(v)\|_1$

c) $\forall\ e \in E,\ |\{r \in \Pi : e \in r\}| \leq \omega(e)$

The objective is to find the bus matrix graph with minimal total wire length $L(\Theta) = \sum_{(u,v)\in E} \omega((u, v))\|P(u) - P(v)\|_1$.

The bus matrix graph is defined above to have the capability of efficient communications. Constraint **i)** and **ii)** ensures that the graph covers all the devices. Constraint **iii)** dictates that for any set of disjoint arcs in A, there is a set of connection paths Π, where each path is shortest (by **iii-*b***) and the weighted edges in Θ can hold all the paths in Π (by **iii-*c***). The total weighted edge length, i.e. total wire length is to be minimized. So the bus matrix we are looking for should support all possible communication patterns, consume minimal power, and use minimal routing resources.

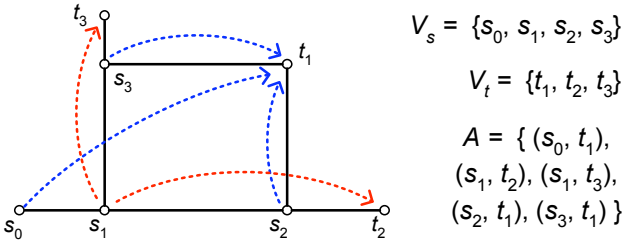

Fig. 5. An example case and the ideal bus matrix graph.

Figure 5 shows an example of a bus matrix graph connecting four masters s_0, s_1, s_2, s_3 and three slaves t_1, t_2, t_3. Five communication arcs are present: s_1 may access t_2 and t_3, and t_1 may be accessed by s_0, s_2 and s_3. The single weight edges in figure 5 (by solid segments) are adequate for this requirement. Notice that (s_0, t_1) is the only arc having more than one shortest paths. And when its connection is on, s_2 and s_3 cannot access t_1 at the same time, i.e. bus lines "$s_2 \leftrightarrow t_1$" and "$s_3 \leftrightarrow t_1$" are both open. Depending on s_1's connection, since s_1 can take at most one of "$s_1 \leftrightarrow s_2$" and "$s_1 \leftrightarrow s_3$", the connection from s_0 can always choose the one other than s_1's and find an open path to t_1.

This formulation defines an ideal high bandwidth low power on-chip communication solution, but with limited practicality. Because first, minimization on the wire length of Θ is computationally expensive due to the exponentially increasing combinations of arc subset A'. And even if we pre-compute the optimal solution, it is still impractical to store the path sets for all the subsets, or to compute the path set Π in real time. Another problem is that, if the communication pattern changes dynamically, when some connections are still on but

need to change paths, it may induce extra delay or timing issues. To make a practical bus matrix achievable with fast response, small silicon footprint and good power efficiency, we slightly weaken the formulation as follows.

C. Practical bus matrix synthesis formulation

Given a communication graph G_C and its placement, we define another bus matrix graph with fixed paths for the arcs in G_C, i.e. each pair of master-slave connection always takes the same path regardless of other connections.

Definition 4: For communication graph $G_C = (V_s, V_t, A)$ and placement function $P : V_s \bigcup V_t \mapsto R^2$, a bus matrix graph is a weighted graph $H = (V, E, \omega)$ with placement $P' : V \mapsto R^2$ and a set of fixed path $\rho : A \mapsto \Pi$ such that

i) $V_s \subseteq V$
$V_t \subseteq V$

ii) $\forall\ v \in V_s \bigcup V_t,\ P'(v) = P(v)$

iii) $\forall\ (u, v) \in A,\ \rho((u, v)) \subseteq V \bigcup E$ *and*
$\sum_{(i,j)\in\rho((u,v))} \|P(i) - P(j)\|_1 = \|P(u) - P(v)\|_1$

iv) *For any $A' \subseteq A$ such that*
$\forall\ (u_i, w_i) \neq (u_j, w_j) \in A',\ u_i \neq u_j \wedge w_i \neq w_j$,
we have $\forall\ e \in E,\ |\{a \in A' : e \in \rho(a)\}| \leq \omega(e)$

In this formulation, the constraints are similar to those in definition 3, except that the connection paths no longer depend on the communication pattern. The same case in figure 5 has a different solution here because of the reduced flexibility on paths. Since the path from s_0 to t_1 is fixed, it conflicts with either path (s_1, t_2) or path (s_1, t_3). For the objective of minimal wire length, we should assign weight 2 on edge (s_1, t_3), and arc (s_0, t_1) always takes path "$s_0 \leftrightarrow s_1 \leftrightarrow s_3 \leftrightarrow t_1$".

We can see that the total wire length of the optimal solution $L(H_m)$ may be larger than that of the ideal formulation by definition 3. However, the fixed paths can be easily stored in bus control units, so that the entire communication protocol can be implemented with very small consumption on area and power. Therefore, the formulation based on definition 4 is more preferable for practical use. Algorithms based on this formulation are elaborated in the following two sections.

IV. Bus Matrix Graph Construction

The flow we use is to first construct a shortest-path Steiner graph based on the given placement of $V_s \cup V_t$ and communication graph G_C, and then decide the weight $\omega(e)$ on each edge. The single-source case is the minimum rectilinear Steiner arborescence (MRSA) problem, which is well studied in previous work such as [7] [21]. Although it is proved to be NP-complete in [22], heuristic algorithms can provide close-to-optimal solutions of Steiner arborescences. Our shortest-path Steiner graph is constructed by multiple iterations of a revised MRSA construction.

A. k-IDeA/G heuristic for MRSA

The RSA/G heuristic for the MRSA problem was first introduced in [21], and is proved to be 2-approximate. Given a single source and n terminals, the basic flow is to start with n subtrees and iteratively merge a pair of subtree roots v and v' such that the merging point is as far from the source as possible, so that the wires can be shared as much as possible. It terminates when only one subtree remains. For efficient implementation, the RSA/G first sorts all the nodes on the Hanan grid [26] with decreasing distance to the source, and visits each node while maintaining a peer set P of subtree roots. Details are shown in the pseudo code in table I, where two operations are used at: terminal merger opportunity (TMO), when a terminal is added into P as a subtree; and Steiner merger opportunity (SMO), when $|X| \geq 2$ and the subtrees in X are merged.

TABLE I
The RSA/G algorithm

Given a source s and n terminals $t_1, \cdots, t_n$,
$v_1, \cdots, v_N$ are the Hanan grid nodes of $\{s, t_1, \cdots, t_n\}$
  sorted by decreasing distance to s
$Q \leftarrow \phi$;
for $i = 1$ to N do
  if there is t_j at v_i, then **(TMO)**
    $Q \leftarrow Q \bigcup \{v_i\}$;
  $X \leftarrow Q \bigcap \{v_j : \|P(s) - P(v_j)\|_1 = \|P(s) - P(v_i)\|_1 + \|P(v_i) - P(v_j)\|_1\}$;
  if ($|X| \geq 2$) then **(SMO)**
    merge the nodes in X rooted at v_i
    $Q \leftarrow (Q \bigcap \overline{X}) \bigcup \{v_i\}$;
return the arborescence rooted at s;

The k-IDeA/G (iterated k-deletion for arborescence) algorithm is developed in [7] based on the RSA/G. In each iteration, it removes up to k nodes from $v_1, \cdots, v_N$ when running the RSA/G algorithm. By removing the nodes, some SMO merges are skipped, which in some cases can result in a better overall solution. All the combinations of the k or fewer skipped nodes are tried in an iteration, and the best set of skipped nodes are marked as permanently deleted. The iterations are repeated until no further improvement occurs.

B. Shortest-path Steiner graph by multiple MRSAs

For a shortest-path Steiner graph with multiple sources $s_1, \cdots, s_m$, the idea behind single source MRSA is still valid. In fact, our algorithm constructs the Steiner graph H just by iteratively constructing the MRSA rooted at every source. While a single arborescence can be optimized by the k-IDeA heuristic, the m arborescences are individually optimized with the same idea, plus that these arborescences also need to share as much wire as possible to optimize the final Steiner graph. For this purpose, we add additional heuristics based on the RSA/G to construct multiple MRSAs one by one.

First, starting from the second MRSA construction, we can reduce terminals by using existing wires. For each MRSA with source s_i, the terminals that need connection from the source can be moved along existing edges of H towards s_i. As the example shown in figure 6, with the wires of previous aborescences, we only need to connect 8 nodes instead of the original 16 terminals to form the MRSA rooted at s_2, because all the other terminals can be reached from one of these 8 nodes by a shortest path from s_2. This set of nodes (denoted as T') can be obtained by checking each terminal t_j, move from t_j towards s_i as much as possible along existing paths until reaching a vertex (can be a terminal or a Steiner node)

in H where no vertex closer to s_i can be reached, and add this vertex to T'. When there are multiple paths in the graph, we pick the final vertex closest to s_i, so the rest part of the path is short and likely to need less wires. Details are in the routine "Necessitate(v)" in table II.

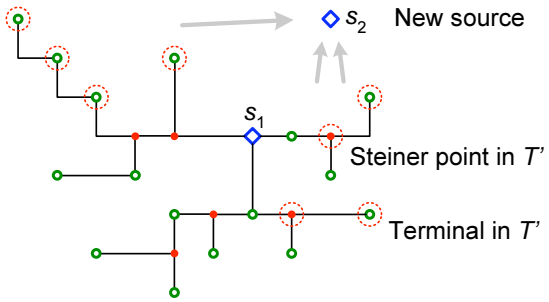

Fig. 6. Nodes requiring connections (in dotted circles).

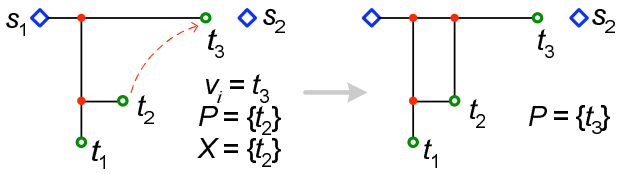

Fig. 7. Connecting a node into the Steiner graph.

Second, we construct the MRSA based on the set of nodes T' using as much existing wires as possible. Compared to the RSA/G heuristic, the TMO condition is changed to $v_i \in T'$; The SMO condition is changed, also for the purpose of wire reusing, from $|X| \geq 2$ to $|X| \geq 2$ or ($|X| = 1$ and $v_i \in H$). Because when v_i is already in the graph, it was added into previous MRSAs and can share wires with the node in X like the case in RSA/G when $|X| \geq 2$. As the example in figure 7 shows, when X contains only one node $\{t_2\}$, it should be connected into H when v_i comes to t_3, and half of the connection length can be saved using the existing horizontal wire. The detailed algorithm is described in table II, where the routine "connect(u, v)" uses existing wires if applicable on shortest connections.

The k-IDeA iterations remain unchanged here. And after the shortest-path Steiner graph is constructed by applying k-IDeA on the m sources, there are possibly some redundant edges that can be removed. So the final step is to check each edge $(v_i, v_j) \in H$, if H still contains all the source-to-terminal shortest paths without (v_i, v_j), then remove it from H.

C. Edge weights

With the graph topology constructed, we can decide the weight on each edge, i.e. number of parallel bus lines along each edge. By the formulation in definition 4, we pick a fixed shortest path in the Steiner graph for each connection. When multiple shortest paths exist between a pair of source and terminal, a random path is chosen, so that the set of paths are less likely to congest on certain edge(s). The weight $\omega(e)$ on each edge e is then determined by the possible communication subset A' in definition 4. The maximum size of the set $\{a \in A' : e \in \rho(a)\}$ for all A', which is used as edge weight $\omega(e)$, can be computed as a maximum bipartite

TABLE II
REVISED RSA/G' ALGORITHM

Given existing Steiner graph G, source s_k, terminals $t_1, \cdots, t_n$, and Hanan grid nodes $v_1, \cdots, v_N$ are same as in RSA/G;

Routine Necessitate(vertex v);
  $U \leftarrow \{u \in G$ and exists a wire path from v to u of length $\|P(s_k) - P(v)\|_1 - \|P(s_k) - P(u)\|_1\}$;
  $T' \leftarrow T' \bigcup \{u_m \in U$ with minimum $\|P(s_k) - P(u)\|_1\}$;
$T' \leftarrow \phi$;
for $i = 1$ to n do Necessitate(t_i);
$Q \leftarrow \phi$;
for $i = 1$ to N do
  if $v_i \in T'$ then $Q \leftarrow Q \bigcup \{v_i\}$; **(TMO)**
  $X \leftarrow Q \bigcap \{v_j : \|P(s_k) - P(v_j)\|_1 = \|P(s_k) - P(v_i)\|_1 + \|P(v_i) - P(v_j)\|_1\}$;
  if ($|X| \geq 1$ and $v_i \in G$) then **(SMO)**
    for each ($u \in X$) connect(v_i, u);
    $Q \leftarrow Q \bigcap \overline{X}$;
    Necessitate(v_i);
  else if ($|X| \geq 2$) then **(SMO)**
    merge the nodes in X rooted at v_i
    $Q \leftarrow (Q \bigcap \overline{X}) \bigcup \{v_i\}$;
return; (the MRSA rooted at s_k is added to G)

matching [25] on the subgraph $G'(e)$ of the communication graph $G_C = (V_s, V_t, A)$. $G'(e)$ contains all the arcs whose corresponding paths go through edge e. The reason is that the number of connections is naturally limited by the bandwidth on each device, i.e. each master or slave has only one interface to the bus matrix, so it can have at most one connection at a time. With fixed path for each master-slave connection, this maximum matching on $G'(e)$ is the upper limit on the number of paths going through edge e simultaneously.

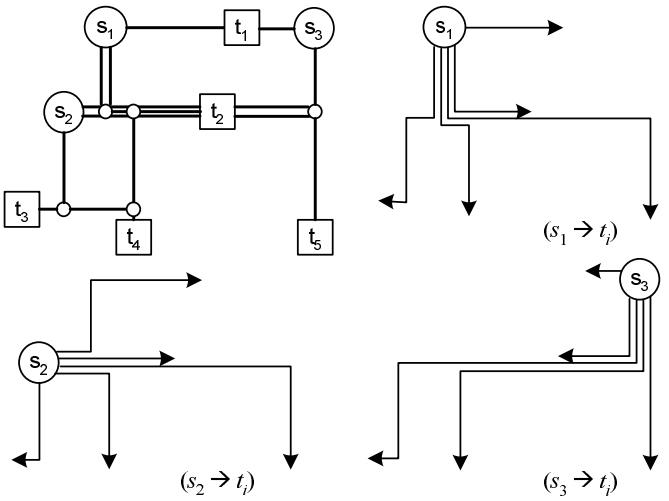

Fig. 8. A bus matrix graph and all its master-slave connections.

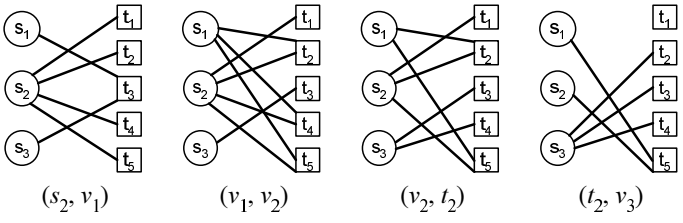

Fig. 9. Bipartite graphs of 4 edges in the bus matrix of Fig. 8.

In the example of figure 8, the communication graph is a biclique between $V_s = \{s_1, s_2, s_3\}$ and $V_t = \{t_1, t_2, t_3, t_4, t_5\}$, i.e. all the $|V_s| \times |V_t| = 15$ arcs are present. The resulting bus matrix graph contains 5 Steiner nodes and 13 edges. Every arc from a master s_i to a slave t_j has a connection of minimal length, and the 15 shortest paths shown in figure 8 are fixed. To assign a weight on each edge, we take $e = (s_2, v_1)$ as example. Six of the fifteen paths go through e, so $G'(e)$ consists of the six corresponding arcs (s_1, t_3), (s_2, t_1), (s_2, t_2), (s_2, t_4), (s_2, t_5), and (s_3, t_3). The maximum matching has two edges, because t_3 can only connect to one of s_1 and s_3. Therefore, $\omega(e) = 2$ is adequate to support all communication patterns. Figure 9 shows the bipartite graphs of four edges on the central horizontal line.

Despite the number of connections, most of the edges are weighted 1. Yet this bus matrix graph is adequate for maximum bandwidth capacity, i.e. wires will not be the bottleneck of multiple simultaneous connections. The total weighted wire length in this bus matrix is 108. Compared to the total path length 266 if implemented as a full bus matrix in figure 2, the Steiner graph approach saves more than half of the routing resources.

V. Tradeoffs on Power, Wire and Bandwidth

The bus matrix graph H constructed by the algorithm above is optimal for minimal power on bus wires, without considering chip area and routing resources. The area overhead can be small because of the simple control mechanism, but routing resource may become a bottleneck depending on other factors in the design. Therefore, we need to explore some tradeoffs among the objectives in order to have more flexible choices. The objective is to achieve a balanced "shallow-light" optimization like in [8], with both short connection lengths (shallow) and small total wire length (light).

A. Steiner graph reduction

Since high bandwidth bus matrices will need significantly more wires to support parallel communications across the chip, routing resource may become another limitation as more components are integrated in to SoCs and interactions increase. Especially when the components are placed in irregular placement instead of cell arrays, the shortest-path Steiner graph generated by the algorithm in table II may contain a lot of loops, which bring additional wire length. We look for changes in the graph structure which can significantly reduce the wires, while preserving the short paths at the same time.

As in figure 10(a), there are some long and narrow rectangles formed by the graph edges, and the long double edges are necessary if we want all the connections to be shortest. But when the double edges are geometrically very close to each other, combining them into one edge only slightly increases the length of some connections, while possibly saving much more wire length. Figure 10(b) and 10(c) shows the effect of merging parallel segments in narrow rectangles. The total edge length is greatly reduced, while the increment on average path length is relatively small. Although fewer edges will generally result in larger edge weight, the total weighted edge length

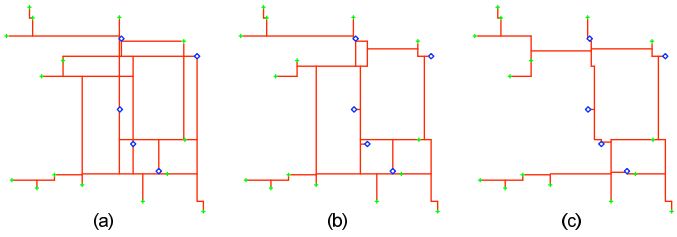

Fig. 10. Merging parallel segments in a bus matrix graph.

(wire length) can still be reduced by this merging operation due to improved wire sharing among paths.

Thus, if we relax the requirement on the path length in definition 4, from the exact Manhattan distance $\|P(u) - P(v)\|_1$ to within $(1+\epsilon)\|P(u) - P(v)\|_1$, we can merge the double parallel edges to save wires. Assume we have a vertical narrow rectangle with dimensions $h \times w$, and we merge the two vertical edges to a single edge placed in middle. The total edge length may be reduced by h, while the lengths of some connection paths increase by $\frac{w}{2} + \frac{w}{2} = w$. So if the h/w ratio is high, this operation can be very helpful on relieving routing congestion, while preserving the low power consumption of a bus matrix.

In the wire length reduction algorithm, we repeatedly search for pairs of parallel double lines in the bus matrix graph, and for each pair, calculate its potential reduction Δl on edge length and possible increment Δp on path lengths. The pair with highest $\Delta l / \Delta p$ ration is merged, and the modified graph will have a new set of connection paths and edge weights. If the added total wire length is really reduced, we keep the merging operation and continue to the next iteration, otherwise discard the operation. Eventually, there will be no positive wire length reduction in the graph, and we have a series of bus matrix graphs with decreasing wire length and increasing path lengths, where a comprise can be chosen.

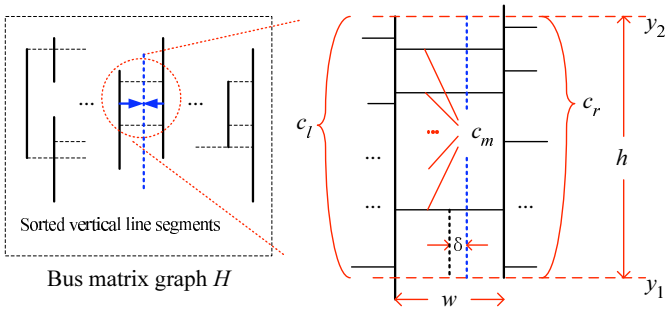

Fig. 11. Searching for mergeable parallel segments (in vertical direction).

The process of searching for vertical mergeable parallel segments is illustrated in figure 11. (Horizontal lines are processed in the same way with x-y coordinates switched.) First, the vertical line segments in the Steiner graphs are sorted by their x coordinates, denoted as $u_1, u_2, \cdots, u_k$. Then for each pair of segments u_i, u_j $(i < j)$ with a common y interval $[y_1, y_2]$, if between i and j there is no other vertical segment on $[y_1, y_2]$, u_i and u_j are a pair of mergeable segments.

On the parallel segments u_i and u_j, let c_l denote the count of horizontal lines connected to the left, c_r denote the count

of lines connected to the right, and c_m the count of lines connecting u_i and u_j in the middle. Assume $c_l < c_r$, so the combined vertical segment may not be at the middle but have an offset δ to the right of the midpoint.

The reduction on total edge length Δl is by combining the vertical segments of length h and changing the lengths of related horizontal connections. The two vertical segments are reduced to one, which reduces edge length by h. The central c_m edges of length w are totally removed. However, the lengths of c_l connections on the left are increased by $\frac{w}{2}+\delta$, and the lengths of c_r connections on the right are increased by $\frac{w}{2}-\delta$. To sum up, $\Delta l = h+c_m w - c_l(\frac{w}{2}+\delta) - c_r(\frac{w}{2}-\delta)$. On the possible increment on path lengths, since the left vertical segment is pushed rightwards by $\frac{w}{2}+\delta$, a path may need to detour and add $\Delta p = w + 2\delta$ of distance. So the ratio is

$$\frac{\Delta l}{\Delta p} = \frac{h + c_m w - c_l(\frac{w}{2}+\delta) - c_r(\frac{w}{2}-\delta)}{w+2\delta}$$

$$= \frac{c_r - c_l}{2} + \frac{h-(c_r-c_m)w}{w+2\delta}$$

The best offset value δ can be decided by the right part $\frac{h-(c_r-c_m)w}{w+2\delta}$ to maximize the ratio. If the upper part $h-(c_r-c_m)w \geq 0$, i.e. $\frac{h}{w} \geq c_r - c_m$, then let $\delta = 0$ so that the merged vertical segment is placed at middle. Otherwise $\frac{h}{w} < c_r - c_m$, let $\delta = w/2$ which is the maximum offset value, and the merged segment is at the right segment u_j's position (note we assume $c_l < c_r$, otherwise δ is negative).

In figure 10, the graphs are the stages of the merging iterations applied on a Steiner graph. First, the long and narrow rectangles are removed, followed by wider rectangles. If we do not require high bandwidth capability, i.e. edge weights all set to 1, the final graph has about half total edge length of the original shortest-path Steiner graph. On weighted bus matrix graphs, the reduction on total wire length is usually smaller, since the number of connections is still $|A|$, and the edge weights are increased by the merging operations. Nevertheless, we still can achieve a significant reduction on total wire length in average cases (test cases in section VII).

Notice that the segment merging operation also helps to merge Steiner nodes which are generated very close to each other. In practice, locally congested Steiner nodes can be hard to implement, because each node needs the area for a switch box and its control unit. Our operation does not guarantee to resolve all closely placed nodes, since it prioritize longer segments, and may leave small square-shaped subgraphs unchanged. Nevertheless, this situation can be easily resolved by a post-processing algorithm, which scans each Steiner node (denoted as v_i), look at v_i's close neighbors within a small $d \times d$ box and compute the density of Steiner nodes in the area. For a box with too many nodes, we can shrink all the nodes in that box into one, and implement it by a single switch. The changes on the bus matrix graph by this operation are limited in the small box areas. The path lengths are not increased if we use the type of crossbar switch in figure 13, because it allows shortest path connections between every pair of ports. The reason we do not implement the whole bus matrix as a crossbar is that it brings excessive wire length, which is not a problem in a local $d \times d$ box area.

B. Overall optimization flow

Combining the heuristic algorithms of graph generation and edge merging, the overall optimization flow is shown in Table III. Step 1 is the construction algorithm of shortest-path Steiner graph in Table II. Step 2 is by the bipartite maximum matching in section IV-C. Step 4 is the iterative graph reduction process by parallel segment merging operations.

The running time of this flow is dominated by step 4, where the maximum matching algorithm need $O(n^2 m)$ time with $n = |V_s + V_t|$ and $m = |A|$. Each edge weight is decided by a maximum matching, and the total number of edges is bounded by the edge count in a Hanan grid [26] which is $O(n^2)$. Therefore the time complexity of one iteration is bounded by $O(n^4 m)$. According to the number of devices in a system-on-chip, the scale of running time will be acceptable for SoC designs within near future. As for the space requirement, $O(n^2)$ of space is adequate for all the algorithms in our flow.

TABLE III
BUS MATRIX PHYSICAL SYNTHESIS

Given a communication graph $G = (U, W, A)$,
and a location function $P : U \bigcup W \mapsto R^2$

Routine Set_Edge_Weights(Steiner graph H)
  For each arc $a = (u, v) \in A$,
    find a shortest path $\rho(a)$ in H from u to v;
  For each edge $e \in E$ in H,
    $A' \leftarrow \{a \in A : e \in \rho(a)\}$;
    $\omega(e) \leftarrow$ Max_matching(A');

1. Generate shortest-path Steiner graph $H_0 = (V, E)$; (by Table II)
2. Set_Edge_Weights(H_0);
3. $k \leftarrow 0$;
4. Repeat
4.1 Find all pairs of parallel segments in H_k,
      and sort them in stack $D[]$ by decreasing $\Delta l/\Delta p$;
4.2 While ($D[]$ is not empty)
      $(u_i, u_j) \leftarrow$ Pop out the segment pair in $D[]$;
      $H_{temp} \leftarrow H_k$ with segment u_i and u_j merged;
      Set_Edge_Weights(H_{temp});
      If ($H_{temp} < H_k$ on total wire length)
        $k \leftarrow k + 1$;
        $H_k \leftarrow H_{temp}$;
        Break the "while" loop;
      Else discard H_{temp};
  Until (no wire length reduction found in H_k)
5. Evaluate the bus matrix graphs $H_0, H_1, \cdots$ by design objectives

The merging operation in step 4 reduces edge length, but does not necessarily reduce the total wire length. In later stages of the iterations, when edges are becoming sparse, merging two edges (e_1, e_2) into one (e') may result in $\omega(e') = \omega(e_1) + \omega(e_2)$. Because with fewer edges left in the graph, the paths have less choices for shortest length, and tend to be congested on critical edges. In such cases, combining e_1 and e_2 does not improve wire sharing, and even increases total wire length because of the increased path lengths.

The iterations of step 4 will terminate when the edges are sparse enough so that total wire length can no longer be reduced. Till that point, our flow has produced a series of bus matrix graphs optimized from device locations and communication requirements (G_C). Among the series of solutions with decreasing wire length and increasing power, designers can choose a best compromise between power and wires, typically the lowest power allowed by available routing resources.

VI. Bus Matrix Control Units and Wires

Apart from path lengths and data wire lengths, the control overhead needs to be considered for a complete optimization. Although the data lines consume the major amount of routing resource because they are usually at least 64-bit (32-bit $\times$ 2-way) wide, control overhead is increased compared to traditional bus architectures by adopting Steiner graphs. We need a lot of switches at Steiner nodes to guide the on-chip traffic, and each switch needs a certain number of control signals depending on its node degree and edge weights.

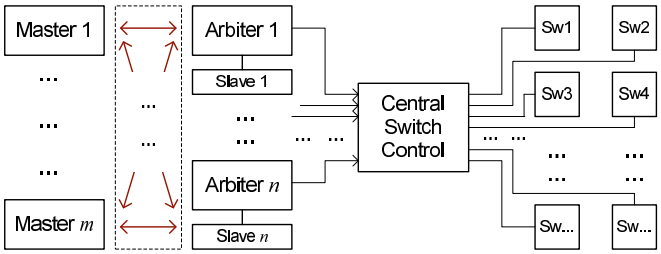

Fig. 12. Control on switches in a bus matrix.

The sketch of bus matrix control scheme is shown in figure 12. Each slave device has an arbiter which handles the requests from masters and decides the connection. The result is sent to the central switch control unit, where all the connection paths are stored. Depending on the set of active paths, the central switch control sends control signals to all the switches on each path, which together instantly create the master-to-slave connection requested by the master device.

A. Control wire overhead

A bus switch is basically a crossbar plus an auxiliary local control which remembers each path going through. The local control handles two types of requests from the central switch control, *create_connection(port1, port2)* and *dispose_connection(port1, port2)*. Typically, a Steiner node in the bus matrix graph has degree 3, and the combinations of *(port1, port2)* is $\binom{3}{2} = 3$, which can be distinguished by 2 control signals. Plus another signal for the create/dispose request, a 3-way switch needs 3 wires connected to the central switch control.

At each slave device, there is an arbiter deciding which master has the access, so we have $\lceil \log_2 m \rceil + 1$ control signals from each arbiter to the central switch control. With a given placement of system components and control units, our algorithm generates the bus matrix graph with all the switches, and the total length of control wires can then be calculated. Compared to a typical bus width of 64 bits, the wire overhead for switch control is relatively low. More details are illustrated in the tables of section VII.

B. Power overhead

First, we assume the power overhead on the control wires can be ignored because of the lower activity rates of control signals. Typically during a data transaction, the control signals do not toggle, and the data is transmitted through the bus with a large amount of toggling on data wires. Plus that the number of control wires is much smaller, the power percentage on control signals has very little impact on the total bus power. We only estimate the power overhead coming from the switches on Steiner nodes.

As constructed in section IV and V, the master-to-slave connections are all along the shortest or near-shortest paths in the Steiner graph. For this purpose, we need to put switches on Steiner nodes to guide all the connections. Compared to traditional bus architectures where the connections are by long lines inserted with buffers, our Steiner graph structure shortens the path length, but also adds crossbar switches which consume more power than basic buffers.

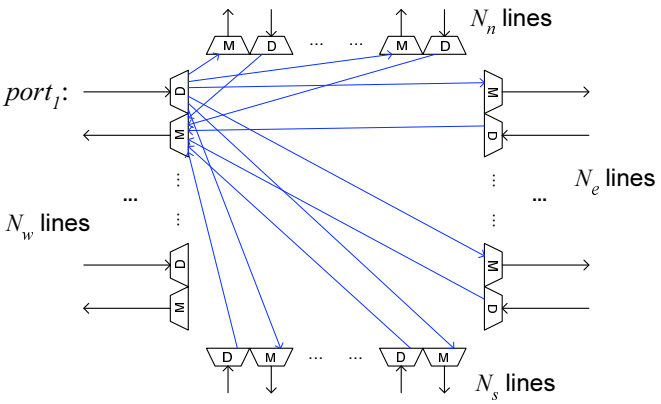

Fig. 13. Implementation of a crossbar switch.

We use a crossbar design illustrated in figure 13. In this example, we have a 4-way switch at a junction of $N_n + N_w + N_s + N_e$ bus lines. It works like a miniaturized bus matrix, enabling 2-way connections between any pair of ports (except two ports at the same side which never need connection). Figure 13 shows the wires connected to the multiplexer and demultiplexer of port 1 on the west side, where each "M" (mux) or "D" (demux) can select among $N_n + N_s + N_e$ connections. Each "M" or "D" can be realized by a binary tree consisting of $\lceil \log_2(N_n + N_s + N_e) \rceil$ levels, where each tree node is a basic 2-to-1 mux or demux.

In a crossbar like this, a path from port_i to port_j is going through $\lceil \log_2(N - N_i) \rceil + \lceil \log_2(N - N_j) \rceil$ muxes and demuxes, where N is the total edge weight at the Steiner node and N_i, N_j are the corresponding edge weights. By using small 2-to-1 muxes/demuxes in binary tree structures, the switch power overhead is on a logarithmic scale of edge weights, lower than that on big n-to-1 muxes/demuxes with sizes on linear scale of edge weights. So we choose this type of crossbar switch for its advantage of low power overhead. Due to the advancing feature size, the overhead can be further reduced to an insignificant level. Also, more details are shown in the following section.

VII. Experimental Results

In our experiments, we implement all the related algorithms, including the shortest-path Steiner graph generation, Steiner graph reduction by parallel line merging, and the edge weight maximum matching. The programs are tested on Windows Vista platform with a 2.2GHz Intel Core2 processor. The running time is short on all the test cases, because the

TABLE IV
POWER AND WIRE LENGTH RESULTS UNDER MAXIMUM BANDWIDTH

Case(m,n)	$\sum L_{v_s,v_t}$	$\overline{L}_{tree}$	Minimal power				Minimal wire			
			$\overline{L}_{path}$	$\overline{P}_{switch}$	$\sum L_{wire}$	$\sum W_{ctrl}$	$\overline{L}_{path}$	$\overline{P}_{switch}$	$\sum L_{wire}$	$\sum W_{ctrl}$
T_0 (3, 16)	305000	84180	6354	5.83%	93000	9.75%	7850	6.95%	62000	10.39%
T_1 (3, 16)	331780	145500	6912	8.52%	105000	9.23%	6912	8.52%	105000	9.23%
T_2 (2, 30)	401220	123050	6687	7.31%	101170	15.68%	7519	11.70%	85850	19.35%
T_3 (3, 16)	331790	82300	6912	8.52%	71680	13.04%	8093	9.98%	69850	12.94%
T_4 (5, 15)	516600	96740	6888	11.83%	141360	10.51%	7232	16.65%	101080	13.83%
T_5 (6, 16)	666260	112100	6940	12.34%	230380	9.57%	7234	16.02%	174600	11.22%
T_6 (8, 8)	440780	89520	6887	10.97%	146060	7.59%	7933	12.15%	113240	9.84%
T_7 (12, 6)	472820	121070	6567	10.40%	157020	7.25%	7100	14.95%	116770	10.57%
T_8 (16, 10)	1092780	143770	6830	18.92%	324290	6.65%	7838	24.41%	207740	9.75%
T_9 (8, 16)	791100	116520	6180	16.52%	272740	8.53%	6646	21.78%	187180	11.00%
T_{10} (8, 16)	958280	128990	7487	15.66%	276630	8.48%	8678	27.90%	163740	12.27%
T_{11} (6, 12)	481300	89700	6685	11.51%	142650	8.78%	6728	11.57%	129420	10.12%
T_{12} (12, 12)	962760	132420	6686	15.85%	274970	7.10%	7299	22.55%	186800	11.83%

algorithms are time/space efficient, and also because most SoC bus matrices will not need to connect too many components (under 32 in our cases).

The test cases we use are mostly artificial, hand made (T_0 and T_1) or randomly generated ($T_{2\sim12}$). They are the same cases used in [23] and [24]. In each test case, the master and slave devices are distributed over a 10mm$\times$10mm square.

The power consumption is estimated by the driven capacitance of data transactions, and can be calculated as a linear combination of path length and switches along the path. Path lengths are minimized by the bus matrix graph construction, since wires are the major power consumer. For the purpose of data completeness, we add the power overhead from the switches on Steiner nodes. According to [27] and [28], we estimate that under 90nm technology, each mux or demux in crossbar switches has about the same capacitance as 25μm of wires. So by using the switch design of figure 13, the dynamic power overhead of going through a switch (port$_i$ – port$_j$) has the power overhead equivalent to adding $25(\lceil\log_2(N-N_i)\rceil+\lceil\log_2(N-N_j)\rceil)\mu$m of wires.

The total wire length on data wires and control overhead are added straightforwardly. Data wire length is the sum of weighted edge length in the bus matrix graph. The control overhead is estimated from figure 12, where the central switch control is placed at the center of the chip, and the control wires consist of those from slave devices to central switch control, and those from central switch control to all switches. We assume the data lines are 64-bit wide, and therefore the percentage on control wires are divided by 64.

A. Maximum bandwidth bus matrix

We list the bus matrix synthesis results on all the test cases in table IV. The unit of all length values is μm. Case $T_i(m,n)$ contains m master devices and n slave devices. The communication graph G_C in these cases is a bipartite connection between the two sides with maximum bandwidth requirement, i.e. a master device can always access any idle slave device without being limited by the number of data lines in the bus matrix.

The objective can be minimum power (i.e. average path length), minimum wire length, or a combination of the two. Table IV shows the results of two single-objective optimizations. At the top of each column,

- $\sum L_{v_s,v_t}$ is the sum of Manhattan distances on all the master-slave pairs;
- $\overline{L}_{tree}$ is the average induced path length (major dynamic power) of master-slave connections in tree structured AMBA AHB buses or bus matrices;
- $\overline{L}_{path}$ is the average path length (major dynamic power) of master-slave connections in the bus matrix graph;
- $\overline{P}_{switch}$ is the added percentage of power overhead in data transactions by the switches on Steiner nodes;
- $\sum L_{wire}$ is the total data wire length;
- $\sum W_{ctrl}$ is the added percentage of control wire overhead.

In the minimal power section, the average path length is exactly $\sum L_{v_s,v_t}/(mn)$, while the total wire length is about one fourth to one third of the total connection length. Compared to traditional bus implementation in [23], the dynamic power saving is mostly over 90% even with the switching overhead added. Overhead on dynamic power increases with the number of components increasing, which requires more bandwidth and larger switches. The percentage is generally under 20% on random cases with under 30 components. So the overall dynamic power here is close to optimal. On the overhead of control wires, the percentage is mostly under 10%, because the number of control signals required is usually very low compared to the 64-bit wide data lines.

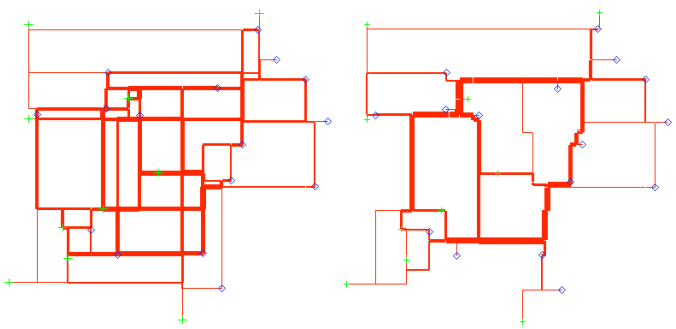

Fig. 14. Case T_8's data wires under minimal power and minimal wire.

In the minimal wire section, the bus matrix graphs are reduced by the parallel line merging heuristic. As a result,

the wire length on most cases is greatly reduced, except for the highly regular case T_1, which is already wire-efficient under minimal power. The data wire length reduction is around 30%, depending on the location distribution of components. And despite the slightly increased percentage, the control wire length is actually reduced, because there are less switches in the reduced bus matrix graph. Figure 14 shows the data wires in case T_8 under minimal power and minimal wire optimizations. The thickness of each segment indicates the number of repeated bus lines (edge weight $\omega(e)$).

Compared to the reduced wire length, the increase on average path length is much lower, mostly around 10% and all under 20%. The power overhead percentage is also increased, because although the Steiner nodes are reduced, the switches along each path are not reduced as much in number, but increased in size. Still, these solutions are relatively power efficient, and we have series of intermediate solutions between minimal power and minimal wire are available for choice.

To see how the path lengths reflect communication power in SoCs, we calculate the bus power consumption with a fixed set of parameters. Assuming 1V of power voltage, 0.2fF/μm of wire capacitance, 4Gbps of transaction bit rate, and 20% of bus matrix activity rate, table V lists the estimated power on bus matrix in each of our test cases. Again we can see a large reduction on total bus power ($\overline{P}_{path} + \overline{P}_{switch}$) compared to $\overline{P}_{tree}$ by traditional Steiner tree structures.

TABLE V
ESTIMATION OF BUS MATRIX POWER CONSUMPTION (IN WATT)

Case	$\overline{P}_{tree}$	Minimal power		Minimal wire	
		$\overline{P}_{path}$	$\overline{P}_{switch}$	$\overline{P}_{path}$	$\overline{P}_{switch}$
T_0	13.5	1.02	0.06	1.26	0.09
T_1	23.3	1.11	0.09	1.11	0.09
T_2	19.7	1.07	0.08	1.20	0.14
T_3	13.2	1.11	0.09	1.29	0.13
T_4	15.5	1.10	0.13	1.16	0.19
T_5	17.9	1.11	0.14	1.16	0.19
T_6	14.3	1.10	0.12	1.27	0.15
T_7	19.4	1.05	0.11	1.14	0.17
T_8	23.0	1.09	0.21	1.25	0.31
T_9	18.6	0.99	0.16	1.06	0.23
T_{10}	20.6	1.20	0.19	1.39	0.39
T_{11}	14.4	1.07	0.12	1.08	0.12
T_{12}	21.2	1.07	0.17	1.17	0.26

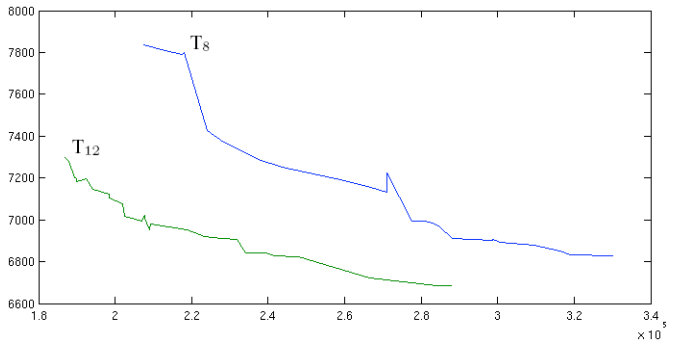

Fig. 15. Average path length vs total (data) wire length in case T_8 and T_{12}.

B. Tradeoffs among power, wire and bandwidth

The parallel segment merging heuristic (in table III's algorithm) can provide a series of bus matrix graphs with decreasing wire length and increasing path length. Figure 15 provides detailed curves in case T_8 and T_{12}. Generally, the paths' length increase as wires being reduced except at a few points. According to power and wire budgets, designers can choose a point on the curve for the best compromise.

If allowed by the system performance requirement, bandwidth capability can be added into the tradeoff. With a bipartite communication graph, the bus matrix should have the capacity of handling $\min(m, n)$ data transactions simultaneously. By reducing this capacity, wires can be saved, i.e. the tradeoff curve is pushed towards lower-left. For bandwidth capacity k, an edge e will need $\min(\omega(e), k)$ copies of bus lines. Applying to our test cases, the wire length start to drop when k goes below 5. But when $k > 5$, the wire reduction is very low. The reason is that the connection paths are randomly distributed over the graph, so very few edges have $\omega(e)$ close to the maximum bandwidth capacity $\min(m, n)$.

Moreover, provided with detailed system communication patterns, bandwidth capacity can possibly be lowered without compromising system performance. Because in real system designs with different cores and peripheral devices, data transactions between certain master-slave pairs may only happen at some specific conditions. So instead of a set of arcs A in the communication graph, we can have a series of arc sets $A_1, A_2, \cdots, A_c$, each one smaller than the original set A, denoting a set of simultaneous connections. Using these sets replacing A in the algorithm of table III, the edge weight can possibly be further reduced. In this way, more detailed system-level behavior information can provide extra capabilities on optimization.

VIII. CONCLUSIONS

We optimize on-chip communications referring to the AMBA AHB bus (matrix) architecture. The weaknesses of original bus matrices, such as low power efficiency and low wire efficiency, are resolved by using a Steiner graph structure. Compared to network-on-chip (NoC) which has better bandwidth flexibility, bus matrix has much less latency because of its centralized control, consumes less power because of the shortest (or close to shortest) paths with minimal control/packet overhead. Therefore, we believe bus matrix architectures will be widely applied for efficient communications in various future systems.

The principle of our work on reducing power is to minimize the data movement on the chip; and that on reducing wires is to maximize wire sharing among different connections. The methods and results have large similarities with city traffic planning and road construction. Like our fixed paths, most people in a city have a fixed route between work and home, and roads are constructed with various number of lanes (width) depending on local traffic density. The two graphs in figure 14 look like roads on a map, which may not be a coincidence but a result of similar principles and approaches.

We devise algorithms which can extensively exploit the on-chip physical design space for a thorough optimization

on power and wire efficiency. The results show promising potentials of bus matrices for low power and high performance on-chip communications. More improvements can be explored in future works on formulations, algorithms and the overall optimization flow.

REFERENCES

[1] S. Adve et al. Parallel computing research at illinois: The UPCRC agenda. Nov. 2008.

[2] C. J. Alpertt, A. B. Kahng, C. N. Szet, and Q. Wang. Timing-driven steiner trees are (practically) free. *ACM/IEEE Design Automation Conf.*, pages 389–392, 2006.

[3] K. Asanovic et al. The landscape of parallel computing research: A view from berkeley. *Technical Report No. UCB/EECS-2006-183*, 2006.

[4] B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Math.*, pages 1029–1055, 2005.

[5] L. A. Ca, Q. Wu, M. Pedram, and X. Wu. Clock-gating and its application to low power design of sequential circuits. *IEEE Custom Integrated Circuits Conf.*, 47:415–420, 2000.

[6] J. Y. Chen, W. B. Jone, J. S. Wang, H. I. Lu, and T. F. Chen. Segmented bus design for low power systems. *IEEE Trans. VLSI Systems*, 7(1):25–29, 1999.

[7] J. Cong, A. B. Kahng, and K.-S. Leung. Efficient algorithms for the minimum shortest path steiner arborescence problem with applications to vlsi physical design. *IEEE Trans. Computer-Aided Design*, 17(1):24–39, Jan. 1998.

[8] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably good performance-driven global routing. *IEEE Trans. Computer-Aided Design*, 11:739–752, 1992.

[9] W. Dally. Keynote: The end of denial architecture and the rise of throughput computing. *ACM/IEEE Design Automation Conf.*, 2009.

[10] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection network. *ACM/IEEE Design Automation Conf.*, 2001.

[11] M. Donno, A. Ivaldi, L. Benini, and E. Macii. Clock-tree power optimization based on rtl clock-gating. *ACM/IEEE Design Automation Conf.*, pages 622–627, 2003.

[12] J. Griffith, G. Robins, J. Salowe, and T. Zhang. Closing the gap: Near-optimal steiner trees in polynomial time. *IEEE Trans. Computer-Aided Design*, 13:1351–1365, 1994.

[13] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings IEEE*, 89:490–504, 2001.

[14] C.-T. Hsieh and M. Pedram. An edge-based heuristic for steiner routing. *IEEE Trans. Computer-Aided Design*, 13(12):1563–1568, Dec. 1994.

[15] K. Lahiri and A. Raghunathan. Power analysis of system-level on-chip communication architectures. *Int'l Conf. Hardware-Software Codesign and System Synthesis*, pages 236–241, 2004.

[16] K. Lahiri, A. Raghunathan, and S. Dey. Efficient exploration of the soc communication architecture design space. *Int'l Conf. Computer-Aided Design*, pages 424–430, 2000.

[17] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane. Floorplan-aware automated synthesis of bus-based communication architectures. *ACM/IEEE Design Automation Conf.*, pages 565–570, 2005.

[18] S. Pasricha, Y.-H. Park, F. J. Kurdahi, and N. Dutt. System-level power-performance trade-offs in bus matrix communication architecture synthesis. *Int'l Conf. Hardware-Software Codesign and System Synthesis*, pages 300–305, 2006.

[19] A. Pinto, L. Carloni, and A. Sangiovanni-vincentelli. Constraint-drive communication synthesis. *ACM/IEEE Design Automation Conf.*, pages 783 – 788, 2002.

[20] M. Powell, S. H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-vdd: a circuit technique to reduce leakage in deep-submicron cache memories. *Int'l Symp. Low Power Electronics and Design*, pages 90–95, 2000.

[21] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[22] W. Shi and S. Chen. The rectilinear steiner arborescence problem is np-complete. *ACM-SIAM Symp. on Discrete Algorithms*, pages 780–787, 2000.

[23] R. Wang, N.-C. Chou, B. Salefski, and C.-K. Cheng. Low power gated bus synthesis using shortest-path steiner graph for system-on-chip communications. *ACM/IEEE Design Automation Conf.*, pages 166–171, 2009.

[24] R. Wang, E. Young, R. Graham, and C.-K. Cheng. Physical synthesis of bus matrix for high bandwidth low power on-chip communications. *ACM Int'l Symp. Physical Design*, 2010.

[25] D. West. *Introduction to Graph Theory*. Prentice Hall, 1999.

[26] M. Zachariasen. A catalog of hanan grid problems. *Networks*, 38:200–1, 2000.

[27] L. Zhang, H. Chen, B. Yao, K. Hamilton, and C.-K. Cheng. Repeated on-chip interconnect analysis and evaluation of delay, power, and bandwidth metrics under different design goals. *Int'l Symp. Quality Electronic Design*, pages 251–256, 2007.

[28] Y. Zhang, X. Hu, A. Deutsch, A. E. Engin, and C.-K. C. James Buckwalter. Prediction of high-performance on-chip global interconnection. *Int'l Workshop on System-Level Interconnect Prediction*, pages 61–68, 2009.

[29] Amba 2.0 specification. *http://www.arm.com/products/solutions/AMBA_Spec.html*, 1999.

[30] Coreconnect bus architecture. *IBM White Paper*, 1999.

[31] Amba 3 specification. *http://www.arm.com/products/solutions/axi_spec.html*, 2003.

[32] Avalon interface specifications. *http://www.altera.com/literature*, 2008.

Renshen Wang received his B.E. degree in Computer Science from Tsinghua University, Beijing in 2005, his M.S. and Ph.D. degrees in Computer Science from University of California, San Diego in 2007 and 2010.

His research interests include CAD algorithms on floorplanning, chip-packaging routing and on-chip communications. Currently, he is a development engineer in the Placement and Route Devision of Mentor Graphics focusing on floorplanning.

Yulei Zhang (S'08) received his B.E. degree in Electrical Engineering from Tsinghua University, Beijing, China, in 2007, and his M.S. degree in Electrical and Computer Engineering from University of California-San Diego (UCSD), La Jolla, in 2009. He is currently working toward the Ph.D. degree in the department of Electrical and Computer Engineering at the University of California-San Diego, La Jolla. Since fall of 2009, he was an intern with Bluetooth IC Design group, Broadcom Corp., San Diego, CA. His research interests include design and optimization of high-speed, low-power on-chip/off-chip interconnects and low-power clock distribution network design.

Nan-Chi Chou is an Engineering Director of timing analysis and physical synthesis in the Design Creation and Synthesis Division of Mentor Graphics. He received his M.S. in management science at National Taiwan University of Science and Technology and M.S. and Ph.D. degrees in computer science from University of California at San Diego. In addition to years of working experience at various EDA companies, Nan-Chi is also the co-founder of CLK Computer-Aided Designs, a startup specialized in ASIC/FPGA placement technologies, acquired by Mentor Graphics Corporation in 1998. Nan-Chi's current interests are in ASIC and FPGA physical timing analysis, optimization, and synthesis.

Evangeline Young received her B.Sc. degree and M.Phil. degree in Computer Science from The Chinese University of Hong Kong (CUHK). She received her Ph.D. degree from The University of Texas at Austin in 1999. Currently, she is an associate professor in the Department of Computer Science and Engineering in CUHK. Her research interests include algorithms and CAD of VLSI circuits. She is now working actively on floorplanning, placement, routing and algorithmic designs. Dr. Young has served on the technical program committees of several major conferences including ICCAD, ASP-DAC, ISPD and GLSVLSI, and also served on the editorial board of IEEE TCAD.

Chung-Kuan Cheng (S'82-M'84-SM'95-F'00) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley in 1984.

From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices Inc. In 1986, he joined the University of California, San Diego, where he is a Professor in the Computer Science and Engineering Department, an Adjunct Professor in the Electrical and Computer Engineering Department. He served as a chief scientist at Mentor Graphics in 1999. He was an associate editor of IEEE Transactions on Computer Aided Design for 1994-2003. He is a recipient of the best paper awards, IEEE Trans. on Computer-Aided Design in 1997, and in 2002, the NCR excellence in teaching award, School of Engineering, UCSD, 1991, IBM Faculty Awards in 2004, 2006, and 2007. He is appointed as an Honorary Guest Professor of Tsinghua University 2002-2008. His research interests include medical modeling and analysis, network optimization and design automation on microelectronic circuits.

Ronald Graham holds the Irwin and Joan Jacobs Endowed Chair in Computer and Information Science in University of California, San Diego, and is Chief Scientist of the California Institute for Telecommunications and Information Technology. He joined the UCSD faculty in 1999, after a 37-year career with AT&T. Graham received his Ph.D. in mathematics from U.C. Berkeley in 1962. From 1962-95, he was director of information sciences at (AT&T) Bell Labs, and from 1996-99 Chief Scientist of AT&T Labs. Graham has held visiting professorships at Rutgers, Princeton, Caltech, Stanford, UCLA, and U.C. Davis, and he holds five honorary doctorates. Graham is the Treasurer of the National Academy of Sciences, a Fellow of Amer. Academy of Arts and Sciences, a Fellow of the Association of Computing Machinery, and a past President of both the American Mathematical Society, and the Mathematical Association of America. He has won numerous awards in the field of mathematics, including the Polya Prize in Combinatorics and the Steele Prize for Lifetime Achievement awarded in 2003 by the American Mathematical Society.