A Multicycle Communication Architecture and Synthesis Flow for Global Interconnect Resource Sharing

Wei-Sheng Huang, Yu-Ru Hong, Juinn-Dar Huang, and Ya-Shih Huang
Department of Electronics Engineering
National Chiao Tung University, Hsinchu, Taiwan
{wshuang,sali}@adar.ee.nctu.edu.tw, yrhong.ee94g@nctu.edu.tw, jdhuang@mail.nctu.edu.tw

Abstract—In deep submicron technology, wire delay is no longer negligible and is gradually dominating the system latency. Some state-of-the-art architectural synthesis flows adopt the distributed register (DR) architecture to cope with this increasing latency. The DR architecture, though allows multicycle communication, introduces extra overhead on interconnect resource. In this paper, we propose the Regular Distributed Register - Global Resource Sharing (RDR-GRS) architecture to enable global sharing of interconnects and registers. Based on the RDR-GRS architecture, we further define the channel and register allocation problem as a path scheduling problem of data transfers. A formal and flexible formulation of this problem is then presented and optimally solved by Integer Linear Programming (ILP). Experimental results show that RDR-GRS/ILP can averagely reduce 58% wires and 35% registers compared to the previous work.

I. Introduction

With the scale evolution of fabrication technology, the delay of a long wire is no longer negligible due to RC delay, coupling effect, inductance, high operating frequency, etc. [1]. In architecture-level synthesis, the system cycle time is decided by the maximum sum of the execution time of the functional units (FUs) and the associated wire delays. If the synthesis flow simply overlooks the wire delays as before, the serious impacts due to long interconnects may be exposed after the physical floorplanning and potentially lead to worse performance because the timing closure is hard to achieve. To overcome this problem, some state-of-the-art synthesis flows perform preliminary floorplanning to obtain more accurate estimation of interconnect delays so that better synthesis outcomes can be expected [2, 3, 4].

In conventional high-level synthesis, the target architecture is assumed to contain a centralized register file (CR). Also, an FU is assumed to be able to access any register within a clock cycle. Under this paradigm, the increasingly long wire delay would drastically lengthen the cycle time. Hence, the distributed register (DR) architecture which partitions a whole system into several clusters is proposed [5, 6, 7]. Each cluster in the DR architecture contains FUs and local registers; an FU can only access the registers within the same cluster where it resides. The intra-cluster data transfers should be completed within a cycle, while the global data transfers, the inter-cluster transfers which go through global interconnects, are allowed to take multiple cycles. As a result, the new paradigm not only prevents the long wire delays from increasing the cycle time but also enables parallel computation and communication. That is, the FUs can perform computing when the global data is transferring.

Under the DR architecture, [5] first performs binding then places the functional units driven by inter-clock slack. With the initial placement, [6] applies the performance-driven scheduling with interconnect delay. Although these two works try to reduce the system latency in different aspects, they both rely on the interconnect delay information extracted from the preliminary placement. The rough result of coarse placement is usually far from the actual implementation obtained after floorplanning and routing. The inaccurate estimation of interconnect delay therefore significantly limit the final quality of synthesis.

To eliminate such inaccuracy, J. Cong et al. proposes the regular distributed register (RDR) architecture and the corresponding synthesis methodology named the architectural synthesis for multicycle communication (MCAS) [7]. The RDR architecture divides the entire chip into a 2-dimensional array of clusters (islands). Due to the highly regular layout, the interconnect delay between each cluster pair can be correctly calculated and recorded into a table. With this delay table, MCAS can perform resource allocation and binding, simulated annealing (SA)-based coarse placement with scheduling-based timing analysis followed by the post-layout scheduling with rebinding in a more accurate fashion.

Because the DR architecture introduces the multicycle communication, it may take more cycles to complete a data transfer than the CR one. Nevertheless, it can keep the cycle time significantly smaller than its counterpart when a large wire delay is present. Hence, the DR architecture can still beat the CR architecture in terms of overall system performance.

However, the DR architecture usually needs more registers and wires compared to the CR one. The reason is that the same data may be demanded by several clusters concurrently. Besides, extra registers are needed to pipeline or hold the data during multicycle communication. Furthermore, the data exchanges among clusters need dedicated wires which connect pairs of clusters in a point-to-point fashion. Thus the number of required wires is lower-bounded by the maximum number of concurrent data transfers. As a result, the DR architecture usually suffers from a significant overhead on interconnect resource, especially on the expensive global wires. It is reported that the DR architecture needs on average 100% more registers and 46% more global wires than the CR architecture [7]. Therefore, the issue of minimizing highly-demanded interconnect resource in the DR architecture should be seriously addressed.

Therefore, in this paper, we propose a new architecture, the Regular Distributed Register - Global Resource Sharing

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

(RDR-GRS) architecture, so that the registers and channels can be globally shared. We further define the channel and register allocation problem as a path scheduling problem of data transfers under the RDR-GRS architecture. Finally, we formulate and optimally solve the path scheduling problem by Integer Linear Programming (ILP). The experimental results demonstrate that significant resource can be saved by evolving the point-to-point interconnect model into the globally sharable model used in the proposed RDR-GRS architecture.

The rest of this paper is organized as follows. Section II gives the motivation by examples. Section III describes the channel and register allocation problem, which is formulated by ILP in Section IV. Section V gives the experimental results and Section VI concludes this paper.

II. Motivation

In this section, we focus on the differences among the RDR-based architectures and their associated resource allocation algorithms.

After operation scheduling, FU binding, and FU-cluster mapping, the global communication demands among clusters in the predefined architecture become obvious. But how to fulfill those communication demands varies in different architectures. In [7], RDR/MCAS assigns a dedicated wire connecting a pair of clusters every time a data transfer between them is initiated. The dedicated wire is exclusively used by a single transfer and is responsible for holding the data from the start of the transmission to the end, usually taking multiple cycles. Therefore, the number of required wires (and register pairs) in RDR/MCAS is the number of maximum possible concurrent data transfers in a cycle. Because of the low utilization of registers and wires, RDR/MCAS requires a massive amount of resources.

To alleviate this problem, an extension named RDR-pipe/ MCAS-pipe [8] is evolved. RDR-pipe chops a long wire into segments by inserting extra pipeline registers. Hence a data transfer no longer occupies the whole wire within a cycle but uses only a wire segment instead. In other words, at a single cycle, a long dedicated wire between two clusters can be shared by several data transfers as long as these transfers do not use the same segment. In addition, the data transfer scheduling algorithm exploits the possible slacks between the actual transfer latency and the arrival-to-deadline interval to reduce the required registers and wires.

Fig. 1 (a) and (b) show a scheduled and bound data flow graph (DFG) implemented by RDR/MCAS and RDR-pipe/MCAS-pipe, respectively. Each small circle with a number ID on it represents a register in the cluster. The large circles labeled as opi represent operations. On the left hand side of (a) and (b) is the DFG, and the right hand side is the architecture with placed operations in the clusters. As the figure shows, RDR/MCAS needs four wires to satisfy all data transfers, while RDR-pipe/MCAS-pipe needs only two wires with pipeline register insertion. However, the global communication in RDR-pipe/MCAS-pipe is still in a point-to-point fashion, and thus the resource sharing is limited to a local scope where a wire is sharable only for those data transfers with the identical source-destination cluster pair.

To further reduce the interconnect resource needed, enabling global sharing of all wires and pipeline registers among all data transfer paths is mandatory. Based on the RDR and RDR-pipe architectures, we propose a new architecture, RDR-GRS, in which a globally sharable interconnect model is innovated. RGR-GRS divides the chip into arrays of clusters; every cluster owns a register station. The neighborhood of a register station is defined as the four register stations in the clusters to its up, down, left and right, respectively. A register residing in a register station can serve as either an intermediate stop (i.e., a pipeline register) for global data transfers or an operand register for computation of local FUs. Then, channels consisting of wires connect every two neighboring register stations. A wire can transfer a datum from a register to its neighboring register station within a cycle. Hence a global data transfer can be broken into a series of neighboring register station-to-register station transfers.

Fig. 2 gives an example of how different architectures can impose different resource demands. Fig. 2 (a), (b) and (c) illustrate the implementation results of RDR/MCAS, RDR-pipe/ MCAS-pipe, and the proposed RDR-GRS architecture with a good resource allocation algorithm. According to the results, RDR-pipe saves 5 wire segments compared to RDR at the cost of adding one extra register as a pipeline register. It is a reasonable register-wire tradeoff since RDR-pipe enables local sharing of wires. However, RDR-GRS, which supports global sharing, outperforms the other two architectures by saving registers and wires simultaneously. This example clearly demonstrates the potential of reducing interconnect resource in the RDR-GRS architecture and the necessity of a good resource allocation algorithm to explore that potential. Therefore, we describe and formulate this allocation problem in the next section, and then present an optimal ILP solution.

III. Problem Description

The channel and register allocation problem is discussed in this section. Without losing generality, we assume that the input to be implemented on the RDR-GRS architecture is an application represented in the form of DFG.

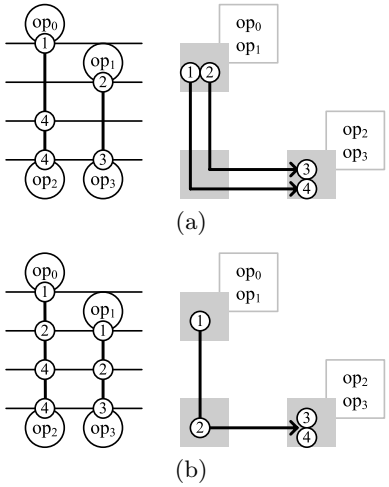

Fig. 1. Different interconnect architectures: (a) RDR/MCAS, (b) RDR-pipe/MCAS-pipe

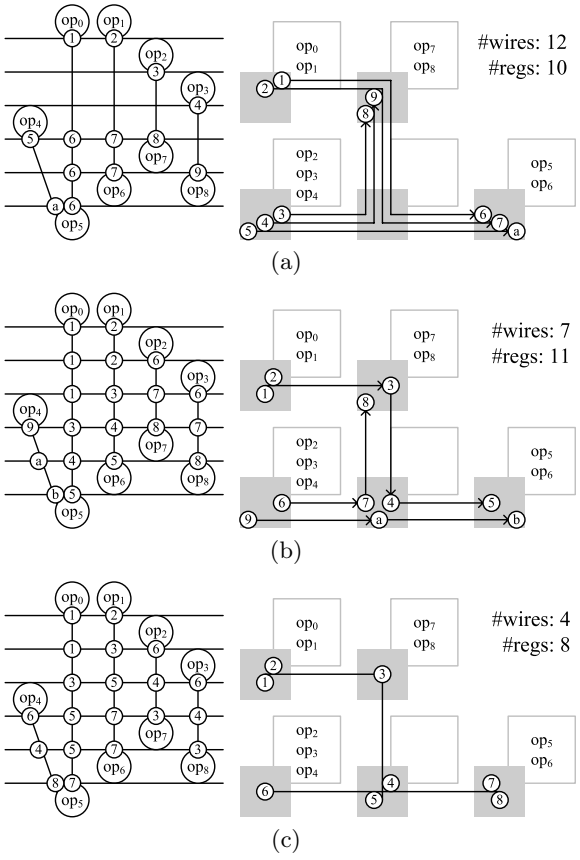

Fig. 2. Examples of different interconnect schemes: (a) RDR, (b) RDR-pipe, (c) RDR-GRS

Definition 1. **Data Flow Graph**
The data flow graph is a directed graph, $G_{DFG}(O_{ps}, E_{dep})$, in which O_{ps} is the set of operations and the directed edge set $E_{dep} \subseteq O_{ps} \times O_{ps}$ corresponds to the data dependency between operations. For an operation $op_i \in O_{ps}$, $d(op_i)$ represents the number of cycles required to execute op_i.

The RDR-GRS architecture is represented as a topology graph (TG). The TG describes the information about available logic clusters with register stations and the corresponding connectivity through channels.

Definition 2. **Topology Graph**
$G_{TOPO}(R_{st}, E_{ch})$, R_{st} is a set of vertices representing register stations and $E_{ch} \subseteq R_{st} \times R_{st}$ is the set of directed edge corresponding to the available channels between register stations. Additionally, $\forall i, m, n : rst_m,\ rst_n \in R_{st}$, $ch_i = (rst_m \rightarrow rst_n) \in E_{ch}$, $\bullet ch_i$ denotes for rst_m and $ch_i \bullet$ for rst_n.

Mapping is a task which schedules the operations and binds, places the FUs to the target architecture. It should also ensure the arrival-to-deadline interval of each data transfer is longer than the shortest possible travel time on the target architecture. After that, the information about where and when the operations are executed can be known.

Definition 3. **Feasible Mapping**
Given a DFG $G_{DFG}(O_{ps}, E_{dep})$ and a TG $G_{TOPO}(R_{st}, E_{ch})$, the mapping from $G_{DFG}(O_{ps}, E_{dep})$ to $G_{TOPO}(R_{st}, E_{ch})$ is represented as two functions $t : O_{ps} \rightarrow T$ and $p : O_{ps} \rightarrow R_{st}$. $\forall i : op_i \in O_{ps},\ \exists j, k : t_j \in T,\ rst_k \in R_{st}$ such that $t_j = t(op_i)$ and $rst_k = p(op_i)$, which describes the operation op_i is executed in a cluster with rst_k at cycle t_j. The mapping is feasible if $\forall i, m, n : e_i = (op_m \rightarrow op_n) \in E_{dep}$, the constraint $t(op_n) \geq t(op_m) + d(op_m) + sp(p(op_m) \rightarrow p(op_n))$ is satisfied. Here the data transfer between two neighboring register stations is assumed to take one cycle. The function $sp(rst_i \rightarrow rst_j)$ represents the minimum possible cycles required for a transfer from rst_i to rst_j.

The feasible mapping can be obtained by certain synthesis algorithm such as MCAS. For the rest of the paper, we assume the initial input contains a DFG, a TG and a feasible mapping. The goal of our work is to schedule the data transfer paths (when and through what channels) to achieve an optimal solution in terms of minimum number of required wires and registers.

Definition 4. **Data Transfer Set**
The data transfer set T_r includes all required data transfers. With the given TG $G_{TOPO}(R_{st}, E_{ch})$, there exist four functions: $gen_t : T_r \rightarrow T$, $req_t : T_r \rightarrow T$, $gen_r : T_r \rightarrow R_{st}$ and $req_r : T_r \rightarrow R_{st}$. $\forall i : tr_i \in T_r$, $\exists m, n, x, y : rst_m, rst_n \in R_{st}$, $t_x, t_y \in T$, such that $t_x = gen_t(tr_i)$, $t_y = req_t(tr_i)$, $rst_m = gen_r(tr_i)$ and $rst_n = req_r(tr_i)$. That is, t_x, t_y, rst_m and rst_n represents the generated, required cycles and the generating, requiring register stations of tr_i, respectively.

With the given DFG $G_{DFG}(O_{ps}, E_{dep})$ and TG $G_{TOPO}(R_{st}, E_{ch})$, for every edge $e_i = (op_m \rightarrow op_n) \in E_{dep}$, there exists the corresponding $tr_i \in T_r$ such that

$$(gen_t(tr_i), req_t(tr_i), gen_r(tr_i), req_r(tr_i)) = (t(op_m) + d(op_m), t(op_n), p(op_m), p(op_n))$$

Definition 5. **Transfer Path Scheduling**
Given a TG $G_{TOPO}(R_{st}, E_{ch})$, the data transfer set T_r, and $tr_i \in T_r$, a transfer path scheduling $\sigma(tr_i)$ is to select a series of channels to complete the transfer tr_i. That is, $\forall i : tr_i \in T_r$, $\exists ch_{i,gen_t(tr_i)}, ch_{i,gen_t(tr_i)+1}, \ldots, ch_{i,req_t(tr_i)-1} \in E_{ch}$, such that $\sigma(tr_i) = (ch_{i,gen_t(tr_i)}, ch_{i,gen_t(tr_i)+1}, \ldots, ch_{i,req_t(tr_i)-1})$ and the following constraints are satisfied:

- $\bullet ch_{i,gen_t(tr_i)} = gen_r(tr_i)$ and $ch_{i,req_t(tr_i)-1} \bullet = req_r(tr_i)$
- $ch_{i,j} \bullet = \bullet ch_{i,j+1}$, $\forall j : req_t(tr_i) - 1 > j \geq gen_t(tr_i)$

$\sigma(tr_i)$ indicates the transferred data use the channel $ch_{i,gen_t(tr_i)}$ at cycle $gen_t(tr_i)$, then the channel $ch_{i,gen_t(tr_i)+1}$ at cycle $gen_t(tr_i) + 1$, and finally the channel $ch_{i,req_t(tr_i)-1}$ at cycle $req_t(tr_i) - 1$.

Given a TG $G_{TOPO}(R_{st}, E_{ch})$ and a data transfer set T_r, the channel and register allocation problem can be interpreted as finding a set of transfer path scheduling $\sigma's$ that can minimize the requirements of registers and wires.

IV. Problem Formulation

In this section, we formally model the channel and register allocation problem using integer linear programming (ILP).

A. Variables in Formulation

All the variables are classified into two types: the allocation variables and the resource variables. The allocation variables are used to describe the transfer behaviors, while the resource variables indicate the quantity of required registers (wires) in a register station (channel).

Definition 6. **Allocation Variables**
Given a transferred data set T_r and a TG $G_{TOPO}\,(R_{st}, E_{ch})$, $\forall i,\, j,\, k : tr_i \in T_r,\, t_j \in T,\, ch_k \in E_{ch}$, an allocation variable, denoted as $x_{i,j,k}$, has a 1-0 value which indicates whether the data transfer tr_i uses the channel ch_k at cycle t_j or not.

Note that not all combinations of index i, j and k of allocation variable $x_{i,j,k}$ are necessary when considering temporal and spatial feasibility. At any cycle, a datum is only allowed to transfer to a neighboring cluster using an available channel connecting the current and that neighboring cluster, i.e., it is impossible for a datum to reach the channels that are not connecting to the current register station.

At the beginning, for any transfer tr_i at cycle $t_j = gen_t(tr_i)$, the feasible channels within $X_{i,j}$ are those whose source is $gen_r\,(tr_i)$. At the next cycle, for continuity, the channels within $X_{i,j+1}$ should be those whose sources belong to the union of destinations of the channels within $X_{i,j}$. The same also applies to the following cycles. Therefore, the *forward feasible channel set* at any cycle can be recursively derived from the set at the previous cycle. For tr_i, the set $CS_{i,j}$ is the forward feasible channel set of tr_i at cycle t_j and is recursively defined as:

$$CS_{i,j} = \begin{cases} \forall j : req_t\,(tr_i) > j > gen_t\,(tr_i)\,, \\ \left\{ch_k |\, \bullet ch_k = \bigcup_{ch_n \in CS_{i,j-1}} ch_n \bullet, ch_k \in E_{ch}\right\} \\ \\ j = gen_t\,(tr_i)\,, \\ \{ch_k |\, \bullet ch_k = rst_{gen_r(tr_i)}, ch_k \in E_{ch}\} \end{cases}$$

Similarly, a transfer can be backtraced from the destination register station to define the *backward feasible channel set* at each cycle. For any transfer tr_i at cycle $t_j = req_t\,(tr_i) - 1$, the feasible channels within $X_{i,j}$ are those whose destination is $req_r\,(tr_i)$. At the previous cycle, for continuity, the channels within $X_{i,j-1}$ should be those whose destinations belong to the union of sources of the channels within $X_{i,j}$. The same also applies to the previous cycles. Therefore, the backward feasible channel set at any cycle can be recursively derived from the set at the next cycle. For tr_i, the set $CD_{i,j}$ is the backward feasible channel set of tr_i at cycle t_j and is recursively defined as:

$$CD_{i,j} = \begin{cases} \forall j : req_t\,(tr_i) - 1 > j \geq gen_t\,(tr_i)\,, \\ \left\{ch_k | ch_k \bullet = \bigcup_{ch_n \in CF_{i,j+1}} \bullet ch_n, ch_k \in E_{ch}\right\} \\ \\ j = req_t\,(tr_i) - 1, \\ \{ch_k | ch_k \bullet = rst_{req_r(tr_i)}, ch_k \in E_{ch}\} \end{cases}$$

For a given transfer and a given cycle, the *feasible channel set* is the intersection of the forward and backward feasible channel sets. As a result, $\forall i, j, k : tr_i \in T_r,\ req_t(tr_i) > j \geq gen_t(tr_i)$ and $ch_k \in E_{ch}$, $\{x_{i,j,k} | ch_k \in CF_{i,j} \cap CD_{i,j}\}$ is the set which includes all allocation variables required in the ILP formulation. Obviously, the number of required variables can be greatly reduced.

Definition 7. **Resource Variables**
A resource variable represents the quantity of a channel or register station. $\forall i : rst_i \in R_{st}$, qr_i is the resource variable which represents the number of registers allocated in register station rst_i; $\forall i : ch_i \in E_{ch}$, qw_i is the resource variable which represents the number of wires allocated in channel ch_i.

A simple example of a mapped DFG is given in Fig. 3. The target RDR-GRS architecture contains a 2×2 cluster array in which each register station is connected to its neighbors. A self-loop channel is also present to conceptually represent that the transfer is held in the register station for a cycle if the channel is taken; but there is no need to implement those loops physically. In this example, there are 4 register stations and 12 channels (self-loops are included). The resultant forward and backward feasible channel set, $CS_{i,j}$, $CD_{i,j}$, as well as the intersection of them are shown in TABLE I. Then the variables actually used in ILP are shown below:

Allocation variables:
$x_{7,6,1}$, $x_{7,6,3}$, $x_{7,6,6}$, $x_{7,7,0}$, $x_{7,7,3}$, $x_{7,7,4}$,
$x_{7,7,6}$, $x_{7,7,9}$, $x_{7,7,11}$, $x_{7,8,4}$, $x_{7,8,9}$, $x_{7,8,10}$

Resource variables of register stations:
qr_0, qr_1, qr_2, qr_3

Resource variables of channels:
qw_0, qw_1, qw_2, qw_3, qw_4, qw_5,
qw_6, qw_7, qw_8, qw_9, qw_{10}, qw_{11}

B. Objective Function

The goal of minimizing the required resources (registers and wires) can be formulated as the minimization on the

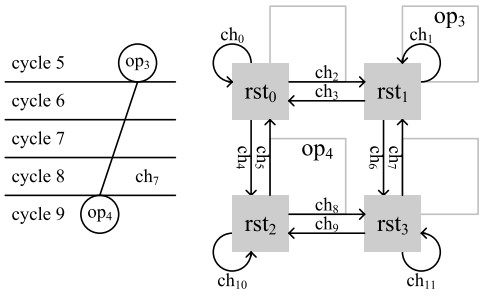

Fig. 3. An example of ILP formulation

TABLE I
EXAMPLES OF FEASIBLE CHANNEL SETS

j	$CS_{7,j}$	$CD_{7,j}$	$CS_{7,j} \cap CD_{7,j}$
6	ch_1, ch_3, ch_6	$ch_0 \sim ch_{11}$	ch_1, ch_3, ch_6
7	ch_0,ch_1,ch_2, ch_3,ch_4,ch_6, ch_7,ch_9,ch_{11}	ch_0,ch_3,ch_4, ch_5,ch_6,ch_8, ch_9,ch_{10},ch_{11}	ch_0,ch_3,ch_4, ch_6,ch_9,ch_{11}
8	$ch_0 \sim ch_{11}$	ch_4, ch_9, ch_{10}	ch_4, ch_9, ch_{10}

value of the following objective function:

$$\sum_{\forall i: rst_i \in R_{st}} \alpha_i qr_i + \sum_{\forall j: ch_j \in E_{ch}} \beta_j qw_j$$

where α and β are the tunable weighting factors. The former summation is the weighted number of used registers, and the latter is the weighted number of used wires.

C. Constraints in ILP Formulation

There exist three types of constraints in our ILP formulation: uniqueness, continuity and resource constraints.

The uniqueness constraint is imposed to ensure that a data transfer can only occupy a single channel at any cycle. That is, in each $X_{i,j}$, there is one and only one allocation variable that is set to 1. This constraint can be written as:

$$\forall i, j : tr_i \in T_r, req_t\left(tr_i\right) > j \geq gen_t\left(tr_i\right),$$

$$\sum_{\forall k: ch_k \in X_{i,j}} x_{i,j,k} = 1$$

The continuity constraint is to ensure the transfer path is continuous in both time and space domains. That is, for a data transfer tr_i, the destination of the adopted channel at cycle t_j must be the source of the adopted channel at cycle t_{j+1}. This constraint can be written as:

$$\forall i, j, k : tr_i \in T_r, req_t\left(tr_i\right) - 1 > j \geq gen_t\left(tr_i\right), ch_k \in X_{i,j},$$

$$-x_{i,j,k} + \sum_{\substack{\forall k': ch_{k'} \in X_{i,j+1}, \\ \bullet ch_{k'} = ch_k \bullet}} x_{i,j+1,k'} \geq 0$$

There are two types of resource constraints. The first one is to make sure the interconnect resource is sufficiently allocated in each station, i.e., the number of allocated registers must be larger than the number of incoming data transfers at every cycle. This constraint can be written as:

$$\forall y, j : rst_y \in R_{st}, t_j \in T,$$

$$qr_y - \sum_{\substack{\forall i: req_t(tr_i) > j, \\ j \geq gen_t(tr_i)}} \sum_{\substack{\forall k: ch_k \in X_{i,j}, \\ ch_k \bullet = rst_y}} x_{i,j,k} \geq 0$$

The first summation in the formula finds the data transfers that are active at cycle t_j, and the second summation finds the channels whose destination is rst_y. Hence the second term in the formula indicates how many transfers actually send the data to rst_y at cycle t_j. The inequality must be satisfied to ensure the number of allocated registers qr_y can always accommodate the incoming data transfers.

Similarly, the second type of resource constraint ensures that the number of allocated wires is larger than the number of transfers using this channel at every cycle. This constraint can be written as:

$$\forall y, j : ch_y \in E_{ch}, t_j \in T$$

$$qw_y - \sum_{\substack{\forall i: req_t(tr_i) > j, \\ j \geq gen_t(tr_i)}} \sum_{\forall k: ch_k \in X_{i,j}} x_{i,j,k} \geq 0$$

Again, the second term in this formula indicates how many transfers actually use the channel ch_y at cycle t_j. The number of wires in this channel qw_y must be larger than the number of demanded wires all the time.

Using the same example in Fig. 3, the set of all constraints is shown below:

Uniqueness constraints:

$x_{7,6,1} + x_{7,6,3} + x_{7,6,6} = 1;$
$x_{7,7,0} + x_{7,7,3} + x_{7,7,4} + x_{7,7,6} + x_{7,7,9} + x_{7,7,11} = 1;$
$x_{7,8,4} + x_{7,8,9} + x_{7,8,10} = 1;$

Continuity constraints:

$-x_{7,6,1} + x_{7,7,3} + x_{7,7,6} \geq 0;$
$-x_{7,6,3} + x_{7,7,0} + x_{7,7,4} \geq 0;$
$-x_{7,6,6} + x_{7,7,9} + x_{7,7,11} \geq 0;$
$-x_{7,7,0} + x_{7,8,4} \geq 0;$ $-x_{7,7,3} + x_{7,8,4} \geq 0;$
$-x_{7,7,4} + x_{7,8,10} \geq 0;$ $-x_{7,7,6} + x_{7,8,9} \geq 0;$
$-x_{7,7,9} + x_{7,8,10} \geq 0;$ $-x_{7,7,11} + x_{7,8,9} \geq 0;$

Resource constraints of register stations:

$qr_0 - x_{7,6,3} \geq 0;$ $qr_0 - x_{7,7,0} - x_{7,7,3} \geq 0;$
$qr_1 - x_{7,6,1} \geq 0;$ $qr_2 - x_{7,7,4} - x_{7,7,9} \geq 0;$
$qr_2 - x_{7,8,4} - x_{7,8,9} - x_{7,8,10} \geq 0;$
$qr_3 - x_{7,6,6} \geq 0;$ $qr_3 - x_{7,7,6} - x_{7,7,11} \geq 0;$

Resource constraints of channels:

$qw_0 - x_{7,7,0} \geq 0;$ $qw_1 - x_{7,6,1} \geq 0;$
$qw_3 - x_{7,6,3} \geq 0;$ $qw_3 - x_{7,7,3} \geq 0;$
$qw_4 - x_{7,7,4} \geq 0;$ $qw_4 - x_{7,8,4} \geq 0;$
$qw_6 - x_{7,6,6} \geq 0;$ $qw_6 - x_{7,7,6} \geq 0;$
$qw_9 - x_{7,7,9} \geq 0;$ $qw_9 - x_{7,8,9} \geq 0;$
$qw_{10} - x_{7,8,10} \geq 0;$ $qw_{11} - x_{7,7,11} \geq 0;$

V. Experiments

A. Environment Setup

We have implemented the GRS-ILP synthesis flow in C++ /Linux environment on a workstation with a Xeon 3.2GHz CPU and 2GB RAM. The initial inputs to our flow are the data transfer set and the topology graph. To obtain the inputs, seven pure data flow graphs are first extracted from mpeg2enc, jpeg and rasta of MediaBench [9]. These data flow graphs are generated through the SUIF compiler infrastructure [10] and the Machine SUIF [11]. Then, to map a data flow graph into the RDR-GRS architecture, a basic high-level synthesis algorithm, which performs force-directed scheduling, approximate max-clique based binding, SA-based FU placement in order, is applied.

Throughout the experiments, the specification of the RDR-GRS architecture is defined as a 3×3 cluster array in which each register station is connected to its neighbors and also itself (i.e., 9 register stations and 33 channels in total). The objective function is set as:

$$\sum_{\forall i: rst_i \in R_{st}} 1 \times qr_i + \sum_{\substack{\forall j: ch_j \in E_{ch}, \\ \bullet ch_j \neq ch_j \bullet}} 5 \times qw_j$$

The weighting factor for wires is set higher than that for registers because global wires are scarcer than registers. The adopted ILP solver is lpsolve 5.5.0.0 [12].

We have also implemented two alternative schemes for comparison. Method1 uses dedicated non-pipelined interconnects as RDR/MCAS; Method2 uses a pipelined point-to-pint interconnects as RDR-pipe/MCAS-pipe.

B. Experimental Results

TABLE II gives the information of input DFGs. The number of operations is listed in Column #node. The numbers of allocated ALUs and multipliers are reported in Column FU resource after initial time-constrained scheduling. Finally, the total elapsed cycle count is obtained and listed in the table after FU placement followed by rescheduling to best fit the RDR architecture.

The allocation results of Method1, Method2 and our GRS-ILP approach are shown in TABLE III. For each method, the numbers of allocated registers and wires are reported in two separate columns. For GRS-ILP, the runtime is also given. As expected, Method1 requires the most wires. This implies RDR/MCAS could impose a serious burden on physical routing. Method2, though on average reduces 20% of wires, uses 42% more registers than Method1. Notice that the amount of pipeline registers in Method2 increases with the size of cluster array and usually cannot be eased because only a very limited set of data transfers (depending heavily on the input) can actually share registers. Whether the tradeoff between pipeline registers and wires is worthwhile might be in doubt under different conditions. Meanwhile, GRS-ILP delivers the best results in both aspects, the number of registers and wires. GRS-ILP demands 58% (35%) less wires and 47% (55%) less registers compared to Method1 (Method2). The experimental results clearly demonstrate the superiority of the proposed architecture and synthesis flow over the previous two. We attribute this remarkable improvement to two indispensable keys, RDR-GRS and GRS-ILP. It is impossible to drastically minimize the number of registers and wires without the support of RDR-GRS, which breaks the limited local sharing scope and provide a globally sharable platform for interconnect resources. On top of RDR-GRS, GRS-ILP guarantees to produce optimal synthesis results such that the underlying architecture can be fully exploited.

VI. Conclusions

A new architecture, RDR-GRS, is proposed in this paper to enable global interconnect resource sharing for multicycle communication. Under the new architecture, the issue of data transfer path scheduling is addressed. The channel and register allocation problem is then defined and modeled in an ILP formulation. The experimental results show that the GRS-ILP synthesis flow can obtain the optimal solution with averagely 35% reduction in registers and 58% reduction in wires compared to the previous work (Method1). In summary, the RDR-GRS architecture with GRS-ILP synthesis flow is capable of providing better synthesis outcomes than the previous RDR-based architectures.

Though the proposed ILP formulation is capable of providing the desirable optimality, it may run out of steam as designs are getting larger. Therefore, we are currently concentrating on developing an efficient heuristic resource allocation algorithm to fully exploit the RDR-GRS architecture for large-scale designs.

TABLE II
Information of input DFGs

	#node	FU resource		cycle count
		#ALU	#MUL	
mpeg2enc (1)	66	5	2	23
mpeg2enc (2)	101	9	4	23
mpeg2enc (3)	196	18	8	18
jpeg (1)	93	9	2	35
jpeg (2)	109	9	2	33
jpeg (3)	140	11	6	23
rasta	119	7	5	33

TABLE III
Results of resource allocation

	Method1		Method2		GRS-ILP		
	#wire	#reg	#wire	#reg	#wire	#reg	runtime (sec)
mpeg2enc(1)	42	28	37	45	25	25	0.2
mpeg2enc(2)	76	53	60	76	42	38	102.6
mpeg2enc(3)	130	92	109	133	68	61	6.3
jpeg(1)	81	50	64	78	24	28	8.9
jpeg(2)	78	44	59	68	28	28	0.9
jpeg(3)	75	72	58	87	24	48	235.5
rasta	74	56	57	75	25	27	340.1
average	79.4	56.4	63.4	80.3	33.7	36.4	
Norm. to Method1	1	1	0.80	1.42	0.42	0.65	
	1.25	0.70	1	1	0.53	0.45	

References

[1] International Technology Roadmap for Semiconductor. Semiconductor Industry Association, 1999.

[2] Y. Mori, V. Moshnyaga, H. Onodera, and K. Tamaru, "A performance-driven macro-block placer for architectural evaluation of ASIC designs," *in Proc. Annual IEEE International ASIC Conference and Exhibit*, pp. 233–236, Sep. 1995.

[3] V. Moshnyaga and K. Tamaru, "A placement driven methodology for high-level synthesis of sub-micron ASIC's," *in Proc. International Symposium on Circuits and Systems*, vol. 4, pp. 572–575, May 1996.

[4] P. Prabhakaran and P. Banerjee, "Parallel algorithms for simultaneous scheduling, binding and floorplanning in high-level synthesis," *in Proc. International Symposium on Circuits and Systems*, vol. 6, pp. 372–376, May 1998.

[5] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," *in Proc. International Conference on Computer Aided Design*, pp. 320–325, Nov. 2001.

[6] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," *in Proc. Asia and South Pacific Design Automation Conference*, pp. 662–667, Jan. 2001.

[7] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multicycle communication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, Apr. 2004.

[8] J. Cong, Y. Fan, and Z. Zhang, "Architecture-level synthesis for automatic interconnect pipelining," *in Proc. Design Automation Conference*, pp. 602–607, Jun. 2004.

[9] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," *in Proc. IEEE/ACM International Symposium on Microarchitecture*, pp. 330–335, Dec. 1997.

[10] SUIF 2 Compiler System. [Online]. Available: http://suif.stanford.edu/suif/suif2/

[11] M. Smith and G. Holloway, "An introduction to machine suif and its portable libraries for analysis and optimization," *in Division of Engineering and Applied Sciences, Harvard University*, 2002.

[12] lp solve open source LP and MILP solver. [Online]. Available: http://groups.yahoo.com/group/lp_solve/