Circuit Partitioning for Dynamically Reconfigurable FPGAs*

Huiqun Liu and D. F. Wong
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712
Email: {hqliu, wong}@cs.utexas.edu

Abstract

Dynamically reconfigurable FPGAs have the potential to dramatically improve logic density by time-sharing a physical FPGA device. This paper presents a network-flow based partitioning algorithm for dynamically reconfigurable FPGAs based on the architecture in [2]. Experiments show that our approach outperforms the enhanced force-directed scheduling method in [2] in terms of communication cost.

1 Introduction

One of the major benefits provided by FPGAs is the ability of run-time reconfiguration. Currently there is a growing interest in dynamically reconfigurable FPGAs (DRFPGA), which have the potential to dramatically improve logic density by time-sharing logic.

Several different architectures have been proposed for dynamically reconfigurable FPGAs, such as Xilinx time-multiplexed FPGA configuration model [1], dynamically reconfigurable FPGA [2], Dharma [6], the Dynamically Programmable Gate Array [7,8] and the Virtual Element Gate Array [9]. These dynamically reconfigurable FPGAs allow the dynamically reuse of the logic blocks and wire segments by having more than one on-chip SRAM bits controlling them. Each on-chip configuration is called a *context*, and a device with more than one context is a multi-context device. A large logic design is partitioned into multiple stages to share the same physical device in a time-multiplexed fashion. This is analogous to the virtual memory system where a program can be larger than the actual size of the physical memory, a dynamically reconfigurable FPGA allows a virtually large logic design to be implemented on a smaller physical device.

For a dynamically reconfigurable FPGA, a circuit is partitioned into k stages (or partitions), such that the logic in different stages temporally share the same physical FPGA device (Figure 1). Each stage is called a *micro-cycle* and the k micro-cycles form one *user cycle*. Between the micro-cycles, the logic blocks and interconnect in the FPGA are reconfigured by a different context. One user cycle should produce the same results on the outputs as would be seen by a non-time-multiplexed device.

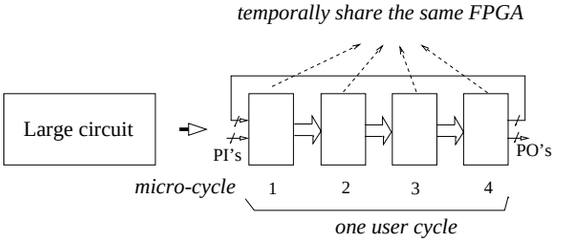

Figure 1: *Temporal partitioning of a circuit for a dynamically reconfigurable FPGA*

The nodes (*i.e* LUTs) in a physical FPGA are called the *real nodes*, while the nodes in any stage (micro-cycle) of the partitioning solution are called the *virtual nodes*. To fit into a physical device, the number of virtual nodes in any stage should be less than or equal to the number of real nodes.

Because the logic blocks and interconnect needed for a circuit is time-multiplexed on a DRFPGA, it is necessary to have a good partitioning strategy to ensure the correctness of the execution, as well as satisfy both the area and pin constraints for a physical FPGA device. It is also crucial to minimize the number of interconnections in order to reduce the overhead for the placement and routing process.

The partitioning problem for dynamically reconfigurable FPGAs was studied in [1, 2, 3, 16]. The traditional directed-acyclic-graph (DAG) scheduling methods are applied, such as list scheduling [1] and force-directed scheduling [2, 3]. Recently [16] proposed a network-flow based approach for multi-way precedence constrained partitioning based on the Xilinx time-multiplexed FPGA architecture [1], and [16] achieved better result than the list scheduling heuristic in [1] in terms of minimizing the communication cost. However, the network-flow based approach in [16] can not be used to solve the partitioning problem for [2], since the architecture in [2] is different from that of [1] and imposes different constraints on the partitioning problem.

In this paper, we focus on partitioning a large logic design into dynamically reconfigurable FPGAs based on the archi-

*This work was partially supported by the Texas Advanced Research Program under Grant No. 003658288.

tecture proposed in [2]. We present a network-flow based approach for multi-way partitioning. We show how to correctly model the nets in both combinational and sequential circuits, so that by the max-flow computation, the min-cut corresponds to the number of communication required. An α-bounded bipartitioning algorithm is presented and then it is iteratively applied to partition a netlist into multiple stages, so that each stage can temporally share the same FPGA device. Experimental results show that our approach outperforms the enhanced force-directed scheduling in [2] in terms of communication cost.

The organization of the paper is as follows. In section 2, we give a brief summary of the time-multiplexed communicating logic model proposed in [2]. In section 3, we introduce the problem formulation of the partitioning for dynamically reconfigurable FPGAs. In section 4 we first present the net modeling method for both combinational and sequential circuits, then present a network-flow based approach for bipartitioning. Section 5 explains the multi-way partitioning algorithm for dynamically reconfigurable FPGAs. Experimental results are discussed in Section 6.

2 Model of Dynamically Reconfigurable FPGA

For dynamically reconfigurable FPGAs, the communication cost, which is the storage needed for buffering a signal from the time it is created to the time it is no longer needed, creates a considerable overhead. Different architectures have been proposed for storing the communication values among the micro-cycles [1, 2], and they impose different constraints on the partitioning problem. For Xilinx's architecture [1], the signal from a cut net is stored in on-chip micro-registers, and [1] requires that the precedence constraints be satisfied in order to guarantee the correctness.

In this paper, we especially examine the partitioning problem for the dynamically reconfigurable FPGA architecture proposed in [2].

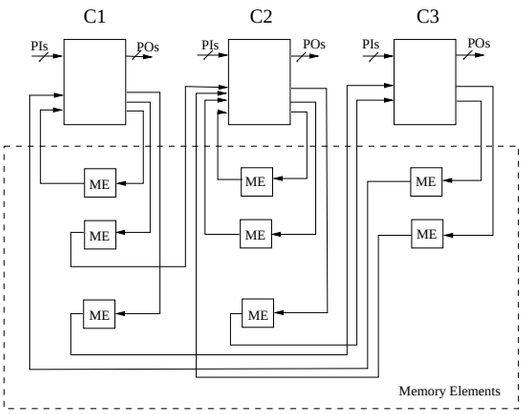

Figure 2: *Model of time-multiplexed communicating logic.*

[2] presented a gate-level model for DRFPGA computation called the time-multiplexed communicating logic (TMCL) model (Figure 2). This model consists of two parts. First, there is a finite set of combinational logic units (CLUs) $\{C_1, C_2, ..., C_k\}$, where each C_i contains a set of logic blocks (*e.g.* LUTs). Secondly, there is a finite set of memory elements (MEs) $\{M_1, M_2, ..., M_m\}$, which can be used to store values for communication between the CLUs. A circuit is partitioned into $(\{C_1, ..., C_k\}, \{M_1, ..., M_m\})$, with the execution sequence being $C_1, ..., C_k$. Each C_i is a subcircuit to be executed at a different micro-cycle on the DRFPGA. Each C_i plus the MEs needed for C_i will be called *context* i since it corresponds to the i-th context on a DRFPGA. The MEs needed for C_i are those that are pseudo primary inputs or pseudo primary outputs of C_i.

One clock cycle of a context i proceeds as follows. First, read the needed pseudo primary input signals from the memory for C_i, and read the primary input signals from input pads. Second, propagate signal values through C_i. Third, latch primary output signal into output pads, and store the pseudo primary output signals for C_i into memory.

In this model, a circuit is represented by $G = (V, N_c, N_f)$. Each node $v \in V$ is a gate. The nets are classified into two types, N_c and N_f. A net $n = \{v_1, ..., v_p\} \in N_c$ if v_1 is the input to the other nodes in this net. A net $n = \{v_1, ..., v_p\} \in N_f$ if there is a flip-flop (FF) between v_1 and the rest of the nodes in net n, *i.e.* v_1 is the input to an FF and the FF is the input to the other nodes $v_2, ..., v_p$. Figure 3 shows a part of a sequential circuit and its conversion to a net $\{v_1, v_2, v_3\}$ in N_f. If there are adjacent FFs in the circuit, then dummy gates can be added between adjacent FFs.

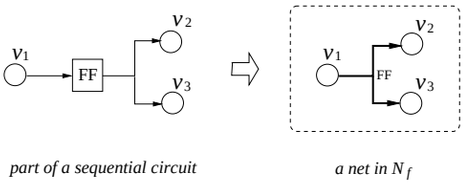

Figure 3: *Example of a net in N_f.*

We define $s(v)$ to be the stage to which a node v is assigned in the partitioning solution.

In a combinational circuit or the combinational part of a sequential circuit, the nodes in a net in N_c must follow the *precedence constraints*, such that if node v is the input of u, then v must be scheduled in a stage no later than u, *i.e.* $s(v) \leq s(u)$.

If a net $n = (v_1, ..., v_p) \in N_c$ is cut, such that $\exists v_j \in n$, $s(v_1) < s(v_j)$, then a ME (memory element) is used to store the value between v_1 and v_j (Figure 4). The output of v_1 will be stored in the ME and be read by v_j in a later micro-cycle within the current user clock cycle. Thus this ME is used for communication within a user cycle.

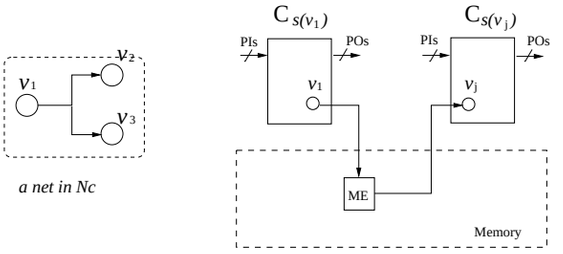

Figure 4: *For a net in N_c, if $s(v_1) < s(v_j)$, one memory element is used to store the communication value for a combinational net.*

For sequential circuit, memory elements are used for

passing values to different micro-cycles of the same user cycle or to the next user cycle. The nodes in a net in N_f can be in any order, but the different ordering will result in different number of memory elements required. There are the following two cases:

First, for a net $n = (v_1, ..., v_p) \in N_f$, if $s(v_1) \geq s(v_j)$ $(v_1, v_j \in n)$, then a ME is inserted between v_1 and v_j (Figure 5). The signal from v_1 is stored in the ME and will not be used until the next user cycle. Thus this ME is used for communication between user clock cycles.

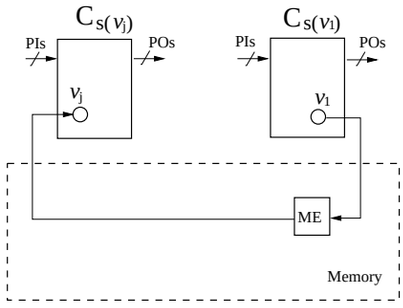

Figure 5: *If $s(v_1) \geq s(v_j)$, then one memory element is used to store the communication value.*

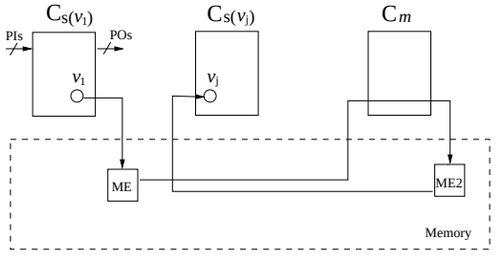

Figure 6: *If $s(v_1) < s(v_j)$, then two memory elements are used to store the communication value.*

Secondly, for a net $n \in N_f$, if $s(v_1) < s(v_j)$ $(v_1, v_j \in n)$, two MEs must be inserted between v_1 and v_j (Figure 6). This is called a ME2 situation. Two MEs are needed because the first ME acts as storage for communication within the current user clock cycle. A second ME (labeled as ME2 in Figure 6) is needed to store the value between user clock cycles. The ME2 can be in any stage later than $s(v_j)$.

Here we further improved the model in [2]. While [2] assumed each net is a two-terminal net, we consider the more general case where a net can be both two-terminal and multi-terminal net.

This TMCL model is different from the Xilinx model [1]. In [1], no memory elements are needed to store the communication values, and micro-registers are used to save values to be passed to a later micro-cycle or the next user-cycle. Each cut net, including combinational and sequential net, must be uni-directional to satisfy the precedence constraints in order to guarantee the correct execution. However, in the model of [2], the sequential nets do not need to be uni-directional, but the different ordering of the nodes will cause different communication cost. Therefore, the different architectures impose different constraints on the partitioning process.

3 Problem Statement

A circuit can be represented by a hypergraph $G = (V, N)$, where V is a set of nodes, N is a set of nets where each net is a subset of nodes which are interconnected, and $N = N_c \cup N_f$. Each node v in V has an area $w(v)$, and the area of a subset X of V, denoted by $w(X)$, is the total area of all the nodes in X. For a net $n = \{v_1, ..., v_p\}$ with p nodes, let v_1 be the *input* to v_j $(2 \leq j \leq p)$, and v_j $(2 \leq j \leq p)$ be the *output* of v_1. If a net only connects two nodes (*i.e.* $p = 2$), then it is a *two-terminal* net, if it connects more than two nodes (*i.e.* $p > 2$), then it is a *multi-terminal* net.

Based on the TMCL model, the partitioning problem for dynamically reconfigurable FPGAs is to partition a circuit $G = (V, N)$ into k non-overlapping subsets $V_1, V_2, ..., V_k$, subject to:

1. $V = \cup_{i=1}^{k} V_i$;
2. Precedence constraints, *i.e.* for a net $n = \{v_1, ..., v_p\} \in N_c$, $s(v_1) \leq s(v_j)$ for $2 \leq j \leq p$;
3. Timing constraint: the number of levels of nodes in any stage is less than D.

The objective is to:

1. minimizing the maximum communication cost for any stage;
2. minimizing the maximum area of any stage, *i.e.* minimizing $max\{w(V_i) | 1 \leq i \leq k\}$;

The precedence constraints guarantee the correctness of the execution, and the timing constraint allows the design to run as fast as possible.

The communication cost for a stage is the total number of memory elements to be used by this stage. The communication cost c_n of a net n is measured as follows.

For a net $n = \{v_1, ..., v_p\} \in N_c$, if it is cut such that $\exists\, v_j$, $s(v_1) < s(v_j)$, then the communication cost is 1; otherwise, if all the nodes in net n are in the same stage, then the communication cost is 0. Notice that the precedence constraints require that $s(v_1) \leq s(v_j)$, $v_1, v_j \in n$.

$$c_n = \begin{cases} 1, & \text{if } \exists v_j \text{ such that } s(v_1) < s(v_j) \\ 0, & \text{if all the nodes are in the same stage} \end{cases}$$

For a net $n \in N_f$, the different ordering of the nodes will result in different communication cost, as discussed in Section 2.

$$c_n = \begin{cases} 1, & \text{if } s(v_1) \geq s(v_j) \\ 2, & \text{if } s(v_1) < s(v_j) \end{cases}$$

For k-way partitioning, it is desirable to balance the total area among the stages so that the design can fit into a smaller device, *i.e.* to have the area of each stage to be close to the average $\frac{w(V)}{k}$. We define *α-bounded bipartitioning* to be partitioning a set of nodes V into two subsets $(X, \overline{X})$ so that $w(X)$ is as close to α as possible, *i.e.* $(1 - \epsilon)\alpha \leq w(X) \leq (1 + \epsilon)\alpha$. ϵ is the variation factor with $0 \leq \epsilon < 1$, *e.g.* $\epsilon = 0.05$. The k-way partitioning problem can be reduced to finding $k - 1$ α-bounded bipartitioning.

4 Network-Flow Based Bipartitioning

Network-flow technique is well known for finding min-cut due to the max-flow min-cut theorem [5]. FBB [14] applied

repeated max-flow min-cut computation to find min-net-cut for balanced circuit bi-partitioning. But [14] did not consider the ordering of the nodes. In our partitioning problem, the nodes in the same net are not symmetric. For a combinational net in N_c, the nodes must satisfy the precedence constraints. For a sequential net in N_f, the ordering of nodes influences the number of memory elements used for communication.

[16] applied network-flow technique to multi-way partitioning for time-multiplexed FPGAs based on the Xilinx architecture [1]. A net modeling method is given in [16] to build a network G' from the netlist G, so that a min-cut in G' corresponds to a uni-directional net cut in G satisfying the precedence constraints. However, since the TMCL model [2] used a different architecture than [1] for storing the communication, the net modeling for sequential circuit in [16] can not be applied here.

In the following sections, we present net modeling for two-terminal and multi-terminal nets in combinational and sequential circuits based on the TMCL model. Then we present a network flow based approach for finding α-bounded bipartitioning.

4.1 Net Modeling for Combinational Circuit

A proper net modeling for combinational circuits must meet two requirements: (1) correctly models a net cut, so that a net is counted exactly once if it is cut; (2) correctly models the precedence constraints among the nodes, *i.e.* a net-cut must be uni-directional. A uni-directional cut is a two-way partitioning $(X, \overline{X})$ such that for any net $n = \{v_1, ..., v_p\} \in N_c$, either all the nodes in n are in the same subset, or v_1 is in X. If we let X be an earlier stage than $\overline{X}$, then it is easy to prove that a uni-directional cut satisfies the precedence constraints.

We construct network $G' = (V', N')$ from G by the following net modeling of a net in N_c (Figure 7).

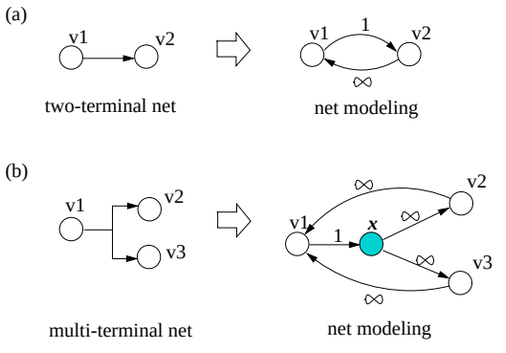

Figure 7: *Net modeling for a two-terminal net and a multi-terminal net in N_c.*

1. All the nodes in V are in V', *i.e.* $V \subset V'$, and each node in V' has the same area as in V.
2. For a two-terminal net $v_1 \to v_2$ in N_c, add a bridging edge $v_1 \to v_2$ in G' with capacity 1, add an edge $v_2 \to v_1$ in G' with capacity ∞ (Figure 7(a)).
3. For a multi-terminal net $n = \{v_1, ..., v_p\}$ with $p > 2$, let v_1 be the input to all other nodes in n. Add a node x in G' with $w(x) = 0$. Add a bridging edge from v_1 to x with capacity 1, add an edge from x to each node v_j $(2 \le j \le p)$ with capacity ∞. Add an edge from node v_j $(2 \le j \le p)$ to v_1 with capacity ∞ (Figure 7(b)).

Here we distinguish the net modeling of a two-terminal net and multi-terminal net, because for two-terminal nets, we add fewer number of edges and nodes, which will reduce the size of the network and speed up the max-flow computation.

For each net, exactly one bridging edge $v_1 \to x$ with capacity 1 is added, and all the other edges have ∞ capacity. Notice that nodes in the same net are asymmetric, the bridging edge starts from v_1 and there is an edge with ∞ capacity from v_j $(2 \le j \le p)$ to v_1. After the max-flow computation on the constructed network G', for a min-cut $(X, \overline{X})$, all the forward edges from X to $\overline{X}$ must be saturated (*i.e.* flow equals to the capacity) and all the backward edges from $\overline{X}$ to X have zero amount of flow. If a net is cut, then only the bridging edge $v_1 \to x$ will be the forward cut edge from X to $\overline{X}$, and therefore v_1 must be in X, which preserves the precedence constraints. Since the capacity on the bridging edge $v_1 \to x$ is one, the cut net contributes exactly 1 to the total cut size.

Figure 8 shows an example of how to get the corresponding net cut in G from a cut in G'. Lemma 1 shows the correctness of the above net modeling for combinational circuits.

Lemma 1: *The min-cut size in G' equals to the minimum number of uni-directional cut-nets in G.*

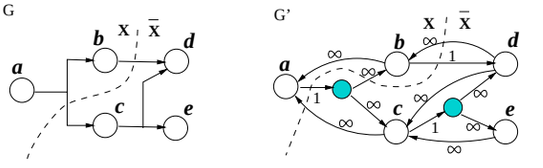

Figure 8: *A cut in G' and the corresponding net-cut in G*

4.2 Net Modeling for Sequential Circuits

In sequential circuits, for nets in N_c, we use the same net modeling as introduced in section 4.1. For nets in N_f, the ordering of the nodes influences the number of memory elements, therefore the communication cost. If $s(v_1) \ge s(v_j)$ for v_1, $v_j \in n$, then one memory element (ME) will be needed. If $s(v_1) < s(v_j)$, then two memory elements (MEs) will be used. We want to find a min-cut which minimizes the number of memory elements needed.

We introduce the following net modeling for nets in N_f (Figure 9).

1. For a two-terminal net (v_1, v_2), add an edge from v_2 to v_1 with capacity 1, and add an edge from v_1 to v_2 with capacity 2 (Figure 9(a)).
2. For a multi-terminal net $n = (v_1, ..., v_p)$, add two nodes w_1 and w_2 with $w(w_1) = 0$, $w(w_2) = 0$. Add an edge from v_1 to w_1 with capacity 2, and add an edge from w_2 to v_1 with capacity 1. Add an edge from w_1 to each of the node v_j $(2 \le j \le p)$ with capacity ∞. Add an edge from each node v_j $(2 \le j \le p)$ to w_2 with capacity ∞ (Figure 9(b)).

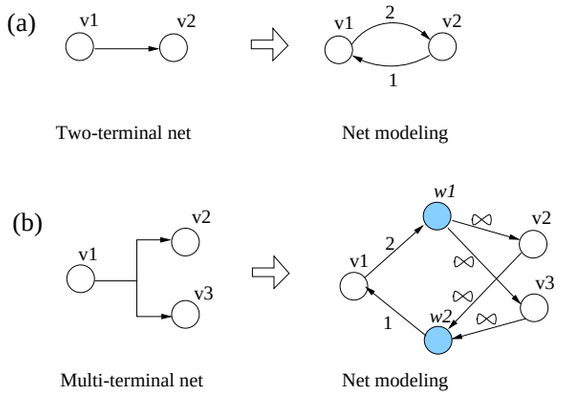

Figure 9: *Net modeling for a two-terminal net and a multi-terminal net in N_f.*

For the above net modeling, the different ordering of the nodes will cause different net-cut size, and the cut size reflects the communication cost. Lemma 2 shows the correctness of the above net modeling for a net in N_f.

Lemma 2: *For a net $n = \{v_1, ..., v_p\} \in N_f$, if $v_1 \in \overline{X}$ and $v_j \in X$, then the cut-size is 1; if $v_1 \in X$ and $v_j \in \overline{X}$, then the cut size is 2.*

Proof: For a two-terminal net, if $v_1 \in \overline{X}$ and $v_2 \in X$, then edge $v_2 \rightarrow v_1$ will be the forward cut edge from X to $\overline{X}$ with cut size 1, which is equal to the capacity on this edge. If $v_1 \in X$ and $v_2 \in \overline{X}$, then $v_1 \rightarrow v_2$ will be the forward cut edge with cut size 2. For a multi-terminal net $n \in N_f$, if $v_1 \in \overline{X}$ and $v_j \in X$, then since only $v_1 \rightarrow w_1$ and $w_2 \rightarrow v_1$ have capacity less than ∞ in the net modeling, $w_2 \rightarrow v_1$ will be the forward cut edge with $w_2 \in X$. So the cut size is 1. On the other hand, if $v_1 \in X$ and $v_j \in \overline{X}$, then $v_1 \rightarrow w_1$ will be the forward cut edge from X to $\overline{X}$. Since the capacity on edge $v_1 \rightarrow w_1$ is 2 and the edge is saturated after the max-flow computation, so the cut size is 2 in this case. ♯

Figure 10 shows a cut in the constructed network G' and the corresponding net cut G. In the example of Figure 10(a), if net n is cut and $v_1 \in X$, then the bridging edge $v_1 \rightarrow w_1$ is cut and the cut size is 2 which equals to the capacity on $v_1 \rightarrow w_1$. Figure 10(b) shows if $v_1 \in \overline{X}$, then the bridging edge $w_2 \rightarrow v_1$ will be cut and contributes 1 to the cut size. Figure 11 shows a netlist G and the corresponding network G' after net modeling of both nets in N_c and N_f. Notice the net $\{a, b, c\}$ is a multi-terminal net belonging to N_f.

4.3 α-bounded Bipartitioning

By the net modeling, we can build a network G' from the netlist G, then apply the repeated max-flow min-cut strategy similar to the algorithm in [16] to find an α-bounded bipartitioning that minimizes the number of crossing nets.

First, a network G' is constructed from G by the net modeling discussed in sections 4.1 and 4.2. Next, a source s and sink t is selected. Then by the max-flow computation, the maximum amount of flow is pushed from the source to the sink, and a min-cut $(X, \overline{X})$ is found in G'. If $(1-\epsilon)\alpha \leq w(X) \leq (1+\epsilon)\alpha$, then return $(X, \overline{X})$ as the result. If $w(X) < (1-\epsilon)\alpha$, then nodes in X are collapsed to s and a node v from $\overline{X}$ is collapsed to s, so that in the next iteration more flows can be pushed through the network and explore a

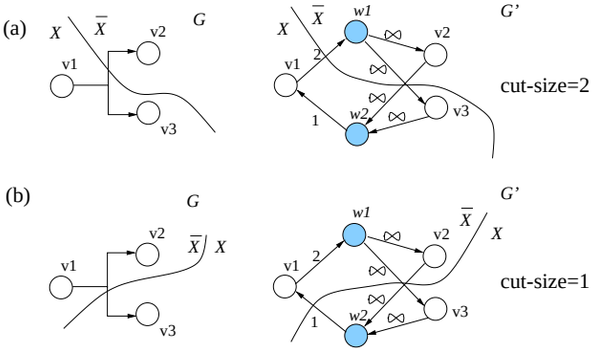

Figure 10: *The corresponding cut in the network G' and the netlist G.*

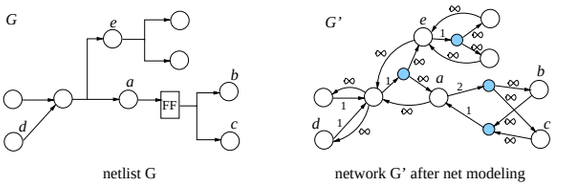

Figure 11: *Example of net-modeling.*

different net cut with a larger area in X. If $w(X) > (1+\epsilon)\alpha$, then all nodes in $\overline{X}$ are collapsed to t, and a node v from X is collapsed to t. Then the max-flow min-cut computation repeats until the area $w(X)$ for subset X is within range.

Incremental flow technique is employed for efficient implementation. It is not necessary to calculate the max-flow from scratch in each iteration. Only additional flow is added through the network from the source to the sink to saturate the bridging edges during the max-flow computation. The time complexity for the repeated max-flow min-cut is asymptotically the same as one max-flow computation, which is $O(|V||E|)$.

Figure 12 shows an example of finding an α-bounded bipartitioning with α=6. The edges with no markings have capacity ∞. In the first iteration, min-cut is 1 after the max-flow computation, and $w(X)$=1. Then node $d \in \overline{X}$ is collapsed to s (*i.e.* $w(s)$=2 now) so that more flow can be pushed through the network in the next iteration. In the second iteration, after pushing the max-flow, the min-cut size is still 1 and $w(X) = 3$. Another node c from $\overline{X}$ is collapsed to X and $w(X) = 4$. In the third iteration, min-cut is 1 and $\overline{X}$ is collapsed to t with $w(\overline{X}) = 3$. In the next iteration, after pushing more flow through the network, min-cut is 2 and X reaches the area limit with $w(X) = 5$. So $(X, \overline{X})$ forms an α-bounded min-cut with cut size 2. We can then find the corresponding net cut in the original netlist G.

5 Multi-way Partitioning

To partition a netlist into k ($k > 2$) stages for dynamically reconfigurable FPGA, we repeatedly apply the network-flow based bipartitioning algorithm $k-1$ times to partition the netlist into k stages.

Since the length of the critical path is usually longer than the number of stages, there will be more than one levels of

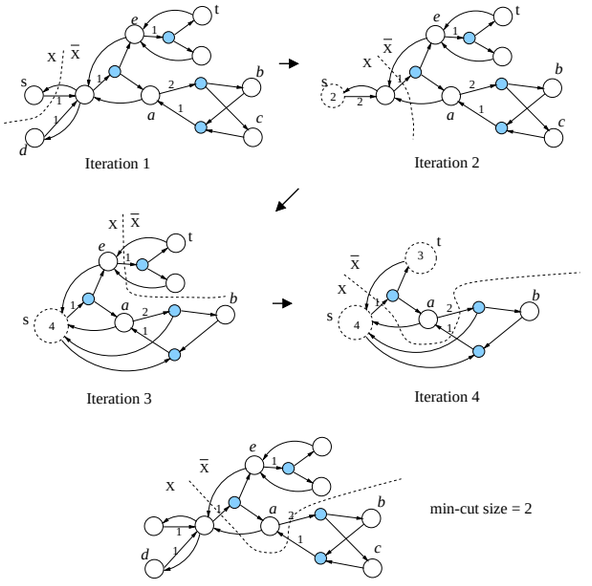

Figure 12: *Example of α-bounded bipartitioning.*

nodes in one stage. Let *depth* be the number of levels of nodes on the critical path in the netlist. When partitioning into k stages, the number of levels in one stage can be $D = \lceil \frac{depth}{k} \rceil$ in order to make the design as fast as possible. If the timing constraint for each stage is known a priori, then the maximum number of levels in one stage can be calculated accordingly. Besides minimizing the communication cost, our objective also includes minimizing the maximum number of nodes in any stage in order to allow the design to fit into a smaller physical FPGA. It is desirable to make each stage have area as close to the average, $\frac{w(V)}{k}$, as possible (*i.e.* $\alpha = \frac{w(V)}{k}$).

Our strategy is to first apply the As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling to assign each node a range of feasible stages. Then the nodes on the critical paths are fixed to certain stages, and those nodes on the shorter paths have the flexibility to be put in more than one stage. Network flow based α-bounded bipartitioning is iteratively applied to partition these flexible nodes between stage i and $i+1$ $(1 \leq i < k)$, with the objective of minimizing the communication cost and balancing the number of nodes in each stage as well. Due to the precedence constraints, the fixed nodes will serve as source and sink when partitioning the flexible nodes.

The partitioning process of Algorithm 1 has four major steps.

Step 1: perform As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling. In the ASAP scheduling, each node is assigned to the earliest possible stage by breadth search. In the ALAP scheduling, each node is assigned to the latest possible stage. Let $AS(v)$, $AL(v)$ be the earliest and latest stage for node v. If $AS(v) = AL(v) = j$, then v must be scheduled in stage j. We call v as a *fixed* node. If $AS(v) < AL(v)$, then v can be assigned to any stage from $AS(v)$ to $AL(v)$. We call v as a *flexible* node.

Step 2: let P_i be the subset of nodes fixed to stage i $(1 \leq i \leq k)$, *i.e.* $P_i = \{v | AS(v) = AL(v) = i\}$. Assign all the nodes in P_i to stage i. The other unassigned nodes are the flexible nodes which can be put in more than one stage. In our partitioning process, the goal is to assign a stage for each of the flexible node while balancing the number of nodes in each stage and minimizing the net-cut size between the stages.

Step 3: iteratively call the network-flow based bipartitioning algorithm to partition the flexible nodes between stages i and $i+1$ $(1 \leq i < k)$. For the i-th iteration, the details of the partitioning process are as follows.

Algorithm 1
Network-flow based multi-way partitioning for DRFPGAs
begin
1. perform ASAP and ALAP scheduling;
   for each node v,
   let $AS(v)$ be the earliest stage by the ASAP scheduling;
   let $AL(v)$ be the latest stage by the ALAP scheduling;
2. **for** $i = 1$ to k **do**
   $P_i = \{v | AS(v) = AL(v) = i\}$;
   for $i = 1$ to k **do**
   assign all nodes in P_i to stage i;
3. **for** $i = 1$ to $k-1$ **do**
   begin
   3.1. $F = \{v | AS(v) \leq i$, and v is unassigned$\}$;
   3.2. source $s = (\cup_{j=1}^{i-1} V_j) \cup P_i$, and $w(s) = w(P_i)$;
   sink node $t = \{v | AS(v) > i\}$, and $w(t) = w(P_{i+1})$;
   3.3. construct network F' from $F \cup s \cup t$ by net modeling;
   3.4. find an α-bounded bipartitioning $(X, \overline{X})$ in F';
   3.5. assign nodes in X to stage i, let $V_i = P_i \cup (X - s)$;
   3.6. for $v \in F$ with $AL(v) = i+1$, assign v to stage $i+1$;
   end
4. Optimally determine the locations of the ME2's.
end

In step 3.1, all the flexible nodes that can be put in stage i form a subset $F = \{v | AS(v) \leq i$, and v is unassigned$\}$. Notice that nodes in F can either be put in stage i or $i+1$. This step guarantees that no node v is assigned to a stage earlier than $AS(v)$. In step 3.2, the source s and sink t of the network are decided. The source is a subset of nodes with $s = (\cup_{j=1}^{i-1} V_j) \cup P_i$ and $w(s) = w(P_i)$. The source s contains all the nodes assigned to stages prior to i and the fixed nodes in P_i. The sink $t = \{v | AS(v) > i\}$ and $w(t) = w(P_{i+1})$. t contains nodes which can only be put in a stage later than i. In step 3.3, network F' is constructed from $F \cup s \cup t$ by the net modeling. In step 3.4, the α-bounded bipartitioning algorithm is applied on F' to find a min-cut $(X, \overline{X})$. Then in step 3.5, the nodes in X are assigned to stage i, such that $V_i = P_i \cup (X - s)$. Next in step 3.6. all the unassigned nodes with $AL(v) = i+1$ are assigned to stage $i+1$, *i.e.* $P_{i+1} = P_{i+1} \cup \{v | AL(v) = i+1\}$. This step is to guarantee that any node v should be assigned to a stage no later than $AL(v)$. Then i is incremented by 1 and control goes back to step 3.1 to start the next iteration. Since the nodes on the critical path are fixed to a stage before the partitioning process, therefore $P_i \neq \phi$, and the source s and sink t are known in each iteration. With fixed source and sink, the network-flow based approach is capable of finding a good min cut.

Step 4: after the k-way partitioning is done, the locations of the ME2's (ME2 situation) are determined. The ME2's are assigned to a stage which will minimally increase the number of MEs needed. This can be optimally solved by the max-flow based algorithm [2].

By Lemma 1, it is easy to prove that the partitioning solution of Algorithm 1 satisfies the precedence constraints among all the nodes. The following Lemma 3 shows that Algorithm 1 also satisfies the timing constraint. Given the maximum number of levels in each stage, $AS(v)$ and $AL(v)$ are the earliest and latest stage that node v can be assigned in by the ASAP and ALAP scheduling. To satisfy the timing constraint, v must be in a stage within the range of $[AS(v), AL(v)]$.

Lemma 3: *The partitioning solution of Algorithm 1 satisfies the timing constraint, such that for any node v, $AS(v) \leq s(v) \leq AL(v)$.*

Proof: In $AS(v) = AL(v) = j$, then $v \in P_j$, v is fixed to stage j and $AS(v) = s(v) = AL(v)$. If $AS(v) < AL(v)$, v is a flexible node, by the construction of F in step 3.3, v is in F only in the i-th iteration when $AS(v) \leq i$, and v can only be assigned to a stage either in step 3.5 or 3.6. If $v \in X$, then v is assigned to stage i in step 3.5, else v is assign to stage $AL(v)$ by step 3.6. In both cases, $AS(v) \leq s(v)$. Step 3.6 guarantees that v is assigned to a stage no later than $AL(v)$, so $s(v) \leq AL(v)$. Therefore, $AS(v) \leq s(v) \leq AL(v)$. ♯

The ASAP and ALAP scheduling in step 1 takes $O(|V|)$ time. Each iteration in step 3 takes $O(|V||E|)$ time, so the time complexity for the $k-1$ iterations is $O(k|V||E|)$. The time complexity for Algorithm 1 is $O(k|V||E|)$.

Table 1: Characteristics of the netlists in our experiment

Circuit	# LUT	# DFF	# Nets	Depth
s5378	422	162	590	10
s9234	317	135	462	13
s13207	688	453	1121	14
s15850	1056	540	1570	23
s35932	2756	1728	4515	6
s38417	3458	1464	4894	18
s38584	3545	1294	4793	20

6 Experimental Results

We implemented algorithm 1 in C++ on Intel Pentium-Pro of 200Mz with 32MB memory, and experimented on the same netlists as in [2]. These netlists are derived by technology-mapping the ISCAS'89 sequential benchmark circuits into 4-LUTs. Table 1 shows the characteristic of these netlists. Columns 2 to 4 show the number of LUTs, FFs and nets in each netlist. In column 5, *depth* refers to the number of levels of nodes on the critical path.

In the experiment, we perform multi-way partitioning into 2, 4 and 8 stages. We fix the number of levels in each stage to be $\lceil \frac{depth}{k} \rceil$ with k being the number of stages. In Table 2, we compare the communication cost with traditional force-directed scheduling (FDS) and enhanced force-directed scheduling (eFDS) [2] when partitioning into 8 stages. T_{CM} is the maximum number of MEs required for any stage. Notice that in our algorithm, the LUTs are balanced among the stages. Table 2 also shows the CPU time of Algorithm 1. From the experiment, our network-flow based method outperforms both the FDS and enhanced FDS methods, with an average improvement of 41.5% and 15% respectively.

Table 3 compares the communication cost of Algorithm 1 with the enhanced FDS method [2] while partitioning into 2, 4 and 8 stages. Our method consistently performs better, with an average improvement of 14.0%, 10.5% and 15.4% respectively. Since our net modeling does not favor the ME2 situation, our experiments also show that the number of ME2's is relatively low and usually does not add to the overall communication cost.

The experimental results show that with proper net modeling, network-flow based partitioning approach can handle the scheduling tasks. The net modeling correctly models the precedence constraints in combinational circuits, and the cut-size reflects the communication cost in both combinational and sequential circuits. The iterative max-flow min-cut computation balances the number of nodes in each stage.

7 Conclusion

Dynamically reconfigurable FPGAs (DRFPGA) have the potential to dramatically improve logic density by time-sharing logic, and have become an active research for reconfigurable computing. The different DRFPGA architectures impose different requirement on the partitioning problem. We present a network-flow based method for multi-way partitioning for dynamically reconfigurable FPGAs based on the TMCL model in [2]. We first give a net modeling for both combinational and sequential circuits so that by the max-flow computation on the constructed network, the min-cut size reflects the number of communication required. We then present a repeated max-flow min-cut based approach to find an α-bounded bipartitioning. Algorithm 1 iteratively applies the bipartitioning algorithm to find a multi-way partitioning. The experiments show that the network-flow based algorithm outperforms the enhanced force-directed scheduling method [2].

8 Acknowledgment

We thank Dr. Douglas Chang for kindly providing us with the data for the experiment.

References

[1] Steve Trimberger, "Scheduling Designs into a Time-Multiplexed FPGA", *International Symposium on Field Programmable Gate Arrays*, Feb, 1998.

[2] Douglas Chang and Malgorzata Marek-Sadowska, "Partitioning Sequential Circuits on dynamically Reconfigurable FPGAs", *International Symposium on Field Programmable Gate Arrays*, Feb, 1998.

[3] Douglas Chang and Malgorzata Marek-Sadowska, "Buffer Minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs", *International Symposium on Field Programmable Gate Arrays*, Feb., 1997.

[4] Sasan Iman, Massoud Pedram, Charles Fabian and Jason Cong, "Finding Uni-Directional Cuts Based on Physical Partitioning and Logic Restructuring", *4th International workshop on physical design*, 1993.

Table 2: Comparing partitioning results with FDS and enhanced FDS (eFDS)

	FDS	eFDS[2]	ours			
Circuit	T_{CM}	T_{CM}	T_{CM}	Impv. over FDS	Impv. over eFDS	CPU(sec.)
s5378	141	98	76	46.1%	22.4%	0.27
s9234	115	80	64	44.3%	20.0%	0.09
s13207	500	173	142	71.6%	17.9%	0.25
s15850	384	225	210	45.3%	6.7%	0.82
s35932	733	664	656	10.5%	1.2%	15.56
s38417	882	743	631	28.5%	15.1%	19.63
s38584	1004	744	561	44.1%	24.6%	22.31
Average				41.5%	15.4%	

Table 3: Comparing the communication cost with enhanced FDS (eFDS)

	T_{CM} $(n=2)$			T_{CM} $(n=4)$			T_{CM} $(n=8)$		
Circuit	eFDS[2]	ours	Impv.	eFDS[2]	ours	Impv.	eFDS[2]	ours	Impv.
s5378	159	132	16.9%	127	109	14.2%	98	76	22.4%
s9234	130	126	3.1%	107	100	6.5%	80	64	20.0%
s13207	305	288	5.6%	237	229	3.4%	173	142	17.9%
s15850	400	336	16.0%	335	285	14.9%	225	210	6.7%
s35932	1472	1308	11.1%	1094	1067	2.5%	664	656	1.2%
s38417	1324	1138	14.5%	1149	875	23.8%	743	631	15.1%
s38584	1054	726	31.1%	810	661	18.4%	744	561	24.6%
Average			14.0%			10.5%			15.4%

[5] J.R. Ford and D.R. Fulkerson, "Flows in Networks", Princeton University Press, 1962.

[6] N.B. Bhat, K. Chaudhary and E.S. Kuh, "Performance-oriented fully routable dynamic architecture for a field programmable logic device", Memorandum No. UCB/ERL M93/42, university of California, Berkeley, 1993.

[7] Jeremy Brown, Derrick Chen, et al. "DELTA: Prototype for a first- generation dynamically programmable gate array", Transit Note 112, MIT, 1995.

[8] Andre DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century", In *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994.

[9] D. Jones and D.M. Lewis, "A time-multiplexed FPGA architecture for logic emulation", In *IEEE Custom Integrated Circuits Conference*, 1995.

[10] R. S. Tsay, E. S. Kuh and C. P. Hsu, "Proud: A sea-of-gates placement algorithm.", in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp318-323, Nov. 1988.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[12] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", In *Proceedings of the 19th Design Automation Conferences*, pp 175-181, June 1982.

[13] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *IEEE Transaction on Computers*, pp1064-1068, Nov. 1978.

[14] Honghua Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *Proc. ICCAD*, 1994.

[15] Xilinx, *The Programmable Logic Data Book*, 1996.

[16] Huiqun Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs", *International Conference on Computer Aided Design*, San Jose, CA, Nov. 1998.