Network Flow Based Circuit Partitioning for Time-multiplexed FPGAs*

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

Abstract

Time-multiplexed FPGAs have the potential to dramatically improve logic density by time-sharing logic, and have become an active research for reconfigurable computing. The partitioning problem for time-multiplexed FPGAs is different from the traditional partitioning problem in that the nodes have precedence constraints among them, and the widely used iterative improvement partitioning methods such as K&L, FM [14,15] are no longer applicable. All previous approaches [1,2,3] used list scheduling heuristics.

In this paper, we present a network flow based algorithm for multi-way precedence constrained partitioning, which can handle the precedence constraints while minimizing the net-cut size. The experimental results on the MCNC benchmark circuits show that our algorithm out-performs list scheduling by a big margin, with an average improvement of over 50% for bipartitioning and 20% for multi-way partitioning.

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). A virtual large logic design is partitioned into multiple stages to share the same smaller physical device in a time-multiplexed fashion.

Time-multiplexed FPGAs have the potential to dramatically improve logic density by time-sharing logic. Several different architectures have been proposed, such as Xilinx model [1], Dharma [6], the Dynamically Programmable Gate Array [7,8], and the Virtual Element Gate Array [9]. These DRFPGs allow dynamic reuse of the logic blocks and wire segments by having more than one on-chip SRAM bits controlling them. Thus logic blocks and interconnect can be changed by reading a different SRAM bit which only takes time in the order of nanoseconds. Currently, there are partially reconfigurable FPGAs available commercially such as AT6000 from Atmel and XC6200 from Xilinx.

Figure 1 shows the Xilinx time-multiplexed FPGA configuration model [1]. The FPGA emulates a large device by sequencing through multiple configurations called micro-cycles. One pass through all the micro-cycles is called a user cycle. In each micro-cycle, the CLBs (Configurable Logic Blocks) are re-used to evaluate logic. The target architecture consists of an array of augmented XC4000E-style CLBs [17, 1]. Each CLB includes a micro register to hold CLB results from the previous micro-cycle. Micro registers hold combinational logic intermediate values for use in later micro-cycles of the same user cycle, and also hold flip-flop values for use in the next user cycle. Every configuration memory cell of the original FPGA is backed by eight inactive memory cells. A micro-cycle starts by saving all CLB results from the previous micro-cycle in micro registers. After this state saving, a new configuration is loaded into active configuration memory.

Figure 1: Xilinx time-multiplexed FPGA configuration model.

Because the logic and interconnect needed for a circuit is time-multiplexed on a DRFPGA, the traditional FPGA design flow needs to be modified. The traditional design flow involves logic synthesis, technology mapping, placement and routing. Now it is essential to have a good partitioning strategy to ensure the correctness of the execution, to minimize the number of interconnection among the partitions as well as satisfy both the area and pin constraints for a physical FPGA.

The multi-way partitioning for time-multiplexed FPGA is different from the traditional circuit partitioning problem. One important reason is that in a time-multiplexed FPGA, the order of the execution of the nodes must follow the precedence constraints. For example, in a combinational circuit, a node must be in a stage no later than all its output nodes. Therefore, a cut should be a uni-directional cut. The traditional partitioning approaches, such as K&L [15] and FM [14] based methods, do not handle this constraint and are no longer applicable for this partitioning problem. The partitioning problem for time-multiplexed FPGA can be formulated as a Directed Acyclic Graph (DAG) scheduling problem.

All previous research [1,2,3] used traditional DAG scheduling methods, such as list scheduling. However, the list scheduling heuristics do not take into account the cost for buffering a signal between non-adjacent time frames.

In this paper, we propose a network flow based approach for multi-way precedence constrained partitioning with application in time-multiplexed FPGAs. It can handle the precedence constraints and minimize the number of interconnections at the same time. We show how to correctly model the

---

*This work was partially supported by the Texas Advanced Research Program under Grant No. 00658288 and by a grant from the Intel Corporation.
nets in both combinational and sequential circuits, so that by the max-flow computation, the min-cut corresponds to a minimum uni-directional net cut which satisfies the precedence constraints among the nodes. A bipartitioning algorithm based on the repeated max-flow min-cut computation is presented and then it is iteratively applied to partitioning a netlist into multiple stages.

The organization of the paper is as follows. In section 2, we give the formulation for the precedence constrained partitioning problem. In section 3 we first present the proper net modeling method for both combinational and sequential circuits, and then present the network flow based approach for α-bounded uni-directional bipartitioning. Section 4 explains the multi-way precedence constrained partitioning algorithm for time-multiplexed FPGA. Experimental results are discussed in section 5.

2 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. The nodes in V are classified into two types, combinational node (c-node) and flip-flop node (FF-node). Each node v in V has a weight w(v), and the weight of a subset X of V, denoted by w(X), is the total weight of all the nodes in X.

For a net n = {v1, ..., vp} with p nodes, let v1 be the fan-out node whose output signal is the input signal to v2 ∈ n (2 ≤ j ≤ p), v1 is the input of vj (2 ≤ j ≤ p), and vj (2 ≤ j ≤ p) is the output of vj. We further define N = Nc ⊔ NF. A net n ∈ Nc if v1 is a c-node, and n ∈ NF if v1 is a FF-node. 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.

![Figure 2: Temporal partitioning of a circuit into 4 stages.](image)

For a time-multiplexed 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 2). In the following discussions in the paper, we use stage and the same results on the outputs as would be seen by a non-time-multiplexed device.

In order for a partitioned time-multiplexed circuit to produce the correct result in one user cycle, the nodes must be evaluated in a proper order. According to Xilinx’s architecture [1], the following three precedence constraints must be satisfied:

1. Each c-node must be scheduled in a stage no later than all its output nodes.
2. Each FF-node must be scheduled in a stage no earlier than all its input c-nodes. This rule guarantees that flip-flop input values are calculated before they are stored.
3. Each FF-node must be scheduled in a stage no earlier than all its output nodes. This rule guarantees that all the nodes that use the value of the flip-flop use the same value: the value of the flip-flop from the previous user cycle.

The above constraints define a partial temporal ordering on the nodes in the circuit. Let P(v) be the precedence of a node v, for two nodes v and u, we define P(v) ≤ P(u) if v must be scheduled no later than node u. We can rephrase the above constraints as follows: for a net n = {v1, ..., vp}, let v1 be the input to vj (2 ≤ j ≤ p).

- If v1 is a c-node, then P(v1) ≤ P(vj) for 2 ≤ j ≤ p;
- If v1 is a FF-node, then P(vj) ≤ P(v1) for 2 ≤ j ≤ p.

The k-way precedence constrained partitioning problem is to partition a circuit G = (V, N) into k non-overlapping subsets V1, ..., Vk, and for each node v, let s(v) = j if v ∈ Vj, subject to

1. V = ∪k i=1 Vi
2. For any nodes v and u, if P(v) ≤ P(u), then s(v) ≤ s(u).

with the objective of

1. Minimizing the interconnection between the partitions;
2. Minimizing max{w(Vi)|1 ≤ i ≤ k}.

The subsets Vi (1 ≤ i ≤ k) share the same physical FPGA device in a time-multiplexed way. s(v) is the stage that v is assigned to. It is important to minimize the amount of interconnection among the different partitions, which greatly influences the placement and routing process. For Xilinx’s architecture [1], the values of the cut-nets to be passed to a later stage are stored in micro registers.

The nodes in Vc are called virtual nodes, while the nodes (e.g. LUTs) in a physical FPGA device are called real nodes. The number of virtual nodes for each stage should be smaller than the number of real nodes ν, i.e. max{w(Vi)|1 ≤ i ≤ k} ≤ r, so that the virtual nodes can be fit into the physical FPGA device. Minimizing max{w(Vi)|1 ≤ i ≤ k} allows the design to fit into a smaller physical FPGA. To do so, we need to let the weight in each stage be as close to w(V) as possible, which is the average of the total weight.

A uni-directional cut is a cut (X, X̄) such that for two nodes v and u, if P(v) ≤ P(u), then either v and u are in the same subset, or v ∈ X and u ∈ X̄. By definition, a uni-directional cut satisfies the precedence constraints. The α-bounded uni-directional min-cut bipartitioning with respect to two nodes s and t, is the problem of partitioning a circuit of total weight W into two disjoint subsets X and X̄, where s ∈ X and t ∈ X̄, with the minimum number of uni-directional cut nets such that w(X) is most close to α. We allow w(X) to deviate from (1 − ε)a to (1 + ε)a, e.g. ε = 5%. For k-way precedence constrained partitioning, α = w(V)/k for each stage.

The k-way precedence constrained partitioning problem can be reduced to finding k − 1 α-bounded uni-directional cuts which partition the circuit into k stages, with each cut (∪j ≤ i Vj, ∪j > i Vj) for 1 ≤ i < k being uni-directional.

Figure 3(a) shows an α-bounded uni-directional min-cut with α = 5. Net n1 = {a, b, f} and node a is the input to nodes b and f. According to the constraint, a must be in a stage no later than b and f. Figure 3(b) shows an example that a min-cut may not necessarily be a uni-directional cut. Net n2 is cut, since P(a) ≤ P(b), a should be in X, a ∈ X̄ and b ∈ X violates the definition for a uni-directional cut.
The problem of finding an \( \alpha \)-bounded uni-directional min-cut is NP-hard even if all \( v \in V \) have \( w(v) = 1 \) [13]. In this paper, we present a network flow based approach which uses iterative max-flow min-cut technique for \( \alpha \)-bounded uni-directional partitioning, then apply the method to multi-way precedence constrained partitioning.

3 Network Flow Based Uni-directional Bipartitioning

Network flow is an excellent approach to finding min-cuts because of the celebrated max-flow min-cut theorem [5]. Yang and Wong [16] successfully applied the network flow approach to balanced bipartitioning and employed incremental flow technique for efficient implementation. [16] gave a net modeling so that the min-cut in the constructed network corresponds to min-net-cut in the original circuit. But with this net modeling, nodes in the same net are symmetric and do not have precedence constraints among them, so the min-cut found is not necessarily a uni-directional cut. For our precedence constrained partitioning problem, it is important and necessary to find a proper net modeling so that the precedence constraints among the nodes will be maintained.

In another related work, Cong et al. [4] first used the iterative max-flow min-cut method to find uni-directional min-cut on combinational circuit in a logic synthesis algorithm. But they only modeled two-terminal nets in a combinational circuit.

In the following sections, we present net modeling for both two-terminal and multi-terminal nets in combinational and sequential circuits, so that by the max-flow computation, the min-cut preserves the precedence constraints. We further prove the correctness of the net modeling.

3.1 Net Modeling for Combinational Circuit

A proper net modeling 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.

For a net \( n = \{v_1, ..., v_p\} \) in \( N_f \), \( v_1 \) is a source node and \( P(v_1) \leq P(v_j) \) for \( 2 \leq j \leq p \) (i.e., \( v_1 \) must be in a stage no later than its output nodes). We construct network \( G' = (V', N') \) from \( G = (V, N) \) by the following net modeling (Figure 4).

1. All the nodes in \( V \) are in \( V' \), i.e. \( V \subseteq V' \).
2. For a two-terminal net \( \{v_1, v_2\} \), add a bridging edge \( v_1 \rightarrow v_2 \) in \( G' \) with capacity 1, add an edge \( v_2 \rightarrow v_1 \) in \( G' \) with capacity \( \infty \).
3. For a multi-terminal net \( n = \{v_1, ..., v_p\} \) where \( p > 2 \), 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 \( v_j \) (\( 2 \leq j \leq p \)) with capacity \( \infty \). Add an edge from node \( v_j \) (\( 2 \leq j \leq p \)) to \( v_1 \), with capacity \( \infty \).

3.2 Net Modeling for Sequential Circuit

For a net \( n = \{v_1, ..., v_p\} \) in \( N_f \), \( v_1 \) is a FF-node and \( P(v_j) \leq P(v_1) \) for \( 2 \leq j \leq p \) (i.e., \( v_1 \) must be in a stage no earlier than all its output nodes). The following is the modeling of a net in \( N_f \) (Figure 6).

1. For a two-terminal net \( \{v_1, v_2\} \) where \( v_1 \) is a FF-node, add a bridging edge from \( v_2 \) to \( v_1 \) with capacity 1, and add an edge from \( v_1 \) to \( v_2 \) with capacity \( \infty \).
2. For a multi-terminal net \( n = \{v_1, ..., v_p\} \), add a node \( x \) with \( w(x) = 0 \). Add a bridging edge from \( x \) to \( v_1 \) with capacity 1. Add an edge from \( v_j \) (\( 2 \leq j \leq p \)) to \( x \) with capacity \( \infty \). Add an edge from \( v_1 \) to each node \( v_j \) (\( 2 \leq j \leq p \)) with capacity \( \infty \).

For each net \( n \in N_f \), only the bridging edge has capacity 1, all the other edges have capacity \( \infty \). The FF-node \( v_1 \) has
an edge with infinite capacity to each of its output nodes \( v_j \) \((2 \leq j \leq p)\), so if it is cut, \( v_1 \) must be in \( \overline{X} \). This guarantees the precedence constraints that \( v_1 \) must be in a stage no earlier than its output nodes.

Figure 7 shows an example of the net modeling and the corresponding net cut in \( G \) from a min-cut in \( G' \). There are three nets, \( n_1 \in N_f \) and \( n_2, n_3 \) are \( N_c \) nets. For min-cut \((X, \overline{X})\), \( n_1 \) and \( n_2 \) are cut. In \( G' \), all the cut edges from \( X' \) to \( \overline{X'} \) have capacity 1. For net \( n_1 \), the FF-node \( b \) and c-node \( f \) are in \( \overline{X} \), and \( d \) and \( e \) are in \( X \).

Figure 7: A cut in \( G' \) and the corresponding net-cut in \( G \).

We have the following lemmas about the correctness of the net modeling for nets both in \( N_c \) and \( N_f \).

**Lemma 1:** Any min-cut in \( G' \) corresponds to a net-cut in \( G \).

**Proof:** After the max-flow min-cut computation in \( G' \), every cut edge from \( X' \) to \( \overline{X'} \) has capacity 1 and is saturated. Only the bridging edge for a net can be the forward cut edge from \( X' \) to \( \overline{X'} \). Since a net has exactly one bridging edge, if it is cut, it contributes exactly 1 in the min-cut in \( G' \). On the other hand, only a cut net will be counted in the min-cut. Therefore, the min-cut size in \( G' \) equals to the number of cut-nets in \( G \).

**Lemma 2:** Any min-cut in \( G' \) corresponds to a minimum uni-directional cut in \( G \).

**Proof:** For a min-cut \((X', \overline{X'})\) in \( G' \), all the forward edges from \( X' \) to \( \overline{X'} \) must be saturated after the max-flow computation and all the backward edges from \( \overline{X'} \) to \( X' \) have zero amount of flow. In both the two-terminal and multi-terminal net modeling for nets in \( N_c \) and \( N_f \), for any two nodes \( v, u \), if \( P(v) \leq P(u) \), then there is an edge from node \( u \) to \( v \) with capacity \( \infty \). So it will never happen that \( u \) is in \( \overline{X'} \) and \( v \) is in \( X' \). Thus for any min-cut \((X', \overline{X'})\), either \( v \) and \( u \) are in the same partition, or \( v \in X' \) and \( u \in \overline{X'} \). Therefore a min-cut in \( G' \) corresponds to a uni-directional cut in \( G \).

We now prove that the uni-directional cut in \( G \) is minimum. Suppose there is another uni-directional cut \((Y', \overline{Y'})\) with a smaller cut size, then let \((Y', \overline{Y'})\) be the corresponding cut in \( G' \). Then \((Y', \overline{Y'})\) would be a smaller cut than \((X', \overline{X'})\), which leads to contradiction that \((X', \overline{X'})\) is a min-cut in \( G' \). Hence, the corresponding cut is a minimum uni-directional cut in \( G \).

Lemma 1 and Lemma 2 lead to the following theorem.

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

### 3.3 \( \alpha \)-bounded Uni-directional Bipartitioning

In this section, we present a network flow based uni-directional bipartitioning algorithm FBP-u. As by the max-flow min-cut computation, the nodes on one side of the min-cut can have arbitrary total weight, we apply repeated max-flow min-cut heuristic to find an \( \alpha \)-bounded uni-directional bipartition which minimizes the number of crossing nets.

**Algorithm FBP-u:**

1. Construct \( G' \) from \( G \) by net-modeling;
2. Pick a pair of nodes \( s \) and \( t \) in \( G' \) as source and sink;
3. Max-flow computation, find a min-cut \( C \) in \( G' \);
   Let \( X \) be the sub-circuit reachable from \( s \) through augmenting paths, and \( \overline{X} \) be the rest;
4. if \((1 - \epsilon) \alpha \leq w(X) \leq (1 + \epsilon) \alpha \) then return \( C \) as the answer;
5. if \( w(X) < (1 - \epsilon) \alpha \) then collapse all nodes in \( X \) to \( s \);
6. if \( w(X) > (1 + \epsilon) \alpha \) then collapse all nodes in \( \overline{X} \) to \( t \);
7. pick a node \( v \in X \), and collapse \( v \) to \( t \);
8. goto step 3;

In step 1 of algorithm FBP-u, the network \( G' \) is constructed from \( G \) by the net modeling discussed in sections 3.1 and 3.2. Step 2 selects the source \( s \) and sink \( t \). Unlike FBB [16], the source and sink cannot be selected randomly. The source \( s \) should be a node such that there is no \( v \) with \( P(v) \leq P(s) \), and the sink \( t \) should be a node such that there is no \( v \) with \( P(t) \leq P(v) \).

In step 3, a min-cut is found in \( G' \) by the max-flow computation. In step 4, if the total weight for \( X \) is within range, then return \( X \) as the result. In step 5, 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 to explore a different cut with a larger weight in \( X \). The node \( v \) collapsed to \( s \) is chosen such that for any \( u \) with \( P(u) \leq P(v) \), \( u \) is already in \( X \). In step 6, 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 \( t \) in step 6.2. The node \( v \) collapsed to \( t \) is chosen such that for any \( u \) with \( P(u) \leq P(v) \), \( u \) is already in \( \overline{X} \).

Similar to FBB [16], 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. Similar to the proof in [16], the time complexity for the repeated max-flow min-cut is asymptotically the same as one max-flow computation. The time complexity for FBP-u is \( O(|V||E|) \).

Figure 8 shows an example of finding an \( \alpha \)-bounded uni-directional bipartitioning with \( \alpha = 6 \). In the first iteration, min-cut is 1 after the max-flow computation, and \( w(X) = 1 \).
Then node $a \in X$ is collapsed to $s$ (i.e. $w(s) = 2$ now) so that more flow can be pushed through the network in the next iteration. After the max-flow in the second iteration, the min-cut size is $2$, $w(X) = 7$ and $w(\overline{X}) = 4$. Nodes in $\overline{X}$ are merged to $t$ and node $i$ from $X$ is collapsed to $t$. In the third iteration, $\text{min-cut} = 3$ and $w(X) = 5$. So $(X, \overline{X})$ forms an $\alpha$-bounded min-cut with cut size 3. We can then find the corresponding uni-directional net cut in the original netlist $G$.

4 Multi-way Precedence Constrained Partitioning

Now we present our network flow based multi-way precedence constrained partitioning algorithm FBP-m. Since each cut $(\cup_{j \leq i} V_{j}, \cup_{j > i} V_{j})$ between the two adjacent stages $i$ and $i+1$ (for $1 \leq i < k$) must be uni-directional, we repeatedly apply the bipartitioning algorithm FBP-u $k-1$ times to find a uni-directional cut.

Since the length of the critical path is usually longer than the number of stages, there will be more than one levels of nodes in one stage. Let $depth$ be the number of nodes on the longest critical path in the netlist. For sequential circuits, $depth$ is the longest path of the combinational part between flip-flops. When partitioning into $k$ stages, the number of levels in one stage is $L = \lceil \frac{\text{depth}}{k} \rceil$ in order to make the design as fast as possible. If the time limit for each stage is known a priori, then the number of levels can be calculated accordingly. Since we also want to minimize the maximum number of nodes in any stage in order to allow the design to fit into a smaller physical FPGA, we want to make each stage have weight as close to the average, $\frac{w(V)}{k}$, as possible (i.e. $\alpha = \frac{w(V)}{k}$).

Algorithm FBP-m:

Flow-based multi-way precedence constrained partitioning
begin
1. perform ASAP and ALAP scheduling;
   let $AS(v)$ be the stage assigned to $v$ in ASAP scheduling; 
   let $AL(v)$ be the stage assigned to $v$ in ALAP scheduling; 
2. for $i = 1$ to $k$ do 
   $P_{i} = \{ v | AS(v) = AL(v) = i \};$
3. for $i = 1$ to $k - 1$ do 
   begin
      3.1. $s = (\cup_{j = i-1} V_{j}) \cup P_{i}$, and let $w(s) = w(P_{i});$
      $t = \{ v | AS(v) > i \}$, and let $w(t) = w(P_{i+1});$
      3.2. $F = \{ v | AS(v) \leq i, \text{ s.t. } v \notin V_{j}, 1 \leq j < i \};$
      3.3. construct $F'$ from $F \cup s \cup t$ by net modeling;
      3.4. find an $\alpha$-bounded uni-directional min-net-cut $(X, \overline{X})$ by algorithm FBP-u;
      3.5. assign nodes in $X$ to stage $i$, $V_{i} = P_{i} \cup (X - s);$ 
      3.6. for $v \in F$ with $AL(v) = i$, assign $s(v) = i + 1;$
   end
end

The partitioning process has three major steps.
Step 1 performs 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. In the ALAP scheduling, each node is assigned to the latest possible stage. For each node $v$, let $AS(v)$, $AL(v)$ be the stage assigned to $v$ in the ASAP, ALAP scheduling respectively.

We decide $AS(v)$ and $AL(v)$ as follows. In the ASAP scheduling, each node $v$ is first labeled with the earliest level by the breadth search. Let $e(v) = \{ u | P(u) \leq P(v) \}$ be a subset of nodes which have a higher precedence than $v$, let $l_{v}$ be the earliest level for $v$. If $e(v) = \phi$, then $l_{v} = 1$, else $l_{v} = \max[\{ u | v \in e(u) \}] + 1$. The earliest stage for $v$ is $AS(v) = \lceil \frac{l_{v}}{k} \rceil$. In the ALAP scheduling, each node is first labeled with the latest level by the breadth search. Let $e'(v) = \{ u | P(u) \geq P(v) \}$ be a subset of nodes which have lower precedence than $v$, and let $l'_{v}$ be the latest level for $v$. If $e'(v) = \phi$, then $l'_{v} = \text{depth}$, else $l'_{v} = \min[\{ l(u) | u \in e'(v) \}] - 1$. Then the latest stage for $v$ is $AL(v) = \lfloor \frac{l'_{v}}{k} \rfloor$.

Each node $v$ is assigned an interval $[AS(v), AL(v)]$ after the ASAP and ALAP scheduling. If $AS(v) = AL(v) = i$, then $v$ must be scheduled in stage $i$. In this case we call $v$ a fixed node. If $AS(v) < AL(v)$, then $v$ can be assigned to any stage from $AS(v)$ to $AL(v)$. We call $v$ a flexible node.

In step 2, let $P_{i}$ be the subset of nodes fixed to stage $i$ ($1 \leq i \leq k$) based on the ASAP and ALAP scheduling, i.e. $P_{i} = \{ v | AS(v) = AL(v) = i \}$. Note that the nodes on a critical path are fixed, but many nodes on the non-critical paths are flexible to be assigned to different stages. The assignment of a fixed node influences other nodes by the precedence constraints. 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 number of interconnections between the stages.

Step 3 iteratively calls the network flow based bipartitioning algorithm FBP-u 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 to find $V_{i}$ are as follows.

In step 3.1, the source $s$ and sink $t$ of the network are decided. The source is a subset of nodes where $s = (\cup_{j = i}^{i+1} V_{j}) \cup P_{i}$, 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

Figure 8: Example of $\alpha$-bounded uni-directional bipartitioning.
which can only be put in a stage later than \(i\). In step 3.2, all the flexible nodes that can be put in stage \(i\) or \(i + 1\) form a subset \(F = \{v \mid AS(v) \leq i, s.t. v \not\in V_i, 1 \leq j < i\}\). Notice that nodes in \(F\) can either be put in stage \(i\) or \(i + 1\). We want to find a uni-directional min-cut \((X, X)\) in \(F\) such that \(X\) has the desired total weight.

In step 3.3, network \(F'\) is constructed from \(F \cup s \cup t\) by the net modeling. Algorithm FBP-u is applied on \(F'\) to find an \(\alpha\)-bounded min-cut \((X, X)\). 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. \(P_{i+1} = P_i \cup \{v \mid AL(v) = i + 1\}\).

Then \(i\) is increased by 1 and control goes back to step 3.1 to start the next iteration. In step 1, the ASAP and ALAP scheduling takes \(O([V]|E|)\) time. Each iteration in step 3 takes \(O(|V| \cdot |E|)\) time, so the time complexity for algorithm FBP-m is \(O(k|V|\cdot|E|)\).

![Figure 9](image_url)

Figure 9: Example of multi-way partitioning into 3 stages.

Figure 9 shows a simple example of multi-way partitioning into 3 stages. The depth of the critical path is 9, each stage has a maximum of 3 levels of nodes. Since there are a total number of 15 nodes, each stage should have an average of 5 nodes assuming each node has the same weight. Figure 9(a) shows the \([AS(v), AL(v)]\) interval for each node after the ASAP and ALAP scheduling. Nodes on the critical path are fixed, such that \(P_1 = \{a, b, c, e\}\), \(P_2 = \{f, i, j\}\), and \(P_3 = \{t, n, o\}\). Node \(d\) is a flexible node that can be either put in stage 1 or 2, and node \(g\) can be put in stages 1 to 3. Figure 9(b) shows the partitioning to find a cut between stage 1 and 2. First \(s, t, F\) and \(F'\) are constructed, where \(F = \{d, g, h\}\), \(w(s) = 4\) and \(w(t) = 3\). Then \(F'\) is constructed from \(F\) and a min-cut with size 3 is found by max-flow computation, and \(V_1 = \{a, b, c, d, e\}\). Since \(AL(h) = 2\), \(h\) is assigned to stage 2. Figure 9(c) shows the next iteration to find a cut between stage 2 and 3, and Figure 9(d) shows the final 3-way partitioning result.

lemmas 3 and 4 show the correctness of algorithm FBP-m.

**Lemma 3:** 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 a stage \(i\) in step 3.5, else \(v\) is assigned to stage equal to \(AL(v)\) 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, \(AS(v) \leq s(v) \leq AL(v)\).

**Lemma 4:** The precedence constraints among the nodes are satisfied, i.e. for any two nodes \(u\) and \(v\), if \(P(v) \subseteq P(u)\), then \(s(v) \leq s(u)\).

**Proof:** Let \(S_i\) be the set of nodes which are assigned to a stage later than \(i\), such that \(S_i = \cup_{j \geq i} V_j\). First, we prove that between the nodes in \(V_i\) and nodes in \(S_i\), the precedence constraints are satisfied. Since when deciding stage \(i\), all the nodes with \(AS(v) > i\) are in the sink \(t\) and by the net modeling and algorithm FBP-u, the min-cut found is uni-directional, therefore, nodes in \(X\) preserve precedence constraints with nodes in \(S_i\). So the precedence constraints are satisfied among nodes in \(V_i\) and \(S_i\). Since this is true for all stages \(1 \leq i \leq k\), the nodes in stage \(i\) and \(j\) \((i < j)\) meet the precedence constraints. Therefore, all the nodes satisfy the constraints in the multi-way partitioning.

## 5 Experimental Results

We implemented algorithms FBP-u and FBP-m in C++ on Intel Pentium-Pro of 200Mz with 32MB memory, and experimented on the MCNC Partitioning93 benchmark circuits. Table 1 shows the characteristic of the MCNC benchmark circuits. In column 5, depth refers to the number of nodes on the longest critical path.

<table>
<thead>
<tr>
<th>Circuit</th>
<th># Nodes</th>
<th># Nets</th>
<th># PIO</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3540</td>
<td>1038</td>
<td>1016</td>
<td>72</td>
<td>38</td>
</tr>
<tr>
<td>c5315</td>
<td>1178</td>
<td>1055</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>c6268</td>
<td>2856</td>
<td>2824</td>
<td>64</td>
<td>14</td>
</tr>
<tr>
<td>c7552</td>
<td>2247</td>
<td>2140</td>
<td>313</td>
<td>29</td>
</tr>
<tr>
<td>c1423</td>
<td>831</td>
<td>750</td>
<td>26</td>
<td>61</td>
</tr>
<tr>
<td>c52 1</td>
<td>340</td>
<td>314</td>
<td>41</td>
<td>12</td>
</tr>
<tr>
<td>c3586</td>
<td>495</td>
<td>459</td>
<td>41</td>
<td>57</td>
</tr>
<tr>
<td>c2334</td>
<td>600</td>
<td>581</td>
<td>53</td>
<td>69</td>
</tr>
<tr>
<td>c13207</td>
<td>944</td>
<td>865</td>
<td>150</td>
<td>60</td>
</tr>
<tr>
<td>c15850</td>
<td>1107</td>
<td>1038</td>
<td>105</td>
<td>83</td>
</tr>
<tr>
<td>c59532</td>
<td>1988</td>
<td>1783</td>
<td>359</td>
<td>31</td>
</tr>
<tr>
<td>c3417</td>
<td>2589</td>
<td>2824</td>
<td>138</td>
<td>84</td>
</tr>
<tr>
<td>c38584</td>
<td>2247</td>
<td>2079</td>
<td>284</td>
<td>57</td>
</tr>
</tbody>
</table>

Because of the precedence constraints, all the related research [1, 2, 3] used a variance of list scheduling heuristic. List scheduling labels each node with a priority and the nodes are greedily assigned to a stage one at a time according to its priority. The assignment of one node influences the priority of its neighboring nodes. Our experiments show that
Table 2: Comparing partitioning results of uni-directional bipartitioning

<table>
<thead>
<tr>
<th>Circuit</th>
<th>List</th>
<th>FBP-m (k = 2)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cut nets</td>
<td>runtime(sec.)</td>
<td>Improv.</td>
<td>cut nets</td>
<td>runtime(sec.)</td>
<td>Improv.</td>
</tr>
<tr>
<td>c3540</td>
<td>127</td>
<td>0.14</td>
<td>23.6%</td>
<td>97</td>
<td>0.25</td>
<td>30.5%</td>
</tr>
<tr>
<td>c3315</td>
<td>151</td>
<td>0.25</td>
<td>30.5%</td>
<td>105</td>
<td>0.33</td>
<td>30.4%</td>
</tr>
<tr>
<td>c7288</td>
<td>102</td>
<td>0.03</td>
<td>30.4%</td>
<td>71</td>
<td>0.04</td>
<td>18.6%</td>
</tr>
<tr>
<td>c7552</td>
<td>382</td>
<td>0.19</td>
<td>64.5%</td>
<td>311</td>
<td>0.19</td>
<td>64.5%</td>
</tr>
<tr>
<td>s1423</td>
<td>152</td>
<td>0.01</td>
<td>35.3%</td>
<td>64</td>
<td>0.01</td>
<td>35.3%</td>
</tr>
<tr>
<td>s20</td>
<td>68</td>
<td>0.01</td>
<td>35.3%</td>
<td>44</td>
<td>0.01</td>
<td>35.3%</td>
</tr>
<tr>
<td>s38</td>
<td>122</td>
<td>0.03</td>
<td>72.9%</td>
<td>33</td>
<td>0.03</td>
<td>72.9%</td>
</tr>
<tr>
<td>s9234</td>
<td>550</td>
<td>1.24</td>
<td>54.7%</td>
<td>249</td>
<td>1.24</td>
<td>54.7%</td>
</tr>
<tr>
<td>s13207</td>
<td>1192</td>
<td>3.44</td>
<td>75.5%</td>
<td>292</td>
<td>3.44</td>
<td>75.5%</td>
</tr>
<tr>
<td>s15850</td>
<td>1154</td>
<td>1.63</td>
<td>78.6%</td>
<td>247</td>
<td>1.63</td>
<td>78.6%</td>
</tr>
<tr>
<td>s35932</td>
<td>3973</td>
<td>1.88</td>
<td>77.9%</td>
<td>838</td>
<td>1.88</td>
<td>77.9%</td>
</tr>
<tr>
<td>s38417</td>
<td>3896</td>
<td>75.01</td>
<td>73.8%</td>
<td>1022</td>
<td>75.01</td>
<td>73.8%</td>
</tr>
<tr>
<td>s38584</td>
<td>5127</td>
<td>57.72</td>
<td>93.8%</td>
<td>316</td>
<td>57.72</td>
<td>93.8%</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>56.1%</td>
</tr>
</tbody>
</table>

Table 3: Partitioning into 4 stages

<table>
<thead>
<tr>
<th>Circuit</th>
<th>List</th>
<th>FBP-m (k = 4)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max comm</td>
<td>ave comm</td>
<td>max comm</td>
<td>ave comm</td>
<td>runtime(sec.)</td>
<td>max comm</td>
</tr>
<tr>
<td>c3540</td>
<td>149</td>
<td>90</td>
<td>104</td>
<td>69</td>
<td>0.21</td>
<td>30.2%</td>
</tr>
<tr>
<td>c3315</td>
<td>261</td>
<td>149</td>
<td>133</td>
<td>80</td>
<td>0.44</td>
<td>4.00%</td>
</tr>
<tr>
<td>c7288</td>
<td>111</td>
<td>48</td>
<td>102</td>
<td>69</td>
<td>0.05</td>
<td>8.1%</td>
</tr>
<tr>
<td>c7552</td>
<td>424</td>
<td>253</td>
<td>395</td>
<td>214</td>
<td>0.57</td>
<td>9.00%</td>
</tr>
<tr>
<td>s1423</td>
<td>129</td>
<td>106</td>
<td>101</td>
<td>88</td>
<td>0.05</td>
<td>21.7%</td>
</tr>
<tr>
<td>s20</td>
<td>81</td>
<td>54</td>
<td>88</td>
<td>49</td>
<td>0.05</td>
<td>3.7%</td>
</tr>
<tr>
<td>s38</td>
<td>131</td>
<td>91</td>
<td>76</td>
<td>52</td>
<td>0.05</td>
<td>42.0%</td>
</tr>
<tr>
<td>s9234</td>
<td>569</td>
<td>447</td>
<td>497</td>
<td>382</td>
<td>2.47</td>
<td>12.7%</td>
</tr>
<tr>
<td>s13207</td>
<td>1116</td>
<td>951</td>
<td>773</td>
<td>698</td>
<td>4.80</td>
<td>30.7%</td>
</tr>
<tr>
<td>s15850</td>
<td>1041</td>
<td>892</td>
<td>719</td>
<td>637</td>
<td>4.61</td>
<td>30.9%</td>
</tr>
<tr>
<td>s35932</td>
<td>4376</td>
<td>3256</td>
<td>2909</td>
<td>2442</td>
<td>4.02</td>
<td>33.8%</td>
</tr>
<tr>
<td>s38417</td>
<td>3351</td>
<td>2729</td>
<td>2493</td>
<td>2178</td>
<td>5.59</td>
<td>25.6%</td>
</tr>
<tr>
<td>s38584</td>
<td>4914</td>
<td>3665</td>
<td>1944</td>
<td>1748</td>
<td>17.29</td>
<td>60.4%</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Partitioning into 8 stages

<table>
<thead>
<tr>
<th>Circuit</th>
<th>List</th>
<th>FBP-m (k = 8)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max comm</td>
<td>ave comm</td>
<td>max comm</td>
<td>ave comm</td>
<td>runtime(sec.)</td>
<td>max comm</td>
</tr>
<tr>
<td>c3540</td>
<td>177</td>
<td>106</td>
<td>166</td>
<td>84</td>
<td>0.31</td>
<td>6.5%</td>
</tr>
<tr>
<td>c3315</td>
<td>265</td>
<td>158</td>
<td>165</td>
<td>107</td>
<td>0.78</td>
<td>37.3%</td>
</tr>
<tr>
<td>c7288</td>
<td>117</td>
<td>83</td>
<td>114</td>
<td>80</td>
<td>0.15</td>
<td>2.6%</td>
</tr>
<tr>
<td>c7552</td>
<td>455</td>
<td>275</td>
<td>392</td>
<td>216</td>
<td>0.54</td>
<td>13.5%</td>
</tr>
<tr>
<td>s1423</td>
<td>130</td>
<td>107</td>
<td>120</td>
<td>96</td>
<td>0.10</td>
<td>7.7%</td>
</tr>
<tr>
<td>s20</td>
<td>91</td>
<td>56</td>
<td>87</td>
<td>54</td>
<td>0.01</td>
<td>12.3%</td>
</tr>
<tr>
<td>s38</td>
<td>131</td>
<td>93</td>
<td>71</td>
<td>54</td>
<td>0.05</td>
<td>45.8%</td>
</tr>
<tr>
<td>s7234</td>
<td>640</td>
<td>484</td>
<td>502</td>
<td>420</td>
<td>2.00</td>
<td>21.6%</td>
</tr>
<tr>
<td>s13207</td>
<td>1118</td>
<td>971</td>
<td>901</td>
<td>885</td>
<td>5.00</td>
<td>29.4%</td>
</tr>
<tr>
<td>s15850</td>
<td>1070</td>
<td>899</td>
<td>877</td>
<td>927</td>
<td>3.97</td>
<td>18.0%</td>
</tr>
<tr>
<td>s35932</td>
<td>3806</td>
<td>2884</td>
<td>2950</td>
<td>2463</td>
<td>19.14</td>
<td>22.7%</td>
</tr>
<tr>
<td>s38417</td>
<td>3546</td>
<td>2834</td>
<td>2892</td>
<td>2531</td>
<td>218.45</td>
<td>18.4%</td>
</tr>
<tr>
<td>s38584</td>
<td>5131</td>
<td>3640</td>
<td>2796</td>
<td>2291</td>
<td>84.86</td>
<td>45.3%</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
network flow based partitioning method can be successfully used to solve the scheduling problem and has the advantage of yielding much smaller net cuts.

Table 2 compares the number of uni-directional cut nets of bipartitioning by FBP-m (when $k = 2$) with that of list scheduling. The number of levels in each of the two stages is set to be $\left\lceil \frac{\text{net depth}}{2} \right\rceil$ and $\gamma$ is set to be $\frac{\text{net depth}}{2}$ with $\pm 5\%$ variation.

From our experiments on all the benchmark circuits, FBP-m results in fewer number of cut nets with an average of 56% improvement over list scheduling. It is observed that our algorithm consistently achieved larger improvement on the larger circuits. Table 2 also shows the CPU time for FBP-m. Since incremental flow computation is employed, FBP-m is very efficient. For netlists with more than 20000 nodes, FBP-m only takes one minute of CPU time.

In the next set of experiment, we performed multi-way partitioning into 3 to 12 stages by FBP-m. Tables 3 and 4 show the experimental results of partitioning into 4 and 8 stages, and compare the communication cost with that of list scheduling. Columns 2 and 4 show the maximum number of micro registers required to store values of the cut nets to be used by a later stage or the next user cycle. If the nodes in a net $n \in N_s$ span from stage $s_1$ to $s_2$, then one micro register is used in stages from $s_1$ to $s_2 - 1$ to store the value. For a net $n \in N_f$, the value from a FF-node will need to be stored in a micro register to be passed to the next user cycle.

Minimizing the number of communications among the stages will facilitate the task of placement and routing. Columns 3 and 5 show the average number of micro registers for a stage. Column 6 shows the run-time of FBP-m. Columns 7 and 8 show the percentage of improvement of FBP-m over list scheduling. All of our experiments show that FBP-m outperforms list scheduling by a big margin with an average improvement over 20%.

Here we used the flat netlists from MCNC benchmark in the experiment, instead of the CLB-level netlists which are derived after the technology mapping step. Our algorithm can easily run on those CLB-level netlists.

From the experiments we show that with proper net modeling, the network flow based approach is suitable for solving the precedence constrained partitioning problem. First, the net modeling captures the precedence of the nodes, so that a min-cut satisfies the precedence constraints. Second, by using ASAP and ALAP scheduling first, some nodes are fixed in a certain stage. Therefore, instead of random selection, the source and sink are prefixed before partitioning between stages $i$ and $i + 1$. This facilitates the max-flow min-cut computation. Next, the repeated max-flow min-cut computation balances the number of nodes in each stage.

Our FBP-m algorithm can also be applied to partitioning for time-multiplexed I/O and buffer minimization [3], and pipeline design. Algorithm FBP-u finds many other applications in logic synthesis and placement algorithms [10, 11, 12].

### 6 Conclusion

Time-multiplexed FPGAs have the potential to dramatically improve logic density by time-sharing logic, and have become an active research area for reconfigurable computing.

We present a network flow based method for precedence constrained partitioning. We first give a general net modeling for both combinational and sequential circuits to find a uni-directional min-net-cut in a netlist, then algorithm FBP-u is developed which used repeated max-flow min-cut computation to find an $\alpha$-bounded uni-directional min-net-cut. Algorithm FBP-m iteratively applies algorithm FBP-u to find a multi-way precedence constrained partitioning. All previous works for precedence constrained partitioning have used list scheduling, our experiments show that the network flow based algorithm out-performs list scheduling by a big margin.

### References


