Optimal Temporal Partitioning based on Slowdown and Retiming

Christian Plessl [#1], Marco Platzner [*2], Lothar Thiele [#3]

[#]*Computer Engineering and Networks Lab, ETH Zurich*
8092 Zurich, Switzerland
[1]plessl@tik.ee.ethz.ch
[3]thiele@tik.ee.ethz.ch

[*]*Department of Computer Science, University of Paderborn*
33098 Paderborn, Germany
[2]platzner@upb.de

***Abstract*— This paper presents a novel method for optimal temporal partitioning of sequential circuits for time-multiplexed reconfigurable architectures. The method bases on slowdown and retiming and maximizes the circuit's performance during execution while restricting the size of the partitions to respect the resource constraints of the reconfigurable architecture. We provide a mixed integer linear program (MILP) formulation of the problem, which can be solved exactly. In contrast to related work, our approach optimizes performance directly, takes structural modifications of the circuit into account, and is extensible. We present the application of the new method to temporal partitioning for a coarse-grained reconfigurable architecture.**

I. Introduction

Multi-context reconfigurable architectures (also denoted as time-multiplexed architectures) enable a number of novel application modes that rely on rapid dynamic reconfiguration. Multi-context FPGAs have been introduced to overcome resource limitations of early FPGAs, e. g. [1], [2]. Later on, also coarse-grained reconfigurable architectures started to include multi-context features.

This work is concerned with the time-multiplexed execution of sequential circuits on such architectures. If a circuit exceeds the size of one context, temporal partitioning decomposes the circuit into a set of communicating subcircuits. Temporal partitioning is performed at compile-time. At runtime, the subcircuits are executed in a cyclic schedule and perform the same task as the original circuit – with less hardware resources, but at the expense of a reduced performance. The communication between subcircuits requires to store and share intermediate data among the different contexts. To enable this inter-context communication, the reconfigurable architecture has to provide adequate storage resources.

Any method for temporal partitioning has to deal with two types of constraints: First, the circuit's netlist must be decomposed and mapped to partitions, such that the precedence constraints between the netlist elements are satisfied. Second, the decomposition and mapping must respect the architecture's resource constraints for both logic elements and inter-context communication. Earlier work on multi-context FPGAs formulated temporal partitioning as a precedence-constrained scheduling problem [3]. Other approaches used approximation algorithms [4], [5], [6] and, to a far lesser extend, exact methods [7]. All these previous approaches treat the minimization of used logic elements and inter-context communication resources as objectives.

The main contribution of this paper is a new method for temporal partitioning that is conceptually different from related work. We start with a circuit's netlist and generate partitions by applying slowdown and retiming, two transformations well-known in digital design. While these transformations increase the performance of the temporally partitioned circuit by modifying the circuit structure, they fully preserve the circuit's function. We formulate the resulting problem as a mixed integer linear program, which can be solved to optimality.

Although we have developed the method for our coarse-grained ZIPPY [8] architecture, we also adopt the TMFPGA architecture model [3] that has been used in related work on temporal partitioning for time-multiplexed FPGAs. Two features of the TMFPGA architecture are relevant to this work: (a) the architecture stores multiple configurations on-chip and can switch rapidly between them, (b) each configurable logic block (CLB) provides an output register-file with one dedicated register per context.

II. Approach

The input to the temporal partitioning algorithm is the technology mapped netlist of a synchronous digital circuit consisting of combinational operators, registers, primary inputs and primary outputs.

Fig. 1(a) illustrates the implementation of a synchronous circuit in a single partition. All signals originate at a register and propagate through purely combinational logic that computes the new contents of the registers.

Our new temporal partitioning method uses two well-known digital design techniques known as *slowdown* and *retiming*. Fig. 1 illustrates the basic idea of the method with an example in which a circuit is partitioned into two partitions. Fig. 1(b) shows how a 2-slow transformation of the initial circuit is obtained by replacing each register with a sequence of two

Published in: *Proc. Int. Conf. on Field Programmable Technology (ICFPT)*
 DOI: 10.1109/FPT.2006.270344

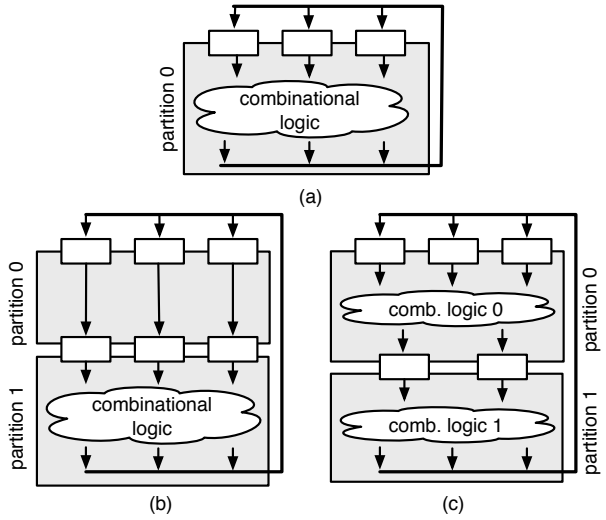

Fig. 1. Concept of Temporal Partitioning with Slowdown and Retiming

registers. A 2-slow circuit performs the same operation as the initial circuit, but works at $1/2$ of the data-rate. That is, each operator works only in every 2nd clock cycle. We interpret this 2-slow circuit as a special case of a partitioning into 2 partitions, where partition 0 is empty. The data-transfer between partitions is implemented with registers. Since partition 0 is empty, it simply forwards the unmodified register contents.

To balance the size of the partitions we use *retiming*. Retiming redistributes registers in a sequential circuit such that the critical path is minimized while the circuit's functionality is preserved. Fig. 1(c) indicates that the combinational logic is distributed into two partitions. We apply additional constraints to the retiming process, to ensure that the partitions satisfy the resource constraints of the architecture.

III. Basic Problem Formulation

This work is founded on two fundamental digital design techniques called *retiming* and *slowdown* [9].

A. Retiming

Retiming minimizes the critical path in a digital circuit by adding and removing registers while preserving the functional behavior of the circuit. Retiming models a circuit as a graph $G = (V, E, w, d)$ that consists of a set of vertices V and a set of edges E. The vertices $v \in V$ model combinational operators in the circuit and are annotated with a propagation delay $d(v)$. Directed edges $(u \xrightarrow{e} v) \in E$ model the connections between operators. Each edge is annotated with a weight $w(e)$ which represents the number of registers along this path in the circuit. Primary inputs and outputs are modeled as ordinary vertices with a propagation delay of 0.

To restrict the graph models to circuits with a physical implementation as synchronous digital circuits, we require that all propagation delays and edge-weights are non-negative ($d(v) \geq 0, w(e) \geq 0$) and that no combinational feedback exists, i. e., the path weight of any cycle in G is strictly positive.

The minimum feasible clock period is a metric for the performance of a circuit and is bounded by the longest purely combinational path in the circuit, which is denoted as *critical path* $\Phi(G)$.

Retiming computes a vertex labeling that assigns each vertex v_i a retiming value $r(v_i)$. The retimed circuit has the same structure as the original circuit, but the edges $u \xrightarrow{e} v$ have a different weight $w_r(e)$ which is defined as $w_r(e) = w(e) + r(u) - r(v)$.

There exists a retiming G_r of G such that the critical path $\Phi(G_r) \leq c$ if and only if the Mixed-Integer Linear Program (MILP) given in Equations 1–4 has a solution. The variables $s(v)$ are auxiliary variables and $c_{\max} \in \mathbf{R}^+$ is an upper bound on the length of $\Phi(G_r)$. We treat c as the optimization objective and solve the optimization problem defined by Eq. 1–4 to compute a circuit implementation that maximizes performance, i. e., $\Phi(G_r)$ is minimal.

For all operators $v \in V$ and edges $(u \xrightarrow{e} v) \in E$, minimize c such that:

$$r(u) - r(v) \leq w(e) \cdot P \tag{1}$$

$$s(v) \geq d(v) \tag{2}$$

$$s(v) \leq c \tag{3}$$

$$r(u) - r(v) + \frac{s(u) - s(v)}{c_{\max}} \leq P \cdot w(e) - \frac{d(v)}{c_{\max}} \tag{4}$$

where $s(v) \in \mathbf{R},\ 0 < s(v) < c_{\max}, r(v) \in \{0, \ldots, P-1\}$.

B. Slowdown

In a P-slow circuit each register is replaced by a sequence of P registers, i. e., each edge weight is multiplied with the slowdown factor P, see Eq. 1 and Eq. 4. For optimal temporal partitioning we exploit the fact that the P-slow transformation decomposes the circuit into P completely independent partitions. Each operator needs to run only every P-th cycle and the inputs and outputs are evaluated also only every P-th cycle. These properties are also preserved if the slowed-down circuit is additionally retimed to minimize the critical path.

C. Partitioned Execution and Performance

The partition to which a node v is mapped directly corresponds to the node's retiming value $r(v)$. While the unconstrained retiming process maximizes performance, it does not consider resource constraints. Sec. IV discusses additional constraints, that we apply to the $r(v)$ values to limit the number of operators that are mapped to a particular partition and to model resource requirements imposed by inter-partition communication.

All operators in the same partition define a sub-netlist which is implemented in one configuration of the reconfigurable architecture. At runtime, these contexts are executed in the order c_0, c_1, ..., c_{P-1}, each for a single cycle. Since the temporal partitioned circuit is P-slow, all P contexts must be executed once to perform the same computation as the initial circuit computes in a single cycle.

In our model, each context is executed for the same cycle time, that is bounded by the critical path $\Phi(G_r) = c$ of the

retimed circuit. The relative performance η of the temporally partitioned execution relative to the execution of the initial circuit is thus given as $\eta = \Phi(G)/(\Phi(G_r) \cdot P)$. Since $\Phi(G)$ is constant, optimal temporal partitioning should minimize $\Phi(G_r) \cdot P$. This objective function is non-linear in P and hence cannot be directly optimized by a MILP solver. But since P is bounded by the number of physical contexts in a concrete reconfigurable architecture, the optimal value for P can be determined by binary or exhaustive search, i. e., solving the MILP for all possible values of P.

IV. Resource Constraints

This section introduces the resource constraints posed by the execution architecture and their mathematical modelling. We introduce a binary variable $x_{i,p}$ per operator and partition to formulate mapping, capacity and communication constraints. $x_{i,p} = 1$ if node v_i is mapped to partition p, otherwise $x_{i,p} = 0$. Eq. 5 defines that each operator must be mapped to exactly one partition, and the set of partitions to which an operator can be mapped to is restricted to $r(v_i) \in \{0, \ldots, P-1\}$ in Eq. 6. The actual value $r(v_i)$ for each node is computed by retiming.

$$\sum_{p=0}^{P-1} x_{i,p} = 1, \quad \forall v_i \in V \tag{5}$$

$$r(v_i) = \sum_{p=0}^{P-1} p \cdot x_{i,p}, \quad \forall v_i \in V, r(v_i) \in \{0, \ldots, P-1\} \tag{6}$$

The implementation of a partition in a context of the architecture requires resources for implementing the operator nodes and additional resources for inter-context communication. We account for these costs by defining two *cost* variables: $nodecost$ and $comcost$.

For modelling the resource demand we subdivide the operators into three subsets that represent the combinational operators (V_{op}), the primary inputs (V_{pi}), and the primary outputs (V_{po}).

A. Operator Resource Demand

We define $nodecost_{i,p}$ as the resource demand caused in partition p by the operator v_i. For this work, we assume unit cost, i. e., $nodecost_{i,p} = x_{i,p} \quad \forall v_i \in V_{\text{op}}$. Primary input and output nodes are distinct from the CLBs and hence do not reduce the capacity for implementing combinational operators: $nodecost_{i,p} = 0 \quad \forall v_i \in \{V_{\text{pi}} \cup V_{\text{po}}\}$ We assume, that the reconfigurable architecture provides K CLBs and thus can accommodate K combinational operators. We limit the number of combinational operators mapped to each partition with the following constraint:

$$\forall p \in \{0, \ldots, P-1\} : \sum_{\forall v_i \in V} nodecost_{i,p} \leq K \tag{7}$$

B. Inter-Context Communication Resource Demand

The CLB's output register-files are used for inter-context communication, i. e., transfering the value of the circuit's nets over the boundaries of a partition. Since only one register of the register-file can be read at once, the number of intermediate signals that can be read in a context is bounded by the number of CLBs. This forms a resource constraint. At most one communication resource is used per net and partition, because as soon as an operator reads the register contents stored in a different context, the respective net becomes local to the partition and can be accessed by all operators in the same context at no additional cost.

We denote the communication resource demand caused in partition p by accessing net i as $comcost_{i,p}$, which is 1 if a net is used (at least once) but not defined in partition p. We model $comcost$ by introducing a number of auxiliary variables as follows:

$$defined_{i,p} = x_{i,p} \tag{8}$$

$$netuses_{i,p} = \sum_{j \in sinks(v_i)} x_{j,p} \tag{9}$$

$$netuses_{i,p} - used_{i,p} \cdot G_{\max} \leq 0 \tag{10}$$

$$-netuses_{i,p} - (1 - used_{i,p}) \cdot G_{\max} < 0 \tag{11}$$

where $netuses_{i,p} \in \{0, \ldots, G_{max}\}$ and $used_{i,p} \in \{0, 1\}$. Eq. 10–11 need an upper bound for $netuses$ for which we use the maximum degree of the circuit's graph $G_{\max} = \max_{\forall v_i \in V} |sinks(v_i)|$.

Using the $used$ and $defined$ variables we can now define a lower bound on $comcost_{i,p}$ as follows:

$$comcost_{i,p} \geq used_{i,p} - defined_{i,p} \quad \forall v_i \in V_{\text{op}} \tag{12}$$

$$comcost_{i,p} = 0 \quad \forall v_i \in \{V_{\text{pi}} \cup V_{\text{po}}\} \tag{13}$$

Note, that $comcost_{i,p}$ is a lower bound on the communication resource cost, rather than the actual cost. We prefer this approach, since formulating $comcost_{i,p}$ as lower bound is easier than computing the actual cost, but serves the MILP problem formulation equally well.

Since the reconfigurable architecture provides K CLBs, a partition can read at most K output register-files. We formulate the resource constraint on the communication resources as:

$$\forall p \in \{0, \ldots, P-1\} : \sum_{\forall v_i \in V} comcost_{i,p} \leq K \tag{14}$$

C. Delay Registers

Using a register for delaying a signal (as opposed to using the register for inter-context communication) adds a delay of P to the respective signal, since the register is updated only after all P contexts have been executed once. Hence, if slowdown and retiming cause edges with more than P registers, additional delay registers are required. These registers increase the resource demand, since each register is coupled with a corresponding cell.

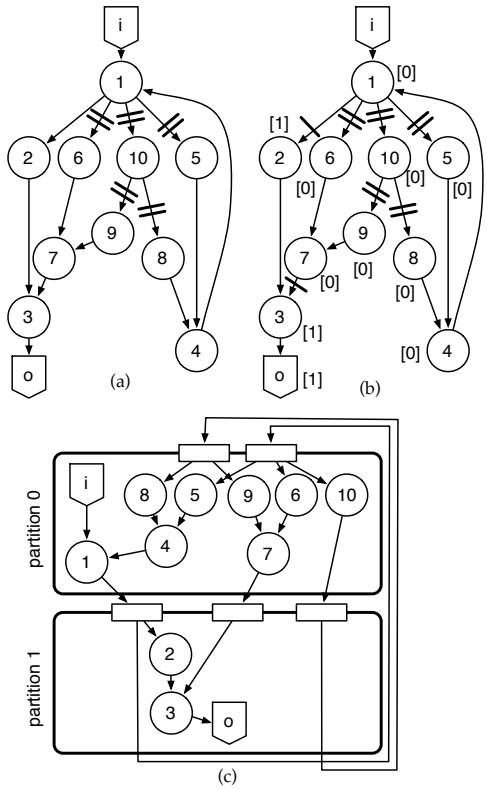

Fig. 2. Temporal Partitioning of a 2nd order IIR filter: (a) 2-slow circuit, (b) 2-slow and retimed circuit, (c) temporal partitioned circuit

In this paper, we do not present the modeling of delay registers, but we apply additional constraints to avoid delay registers. The number of registers on an edge $u \xrightarrow{e} v$ after slowdown and retiming is given as: $P \cdot w(e) + r(v) - r(u)$. To avoid delay registers, we prohibit edges with more than P registers by adding the following constraint:

$$\forall \text{ edges } u \xrightarrow{e} v : P \cdot w(e) + r(v) - r(u) \leq P \qquad (15)$$

V. Example

The presented optimal method for temporal partitioning has been developed for ZIPPY, a reconfigurable processor that integrates a coarse-grained reconfigurable array with an embedded CPU core [8]. The ZIPPY architecture is widely parametrized and features an array of 4×4 cells and 8 contexts in the default architecture. A key idea of the ZIPPY architecture is to use dynamic reconfiguration to compensate for the limited size of the reconfigurable array.

We illustrate our optimal temporal partitioning technique with the example of a 2nd-order Infinite Impulse Response (IIR) filter. Fig. 2(a) shows the schematics of the IIR2 circuit after a 2-slow transformation. The circuit is composed of 10 combinational operators and 5×2 registers. The functions of the operators are irrelevant for the temporal partitioning process and are thus not further detailed. We have implemented a tool flow that generates the MILP formulation for the temporal partitioning problem from the circuit's netlist and the architecture parameters. We solve the MILP with the CPLEX optimizer on a 900 MHz SunFire280R machine in less than 1ms using the following parameters: context capacity $K = 8$, number of partitions $P = 2$, and the upper bound for the clock period $c_{\max} = 10$. The retiming values for the nodes are annotated in brackets in Fig. 2(b). The registers in this figure already reflect the modified register placement due to retiming. The temporally partitioned implementation on a two context architecture is shown in Fig. 2(c).

VI. Discussion

In this paper we have presented a novel method for temporal partitioning of sequential circuits that can be applied to architectures with a TMFPGA-like architecture model. When comparing our approach to previous work, we see the following advantages: (a) we treat temporal partitioning as an optimization problem and optimize a *single performance-centric metric*, (b) we allow for *structural circuit modifications* to maximize the circuit's performance, (c) we use a *strict mathematical problem formulation* that can be solved *optimally*, and (d) the MILP formulation can be easily *extended* to cope with additional constraints, while the problem can still be solved optimally. A interesting extension is to allow for moving operations to later iterations (automated, optimal pipelining), which can increase the performance at the expense of additional delay.

We see two drawbacks of using a MILP problem formulation: (a) any extension to the MILP problem must be a linear constraint, and (b) in the worst case, the time for solving a MILP grows exponentially with the problem size, but there exist efficient relaxation methods that allow for approximation solutions with guaranteed bounds on the quality of the solution.

References

[1] A. DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century," in *Proc. 2nd IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, 1994, pp. 31–39.

[2] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, "A time-multiplexed FPGA," in *Proc. 5th IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 1997, pp. 22–28.

[3] S. Trimberger, "Scheduling designs into a time-multiplexed FPGA," in *Proc. 6th ACM Int. Symp. on Field-Programmable Gate Arrays (FPGA)*. ACM, 1998, pp. 153–159.

[4] D. Chang and M. Marek-Sadowska, "Partitioning sequential circuits on dynamically reconfigurable FPGAs," *IEEE Trans. on Computers*, vol. 48, no. 6, pp. 565–578, June 1999.

[5] H. Liu and D. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD)*. ACM, Nov. 1998, pp. 497–504.

[6] M. C.-T. Chao, G.-M. Wu, I. H.-R. Jiang, and Y.-W. Chang, "A clustering- and probability-based approach for time-multiplexed FPGA partitioning," in *Proc. IEEE/ACM Int. Conf. on Computer Aided Design (ICCAD)*. Piscataway, NJ, USA: IEEE Press, Nov. 1999, pp. 364–369.

[7] G.-M. Wu, J.-M. Lin, and Y.-W. Chang, "Generic ILP-based approaches for time-multiplexed FPGA partitioning," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 10, pp. 1266–1274, Oct. 2001.

[8] C. Plessl and M. Platzner, "Zippy – a coarse-grained reconfigurable array with support for hardware virtualization," in *Proc. IEEE Int. Conf. on Application-Specific Systems, Architectures, and processors (ASAP)*. IEEE Computer Society, July 2005, pp. 213–218.

[9] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry." *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.