Optimization of Hierarchical Designs Using Partitioning and Resynthesis

Heinz-Josef Eikerling†, Ralf Hunstock†, Raul Camposano††

†University of Paderborn
Warburger Str. 100
33 095 Paderborn, Germany

††Synopsys, Inc.
700 E. Middlefield Road
Mountain View, California 94 043, USA

Abstract

This paper explores the influence of optimization along the boundary between hierarchically described components. A novel technique called repartitioning combines partitioning and sequential resynthesis of the design under various quality measures. It is applied to various digital circuits which consist of a controller and a datapath. The outcome of this effort is a versatile, parametrizable resynthesis tool which preserves this hierarchy. Due to the cost measures, an average improvement ranging between 5% and 15% was obtained.

1 Introduction

High-level synthesis turns a behavioral description into a hierarchical, structural description at the RT-level. Passing through other design steps (logic and layout synthesis, simulation, verification) the initial design becomes manufacturable.

A hierarchical RTL description can be divided into two parts: the controller which is the result of scheduling operations to time-steps and the datapath which is the result of allocating modules which have to execute the specified operations. This partitioning is also used by designers most often. Both parts are connected by status and command lines; therefore, this target architecture is called *Finite State Machine with Datapath* (FSMD) [5]. In further design steps both parts are handled separately: the controller is synthesized using sequential logic synthesis and for the datapath module generators are sometimes used. A full optimization of the whole design by means of sequential logic synthesis is not possible at present.

On the logic or gate level the control/data partitioning seems to be arbitrary since it could be possible to obtain a superior circuit by incrementally moving components from one portion to another and by then applying combinational and sequential optimization to the new partitions. A simple example describing a transformation at the controller/datapath interface is shown on Figure 1. Generally, improvements can be achieved by applying state count minimization, state re-encoding, repartitioning of the automaton and Boolean optimization of the state-transition and the output logic to the selected parts.

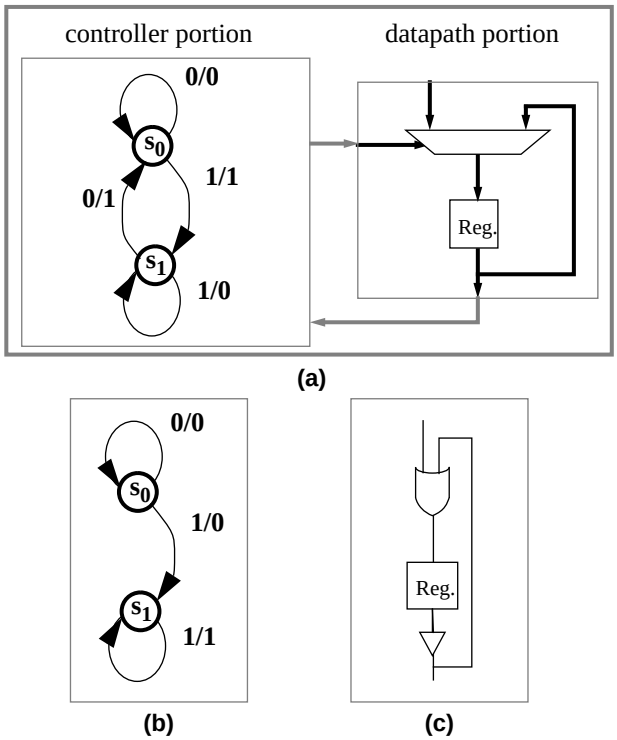

Figure 1. (a) Controller and datapath (partial implementation). (b) Reduced machine representing the product of both automata. (c) Optimized and mapped logic for (b).

Very little is known about optimization at the boundary of the datapath and the controller. Tradeoffs on behavioral level are studied by Mlinar [11]. Camposano et al. [2] combined Boolean logic synthesis of controller and datapath which is limited to rather small designs. Some logic synthesis tools perform propagation of constants (e.g., [1]) to subsequent modules in order to optimize them more efficiently. Recently, Huang and Wolf made some progress in predicting and optimizing the performance (i.e. the delay in the overall system) of control/datapath systems [8].

If the final design still does not meet the constraints specified by the designer, adjustment on lower levels must be made; this is frequently referred to as *resynthesis.*

This work introduces an optimization method for the controller / datapath repartitioning at logic level considering different aspects. We extend the previous attempts by introducing a technique for optimizing designs at the controller/datapath interface for various quality measures by combining *partitioning* with local *sequential resynthesis*. The hierarchy in the structural description is preserved; no additional information about the previous synthesis process is required.

The remainder of this paper is organized as follows: section 2 discusses cost measures and introduces the generic repartitioning framework. A new partitioning algorithm is introduced and compared to classical Mincut heuristics. In section 3 the evaluation of the cost function and the resynthesis aspects are examined. Section 4 gives some results for benchmark examples. Finally, we give conclusions and an outlook on future work.

2 Approach

As detailed below, only the problem of optimizing bipartitions is considered. The extension to the general case in which the design consists out of m hierarchical blocks (multiway partitioning) is straight-forward by decomposition into a set of bipartitioning problems. Distinguishing control- and data-dominated components, an initial partitioning is assumed to be given.

2.1 Definitions

The circuit is represented by a hierarchical *netlist* $N(V, E)$. On the top level, V denotes the set of components which can be either of combinational or sequential nature. E denotes the set of connections running between these components. These components are referred to as *instances* of *cells* having a certain behavior. $N_{cp} = (V_{cp}, E_{cp})$ and $N_{dp} = (V_{dp}, E_{dp})$ depict the controller and datapath, respectively.

$P(N) = (V_{cp}, V_{dp})$ is a *partitioning* of the circuit, i.e., a decomposition of the set of nodes N in the flat netlist into disjoint sets such that $\forall v \in V (v \in V_{cp} \vee v \in V_{dp})$ and $V_{cp} \cap V_{dp} = \emptyset$ holds. The *cost* of an implementation of a design (i.e., a circuit) under a partition is described by a vector $c_{(P(N))}$ with the following components:

- $Area(P(N))$ characterizes the area of the design.
- The communication between the controller and datapath is measured by $Comm(P(N))$. A certain weight proportional to the bitwidth of the wires connecting the subcircuits is also considered.
- $Power(P(N))$ denotes the (average) dynamic power dissipation of the circuit. Since this measure is highly data dependent, it is only estimated.
- $Time(P(N))$ is the *slack*, i.e., the maximal difference between the expected and the actual arrival of a signal at sequential elements or primary outputs. The overall performance of the system is related to this value.

In the following sections it is assumed that the above mentioned measures are additive: if $Area(N_{cp})$ and $Area(N_{dp})$ denote the area of the controller and datapath, respectively, $Area(P(N)) = Area(N_{cp}) + Area(N_{dp})$ is the calculated area of the entire circuit which is the result of combining both subcircuits[1].

The introduction of a *scalar cost function* for a circuit with respect to a given cost vector and a partition $P_{ref}(N)$ allows to distinguish better designs from worse ones by considering multiple cost metrics simultaneously, i.e., tradeoffs concerning the metrics can easily be expressed. This assessment of a partition is done by applying a real weighting vector $\langle\alpha\rangle = \langle\alpha_1, \alpha_2, \alpha_3, \alpha_4\rangle$ to the components of the cost vector

$$\hat{c}(P(N)) = \alpha_1 \cdot \frac{C_1(P(N))}{C_1(P_{ref}(N))} + \alpha_2 \cdot \frac{C_2(P(N))}{C_2(P_{ref}(N))} + \alpha_3 \cdot \frac{C_3(P(N))}{C_3(P_{ref}(N))} + \alpha_4 \cdot \frac{C_4(P(N))}{C_4(P_{ref}(N))}$$

, where for the weights $\alpha_i \in [0, 1]$, $\sum_i \alpha_i = 1$ holds and $C_i = \{Area, Comm, Time, Power\}$.

We also define the weighted cost with respect to an initial partition $P_{ref}(N)$ for one part $N^{part} \subseteq N$ of the bipartition by $\hat{c}(N^{part})$.

2.2 Starting Point

Basically, we have basically 3 possible choices to start a repartitioning a system: A *0-Partitioning* denotes a entire design to be treated as the datapath ($V_{cp} = \emptyset, V_{dp} = V$). On the other hand, starting from a *1-Partition* means the whole design is to be treated as a controller ($V_{cp} = V, V_{dp} = \emptyset$). More generally speaking, a so called *p-Partitioning* can be defined; p describes the participation of the controller in the overall cost regarding a particular weighting $\langle\alpha\rangle$ and P_{ref}; more precisely:

$$p = \hat{c}(N_{cp}) / \hat{c}(P(N)) = 1 - \hat{c}(N_{dp}) / \hat{c}(P(N))$$

The goal of the optimization is to find a partitioning $P_{opt}(N) = (V_{cp}^{opt}, V_{dp}^{opt})$ starting from an initial partition $P(N)$ for which the cost function $\hat{c}$ is minimized.

---

1. The area for the wiring of the interface between both partitions is neglected. If the system is pipelined the additional circuitry (i.e. latches) is regarded to be accumulated in one partition.

2.3 Generic Partitioning Algorithm

We will consider a class of partitioning algorithms for which during one pass cell instances (representing combinational or sequential behavior) are moved or swapped over the initial or prescribed cutline. A log which protocols each change of the configuration (i.e., moves of cell instances over the cutline) is kept.

```
P_init(N)=(N_cp,N_dp) = getInitialPartition (N);
Log.init(); cost = calculateCost(P(N));
P(N) = P_init(N); optcost = cost;
while (!abort(N, Log, cost)) {
  nodelist = getPromisingNodes (P(N)) ;
  forall (v ∈ nodelist) {
    if (v ∈ N_cp) P(N) = (N_cp \ {v}, N_dp ∪ {v});
    else P(N) = (N_cp \ {v}, N_dp ∪ {v});
    Log.append(v);
  }
  cost = calculateCost(P(N));
  if (cost < optcost)
     optcost = cost;
}
P_opt(N) = bestPrefix(P(N), Log));
```

Figure 2. The Generic Partitioning Algorithm.

Figure 2 shows the approach: Starting from an initial partition $P(N) = (V_{cp}, V_{dp})$ which is determined by *getInitialPartition* (*N*), iteratively a number of candidates (cell instances) to be considered for moving is determined by *getPromisingNodes* (*P(N)*). The selected components are moved and the cost of the new design is calculated. This step is repeated until a stopping criterion is fulfilled. By analyzing the log via the function *bestPrefix(P(N),* Log*))* the best intermediate configuration is recovered.

2.4 Resolving the Non-determinism

Some classical approaches appear as special cases of this generic algorithm. Depending on the function *getPromisingNodes* (*P(N)*) and *abort*(*N*, Log, cost) that comprises the stopping criterion several resolutions can be considered.

In order to avoid oscillations (i.e., a set of instances is repeatedly moved over the cutline) previously moved nodes are locked in their partition. If only the cost measure *Comm(P(N))* in *getPromisingNodes (P(N))* is considered and |`nodelist`| = 1, the algorithm is identical to the *Fiduccia-Mattheyses* (FM) Mincut heuristics [4] which is an improvement of the *Kernighan-Lin algorithm* (KL) [9] due to the minor number of iterations. For the KL partitioning scheme |`nodelist`| = 2 counts; furthermore it is assumed that the instances to be swapped belong to different partitions. The efficiency especially of the FM algorithm is based on a sophisticated data structure that permits one pass to be carried out in linear time.

The disadvantage of these heuristics is that they are not capable of treating more complex cost measures (area, time, power). Therefore one can make the algorithm terminate if a certain upper bound *THRESHOLD* on the cost cannot be improved; for instance, if partition $P'(N)$ results from $P(N)$ by moving the instances stored in `nodelist`, it is possible to apply the different strategies for the selection of the nodes `nodelist`.

In a *Greedy* approach `nodelist` is composed of the most promising nodes which are expected to achieve the maximal reduction of the cost $\hat{c}(P'(N))$ of the new configuration. The process is aborted when no further optimization is possible. The probability to end up in high local minimum is considerable. Significant improvement in the final cost can be achieved by applying a new optimization procedure called the G*reat Deluge Algorithm*[1] (GDA) [3] which appears to be superior to Simulated Annealing for the Travelling Salesman and other intractable problems. This method is particularly well-suited for problems which have a fairly small number of local minima.

The Deluge algorithm overcomes the disadvantage of the Greedy approach that deteriorations of the cost function are not allowed during one pass by permitting deteriorations up to a certain limit $(1+\varepsilon)$ with respect to the cost of the configuration:

$$\hat{c}(P'(N)) \leq (1+\varepsilon) \cdot \hat{c}(P_{opt}(N))$$

GDA chooses in *getPromisingNodes* (*P(N)*) an arbitrary component for which the inequality above holds; this means that

$$(1+\varepsilon) \cdot \hat{c}(P_{opt}(N)) \leq THRESHOLD$$

is satisfied. The *THRESHOLD* variable is updated and refers to the newly found optimum. For practical instances it is recommended to choose in the range $\varepsilon \approx 0.05 \ldots 0.10$. Now, the algorithm will be modified to avoid evaluating the complete design; instead of a global evaluation only a local gain analysis is performed.

2.5 Local Gain Analysis for GDA

The computation of the set of candidates is performed by estimating the success of a transformation for the given cost

1. "Deluge" refers to an illustration for maximization problems: a lower bound (water level) for the cost of the actual configuration is increased steadily. The speed of this rise is determined by ε.

function. Because each move of a set of nodes can be simulated by a move of a hierarchical module that contains all nodes of the set, the task can be reduced to the selection of one single candidate at a time.

To decide whether to accept or to dismiss a candidate $v \in V$, the region $N^{local} \subseteq N$ around $v \in V$ is analyzed. By the actual partitioning $P(N)$, a local partitioning $P(N^{local}) = (V_{cp}^{local}, V_{dp}^{local})$ on the set of nodes in this part of the netlist is induced with

$$\varnothing \subset V_{cp}^{local} \subseteq V_{cp}, \varnothing \subset V_{dp}^{local} \subseteq V_{dp}$$

If $P'(N)$ is the result of moving $v \in V$, the *absolute gain* (or global gain) is defined as:

$$gain(v) = \hat{c}(P(N)) - \hat{c}(P'(N))$$

The procedure is explained in Figure 3.

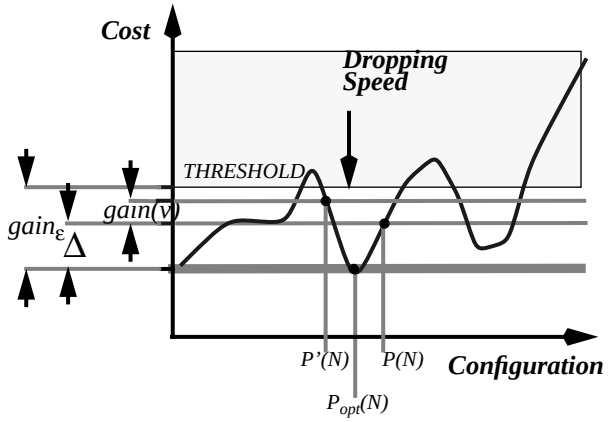

Figure 3. Illustration of the Great Deluge Algorithm; a configuration corresponds to a partition $P(N)$ which is compared to the actual optimal partition $P_{opt}(N)$. The forbidden zone is shaded.

The global gain estimation is performed by a *local gain analysis*:

$$gain^{local}(v) = \hat{c}(P(N^{local})) - \hat{c}(P'(N^{local}))$$

This is a lower bound on the overall gain because $gain(v) \geq gain^{local}(v)$ always holds. Δ describes the distance of the cost of the actual partition from the optimal cost. For the local gain analysis the following rules are established:

- A node $v \in V$ is appended to `nodelist` if
$$gain^{local}(v) > \Delta - gain_{\varepsilon}$$
holds. So, a negative gain denotes a deterioration of the cost (which is allowed within certain bounds).
- If $gain(v) > \Delta$ holds a new optimum configuration is obtained which is derived out of $P(N)$ by swapping v into the other partition, resetting
$$gain_{\varepsilon} = gain_{\varepsilon} + \varepsilon \cdot (\Delta - gain^{local}(v))$$
and setting $\Delta = 0$.
- If no new optimum is reached $\Delta = \Delta - gain^{local}(v)$ is set. The process is aborted if during the last k trials no change to Δ occurred. In this application, k depends on the cutsize because we do not want to consider members of the cutline twice.

The algorithm can be carried out without evaluating the cost of the entire design successively if $gain_{\varepsilon} = \varepsilon \cdot \hat{c}(P(N))$ is initialized with the cost of the initial partition $P_{init}(N)$. Note, that $gain_{\varepsilon} = \varepsilon$ holds if $P_{init}(N) = P_{ref}(N)$. The decision if a move should be done or not for a certain candidate is made upon the result of the local gain analysis.

3 Resynthesis

Two policies are viable for resynthesis. First, it is possible to *resynthesize after the partitioning* algorithm has terminated. Since each move depends heavily on all other previous and influences all successive moves we favor *resynthesis during the partitioning* process.

In order to get an accurate estimation of the global gain the local gain of a move is determined by a fast synthesis[1] of the region around the candidate. Now, we will explain how this region is determined.

3.1 Extracting a Subcircuit for Examination

The above mentioned heuristics are established in order to evaluate the cost of a configuration rapidly. Hence, similar to the Mincut heuristics, the instances at the border of the partitions are analyzed for their expected gains. Further constraints are implied by the size of the extracted state transition graphs: If a circuit considered for resynthesis contains i inputs, o outputs and l latches, the corresponding STG representing the behavior of the circuit is of size $|G_{st}| = o \cdot 2^i \cdot 2^l$, i.e., the STG grows exponentially with the number of latches and inputs. The problem can be stated as follows:

Find a subcircuit $N'(V', E') \subseteq N(V, E)$ for a candidate $v \in V$ of maximal size (number of internal nodes) with at most l latches and i inputs, for which $v \in V'$ holds.

Because this is a NP-hard optimization problem, we propose an *breadth-first search* (BFS) clustering process start-

1. Not all options of sequential logic synthesis are exploited.

ing from a node $v \in V$ (adjacent to the cutline) that obeys the parameters i, l. On each level of the BFS tree built during the clustering the size and the number of inputs of the cluster is analyzed. If the size of the actual cluster reaches the penalty implied by these restrictions the clustering process terminates; if not, the traced instances are added to the cluster. The algorithm can be parametrized over these two parameters.

Now, the cost of the previously determined region will be computed. This is done by simply doing Boolean optimization and mapping the region to gates separately for all four induced partitions ($N'^{local}_{cp}, N'^{local}_{dp}, N^{local}_{cp}, N^{local}_{dp}$), so that the local (estimated) gain is

$$gain^{local}(v) = \hat{c}(N^{local}_{cp}) + \hat{c}(N^{local}_{dp}) + \hat{c}(N'^{local}_{cp}) + \hat{c}(N'^{local}_{dp})$$

Since we only want an estimation representing a lower bound on the overall gain, this is sufficient.

After the gain has been determined and if the considered component is accepted for moving, the STG for the subcircuit is extracted and state-minimization and state-assignment, Boolean optimization and mapping is performed. Finally, the subcircuit is replaced by the optimized design.

3.2 Ordering the Set of Candidates

By influencing the order in which the candidates are examined the algorithm for the local gain estimation can be accelerated. Ranking the most prospective ones first leads to an improved runtime. For this ranking we only consider topological aspects which can be analyzed rather fast. We give two heuristics:

- If *Area*, *Power* and *Comm*(unication) is the main objective we find it suitable to order the set of candidates for their expected hypergain, i.e., the reduction in connection cost between the hierarchical entities.
- For speeding up the circuit (regarding *Time*) and if only pure combinational modules lying at the boundary are considered it is useful to choose that entity which balances the *sequential depth* (i.e. minimal distance over both partitions to a latch) because this increases the optimization potential concerning *Time*.

4 Experimental Results

The proposed methods have been implemented in C++ (5780 lines of code). The repartitioning tool ***cdpart*** reads in a configuration file in which the parameters (such as the number of passes, parameters for GDA, options for resynthesis etc.) can be specified.

In order to determine the relevance of the approach, several of the bigger sequential logic synthesis examples (viterbi processor, sbc, s344, mult16, mult32) were examined. Because no structural hierarchy on the top level was given, a 0.5-partition for each circuit was determined and the cost was computed by running it through SIS [12] using mcnc.genlib for mapping (was used throughout all experiments). For the estimation of power consumption which depends heavily on the input patterns the tool described in [7] was used. The development of local criteria is rather difficult because the local gain analysis technique is not capable of propagating these patterns to the local partitions. We conducted experiments for which the combined synthesis leads to only slight differences due to the measures. The initial partition then was optimized by passing it to the FM Mincut heuristics.

	MULT16	*MULT32*	*S344*	*SBC*	*VITERBI*
Area	285.00 / 200.00	617.00 / 349.00	293.00 / 68.00	1357.00 / 224.00	1541.00 / 184.00
(opt.)	218.00 / 239.00	631.00 / 277.00	291.00 / 62.00	1271.00 / 232.00	1501.00 / 112.00
Comm.	18 / 8	31 / 11	16 / 20	66 / 105	10 / 38
(opt.)	18 / 8	27 / 11	18 / 17	59 / 111	10 / 23
Slack	-390.6 / -176.80	-628.30 / -444.50	-286.30 / -42.80	-1740.6 / -184.5	-928.70 / -99.80
(opt.)	-235.40 / -239.30	-637.00 / -381.50	-286.30 / -42.80	-1408.00 / -206.40	-905.60 / -104.20
Power	50.32 / 66.13 µW	95.54 / 109.72 µW	67.35 / 35.07 µW	673.34 / 110.74 µW	476.80 / 11.70 µW
(opt.)	64.13 / 59.72 µW	89.64 / 111.38 µW	118.68 / 3.93µW	501.44 / 118.05 µW	416.60 / 23.28 µW

Table 1. Examples for which initial partition is not known.

For further optimization the designs were passed to ***cdpart*** (using GDA, a 100% weight was given to each cost measure under consideration) and the partitions were synthesized again. Table 1 shows that an average reduction for

	GCD	*DIFFEQ*	*ELLIP*	*ATOI*	*FIBON*
Area	142.00 / 495.00	284.00 / 1398.00	954.00 / 2833.00	114.00 / 1269.00	94.00 / 463.00
(opt.)	149.00 / 484.00	401.00 / 1218.00	1874.0/ 1938.0	162.00 / 952.0	233.00 / 300.00
Comm.	3 / 10	2 / 24	2 / 46	2 / 20	2 / 10
(opt.)	3 / 10	2 / 23	2 / 43	2 / 20	2 / 10
Slack	-142.90 / -299.70	-487.00 / -752.40	-1687.00 / -2425.70	-127.20 / -681.90	-116.10 / -259.50
(opt.)	-146.60 / -299.70	-761.60 / -454.00	-2294.2 / -1599.9	- 156.90 / -526.10	-231.20 / -214.60
Power	- /-	40.96 / 412.25 µW	42.47 / 71.41 µW	13.93 / 76.71 µW	- /-
(opt.)	- /-	66.18 / 308.49 µW	40.96 / 30.89 µW	11.33 / 82.47 µW	- /-

Table 2. Examples from high-level synthesis.

comm. (-10.13%), area (-6.62%), slack (-7.89%) and power (-11.16%) can be obtained.

Then another examination was done on a number of designs coming from the behavioral synthesis system PMOSS [6] (gcd, elliptic, diffeq, atoi, fibonacci). Each controller (mostly counters) was synthesized and optimized using JEDI. Once again, only slight differences to the results for the 0-partition can be reported. Table 2 shows the results for the optimized partitions (carrying out one pass of GDA). For each cost measure the above mentioned heuristics to speed-up the candidate selection have been employed. Regarding the initial partition, an average improvement for area (-5.12%), slack (-4.43%) and power (-17.96%) can be achieved when compared to the cost of the entire design. If the improvement is related to the actually resynthesized region the effects become more remarkable. Note, that this also holds when compared to the flattened design. The optimization for communication between controller and datapath is difficult for these examples because the controller was already mapped and rather little can be gained by moving single-output gates/latches from the controller over the cutline if there is no compensation by a controller input representing a feedback from the datapath. For none of the examples, a reduction by applying FM to the initial partition was possible.

5 Conclusion and Future Work

In this paper the optimization of hierarchical designs of digital systems across the boundary of controller and datapath was studied. An algorithm that is capable of optimizing a design under various cost measures was introduced and analyzed. The presented algorithms detect control-dominated parts in an initial design for which efficient redesign by means of sequential logic synthesis can be done. Based on the cost measures, an average improvement ranging from 5% to 15% was obtained. The presented methods can be applied to designs which have been automatically synthesized in order to have a optimization methods which preserves hierarchy and resynthesizes incrementally.

For huge clusters the sizes of the extracted STGs during local resynthesis increase enormously. We are working on BDD based techniques similar to those described in [10] to carry out sequential resynthesis (state count minimization, state assignment, boolean synthesis) fully implicit, i.e., without extracting the STG.

Acknowledgments

This work was supported by the DFG under grant SFB 358, project *Automated System Design*. We thank S. Devadas of MIT for supplying us with the power estimation tool and A. Hoffmann of Paderborn University for giving us helpful comments.

References

[1] Design compiler (tm) reference manual. Technical Report 3.0, Synopsys, inc., December 1992.

[2] R. Camposano and J.T.J. van Eijndhoven. Combined synthesis of control logic and datapath. In *Proc. of the ICCAD*, pages 327–329, Santa Clara, CA, 1987. ACM/IEEE.

[3] G. Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record-travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

[4] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristics for improving network partitions. In *Proc. of the 19th DAC*, pages 175–181, Miami, FL, 1982. ACM/IEEE.

[5] D.D. Gaijski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[6] R. Genevriere and A. Hoffmann. PMOSS - a modular synthesis and HW/SW-codesign system. Technical Report SFB - 358 - B2 - 2/94, Universität Paderborn, Fachbereich 17, Germany, March 1994.

[7] A. Ghosh, S. Devadas, K. Keutzer, and J. White. Estimation of average switching activity in combinational and sequential circuits. In *Proc. of the 29th DAC*, pages 153–159. ACM/IEEE, 1992.

[8] S. C.-Y. Huang and Wayne Wolf. How datapath allocation influences controller delay. In *Seventh ACM / IEEE Int. WS on High-Level Synthesis*, Niagara Falls, Canada, May 18 - 20 1994.

[9] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. In *Bell Systems Technical Report*, volume 49, pages 291–307, 1970.

[10] B. Lin and A.R. Newton. Implicit manipulation of equivalence classes using binary decision diagrams. In *Proc. of the ICCD*, pages 81 – 85, Cambridge, MA, 1991. IEEE.

[11] M.J. Mlinar. Control path/data path tradeoffs in VLSI design. Technical Report CEng 91-16, University of Southern California, May 1991.

[12] E.M. Sentovich, K.J. Singh, and L. et al. Lavagno. SIS: A system for sequential circuit synthesis. Technical Report Memorandum No. UCB/ERL M92/41, University of California at Berkeley, May 1992.