ENGINEERING CHANGE PROTOCOLS FOR BEHAVIORAL SYNTHESIS

Darko Kirovski and Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles

ABSTRACT

Rapid prototyping and development of in-circuit and FPGA-based emulators as key accelerators for fast time-to-market has resulted in a need for fast error correction mechanisms. The fabricated or emulated prototypes upon error diagnosis require quick and as much as possible flexible engineering change (EC). However, this problem has recently initiated research activity mainly in the logic synthesis domain. We introduce the first set of EC protocols for behavioral synthesis. The protocols support both the pre- and post-processing EC paradigms. In addition, instead of developing special algorithms for EC which is the adopted research model, as a key contribution, we show that using protocols which facilitate constraint manipulation of the initial design specification there is no need for development of specialized EC algorithms. The EC process is performed using the standard optimization algorithms on the modified design. Nevertheless, as shown on a number of behavioral synthesis tasks including: resource assignment, design partitioning, and operation scheduling, the approach provides variable and guaranteed flexibility for incremental synthesis with minimal hardware overhead.

1. INTRODUCTION

Rapid prototyping and development of in-circuit and FPGA-based emulators has been recently adopted as the key accelerating technology for fast time-to-market. However, upon development, fabricated or emulated prototypes are yet to be modified due to an extensive debugging process. The modifications succeed the diagnosis of a smaller number of design errors. While in the case of FPGA-based emulators or designs, engineering change (EC) requires a time-consuming update [Fan97], in the case of fabricated circuits, the modifications are performed using mask updating, the Focused Ion Beam apparatus for cutting and implanting new wires on a die, and Electron Beam Lithography for implanting logic structures into an already fabricated design [Tho68]. Since both rewiring and logic post-implanting are expensive and time-consuming processes, the design requirements for EC should enable as much as possible flexibility for design update. There are two fundamental approaches to EC: pre-processing where certain amount of logic or programmable interconnects (with no effect on the design functionality) is augmented into the design before fabrication, and/or post-processing where knowing the correct functionality of the design, the fabrication is minimally altered such that the error is corrected. While the first technique has a difficult goal to anticipate which extra hardware might be useful in the case of an error, the second one has a difficult task to use a limited amount of resources to update the design with minimal hassle.

The problem of EC has initiated research activity mainly in the logic synthesis CAD domain. However, due to the increasing complexity of behavioral specifications, designers are commonly faced with behavioral level modifications which are by no means minor at the physical level of abstraction. In this paper, we introduce the first set of engineering change protocols for behavioral synthesis. Achieving flexibility for EC at this stage of the design can enable more economic modifications at lower design levels [Pra94]. As a key novelty, we present an approach to EC which facilitates the idea of constraint manipulation. Such approach focuses on intelligent modification of the initial design specification which results in application of the original optimization algorithm. The key advantage is that this algorithm would have been used, if no flexibility for EC was ever requested. That is in oppose to the currently adopted research model for EC problems which, by default, seeks for new synthesis solutions.

The developed protocols (input modification strategies) support both the pre- and post-processing EC paradigms. In the pre-processing step, the design specification is embedded with extra constraints. After the optimization algorithm is applied to such input, these constraints impose a set of additional functionalities that the design can also perform. This technique for enabling EC provides variable and guaranteed flexibility for incremental synthesis with overhead bound by the efficiency of the optimization algorithm. To facilitate this flexibility, in the post-processing step, again, the output (optimized behavioral specification) is modified in such a way that the initial optimization algorithm can be now applied only to the selected area where the error is diagnosed. In the remainder of this paper, this high level description of the approach is explained in more technical detail on three behavioral synthesis tasks and a set of benchmarks extracted from real-life designs.

2. RELATED WORK

A number of developed algorithmic techniques for EC facilitate rectifying logic networks [Wat91, Kha96]. Different approaches to EC for logic synthesis have been developed [Bra94, Swa97]. Lin et al. have introduced alternative wires as a solution to the EC problem [Lin95]. They have also shown that alternative wires may aid optimization algorithms for routing in FPGAs. A number of systems targeting ASIC design have been developed for incremental synthesis from the RT-level to the gate-level [Pra94]. Fang et al. have developed an RT-level EC method that supports on-line debugging for FPGA-based logic emulators [Fan98]. Buch et al. have applied an EC technique to a different problem: optimization of logic networks for low-power [Buc97].

3. HARDWARE AND COMPUTATIONAL MODEL

We have selected as our computational model the synchronous data flow (SDF) model [Lee87]. The SDF is a special case of data flow in which the number of data samples produced or consumed by each node on each invocation is specified a priori. Nodes can be scheduled statically at compile time onto programmable processors. We restrict our attention to homogeneous SDF (HSDF), where each node consumes and produces exactly one sample on every execution. The HSDF model is well suited for specification of single task computations in numerous application domains such as DSP, communications, and multimedia. The syntax of a targeted computation is defined as a hierarchical control-data flow graph (CDFG) [Rab91]. The CDFG represents the computation as a flow graph, with nodes, data edges, and control edges. The semantics underlying the syntax of the CDFG format, as we already stated, is that of the SDF model.

4. THE GLOBAL ENGINEERING CHANGE APPROACH

Behavioral synthesis transforms a given behavioral specification into an RT-level description. The research work in behavioral synthesis encompasses a variety of tasks, such as scheduling, allocation, binding, partitioning, module selection, and transformations. An overview of existing synthesis techniques can be found in [Gaj92, DeM94]. We demonstrate the developed EC paradigm only for several synthesis tasks: resource allocation and assignment [Pau89, Kur87, Sto89], operation scheduling [Pau89], and partitioning [Lag91].

The developed EC paradigm can be applied to any of the synthesis tasks as follows. In the pre-processing step, as shown in Figure 1, the behavioral design description BD is augmented with additional design constraints (BD_a). The application of the optimization algorithm to BD_a provides a solution $OptD_a$ that can satisfy both the original and EC-targeted constraints. For example, in the case of register assignment, i.e. graph coloring, the additional constraints are modeled as new edges and nodes. The additional design constraints can be focused towards a particular type of an error, augmented to provide a guaranteed flexibility for EC after an arbitrary error is diagnosed, and random. In all cases, the trade-off of having significant design flexibility with respect to a small hardware overhead can be tuned according to the designer's needs. A correction in the optimized design is performed by manipulating the constraints that are additionally augmented into the design. Methods that facilitate constraint manipulation have been already used for other VLSI optimization tasks [Koi94].

In post-processing for EC, upon detection and diagnosis of a functional misbehavior, the error is corrected in the initial design specification cBD. The optimized design $OptD_a$ is altered to satisfy the corrected functional behavior. The goal is to change minimally the optimized design $OptD_a$, while achieving the desired modification of its functionality $cOptD_a$.

We have developed a method which searches for the best design alteration iteratively until the desired modification is not accomplished with minimal hassle. In the first step of each iteration, the optimized design specification $OptD_a$ is partitioned into two parts, part $OptD_a^A$, where the design changes should be located, and part $OptD_a^B$, which should be left intact by the modification procedure. In order to reduce the number of iterations, both parts are selected using a modified binary search of the solution space domain. In the next step, the constraints of part $OptD_a^B$ are compressed (creating $OptD_{ac}^B$) using protocols specific to the problem domain (details of several protocols are presented in the Section 5). In the last step, the optimization algorithm is applied to the merger of parts $OptD_a^A$ and $OptD_{ac}^B$. Since the domain cardinalities of $OptD_{ac}^B$ are much smaller than of $OptD_a^B$, the optimization algorithm is applied mainly to the isolated (changed) part $OptD_a^A$. The increased flexibility for EC of the initial optimized specification $OptD_a$ enables more efficient search for an optimization solution (update) $cOptD_a$ that satisfies the correction constraints.

5. THE ENGINEERING CHANGE PROTOCOLS

We applied the proposed EC paradigms on three behavioral synthesis tasks: register allocation and binding, operations scheduling, and design partitioning. For each of these tasks, we have defined their pre- and post-processing EC protocols, outlined effective algorithms for constraint manipulation, and presented the approach on a second order Gray-Markel ladder filter.

5.1. Register Allocation and Binding

Values that are generated in one control step and used in a later step must be stored in a register during the intermediate control step transitions. A variable is *live* between the time it is generated (written) and the last use (read) of it. This interval is called the *lifetime* of the variable. Two variables whose lifetimes do not overlap can be stored in the same register. The interval graph [DeM94] can be constructed as follows. For each variable, a node is made in the interval graph. Two nodes are connected if the lifetimes of the corresponding variables overlap. Register allocation can be performed by coloring the interval graph. The GRAPH K-COLORABILITY problem is solvable in polynomial time for $K = 2$, but remains NP-complete for all fixed $K \geq 3$ [Gar79]. The left-edge algorithm [Kur87] is optimal only for the interval graphs constructed from CDFGs with no loops.

We assume that in the design specification two types of errors may occur. The first type of errors are the ones where a variable V (with a lifetime $[C_V^S, C_V^E]$) is not used as an operand in an operation O_i that is out of the range $(C_{O_i} > C_V^E)$ of the preliminarily specified lifetime of V. Such error is modeled by adding edges to the interval graph. The procedure that we used to add edges of type-I for flexibility in EC is presented in Figure 1. The goal of this procedure is heuristically defined and targets expansion of variables with short lifetimes. Given M, the maximal number of alive variables at any control step C_i ($M = max(AliveVars(C_i), i = 1, \ldots, |C|)$), the procedure expands the lifetime of a variable if its expansion does not increase the number of alive variables at any control step over $M - 1$. In addition, at each control step C_i, $M - 1 - AliveVars(C_i)$ variables with the shortest lifetimes can be expanded for a single step. In Figure 2, it is shown how the lifetimes of variables $A1, C1, A5, A3, C2$, and $C3$ (bold edges in the CDFG and interval graph) are expanded. Although no overhead occurred, as shown in Figure 5, such register assignment can be used to resolve a number of corrections.

$M = max(AliveVars(C_i), i = 1, \ldots, |C|)$
Repeat
  For each control step C_i
    Subset of variables $W = \{V_i, C_{V_i}^E = C_{i-1}\}$
    Select subset $W_k \in W$ of $K < M - 1 - AliveVars(C_i)$ variables with shortest lifetimes
    For each $V_i \in W_k$
      $C_{V_i}^E = C_i$
until no more edges of type-I can be added

Figure 1: Procedure used to embed edges of type-I into a CDFG.

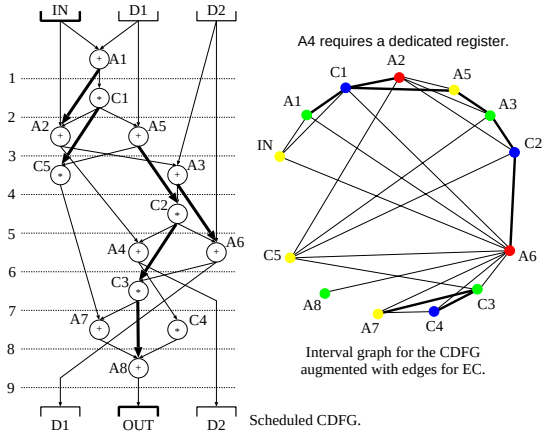

Figure 2: An example of addition of type-I constraints to the graph coloring problem for EC.

The second type of errors are the ones where entire operations (variables) are added to the spec. If such an operation is added

at the part of the interval graph where a maximal clique occurs, the EC process would require an addition of a new register and/or rescheduling. While the first type of a consequence can be trivially solved, the second one requires more attention.

To enable effective rescheduling, we identify and/or enable tuples of variables which can switch their registers arbitrarily. For example, consider two adjacent nodes A and B. If the sets of adjacent nodes to A and B are identical then nodes A and B can be colored with colors $C1$ and $C2$ or $C2$ and $C1$ respectively in any valid coloring of the graph. At the time of correction, this property can be used for faster minimal-hassle graph recoloring. We have used the lsII [Kir98gc] graph coloring algorithm modified to facilitate only the local changes enabled for EC.

The procedure which augments type-II edges into the graph coloring problem is outlined using the pseudo-code in Figure 4. The goal of this procedure is to involve as many as possible variables to become part of tuples for arbitrary coloring. The procedure initially sorts the set of control steps in descending order of the number of alive variables. Then for each control step C_i it identifies the variables which constitute a clique (its cardinality is $AliveVars(C_i)$). The neighborhood of the clique is analyzed whether it has a good potential for embedding edges of type-II. "Good potential" is heuristically defined with a bound on the number of edges adjacent to the nodes in the clique that has to be added to the graph in order to enable validity of arbitrary coloring permutation of the clique. In addition, for each control step, the added edges should not increase $AliveVars(C_i)$ above M. The bound M can be increased if the designers decide to include extra EC registers.

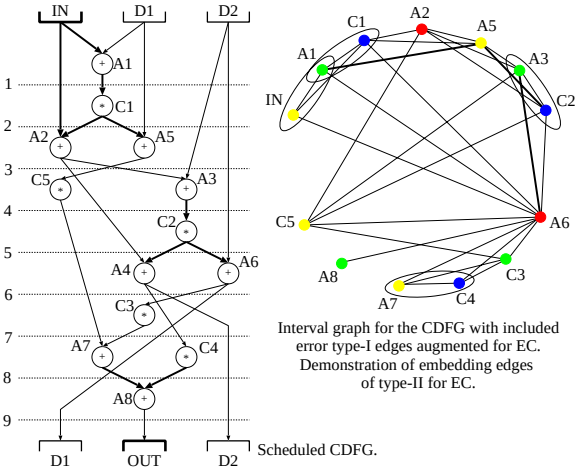

Figure 3: An example of addition of type-II constraints to the graph coloring problem for EC.

Finally, to identify all pairs of adjacent K-tuples of nodes which can be arbitrarily colored with K colors, we assign weights to edges in the interval graph. The weight for an edge between nodes A and B is equal to the sum of number of nodes which are adjacent to one but not both nodes A and B. Next, all edges with weights greater than some predetermined threshold value α are removed from the graph. For each edge $E_{A,B}, W(E_{A,B}) < \alpha$, we add a set of edges $E+$ to nodes A an B such that can be arbitrarily colored. Of course, the addition of each edge $E \in E+$ is bounded by the increase of $AliveVars(C_i)$ beyond M for any control step C_i. An example of addition of such edges is shown in Figure 3. Pairs of nodes $\{IN, A1\}$, $\{A1, C1\}$, $\{A3, C2\}$, and $\{A7, C4\}$ are enabled for arbitrary colorability by addition of edges drawn in bold in the appropriate interval graph.

Sort $CS = Sort(C)$ according to ascending $AliveVars(C_i)$
$M = max(AliveVars(C_i), i = 1, \ldots, |C|)$
For each control step CS_i and its clique CL
  Find a set of edges $E+$ necessary to be added to all nodes $V_i \in CL$ such that CL can be arbitrarily colored.
  For each edge $E_i \in E+$
    If for any C_i addition of E_i results in $AliveVars(C_i) < M$ **break**
  If no **break**
    Add each edge $E_i \in E$ to the interval graph
    Remove nodes in CL and adjacent edges from $IntervalGraph$
For each edge $E_i \in IntervalGraph$ between nodes A and B
  If $|Nei(A) \notin Nei(B) \cup Nei(B) \notin Nei(A)| > \alpha$
    Add edges $A - Nei(B) \notin Nei(A) \cup B - Nei(A) \notin Nei(B)$ if none of them results for any C_i that $AliveVars(C_i) < M$

Figure 4: Procedure used to embed edges of type-I into a CDFG.

Once the error is detected its correction requires updating the optimized spec. We have developed an approach which iteratively identifies a locality around the zone that needs to be updated, compresses the interval graph that is out of the identified locality, and performs the optimization algorithm on the compressed and updated spec. The advantage of this approach is that it tries to modify the smallest part of the spec while quickly searching for solution only on the part of the spec which is updated.

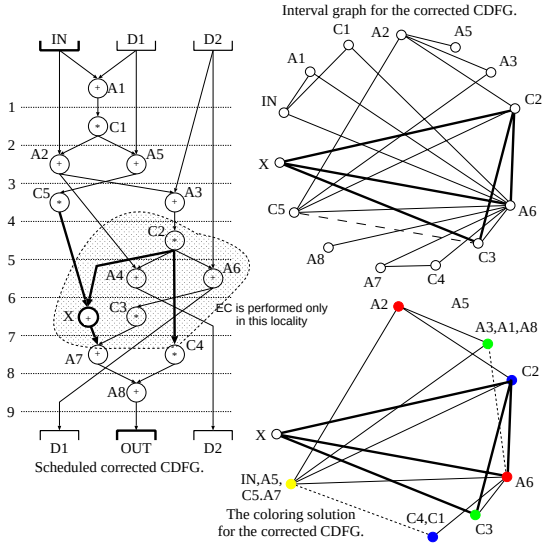

Figure 5: Individual steps in the post-processing for EC: graph bipartitioning, subgraph compression, and graph coloring on the compressed and updated interval graph.

The first step in the post-processing function performs simple binary search on the size of the subgraph B which will be left intact by the EC process. The linear parameter on which we perform the binary search is the maximal distance from any update on the interval subgraph A to an arbitrary node in the subgraph which will be changed. The second step involves compression of constraints in subgraph A. The compression is conducted in such a way that nodes colored with the same color in the optimized solution are merged. The new node inherits all edges adjacent to the parent nodes. An example of such merger is shown in Figure 5 where nodes which are merged are specified as labels to each node. In the same figure, the shaded area that is selected to be updated is left intact while the remainder of the interval graph is compressed. In the last step a graph coloring algorithm is applied to the compressed and updated interval graph.

5.2. Operation Scheduling and Design Partitioning

For the sake of brevity, in this section we briefly outline the main properties of developed protocols for pre- and post-processing for EC of operation scheduling and design partitioning solutions. The developed protocols are discussed in detail in [Kir98r].

Constraints are augmented in operation scheduling specs using a K-input single output additional computation unit. A chain O_{ADD} of N successive operations are added to the CDFG in order to force a critical path of length N. Next, at control steps at which a particular computation unit U is desired to be idle, operations of type U are attached to the augmented chain O_{ADD} as shown in Figure 6. Obviously K is equal to the maximum number of idle units at a single control step. In order to guide the process of forcing particular units to be idle at particular control steps, we used a heuristic identical to the one presented in previous subsection.

The procedure for compression of constraints which should be left intact after the update of operation scheduling is performed in the following way. A K-input single output computation unit is added to the CDFG. The chain O_{ADD} of N successive operations is added to the CDFG in order to force a critical path of length N. Scheduled operations of the subgraph B are connected to the chain O_{ADD} in a way that any algorithm can retrieve, as trivial, only one solution for scheduling the subgraph B. An explanatory example of such modification is shown in Figure 6.

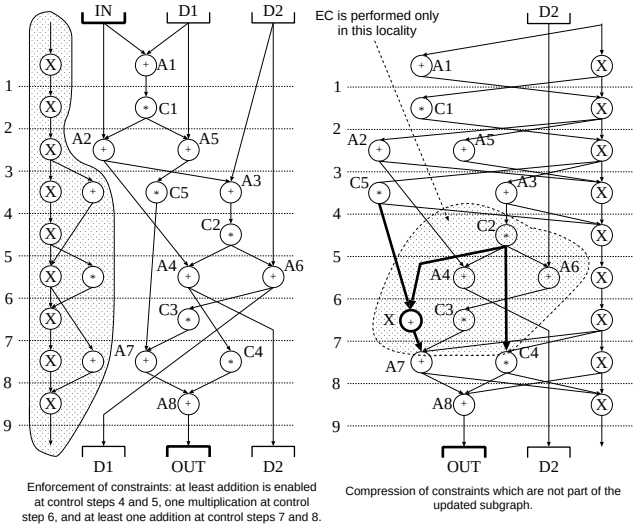

Figure 6: Constraint augmentation and subgraph compression for pre- and post-processing for EC of operation scheduling.

For partitioning, the standard constraint addition techniques such as edge augmentation and node merging (to force appearance of particular nodes in the same partition) can also be applied. However, the constraint compression protocol is specific to the nature of the partitioning problem. For each partition P, all nodes V that are part of the intact subgraph $V \in B$, are merged into a single node V_P in a way that each node $V_i \notin P$, if connected to at least one node in P is assigned a single edge E_{V_i,V_P} with weight $W(E_{V_i,V_P}) = \sum_{V_j \in B \cap P} W(E_{V_i,V_j})$.

6. EXPERIMENTAL RESULTS

We present the experimental data only for the EC of graph coloring related problems [Kir98r]. Table 6 shows in the first four columns the public domain benchmark, its number of nodes, edges, and chromatic number. In the next three columns it shows how many edges have been added for EC, the overhead in registers, and the percentage of corrected errors within a locality that equals 2% of the overall instance node cardinality. The errors, including adding 1% nodes and 0.1% edges, were generated randomly. Each instance was augmented with edges and corrected according to the protocols presented in subsection 5.1.

Original Instance			χ	Add Edge		EC
Instance	Vertices	Edges		Edges	Overhead	ratio
fpsol2.i.1.col	496	11654	65	496	0	99%
fpsol2.i.2.col	451	8691	30	451	0	99%
fpsol2.i.3.col	425	8688	30	425	0	97%
inithx.i.1.col	864	18707	54	864	0	97%
inithx.i.2.col	645	13979	31	645	0	96%
inithx.i.3.col	621	13969	31	621	0	95%
mulsol.i.1.col	197	3925	49	197	0	97%
mulsol.i.2.col	188	3885	31	188	1	100%
mulsol.i.3.col	184	3916	31	184	1	99%
mulsol.i.4.col	185	3946	31	185	0	92%
mulsol.i.5.col	186	3973	31	186	0	89%
zeroin.i.1.col	211	4100	49	211	0	93%
zeroin.i.2.col	211	3541	30	211	0	93%
zeroin.i.3.col	206	3540	30	206	0	91%

Table 1: Success of error correction for coloring register allocation instances augmented with constraints for EC.

7. CONCLUSION

We have introduced the first set of protocols for behavioral synthesis EC which support both the pre- and post-processing paradigms. As a key contribution, our approach to EC does not rely on developing special EC algorithms. It manipulates constraints to achieve design flexibility. The actual EC process is performed using standard optimization tools on the modified design. As shown on a number of behavioral synthesis tasks including: resource assignment, design partitioning, and operation scheduling, the approach provides variable and guaranteed flexibility for EC with minimal hardware overhead.

8. REFERENCES

[Bra94] D. Brand, et al. Incremental synthesis. ICCAD, pp.14-18, 1994.

[Buc97] P. Buch, et al. Engineering change for power optimization using global sensitivity and synthesis flexibility. Intl. Sym. on Low Power Electronics and Design, pp.88-91, 1997.

[DeM94] G. De Micheli. Synthesis and optimization of digital circuits. McGraw-Hill, New York, NY, 1994.

[Fan97] W.-J. Fang, et al. A real time RTL engineering change method supporting online debugging for logic emulation applications. DAC, pp.101-6, 1997.

[Gaj92] D.D. Gajski, et al. High-level synthesis: introduction to chip and system design. Kluwer, 1992.

[Kha96] S.P. Khatri, et al. Engineering change in a non-deterministic FSM setting. DAC, pp.451-6, 1996.

[Kir98gc] D. Kirovski and M. Potkonjak. Efficient coloring of a large spectrum of graphs. DAC, pp.427-32, 1998.

[Kir98r] D. Kirovski and M. Potkonjak. EC protocols for behavioral synthesis. Technical report, UCLA.

[Koi94] T. Koide, et al. A floorplanning method with topological constraint manipulation in VLSI building block layout. Trans. on Fund. of Electronics, Comm. and Computer Sciences, Vol.E77-A, (no.12), pp.2053-7, 1994.

[Kur87] F.J. Kurdahi and A.C. Parker. REAL: a program for REgister ALlocation. DAC, pp.210–215, 1987.

[Lag91] E.D. Lagnese and D.E. Thomas. Architectural partitioning for system level synthesis of integrated circuits. TCAD, Vol.10, (no.7), pp.847–860, 1991.

[Lee87] E.A. Lee and D.G. Messerschmitt. Synchronous dataflow. Proc. of the IEEE, Vol.75, (no.9), pp.1235–45, 1987.

[Lin95] C.-C. Lin, et al. Logic synthesis for engineering change. DAC, pp.647-52, 1995.

[Pau89] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of ASICs. TCAD, Vol.8, (no.6), pp.661–679, 1989.

[Pra94] S.C. Prasad, et al. System for incremental synthesis to gate-level and reoptimization following RTL design changes. DAC, pp.441-6, 1994.

[Rab91] J. Rabaey, et al. Fast prototyping of data path intensive architectures. Design & Test, Vol.8, (no.2), pp.40–51, 1991.

[Sto89] L. Stok and R. van den Born. EASY: multiprocessor architecture optimisation. Intl. Wrsp. on Logic and Architecture Synthesis for Silicon Compilers, pp.313–328, 1989.

[Swa97] G. Swamy, et al. Minimal logic re-synthesis for engineering change. ISCS, Vol.3. pp.1596-9, 1997.

[Tho68] P.R. Thornton. Scanning Electron Microscopy. Chapman and Hall, 1968.

[Wat91] Y. Watanabe and R.K. Brayton. Incremental synthesis for EC. ICCAD, pp.40-3, 1991.