Resource Constrained Modulo Scheduling with Global Resource Sharing

Christoph Jäschke Rainer Laur

Institute for Electromagnetic Theory and Microelectronics, University of Bremen/Germany
{jaeschke,laur}@item.uni-bremen.de

Abstract

Commonly used scheduling algorithms in high-level synthesis are not capable of sharing resources across process boundaries. This results in the usage of at least one resource per operation type and process. A new method is proposed in order to overcome these restrictions and to share high-cost or limited resources within a process group. This allows the use of less than one resource per operation type and process, while keeping the mutual independence of the involved processes. The method which represents an extension of general scheduling algorithms is not tied to a specific algorithm. The method is explained by using the common List Scheduling and further on applied to examples.

1. Introduction

The most customary model for the specification of an independent task for implementation into hardware is the process. For the realization of a complex system, mostly several such processes are needed. When using this systems in a non deterministic environment, events may occur at unpredictable times. In this case, implementing the system by using independent processes is mandatory. Usually the tasks have to be finished within predefined time limits. Systems of this kind are called real-time systems, or, if a timing violation causes a serious system failure, hard real-time systems.

Synthesis of such systems using traditional scheduling algorithms leads to a minimum of one resource per operation type and process. The same problem can be examined at loops with an unbound iteration count. Traditional scheduling algorithms won't allow resource sharing with blocks outside these loops.

1.1. Resource sharing of conventional static scheduling algorithms

In the following a survey of resource sharing concepts within a group of processes or across loop hierarchies is given. In this paper, resources are defined by elementary operations like multiplications or single bus- or memory-accesses. We only consider synchronous systems with a common clock.

Common static scheduling algorithms [1, 2, 3, 4] for high-level synthesis assign a control step to each operation of a block. Here, a block is understood as a connected subset of a process description. The control step determines the execution time of the operation relatively to the starting time of the block, all scheduled operations receive a fixed temporal relation. Thus, an assignment of resources to operations within this group at synthesis time is possible. However, the set of the processed operations in these algorithms is always a subset from one process. Therefore, resource sharing at synthesis time can only be considered by these procedures within *one* process.

Process merging transformations are able to extend the set of operations for scheduling and therefore the scope of resource sharing initially defined by the process boundaries. However, strong restrictions to process behaviours are imposed by techniques like process unrolling or latency adaption [5]. E.g. merging processes is not applicable in case of unpredictable block starting times. So if a process contains operations of unknown execution time or loops with unbound iteration count, a different method must be used.

A less restrictive approach to share resources within a set of processes is proposed by the Interface Matching algorithm [6]. Blocking communication results in a temporal synchronisation of two processes. By iterative scheduling the algorithm attempts to maximize the synchronized period. In addition to communication channels also common resources can be shared within this time range. However, that also means resource sharing is only possible in this limited time period and when blocking operation pairs exist.

The problem of sharing resources across loop hierarchies is discussed in the CADDY-II synthesis system [7]. The

blocks are synthesized in a bottom-up manner. At each level resources from already scheduled blocks can be used in the current block if no access conflicts may arise. The conflicts are detected by the calculation of Clock Cycle Spaces of the involved modules (block hierarchies) followed by an examination of the corresponding Collision Set. Like in the method mentioned above, blocking operations are used as calculation anchors. Still both methods cannot cope with loops of unbound iteration count or an operation with unknown execution time.

2. Problem definition

In minimum area applications it is necessary to maximize the resource sharing under given timing constraints. Certain constraints like reactivity, performance and independence of individual processes needed by the former systems, will restrict or not allow the use of the methods for static resource sharing described above. Even if there is only low utilization of limited or high-cost resources in such a system, *one full* resource is needed by each operation type and process when using traditional static scheduling algorithms. A scheduling algorithm without these restrictions is needed to further reduce the resource requirements of such systems.

3. Modulo Scheduling

A new universal method for the extension of conventional scheduling algorithms is presented. The method is based upon a time-dependent and periodic assignment of resources to processes and does not require a certain scheduling algorithm. This assignment is determined at synthesis time and must not be mixed up with a runtime-executive solving access conflicts. The method is explained by a resource constrained scheduling procedure but can also be applied to a time constrained algorithm.

The extension allows a static resolution of conflicts arising from independent processes with at synthesis time unknown execution times or data dependent loops trying to access shared resources. In this way, the method allows the use of less than *one* resource per operation type and process needed by traditional scheduling algorithms. Possible minimum area implementations can now be explored beyond the traditional limit.

Any block composition of a process is supported by the method if the following two conditions are met:

(C1) Each single block must also be processable by the *non modified* scheduling algorithm. Time steps must be assigned statically by the scheduling procedure.

(C2) Two blocks with unknown timing relation using at least one common resource within one process are not allowed to overlap in execution. A possibly overlapping block in this context must be considered as a separate process.

Operations having unknown execution delays at synthesis time may be placed arbitrarily between the individual blocks. In this way also loops of any depth with unbound iteration count running concurrently with other blocks can be handled. The first condition concerning the static time step assignment can be relaxed for algorithms such as the Relative Scheduling [8] if additional restrictions are applied to the allowed operations of unbound execution delay: The operation following e.g. an alternation with case dependent execution latency must be considered as a new block start. The timing restriction concerning block starting times will be derived in the following text.

When using a resource constrained scheduling algorithm the procedure can be subdivided into three steps.

(S1) The available resource count of each type is selected. The resources are assigned to the processes. For each assignment a decision between a local and global assignment must be done.

(S2) An authorization function is assigned to each of the global resources. This function determines a time-dependent selection of one process permitted to access the resource.

(S3) A modified scheduling algorithm is applied separately to every process block.

The first two steps will establish a time-dependent assignment from resources to processes. As opposed to traditional static scheduling algorithms, an assignment to a process group is allowed. In the last step these assignment informations are used by the modified scheduling algorithm.

Periodicity over time is an important quality of every global assignment. Only by taking care of this periodicity an independent and conflict-free resource access of all processes can be achieved. However, restrictions will arise for the possible starting times of individual blocks. The information from step (S1) and the periodicity of global assignments from step (S2) are required for their determination. On the other hand, restrictions of the block starting times, resulting e.g. by latency constraints or maximum reaction times to spontaneous input events, affect the possible decisions of the first two steps. This correlation is pointed out later in section 3.2.

3.1. Assignment of resources to processes

The current implementation requires a manual choice for the first step (S1). While a local assignment is corresponding to the traditional resource allocation step, the newly

introduced global assignment defines a group of processes sharing a single resource.

Let R be the set of all resources and P the set of all processes of the overall system. All local resources $r_l \in R$ are assigned only to one process p. A resource $r_g \in R$ is called global if it is assigned to more than one process. Every resource r_g defines a subset $P_{r_g} = \{\, p \mid p \text{ uses } r_g \,\} \subseteq P$ of all processes with $|P_{r_g}| > 1$. Let $R_{p,l}$ be the set of all resources locally assigned to the process p and $R_{p,g}$ the set of the globally assigned resources.

3.2. Authorization function of a global resource

An authorization function $a_{r_g}(t)$ is attached to each globally assigned resource. This function unambiguously assigns the resource r_g to a process $p_{r_g} \in P_{r_g}$ depending on control step t. Let A_g be the set of all global authorization functions $a_{r_g}(t)$. Local assignments of a resource r_l to a process are defined by a time invariant authorization function a_{r_l}. With A_l being the set of all a_{r_l}, the union set $A = A_g \cup A_l$ defines the complete unambiguous assignment of all resources R to all processes P, which is time-dependent in the case of existing global resources. Assuming that every process p_{r_g} only accesses the resource r_g at its authorized times, the conflict-free access of all processes P onto the resources R follows from the unambiguity of A.

The authorization function $a_{r_g}(t)$ is periodic in t. With ap_{r_g} being the period, the following equation holds:

$$\forall\, t \in \mathbb{N}_0^+ \;:\; a_{r_g}(t) = a_{r_g}(t \bmod ap_{r_g}). \tag{1}$$

The resource assignment chosen in step (S1) must be guaranteed by $a_{r_g}(t)$. Therefore, all processes $p_{r_g} \in P_{r_g}$ must be authorized at least once within the period. Considering the unambiguity of $a_{r_g}(t)$, we obtain

$$ap_{r_g} \geq |P_{r_g}| > 1 \tag{2}$$

for the minimum period ap_{r_g} of a 1-cycle or pipelined operation. Non-pipelined multi-cycle operations are using a resource for a certain count of consecutive cycles, the resulting ap_{r_g} of equation 2 have to be multiplied with this cycle count.

Steps (S1) and (S2) imply restrictions on the block starting times. They can be formulated as follows: Under the assumption $A_g \neq \emptyset$ the resource assignment A defines a time-dependent access regulation of processes p_{r_g} onto resources r_g through its functions $a_{r_g}(t)$. A static scheduling algorithm assigns a control step relative to a starting time t_{st}, chosen implicitly at procedure entry, to all operations of a block. For this task, the modified scheduling algorithm additionally takes into account the time dependent assignment A. Unlike non modified procedures, the scheduling result of the modified algorithm becomes dependent on starting time t_{st}.

Let pb_g be a block of process p_g using at least one global resource. If *exactly one* global resource is assigned to a process p_{g1} by A, then the assignment A concerning a block pb_{g1} using this resource behaves periodic after $ap_{r_{g1}}$ control steps. Therefore, a scheduling result of pb_{g1} is valid for the block starting times

$$t_{st,pb_{g1}} \in \{\, t \mid t - t_{st} \mod ap_{r_{g1}} = 0 \,\}. \tag{3}$$

In other words, an equidistant time grid with a spacing of $ap_{r_{g1}}$ is formed by the permitted block starting times.

If *more than one* global resource is assigned to a process p_g by A, then the periodicity of the assignment A concerning a block pb_g using all resources is determined by multiple possibly different periods ap_{r_g}. Each resource $r_g \in R_{p_g,g}$ generates a restriction onto the starting times of block pb_g by equation 3. Making use of the least-common-multiple function (lcm), this can be formulated for any A:

$$t_{st,pb_g} \in \left\{\, t \mid t - t_{st} \mod \operatorname*{lcm}_{r_g \in R_{p_g,g}} ap_{r_g} = 0 \,\right\}. \tag{4}$$

If a block pb_{gp} uses a subset $R_{pb_{gp},g} \subset R_{p_g,g}$, only the periods of the resources $r_g \in R_{pb_{gp},g}$ determine the valid starting times defined by equation 4. Consequently, blocks without any global resource usage can arbitrarily started if condition (C2) is still met.

Figure 1 illustrates a time controlled assignment of two global resources to three processes. The mappings of the authorization functions are drawn into their boxes. Following all dashed lines from one control step on the left time axis to the authorization functions, one will get the assignments from resources to processes depicted between them. On the other hand, the authorized resource accesses are drawn inside the process boxes, each also representing a control step from the time axis above. The block starting time restrictions formulated by equation 4 are now obvious for e.g. process p_1, which must to be on a grid with distance $\operatorname{lcm}(2,3) = 6$.

The dependencies between the assignment A and the restriction of the block starting times by equation 4 can also be formulated into the reverse direction. The possible block starting times can be quantified by the time spacing td_{p_g} on an equidistant grid. To achieve independence from the specific resource usage of the different blocks, we consider the spacing obtained by using all resources $r_g \in R_{p_g,g}$. This maximum restricted spacing can be taken from equation 4:

$$td_{p_g} = \operatorname*{lcm}_{r_g \in R_{p_g,g}} ap_{r_g}. \tag{5}$$

If the grid spacings td_{p_g} of all processes $p_g \in P_{r_g}$ of a global resource r_g are fixed, from equation 5 follows $ap_{r_g} = \operatorname*{gcd}_{p_g \in P_{r_g}} td_{p_g}$ for the minimal period of an authorization function.

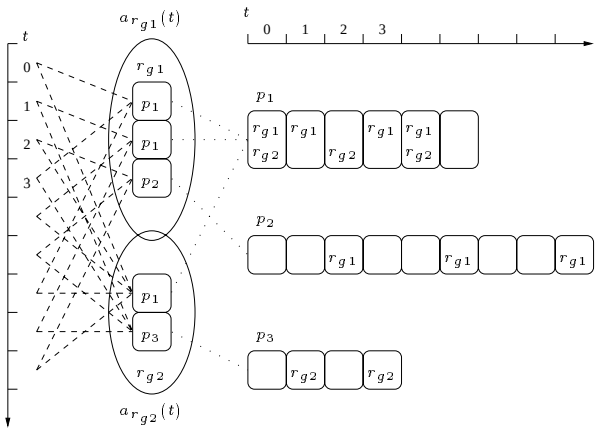

Figure 1. Assignment of two global resources to three processes

3.3. Modified List Scheduling

Due to space limitations, the basic List Scheduling (LS) algorithm [9] is not described here.

The unmodified LS algorithm uses a constant number of resources $|R_y|$ for each operation type y to decide whether or not the current control step will be assigned to an operation. For the modified algorithm, the number of available resources for one operation type is replaced by a function $rc_y()$. The function set $RC()$, defined by all $rc_y()$, serves as an interface between the modified scheduling procedure and the resource assignment A. Consequently, these functions are dependent from the current control step t and the process p currently processed by the algorithm. $rc_y(p,t)$ results from an addition of a local part $rc_{y,l}(p)$ and a global part $rc_{y,g}(p,t)$.

Let $R_{y,g}$ be the set of all global resources r_g of a type y and $R_{y,l}$ be the corresponding local set. The part $rc_{y,l}(p)$ is determined by the assignment A_l of the local resource set R_l and corresponds to $|R_y|$ of the traditional LS :

$$rc_{y,l}(p) = \sum_{r_l \in R_{y,l}} \begin{cases} 1 & \text{if } a_{r_l} = p, \\ 0 & \text{otherwise.} \end{cases} \qquad (6)$$

The global part $rc_{y,g}(p,t)$ is determined analogously by the assignment A_g of the global resource set R_g:

$$rc_{y,g}(p,t) = \sum_{r_g \in R_{y,g}} \begin{cases} 1 & \text{if } a_{r_g}(t) = p, \\ 0 & \text{otherwise.} \end{cases} \qquad (7)$$

The LS algorithm modified by $RC(A)$ can now schedule each block pb of all processes $p \in P$. The realizable block structure of a process has to follow condition (C2). The processed blocks can be started at any time of the grid defined by equation 4. Operations of unknown execution time can be placed arbitrarily between the single blocks of a process. However, the restrictions of the block starting times still apply.

4. Implementation

The Modulo Scheduling method using the LS algorithm was implemented in a system called IPS and is based on parts of the Olympus Synthesis System [10]. An enhanced version of the tool Hercules is used to transform a HardwareC-description into a control-data flow graph. The LS algorithm uses the ALAP time as the priority function.

Step (S1) and specifying the allowed range of time spacing td_{p_g} is to be done manually first. Based on these informations, all possible assignments A are generated through a 3-stage permutation. The first stage calculates a set of periods ap_{r_g}, the second gives out the count of slots a process owns at $a_{r_g}(t)$ and the last places these slots at a specific time. The number of assignments A for a chosen set of periods ap_{r_g} is limited by $\prod_{r_g \in R_g} |P_{r_g}|^{ap_{r_g}}$, while maximal $\prod_{p_g \in P_g} (td_{p_g,max} - td_{p_g,min} + 1)$ sets of ap_{r_g} are possible. At each stage various checks are made to detect invalid and redundant assignments as soon as possibly. Mainly equations 2 and 5 are used for this purpose at stage 1. The need of at least one assigned resource for each operation type used in a process is exploited at stage 2.

For each assignment A, a modified LS of all processes $p \in P$ is carried out and results are recorded. In this way the procedure explores the whole scheduling space of a hardware description available by the modified LS under the chosen resource constraints.

5. Experiments

The 5th order elliptical wave filter and the main loop of the differential equation solver from the High-Level Synthesis (HLS) Benchmarks 1991 were used for the first two examples 1a and 1b. The equation solver was modified by replacing the comparator with a subtracter. The execution time of an addition and a subtraction was fixed to 1 cycle and that of a pipelined multiplication to 2. The process set was formed by two elliptical filters p_0 , p_1 and the main loop of the equation solver p_2. One local adder was assigned to each filter and one local subtracter was assigned to the equation solver. All processes were sharing an adder $r_{add,g}$ and a multiplier $r_{mul,g}$ as global resources. The grid spacing td_{p_g} was restricted to a range from 1 to 5 for all processes. Please note, that although these processes can be merged into one, we consider the processes as triggered by spontaneous events. This is impossible to handle by merging, but *can* be scheduled with our new approach.

The resulting 22068 assignments A were calculated and a modified LS for all processes was carried out. The whole task takes 88 sec. on a Pentium II 233 MHz. By removing entries with identical latencies for all processes, the number of results was reduced to 1250. For illustrating the results, two of three latency pair plots are shown in figure 2 and 3. Figure 2 shows the latency pair results of both elliptical filters for each of the generated assignments A. In addition to the direct results, the latency pairs corrected by equation 4 are plotted in the figure as squares. The obvious symmetry of the results is caused by the identical filters and the symmetric assignment of resources concerning step (S1).

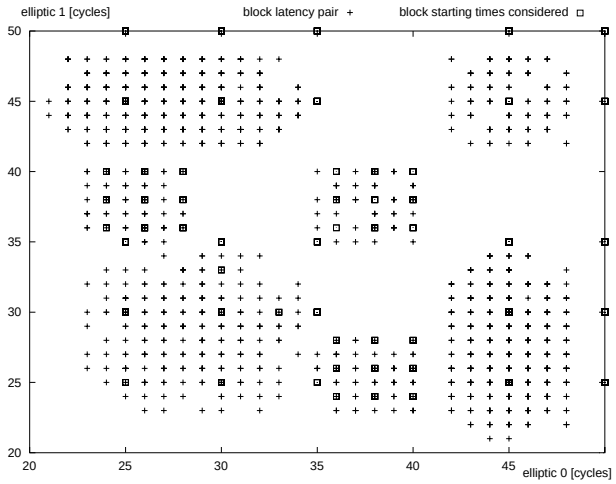

Figure 2. Latency pairs of both filters parametrized by assignments A.

Figure 3 contains the results of the modified LS for the first filter and the equation solver. Due to the former mentioned symmetry, the results of the third possible constellation are redundant and therefore not depicted here.

For the evaluation of the latency results tl, fitted to the time grid, $\sqrt{\sum_{p_i \in P}(c_i \cdot tl_{p_i})^2}$ is chosen for the costs.

In example 1a, all weights c_i are set to one. A minimum cost of 47.34 is obtained by two assignments (twelve in the raw result set). Both assignments, A_{13} and A_{19}, lead to 30 cycles for each filter and to 21 cycles for the equation solver main loop. The assignment functions of both global resources can be taken from table 1.

Table 1. A_{13} and A_{19} of example 1a.

no.	$a_{r_{add,q}}(t) : t \to p_i$	$a_{r_{mul,q}}(t) : t \to p_i$
13	$0 \to 2, 1 \to 1, 2 \to 0$	$0 \to 1, 1 \to 2, 2 \to 0$
19	$0 \to 2, 1 \to 1, 2 \to 0$	$0 \to 0, 1 \to 2, 2 \to 1$

Using asymmetrical weights $c_1 = 0.5$ for the first filter and $c_2, c_3 = 1$ for the other processes in example 1b, a minimum cost of 38.41 is achieved. The latency time of the first filter is now 50 cycles, while the second filter needs 25 cycles and one equation solver iteration executes in 15 cycles.

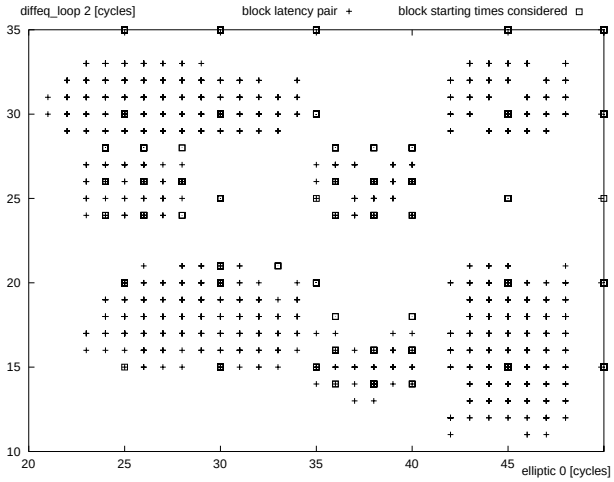

Figure 3. Latency pairs of first filter and equation solver parametrized by assignments A.

A minimum, pure local resource assignment needs one subtracter, three adder and three multiplier. Scheduling this system will lead to 29, 29 and 9 cycles latency with a cost of 41.99. Compared to example 1a using the same cost function, we need two more multipliers while only achieving a cycle cost improvement of 11.3 %. We assume that all resources have an area of 1, except the multiplier having an area of 8. In this case, the cycle cost improvement implies a worsening of area from 12 to 28. Considering the area-cycle cost product, we get a factor of 2.07 in favour of the Modulo Scheduling method.

The second example uses one frisc processor from the same Benchmark set and two kalman filters of the HLS Benchmarks 1992. Both designs contain loops and alternations. The implemented LS algorithm partitions the control-data flow graph automatically to prevent possible violations of condition (C2) and considers exclusive resource usage in alternations. The block starting times are considered here only for a loop (or process) not interrupted by any other loop. Synchronization loss on other block boundaries are not calculated. This is applicable, because inner loops contribute the main part to the now hierarchical cost function, reflecting the loop hierarchy.

In this example a weight of 1 is chosen for all process blocks (outside any loop). The loop weights are recursively calculated by a multiplication of the assumed iteration count of a loop with their outer block weight. The iteration counts of the four first level loops in the kalman filters are (in the

order of their execution) 16, 3, 16 and 4. Both second level loops iterate 16 times. The data dependent loop of the frisc is assumed to be executed 128 times. Please note, that the assumed iteration counts are used only for the cost function and not for a resource conflict detection. In this way, the loop exit conditions in the kalman filters may also be data dependent.

Both filters were assigned a local equality comparator and the processor a logical alu, each needing 1 cycle to complete and consuming an area of 1 altogether. The filters are sharing one global pipelined multiplier $r_{mul,g}$ needing 2 cycles and an area of 8. All processes also access a global arithmetic alu $r_{alu,g}$ executing in 1 cycle and occupying an area of 1.

We scheduled 39616 assignments for a grid spacing td_{p_g} ranging from 1 to 6 for all processes. This needs 654 sec. to complete and results in a best cost of 5996.79. The assignment of this run is shown in table 2.

Table 2. A_{24143} of example 2.

	$t \rightarrow p_i$
$a_{r_{alu,g}}(t)$	$0 \rightarrow 1, 1 \rightarrow 0, 2 \rightarrow 0, 3 \rightarrow 2, 4 \rightarrow 2, 5 \rightarrow 1$
$a_{r_{mul,g}}(t)$	$0 \rightarrow 2, 1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 1, 4 \rightarrow 1, 5 \rightarrow 2$

A minimum, pure local resource assignment will additionally need one multiplier and two arithmetic alus. A cycle cost of 4354.41 results from scheduling this system. Again compared to the Modulo Scheduling method, the cycle cost is decreased by 27.38 % while the area is doubled, leading to a 1.45 times higher area-cycle cost product. The less significant advantage of our new method in this example is caused by a lower sharing of the cost intensive multiplier.

6. Conclusion and future work

A new method for extending the scope of resource sharing in static scheduling algorithms beyond process boundaries was presented. It is based on a time dependent and periodic assignment of resources to processes. The method allows a static resolution of access conflicts from independent processes onto shared resources without the need for a runtime-executive. In this way, the requirements of traditional scheduling procedures of at least one resource per operation type is further lowered to a fraction. This can lead to substantial area savings. The method remains applicable in reactive systems if the use of other methods such as process merging or Interface-Matching is impossible or ineffective.

Restrictions concerning the block starting times and their dependencies from the resource assignment were explained. The non modified LS was provided with an interface to the time dependent resource assignment for usage with the Modulo Scheduling method.

The method was successfully applied to two systems containing independent running processes accessing shared resources, some including alternations and nested loops assumed to be data dependent. These systems can not be handled by traditional scheduling algorithms.

After manually choosing parts of the resource assignment, the procedure finds the optimal solution by the automatic generation of all further parameters. Currently, the suitability of a Branch and Bound Algorithm for an effective search space reduction is examined.

References

[1] Pierre G. Paulin and John P. Knight. Force-directed scheduling for the behavioral synthesis of asic's. *IEEE Transactions on Computer Aided Design*, 8(6):661–679, June 1989.

[2] Roni Potasman, Joseph Lis, Alexandru Nicolau, and Daniel Gajski. Percolation based synthesis. In *27th Design Automation Conference*, pages 444–449, 1990.

[3] Raul Camposano and Reinaldo A. Bergamaschi. Synthesis using path-based scheduling: Algorithms and exercises. In *27th Design Automation Conference*, pages 450–455, 1990.

[4] Catherine H. Gebotys. Optimal scheduling and allocation of embedded vlsi chips. In *29th Design Automation Conference*, pages 116–119, 1992.

[5] Keshab K. Parhi. Algorithm transformation techniques for concurrent processors. *Proceedings of the IEEE*, 77(12):1879–1895, December 1989.

[6] David Filo, David C. Ku, Claudionor N. Coelho Jr., and Giovanni De Micheli. Interface optimization for concurrent systems under timing constraints. *IEEE Transactions on VLSI Systems*, 1(3):268–281, September 1993.

[7] Oliver Bringmann and Wolfgang Rosenstiel. Resource sharing in hierarchical synthesis. In *International Conference on Computer Aided Design*, pages 318–325, 1997.

[8] David C. Ku and Giovanni De Micheli. Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. CSL Technical Report CSL-TR-91-477, Stanford, June 1991.

[9] S. Davidson, D. Lanfskov, B.D. Shriver, and P.W. Mallet. Some experiments in local microcode compaction for horizontal machines. *IEEE Transactions on Computers*, C-30(7):460–477, July 1981.

[10] David C. Ku and Giovanni De Micheli. *High Level Synthesis of ASICs Under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.