Time Constrained Modulo Scheduling with Global Resource Sharing

Christoph Jäschke[1] Friedrich Beckmann[2] Rainer Laur[1]

[1]Institute for Electromagnetic Theory and Microelectronics, University of Bremen, Germany
[2]Siemens AG, Munich, Germany
{jaeschke, laur}@item.uni-bremen.de, friedrich.beckmann@siemens-scg.com

Abstract

Commonly used scheduling algorithms in high-level synthesis only accept one process at a time and 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 represents an extension of general scheduling algorithms and is not tied to a specific algorithm. It is applied to the time constrained Force-Directed Scheduling algorithm. For this the scope of the scheduling is extended to the processes of the whole system and a two-part modification is applied to the original procedure. A multi-process example illustrates the resource sharing capabilities of the extension.

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 processes are needed. When using these 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 failure, hard real-time systems.

The 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, the considered resources range from simple adders, memories or busses to more complex (pipelined or multicycle) functions including local memories. In the latter case, the local memory must be private and selectable for each using process. We do not consider any problems caused by a sequence of accesses (data inconsistency, starvation or deadlock). 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. Using this assignment strategy, all operations of the processed group receive a fixed temporal relation, thus an assignment of resources to single operations within this group at synthesis time is possible. However, the set of the processed operations in these algorithms is always a subset of 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 synchronization 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 or process boundaries is discussed in the CADDY-II synthesis system [7]. The modules are synthesized in a bottom-up manner. At each level resources from already scheduled modules can be used in the current module if no access conflicts may arise. The conflicts are detected by the calculation of Clock Cycle Spaces of the involved modules 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 reduce the resource requirements of such systems further.

3. Modulo Scheduling

A new universal method for the extension of conventional static scheduling algorithms is presented. The method is based on 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 time constrained scheduling procedure but can also be applied to a resource constrained algorithm [8].

The extension allows a static resolution of conflicts arising from independent processes with at synthesis time unknown execution times or loops with unbound iteration count 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. Within a block, time steps must be assigned statically.

(C2) Two blocks having a non fixed 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.

Under these conditions, loop bodies have to be considered as separate blocks. 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. Summarized, the complete system schedule is comprised of statically scheduled blocks with at synthesis time unknown starting times.

When using the time constrained Force-Directed Scheduling algorithm [9] (FDS) the procedure can be subdivided into three steps.

(S1) Resource types are assigned to the processes. For each assignment a decision between a local and a global assignment must be done.

(S2) A periodicity is assigned to each global resource type. Possible periods are determined by the timing constraints of each process and the assignments of step (S1).

(S3) Coupled Force-Directed Scheduling algorithms are simultaneously applied to all process blocks.

The first two steps will establish the periods of the resource types and their assignment to processes or process groups. Note, that an assignment to a process group is not allowed in a traditional static scheduling algorithm. In the last step this assignment information is used by the modified scheduling algorithm.

3.1. Assignment of resources to processes

In the first step, resource types are assigned to the processes. A decision between a local and global assignment for each resource type has to be made. A local assignment selects the traditional calculation of the resource count for a process. A global assignment defines a process group and therefore distinct processes for which a calculation of the shared resource count will be performed during the scheduling phase.

Let RT be the set of all resource types and P the set of all processes of the overall system. A process is composed of blocks pb. Each block has to comply conditions (C1) and (C2). A resource type $rt_g \in RT$ is called global, if it is assigned to more than one process. The set of all global resource types is RT_g. Every resource type rt_g defines a subset $P_{rt_g} = \{p \mid p \text{ uses } rt_g\} \subseteq P$ of all processes with $|P_{rt_g}| > 1$. Let $RT_g(p)$ be the set of all resource types globally assigned to the process p.

3.2. Periodicity of a global resource type

A global resource type is assigned to a process group. We allow the processes running independently from each other, i. e. having no synchronization points. The multiple use of a single resource instance from blocks within different processes requires an access regulation. The Modulo Scheduling solves this conflict through a periodical sequence of access authorizations of the involved processes. All accessing processes have to obey these authorizations. To reflect this, a periodicity $ap(rt_g)$ is assigned to a global resource type rt_g. The absolute time steps t_{abs} of the entire system are mapped to the time steps of the period $t_{per} \in T_{per}(rt_g) = \{0 \ldots ap(rt_g) - 1\}$ by

$$t_{per} = t_{abs} \bmod ap(rt_g). \qquad (1)$$

The aim of a time constrained scheduling algorithm is to minimize the maximum resource usage over all t_{abs}. This holds also for the time constrained Modulo Scheduling, which tries to minimize the maximum resource usage for global types over $T_{per}(rt_g)$. The maximum usage for a global resource type also defines the needed instance count.

An access authorization for a specific process once granted for a time step t_{per} is valid for all t_{abs} that can be mapped to t_{per}. Figure 1 gives an example for a process p_i and a resource type rt_g of the period $ap(rt_g) = 2$. Two operations of type rt_g are executed by process p_i at time step $t_{abs} = 4$ (upper graph). In the lower usage distribution graph of resource rt_g, this is recorded in time step $t_{per} = 0$. By the mapping of equation 1, the process may execute the same number of operations of this type in all time steps marked with a rippled line without increasing its resource requirements (granted access authorizations).

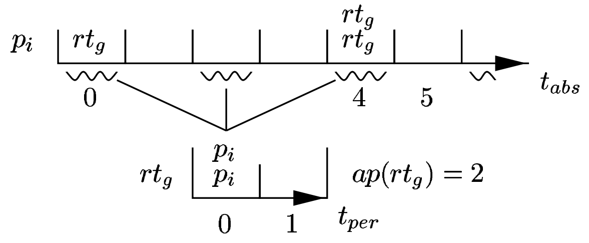

Figure 1. Time steps of access authorization for process p_i onto rt_g

The advantage of the modulo mapping is the unlimited scope of application. Loops of unknown iteration count or independent processes can be integrated into the resource sharing. No blocking operation pairs are needed. Drawbacks may occur due to the fixed periodicity of the resource access authorizations, which restricts possible block starting times.

The restrictions for the block starting times, implied by steps (S1) and (S2), may be formulated as follows: Like any *non modified* static scheduling also the *modified* scheduling assigns a time relative to the starting time t_{st} to each operation of a block. Due to the modification an interdependency of the schedulings for each block exist. The coupling of the simultaneously running schedulings concerning the time is achieved by equation 1 for each commonly used global resource. Opposed to traditional procedures, the scheduling result of a block is now dependent on the starting time t_{st}.

A block pb is using a subset $RT_g(pb) \subseteq RT_g(p)$ of all global resource types of a process p. The modified scheduling of this block is coupled with the parallel scheduling of all other blocks by possibly multiple equations 1. Due to the periodicity of this coupling, the scheduling remains valid for all movements of a block defined by:

$$\Delta t_{st,pb} \in \{\, \Delta t \mid \Delta t \bmod \underset{rt_g \in RT_g(pb)}{\text{lcm}} ap(rt_g) = 0\}. \quad (2)$$

It follows, that blocks without a global resource usage can be started at any time if condition (C2) is still met.

The dependencies between the periods $ap(rt_g)$ and the restriction of the block starting times by equation 2 can also be formulated into the reverse direction. The possible block starting times can be quantified by the time spacing $td(p)$ on an equidistant grid. To achieve independence from the specific resource usage of the different blocks, we consider the spacing obtained by using all resource types of $RT_g(p)$. This maximum restricted spacing can be taken from equation 2:

$$td(p) = \underset{rt_g \in RT_g(p)}{\text{lcm}} ap(rt_g). \qquad (3)$$

For all processes $p \in P_{rt_g}$ accessing the global resource type rt_g with a fixed grid spacing $td(p)$, the coupling through equation 1 is to be considered. At least a global resource type of period

$$ap(rt_g) = \underset{p \in P_{rt_g}}{\text{gcd}} td(p)$$

will comply the defined grid spacings.

Not considering any period combination, the impact of a global resource period is always twofold. On the one hand higher values allow more processes to share a single resource instance, on the other hand the invocation interval of critical loops could be enlarged.

4. Basic Force-Directed Scheduling

The used scheduling algorithm is a further development of the FDS presented by Paulin and Knight. This Improved Force-Directed Scheduling [10] (IFDS) is dealing with gradual time frame reduction and global spring constants. For better understanding of the modifications applied to the IFDS, a brief overview of the unmodified algorithm will be given. For the sake of simplicity, the look-ahead mechanism and the global spring constants are not considered here. Except where otherwise stated, the overview is valid for the original and the improved algorithm.

Input data for the FDS algorithm is the operation set O of a block represented as a graph describing its precedence constraints $\prec$. The algorithm tries to find a scheduling using minimal resources within the specified time range $T = \{0, 1, \ldots, m-1\}$. For this, the possible time frames $fr_o = \{a(o), \ldots, b(o)\}$ for each operation $o \in O$ are computed by an ASAP (a) and ALAP (b) scheduling. The iterative constructive algorithm computes the final schedule through a sequence of partial solutions. A partial solution $\tilde{\tau}$ is defined by the set of time frames fr_o of all operations while considering the precedence constraints $\prec$. From each partial solution $\tilde{\tau}$ all in the next iteration reachable partial solutions $\tilde{\sigma}$ are evaluated. For this, the original algorithm places all operations onto all time steps within their time frames. The improved algorithm only investigates the time steps $a(o)$ and $b(o)$ at the outmost ends of the time frames. It is to be noted, that implicit time frame reductions of other operations may occur due to the precedence constraints $\prec$. Selecting the next partial solution $\tilde{\sigma}$ is done by evaluating a cost function, the so-called force.

4.1. Distribution function

The FDS algorithm tries to find an optimal solution by smoothing the resource requirement over the time range T. For this purpose, the algorithm calculates with each partial solution $\tilde{\tau}$ the needed number of resources per type in a distribution function. Each operation o has a probability $Prob(\tilde{\tau}, o, t)$ to be placed at a specific time step t. The probability for an operation to be placed within its time frame is $1/(b(o) - a(o) + 1)$ and zero otherwise. The set of all operations of a resource type rt is O_{rt}. The distribution function $\tilde{N}_{rt}(\tilde{\tau}, t)$ is calculated by the sum of the probabilities of all operations of a resource type:

$$\tilde{N}_{rt}(\tilde{\tau}, t) = \sum_{o \in O_{rt}} Prob(\tilde{\tau}, o, t). \tag{4}$$

The model introduced by Paulin and Knight considers the individual values of the distribution function as springs with constants equal to the sum of probabilities. Possible neighbourhood solutions $\tilde{\sigma}$ are evaluated by placing an operation $o_{pl} \in O$ with $a(o_{pl}) < b(o_{pl})$ at a time step t_{pl}. This causes a modification of at least one distribution function. The modification corresponds to a displacement of the affected springs:

$$\Delta\tilde{N}_{rt}(\tilde{\tau}, \tilde{\sigma}, t) = \tilde{N}_{rt}(\tilde{\sigma}, t) - \tilde{N}_{rt}(\tilde{\tau}, t). \tag{5}$$

4.2. Forces

The resulting force is the sum of the individual forces needed for the displacement of the springs obeying Hooke's law:

$$F(\tilde{\tau}, o_{pl}, t_{pl}) = \sum_{rt \in RT} \sum_{t \in T} \Delta\tilde{N}_{rt}(\tilde{\tau}, \tilde{\sigma}, t)\, \tilde{N}_{rt}(\tilde{\tau}, t). \tag{6}$$

Negative forces correspond to a decrease of operation concurrency and therefore to a better smoothing of the distribution function. From all possible neighbourhood solutions the original algorithm chooses the operation which caused the least force at its best time step. The operations time frame is reduced to this single time step.

The gradual time frame reduction calculates the maximum $F_{max}(\tilde{\tau}, o_{pl})$ and minimum force $F_{min}(\tilde{\tau}, o_{pl})$ of a tentative placement at $a(o_{pl})$ and $b(o_{pl})$ of operation o_{pl}. If a time frame allows more than two placements, $F^*_{min}(\tilde{\tau}, o_{pl})$ is set to $min(F_{min}(\tilde{\tau}, o_{pl}), 0)$ otherwise to $F_{min}(\tilde{\tau}, o_{pl})$. Thereby a rough estimate for the placements between the outmost ends of the time frame is done. The maximum difference $\Delta F_{gain} = F_{max}(\tilde{\tau}, o_{pl}) - F^*_{min}(\tilde{\tau}, o_{pl})$ of all operations o_{pl} selects the time frame to be reduced. The selected time frame is shortened at the side with the higher force. In this way, the worst neighbourhood solution is taken from the set of all possible ones.

After reducing the affected time frames, all functions are recalculated and the new partial solution is defined as $\tilde{\tau}$. If there are still operations o_{pl} left, a new iteration starts.

5. Modified Force-Directed Scheduling

The IFDS algorithm was extended by a two-part modification. The first part takes care of the periodical access of global resource types and operates with a local block scope. The second part extents the scope of the algorithm to blocks within a group of processes and takes care of balancing the global resource requirements within this group. Except where otherwise stated, only a global resource type rt_g is taken into account and for reason of clarity the index g is left out.

5.1. Periodical alignment of operations

At first, $\tilde{N}_{rt,pb}(\tilde{\tau},t)$ is calculated according to equation 4. Only the probabilities of the operations of block pb are taken into account. Then, a modulo-maximum transformation is carried out:

$$\tilde{N}mm_{rt,pb}(\tilde{\tau},t) = \max_{t_i \in T(pb) \wedge t_i \bmod ap(rt)=t} \tilde{N}_{rt,pb}(\tilde{\tau},t_i) \tag{7}$$

with $T(pb)$ denoting the time range of block pb. The change of the modulo distribution function $\tilde{N}mm_{rt,pb}(\tilde{\tau},t)$ is described in accordance with equation 5:

$$\Delta\tilde{N}mm_{rt,pb}(\tilde{\tau},\tilde{\sigma},t) = \tilde{N}mm_{rt,pb}(\tilde{\sigma},t) - \tilde{N}mm_{rt,pb}(\tilde{\tau},t).$$

For explaining the effect of this transformation, figure 2 shows the last two iterations of a block containing two operations. The unmodified IFDS algorithm is depicted on the left side while the right shows the modified one. The time range of the block is $T(pb) = \{0,1,2\}$. The time frame of operation o_2 is $fr_{o_2} = \{0,1,2\}$ while operation o_1 is already scheduled at time 2. For the sake of simplicity, the indices pb and rt as well as the argument lists were left out.

In figure 2a, $\Delta\tilde{N}$ of the unmodified algorithm for a placement of the operation o_2 onto the time steps $a = 0$ and $b = 2$ is shown. The distribution function $\tilde{N}$ of the current partial solution $\tilde{\tau}$ is located below these functions. For the tentative placement of o_2 onto the time step a, an exemplary calculation of the force F_a is performed: $\Delta\tilde{N}_a$ shows an displacement of +2/3 at time 0 and of -1/3 at times 1 and 2 for the distribution $\tilde{N}$. According to equation 6, the force of this placement is

$$F_a = +2/3 \cdot 1/3 - 1/3 \cdot 1/3 - 1/3 \cdot 4/3 = -1/3$$

and therefore obviously leads to an improvement. A placement on the time step b leads to a force of $F_b = 2/3$ and therefore to an unfavourable solution. The placement b is selected and by reducing the time frame fr_{o_2} to $\{0,1\}$ removed from the possible neighbourhood solutions. Due to identical forces of both placements in the following iteration (figure 2b), the IFDS algorithm arbitrarily selects the placement $a = 0$ and schedules the operation o_2 onto time step $b = 1$ (figure 2c).

Figure 2d shows the correspondent situation of the first placements for the modified algorithm. The graphs on the right ($\tilde{N}, \Delta\tilde{N}_{a/b}$) are based on absolute time t_{abs} while the graphs on the left ($\tilde{N}mm, \Delta\tilde{N}mm_{a/b}$) are using time steps t_{per} transformed by equation 1.

On the left hand of each function $\Delta\tilde{N}$ the modulo maximum transformed $\Delta\tilde{N}mm$ are depicted. For the placement $a = 0$ in the first iteration $\Delta\tilde{N}mm$ has only negative values. Due to the transformation, the positive displacement of $\tilde{N}$ at time 0 is hidden by the value 1 at time 2. This leads to a further improvement of the good result for F_a by $2/3 \cdot 1/3$. The negative displacement for placement $b = 2$ at time 0 is now hidden by an even higher value of 2 for time 2. This missing negative share on force F_b worsens the result by $1/3 \cdot 1/3$. Please note the higher preference of placement $a = 0$ compared to the unmodified algorithm.

In the second iteration, shown by figure 2e, the hiding effect is even more visible. In the placement at time $b = 1$ the complete negative part $1/2 \cdot 1/2$ is hidden. The placement, which was considered neutral in the unmodified IFDS algorithm, now has a significantly lower rating. The placement $a = 0$ is rated $1/2 \cdot 1/2$ better by the modified method than the neutral force of the unmodified algorithm. Because of the preferred placement $a = 0$, the operation o_2 is fixed to this time step in the final solution. The function $\tilde{N}mm$ in figure 2f shows the periodic usage of time step 0 of this resource type and therefore enables another process to use the other time step.

Besides the known effect of smoothing the resource requirements over all time steps, the additional rating for changes of $\tilde{N}$ hidden by the transformation results in an additional periodic alignment of the operations of a global resource type.

5.2. Global balancing

The second part of the modifications grants a balancing of the resource requirements in a group of blocks. Let $PB_{rt}(p)$ be the set of all blocks of a process $p \in P_{rt}$ accessing resource type rt. Blocks belonging to one process can be handled like branches of an alternation (because of condition (C2), the blocks are guaranteed not to overlap). Therefore the maximum of the distribution function $\tilde{N}mm_{rt,pb}(\tilde{\tau},t)$ of all blocks from $PB_{rt}(p)$ has to be computed:

$$\tilde{N}mm_{rt,PB}(\tilde{\tau},t,p) = \max_{pb \in PB_{rt}(p)} \tilde{N}mm_{rt,pb}(\tilde{\tau},t). \tag{8}$$

In order to balance the resource requirements for all processes, the sum of all $\tilde{N}mm_{rt,PB}(\tilde{\tau},t,p)$ over $p \in P_{rt}$ is built analogously to equation 4.

$$\tilde{N}mm_{rt,P}(\tilde{\tau},t) = \sum_{p \in P_{rt}} \tilde{N}mm_{rt,PB}(\tilde{\tau},t,p). \tag{9}$$

5.3. Integration

The force of the modified IFDS algorithm is now given by the following components:

$$F^{+}(\tilde{\tau},o_{pl},t_{pl}) = \sum_{rt \in RT} \sum_{t \in T^{+}(rt)} \Delta\tilde{N}^{+}_{rt}(\tilde{\tau},\tilde{\sigma},t)\, \tilde{N}^{+}_{rt}(\tilde{\tau},t)$$

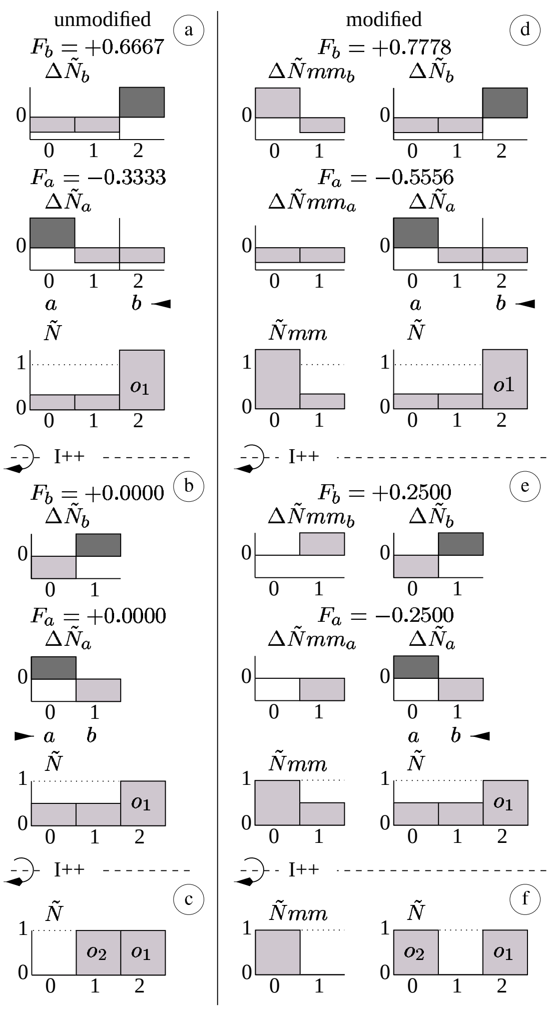

Figure 2. Unmodified and first part modified IFDS algorithm for two iterations

where

$$\Delta\tilde{N}^+_{rt}(\tilde{\tau},\tilde{\sigma},t) = \begin{cases} \Delta\tilde{N}mm_{rt,pb}(\tilde{\tau},\tilde{\sigma},t) & \text{if } rt \in RT_g, \\ \Delta\tilde{N}_{rt}(\tilde{\tau},\tilde{\sigma},t) & \text{otherwise.} \end{cases}$$

$$\tilde{N}^+_{rt}(\tilde{\tau},t) = \begin{cases} \tilde{N}mm_{rt,P}(\tilde{\tau},t) & \text{if } rt \in RT_g, \\ \tilde{N}_{rt}(\tilde{\tau},t) & \text{otherwise.} \end{cases}$$

and

$$T^+(rt) = \begin{cases} T_{per}(rt) & \text{if } rt \in RT_g, \\ T(pb) & \text{otherwise.} \end{cases}$$

with operation o_{pl} defining block pb.

The force $F(\tilde{\tau}, o_{pl}, t_{pl})$ is substituted in the IFDS algorithm by $F^+(\tilde{\tau}, o_{pl}, t_{pl})$. Now all blocks of the system are processed simultaneously. A partial solution $\tilde{\tau}$ therefore describes the time frames of all operations of the system. Still the modification doesn't influence the basic iterative behaviour of the algorithm.

Please note, that the complexity of the IFDS algorithm is not increased by the additional computation of the modulo-maximum transformation and $\tilde{N}mm_{rt,P}(\tilde{\tau},t)$. The additional effort in equations 7, 8 and 9 is bound by a constant multiple of the calculation effort in equation 4.

6. Implementation

The Modulo Scheduling method using the IFDS algorithm was implemented in a system called IPS and is based on parts of the Olympus Synthesis System [11]. At the moment step (S1) have to be done manually, the possible periods of step (S2) are automatically generated by a permutation. The permutation complexity is bound by $\prod_{p\in P}(td_{max}(p) - td_{min}(p) + 1)$, but typically the most sets of $ap(rt_g)$ are filtered out by equation 3 before scheduling.

7. Experimental

For experimental evaluation purposes the elliptical wave filter and the main loop of the differential equation solver from the HLS workshop benchmarks 1991 are used. The comparator function is substituted by a subtraction. This limits the operation types to addition, subtraction and multiplication. The execution time of the addition and subtraction is set to a unit delay of 1, the pipelined multiplication execution time is set to 2. The area cost for a multiplication is set to 4, for an addition and subtraction to 1.

The scheduled system consists of 5 independently running processes. Processes p_0, p_1 and p_2 are elliptical wave filters, processes p_3 and p_4 are the main loops of the equation solver. The total execution time of process p_0 and p_1 was set to 30, p_2 to 25 and p_3 and p_4 to 15. 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 global types adder and multiplier are assigned to all processes. In addition a global subtracter is assigned to the processes p_3 and p_4. The period $ap(rt_g)$ of all resource types is set to 5. The modified IFDS algorithm uses a look-ahead factor of $1/3$ and a global spring constant z of 3.

Table 1 shows the scheduling results of the multi-process example. The left column contains the resource types and processes. The second column shows the modulo-maximum transformation $\tilde{N}mm$ of the probability density function $\tilde{N}$, which is given in the fourth column. The periodical alignment of operations by the first part of the modification in $\tilde{N}$ can be easily seen. The number of required resources are given in column three. All together, four adders,

Table 1. Scheduling results of the multi-process example

+	$\tilde{N}mm$	#	$\tilde{N}$
p_0	11111	1	˜11111..111111111111111111111˜
p_1	1.112	2	˜.1121..121.1121.1121.1121.11˜
p_2	12211	2	˜221..1211112111121.122.˜
p_3	1....	1	1....1.......˜˜
p_4	...1.	1	...1....1....˜˜
all	43444	4	
-			
p_3	1..1.	1	˜....1..1.1...˜
p_4	.11..	1	˜.....1....11.˜
all	1111.	1	
$*$			
p_0	.1.1.	1	˜˜˜˜..1.1..1.1..1.1....1..1˜˜˜
p_1	.11..	1	˜˜˜˜..11...11...1....11...1˜˜˜
p_2	1..11	1	˜˜˜˜11..11...1.1...11.˜˜˜
p_3	1.111	1	1.1111.1...˜˜˜˜
p_4	111.1	1	111.1..1.1.˜˜˜˜
all	33333	3	

one subtracter and three multipliers are required with overall area cost of 17. The scheduling time with 711 iterations on a Pentium 133MHz running Linux was 7 seconds.

A pure local assignment of the resource types with identical parameters resulted after 728 iterations and an execution time of 5 seconds in a resource requirement of six adders, two subtracters and five multipliers. The 1.65 times higher area cost of 28 is mainly contributed by the two additional multipliers.

Whether or not the area saving due to the global adders and subtracters is compensated by additional multiplexors and wires is not considered. The global sharing of all resource types is applicated here to demonstrate the ability of the extension to handle many global sharings concurrently.

8. Conclusions

This paper presents a new method for the extension of static scheduling algorithms, based on a time-dependent and periodic assignment of globally shared resources to processes. The access control is static with no need of a runtime executive. This overcomes the requirement of *at least* one resource per operation type and process, which is given in traditional scheduling algorithms. The method remains applicable in reactive systems, if the use of other methods such as process merging or Interface Matching are impossible or ineffective.

Restrictions concerning the block starting times and their dependencies from the resource assignment were explained. The Improved Force-Directed Scheduling has been extended with two modifications which allow the sharing of global resources in multi-process systems. The capability of lowering the resource usage is shown using an example of three independent elliptical wave filters and two differential equation solvers. With a pure global resource assignment the area is decreased by 40 % compared to a traditional scheduling.

Current work is in progress in order to automatically select the assignment scope of each resource and to find the optimal periods of the global resource types without a complete enumeration.

References

[1] 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.

[2] Roni Potasman, Joseph Lis, Alexandru Nicolau, and Daniel D. 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] Christoph Jäschke and Rainer Laur. Resource constrained scheduling with global resource sharing. In *11th International Symposium on System Synthesis*, pages 60–65, 1998.

[9] 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.

[10] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, J.L. van Meerbergen, and A. van der Werf. Improved force-directed scheduling in high-throughput digital signal processing. *IEEE Transactions on Computer Aided Design*, 14(8):945–960, August 1995.

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