Target Architecture Oriented High-Level Synthesis for Multi-FPGA Based Emulation*

Oliver Bringmann[1,2], Carsten Menn[1], Wolfgang Rosenstiel[1,2]

[1] FZI, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany
[2] Universität Tübingen, Sand 13, 72076 Tübingen, Germany
bringmann@fzi.de, menn@fzi.de, rosen@informatik.uni-tuebingen.de

Abstract

This paper presents a new approach on combined high-level synthesis and partitioning for FPGA-based multi-chip emulation systems. The goal is to synthesize a prototype with maximal performance under the given area and interconnection constraints of the target architecture. Interconnection resources are handled similarly to functional resources, enabling the scheduling and the sharing of interchip connections according to their delay. Moreover, data transfer serialization is performed completely or partially, depending on the mobility of the data transfers, in order to satisfy the given interconnection constraints. In contrast to conventional partitioning approaches, the constraints of the target architecture are fulfilled by construction.

1 Introduction

Due to increasing design complexity, the emulation of complex systems embedded in a real hardware environment as a prototype is becoming more and more important, in design validation. In this context high-level synthesis is gaining importance in accelerating the design flow for rapid prototyping and closing the gap to the system level within a hardware/software codesign environment. During the last years, several commercial and academic emulation platforms have been developed. Particularly, the usage of multiple-chip FPGA-based emulation platforms are very attractive due to their high flexibility, reusability, and scalability. However, state-of-the-art synthesis systems produce insufficient results when a design is to be mapped and partitioned onto a multiple-chip target architecture. Especially the delay and the limited number of interconnection resources have been insufficiently considered during synthesis, up till now.

Partitioning a design onto a multiple-chip target architecture can be performed at various levels of abstraction. Very common are structural partitioning approaches applied at the RT or gate level. An overview can be found in [1]. Due to the highly interconnected structural components, any partitioning results in many interconnections crossing between the chips. A valid implementation according to the limited pin count of the chips quite often enforces the generation of very small partitions consuming not more than 10% of the area of each chip. Moreover, inferior global synthesis decisions, taken due to the unconsidered target architecture during synthesis, can produce non-implementable designs. The missing global knowledge about the circuit semantics and the synthesized circuit timing in lower level partitioning approaches make it difficult to integrate some re-synthesis tasks, like re-scheduling of the I/O operations for improving the partitioning result.

In [2] the insufficient results of structural partitioning and the advantages of functional partitioning have been discussed. But they restrict functional partitioning to partitioning procedures before high-level synthesis. Due to the missing knowledge on the synthesized RT structure, partitioning before high-level synthesis produces insufficient results as well. Especially the sharing of functional datapath units by different operations of the CDFG cause adverse effects on the partitioning before synthesis, because a lot of operations mappable to a single functional resource could result in too many partitions and interchip connections. Hence, a direct interpretation of the CDFG as a datapath to be partitioned produces insufficient results, since the synthesized datapath can have a quite different structure.

The objective of this paper is to incorporate partitioning and an extended interconnection cost model into high-level synthesis, enabling a synthesis-driven partitioning technique that considers the influence of functional resource sharing, schedules interchip connections according to their delay, provides automatic selection of completely or partially serialized data transfers, and supports interconnection sharing of single or multiple processes. Because emulated systems can even run fast enough to be used within the real hardware environment as a prototype, the optimization goal is to maximize the circuit performance under the resource constraint of the given multiple-chip target architecture.

1.1 Related work

State-of-the-art high-level synthesis systems focus mainly on the synthesis of single processor designs. There

* This work is partially supported by the DFG program “Rapid Prototyping of Embedded Systems with Hard Time Constraints” under Ro1030/4.

exist a few approaches that combine high-level synthesis and partitioning. Early approaches perform partitioning before synthesis in order to shorten the synthesis time (YSC [3], [4]), or guide the design space exploration during synthesis (BUD [5]). They use a hierarchical clustering technique based on several closeness metrics to group similar objects, where each iteration generates a partitioning that can be evaluated to choose the best one for implementation. Mapping to a multiple-chip architecture is not intended.

Multiple-chip architectures are specifically addressed by the approaches discussed in the following. APARTY [6] tries to improve hierarchical clustering by applying a multi-stage clustering approach with manually selectable closeness metrics. However an implementation, satisfying given pin or area constraints can not be guaranteed. A bipartitioning approach has been presented in [7], where the partitioning is performed after scheduling and binding is completed. Interchip data-transfers can not explicitly be scheduled, but a re-scheduling is possible. Area, pin, and timing constraints are tried to be satisfied by applying iterative improvement techniques, like simulated annealing or a min-cut algorithm. Another approach based on the iterative improvement technique is given in [8]. Since this approach is carried out before synthesis, procedures are used as partitioning objects in order to reduce inferior design decisions in contrast to a partitioning of fine-grained operations which could share the same resources. The main problems are, finding a well-balanced number of procedures to be partitioned to the given target architecture, and the missing close connection to high-level synthesis.The objective of PARAS is to optimize the system performance for a given homogenous multiple-chip architecture with a fixed distribution of functional units [9]. Advantageous is that partitioning and scheduling are tightly combined, but the approach is restricted to dataflow-oriented specifications without conditional branches and loops, and area and pin constraints can not be considered. Unfortunately, only a bipartitioning approach has been explained. In [10], an ILP model incorporating partitioning, scheduling, and functional unit allocation has been presented. However, functional units have to be distributed in advance and automatic serialization of data transfers are not supported. Interconnection sharing are restricted to bus assignment with a given bus width. Furthermore, external interconnections can not be shared, and the actual interconnection structure is not taken into account. A discussion concerning the synthesis tool performance is missing, especially if all conditions are integrated, including that for control structures. In contrast to this, CHOP does not provide automatic circuit partitioning, but can assist the designer in partitioning behavioral specifications onto multiple chips [11]. The approach presented in [12] does not provide partitioning as well, but tackles bus assignment for a given interconnection structure and performs scheduling, subsequently.

The problem of traditional functional partitioning approaches is illustrated in Figure 1 (a). Due to the missing knowledge about the allocated functional units, the synthesized schedule, and the generated RT structure, usually graph-based techniques are applied in pre-synthesis partitioning approaches. A possible partitioning is given by the cut-line crossing two inter-chip connections. This partitioning separates two of the three multiplications, which would require the generation of additional partitions, after synthesis, if not much more than one multiplier can be implemented on a single FPGA.

An integrated partitioning is depicted in Figure 1 (b). Due to the knowledge about the allocated functional units and the schedule, it can be derived that all multiplications, subtractions, and additions can be mapped to one multiply, subtract, and add unit, respectively, resulting in a reduced interconnection structure. Based on the area and timing characteristics of the allocated functional units a proper number of partitions can be found. Although five cuts seem to be needed, only one inter-chip connection is sufficient to implement the design on two FPGAs, when applying interconnection sharing, as illustrated in Figure 1 (c). For the sake of clarity, only the partitioned datapath of the design are illustrated within the FPGAs.

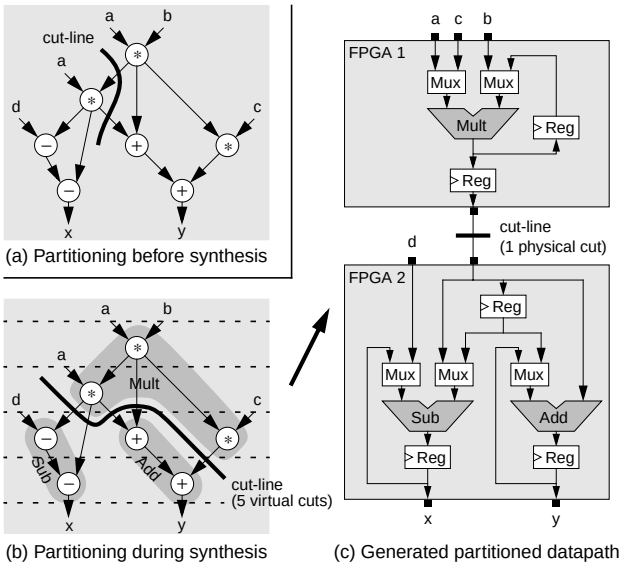


Figure 1. Benefit of integrated partitioning

2 Synthesis flow

As mentioned before, the objective is to incorporate circuit partitioning and high-level synthesis based on an extended interconnection cost model for a given multiple-chip target architecture. Partitioning produces proper results if all functional units have been allocated and the interconnection structure has already been generated. However, the

interconnection structure is not fixed before scheduling and binding is completed. On the other hand, functional unit allocation mainly depends on the circuit specification, the timing constraints, and the resource constraints given by the target architecture. Hence, the approach starts with functional unit allocation, constrained by the overall area of the previously read multiple-chip target architecture, as presented in [13]. Result of the allocation process is to provide several promising sets of allocated functional units to be scheduled and partitioned, in order to select the best one for implementation. The target architecture, including its interconnection structure, can easily be described using a proprietary file format. Since scheduling and partitioning are strongly correlated and we want to utilize the partitioning result during scheduling, the expected scheduling decisions and the influences of the target architecture are estimated. Based on a multi-stage estimation function, an initial partitioning can be calculated. This step can be distinguished in distribution of the already allocated functional units to the FPGAs, as presented in Section 3, and in assignment of the operations to the FPGAs according to the previously distributed functional units, as shown in Section 4.

In order to minimize unfavorable partitioning decisions taken before scheduling, the initial partitioning result is expressed by a weighted distribution of the operations to the partitions. The weights are fixed during the following scheduling phase. Furthermore, pin constraints of the FPGAs are considered and satisfied during scheduling by applying interconnection sharing and data transfer serialization, as described in Section 5. The actual assignment of operations to functional units and FPGAs is done using a conventional binding algorithm [14]. Last of all, the controller has to be distributed to all allocated chips. A structural controller partitioning approach is not recommended, because numerous additional interchip connections could violate the given pin constraints. Therefore, the controller is partitioned implicitly by constructing separately a communicating controller for each scheduled DFG partition. At most one additional signal is needed to control each interchip communication, which can be transferred by a shared interchip connection.

In this paper, we concentrate on the distribution of the functional units and the mapping of the operation to the FPGAs. This allows easy integration of the partitioning approach in other high-level synthesis systems. The extension of the scheduling algorithm for interconnection sharing, data transfer serialization, and consideration of the pin constraints are addressed afterwards. All the other topics are beyond the scope of this paper due to space limitations.

3 Distribution of the functional units

After functional unit allocation is completed, the allocated functional units (FU) are distributed to the FPGAs of the given target architecture. Because several FUs able to perform the same operations can be allocated, proper partitioning decisions of the DFG operation can only be taken if the FUs are already distributed to the FPGAs. Otherwise, concurrent operations could be partitioned onto different FPGAs although an adequate number of FUs are provided within each FPGA. In order to determine a proper distribution of the FUs, the control/dataflow graph (CDFG) has to be analyzed with respect to pairs of interconnected operations and potential critical paths. The approach operates on a global DFG with control flow extensions and references to the corresponding CFG nodes. Loops and conditional branches are realized using multiplexers, and the feedback edges of all loops are immediately known. This enables a partitioning across control structures, which is important in control-dominated designs. Even so, the effects of alternative branches are not neglected during partitioning, especially the potential sharing of mutually exclusive resources.

Before the algorithm for FU distribution is presented, some notations and definitions are given. We assume, that functional units are already allocated and the time frames $T_{op} = \{t_{op}^{asap}, \ldots, t_{op}^{alap}\}$, for all operations $op \in OP$, and $T_e = \{t_e^{asap}, \ldots, t_e^{alap}\}$, for all interconnections $e \in E$, are computed by ASAP/ALAP algorithms, where OP denotes the set of all operations of the DFG and E the edges, interconnecting the operations of the DFG. The probability that an operation or an interconnection a, respectively, will be assigned to clock step t can be calculated by $p_a(t) = 1/|T_a|$. Furthermore, let $\underline{P}(op_1, op_2)$ be the shortest non-directed path within the DFG between the operations op_1 and op_2, and $OT(op) \in OP_{Type}$ be the type of operation op, out of the set of all used operation types OP_{Type}.

Definition 1. The *concurrency(a, b)* of two operations/interconnections a and b denotes the probability that both operations/interconnections are concurrently used in the synthesized design and is calculated, as follows:

$$concurrency(a,b) = \sum_{t \in T_a \cap T_b} p_a(t) \cdot p_b(t).$$

In addition, the concurrency is defined to be zero, if the concerned operations/interconnections a and b are in alternative branches. If the synthesis system supports speculative execution as well, then the concurrency becomes only zero, if the condition

$$t_a^{asap} - t_{op_{c_e}}^{asap} > \left|\underline{P}(op_{c_b}, op_{c_e})\right| \wedge t_b^{asap} - t_{op_{c_e}}^{asap} > \left|\underline{P}(op_{c_b}, op_{c_e})\right|$$

holds, where $\underline{P}(op_{c_b}, op_{c_e})$ denotes the shortest path for evaluating the corresponding branch condition beginning at operation op_{c_b} and ending at operation op_{c_e}. Note that the complementary probability of the concurrency expresses that both operations can share the same resources.

Definition 2. The probability that the operation op_1 and op_2 are scheduled on the critical path are denoted by *criticalPathProb*(op_1, op_2) and can be calculated by

$$criticalPathProb(op_1, op_2) = \sum_{t \in (T_{op_1} + latency(op_1)) \cap T_{op_2}} p_{op_1}(t) \cdot p_{op_2}(t),$$

if $|\underline{P}(op_1, op_2)| = 1$ and $t^{asap}_{op_2} - t^{alap}_{op_1} \leq 1$ holds. Otherwise the function *criticalPathProb* returns zero. Note that the function *latency(op)* are used to enlarge the time frame T_{op_1} by the execution time of the fastest allocated FU, able to execute *op*. Without loss of generality, it is assumed that op_2 is the successor of op_1.

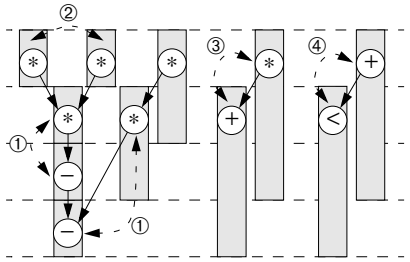
Definition 3. The *attraction* between operators denotes the relative frequency of external data transfers caused when two operators ot_1 and ot_2 are partitioned to different chips. A different calculation is needed for equal and different operators. The attraction for directly connected different operators is defined by

$$attraction(ot_1, ot_2) = \frac{1}{|E|} \cdot \sum_{\substack{\forall op_1, op_2 : OT(op_1) = ot_1 \wedge OT(op_2) = ot_2 \\ \wedge |\underline{P}(op_1, op_2)| = 1}} criticalPathProb(op_1, op_2) \quad ,$$

and the attraction for equal operators with a non-directed distance of two and overlapping time frames is defined by

$$attraction(ot_1) = \frac{1}{|E|} \cdot \sum_{\substack{\forall op_1, op_2, op_3 : OT(op_1) = ot_1 \wedge OT(op_2) = ot_1 \\ \wedge |\underline{P}(op_1, op_3)| = 1 \wedge e_1 = \underline{P}(op_1, op_3) \\ \wedge |\underline{P}(op_2, op_3)| = 1 \wedge e_2 = \underline{P}(op_2, op_3)}} concurrency(e_1, e_2) \quad .$$

In case of equal operators, the attraction has to be calculated differently because consecutive equal operations have a high sharing probability, so there is no need to implement the related FUs in a single FPGA. But if the DFG contains a lot of highly concurrent equal operations with a distance of two, it is beneficial to implement similar FUs, able to execute that operation, in one FPGA. In contrast, a lot of different non-concurrent operations on the critical path force the implementation of FUs, which can execute the different operations, in one FPGA. Thus, the attraction measure tries to ensure that time-critical operations can immediately be executed without much data transfers. Note that accesses to external I/Os are modeled similarly to the other DFG operations. This is important especially for partitioning operations, which mainly depend on external I/Os. The calculation of the attraction measure is illustrated in Figure 2, using the differential equation example taken from [15]. In this example, the multiply and subtraction operators have the highest attraction, forcing the implementation of the corresponding FUs, which are able to perform multiplication and subtraction, in one FPGA. For the sake of clarity, the attraction measures with external I/Os are not illustrated in this example and all FUs have an assumed latency of one.



① *attraction*(∗, −) := 1/8 + 1/(2∗8) = 0.1875
② *attraction*(∗) := 1/8 = 0.125
③ *attraction*(∗, +) := 3/(9∗8) = 0.0416
④ *attraction*(+, <) := 3/(9∗8) = 0.0416

Figure 2. Attraction calculation for the HAL example

Based on the attraction measure, the allocated functional units (*FU)* can now be distributed to the FPGAs. Note that functional units can also be assigned manually to specific FPGAs in advance. Objective of this step is to concentrate or to distribute specific functionality to be realized in the FPGAs depending on the DFG. A high number of edges connecting, for instance, add and multiply operations on the critical path imply a solution implementing functional units which provide add and multiply operators on each FPGA.

---

for all operator pairs ot_1, ot_2 in descending order of *attraction* **do**
  if different operator pairs have same *attraction* measure **then**
    select operator pair with the least number of unmapped FUs;
  fi
  for all $f_1, f_2 \in FU$ in ascending cost order with $ot_1 \in f_1 \wedge ot_2 \in f_2$ **do**
    let F be the FPGA with maximal available area;
    if F has still enough area to implement f_1 and f_2 **then**
      map f_1, f_2 to FPGA F; $FU := FU \setminus \{f_1, f_2\}$;
    fi
  od
  for all $f \in FU$ in ascending cost order able to execute ot_1 or ot_2 **do**
    let $ot \in \{ot_1, ot_2\}$ be executable by f and $ot \neq \overline{ot} \in \{ot_1, ot_2\}$;
    let F be the FPGA, with least number of FUs able to execute ot
      and there exists another FU that can execute $\overline{ot}$;
    if F has still enough area to implement f **then**
      map f to FPGA F; $FU := FU \setminus \{f\}$;
    fi
  od
od
distribute well-balanced all still unmapped FUs to the FPGAs;

Algorithm 1. Distribution of functional units to FPGAs

The presented distribution algorithm and the underlying attraction measure generally provides the intended homogeneous distribution of the functional units, but in cases of a quite non-balanced occurrence of different operations in the DFG, the functional units are distributed inhomogeneously.

4 Operation assignment

The operations of the DFG are assigned to the different FPGAs, depending on the previously distributed functional units, the given pin constraint of each FPGA, and the inter-chip connection delay. The assignment approach based on the force-directed list scheduling algorithm [15] enhanced by a two-phase priority and multi-stage estimation function, which guides the assignment of the operations to the

FPGAs. During the first phase, all ready operations are ordered by the force-directed priority function, according to the given resource constraints. During the second phase, the assignment of the operations to the FPGAs are performed by the following shown multi-stage estimation function, which evaluates all tentatively taken assignments with respect to the previously determined operation priority. The estimation function is called for each operation *op* in ascending order of their forces, terminates once one stage computes a unique result, and returns the assigned FPGA.

1. *Concurrency*: discard all FPGAs with the property that all FUs able to execute the current operation *op*, have already been occupied by concurrent operations.
2. *Connectivity*: minimal number of interchip connections caused by the current operation *op* concerning all FPGAs containing FUs able to execute *op*.
3. *Backward sequential closeness*: minimal distance of the current operation *op* to all preceding operations (*pred(op, F_i)*), with a high sharing probability, in the tentatively chosen FPGA F_i, calculated by
$$min\left\{\frac{1}{n}\sum_{op_i \in\ pred(op,F_i)} concurrency(op,op_i)\cdot\left|\underline{P}(op,op_i)\right| : F_i \in \boldsymbol{F}\right\}$$
where $\boldsymbol{F}$ denotes the set of available FPGAs. This criteria considers the probability that the current operation *op* can share the same FU with previously assigned operations.
4. *Forward sequential closeness:* minimal distance, within a given diameter, to the first subsequent operation in the CDFG which can not be executed on the tentatively chosen FPGA. This stage is applied in order to avoid a potential increase of latency caused by resource conflicts due to unfavorable assignments.
5. *Parallel closeness*: minimal distance to all operations currently in the ready set. This stage has the same focus as the previous stage, but takes the influence of concurrent operators into account.
6. *Utilization*: the last criteria selects the currently lowest utilized FPGA. If all FPGAs have the same utilization, then an arbitrary FPGA can be chosen.

All closeness and connectivity measures are weighted by using the functions *criticalPathProb* and *concurrency*, as introduced in Section 3. This is important because a sequential distance, for instance, is only significant for partitioning, if most interconnections are on the critical path. Otherwise, there is sufficient freedom to find a suitable partition or to apply optimization techniques, like interconnection sharing or data transfer serialization, in order to satisfy given pin constraints.

Figure 3 shows an example for the stages 4 and 5 with two FPGAs. The first FPGA (F_1) contains two multiply and one subtraction unit and the second FPGA (F_2) one add and two multiply units as well. The forward sequential closeness measure determines the distance between an operation of the ready set and a subsequent operation of the CDFG which could not be calculated on the same FPGA. In this example there is only one FPGA (F_1) capable to perform multiplications and subtractions. The decision, for instance, which of the two multiplications m_1 and m_2 or m_3 and m_4 of Figure 3 (a) should be assigned to the FPGA F_1 is assisted by the forward sequential closeness measure that prefers the pair m_1 and m_2, because they have a larger closeness to the subsequent subtraction, which must be calculated by FPGA F_1.

The parallel closeness measure determines the distance between concurrent operations from the ready set and subsequent operations consuming their results. The closer two concurrent operations are, the greater is the benefit when calculating these operations on the same FPGA. Assuming multiplication m_2 of Figure 3 (b) should be assigned to a multiply unit implemented in one of the two given FPGAs. Then, the lowest parallel closeness value (2) prefers the execution of m_2 together with m_1 on the same FPGA resulting in a reduced number of interconnections. Both stages are very important, especially at the beginning of the partitioning process, because there are only a few or even no information about preceding assigned operations available.

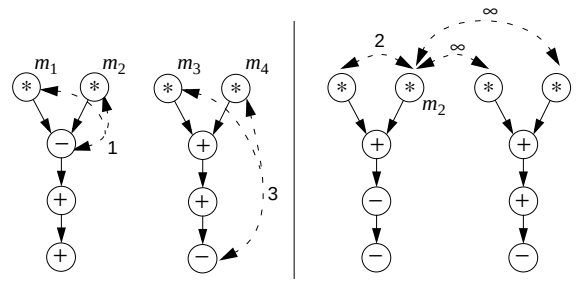


Figure 3. Forward sequential and parallel closeness

The stage ordering has been determined experimentally, using several synthesis benchmarks. At the beginning of the assignment process, mainly the forward sequential closeness and the parallel closeness measures induce the assignment decision. This is important, because initially, the measures at previous stages quite often produce similar results for all considered operations, since no or just a few operations have already been mapped to the FPGAs. Once several operations have been assigned to the different partitions, the connectivity and the backward sequential closeness measure strengthen their influence. The combined application of the presented measures together with the already known operation forces provide a globally enhanced evaluation of the assignment of operations to FPGAs. Note that each access to an external I/O is handled similarly to functional operations, if external I/Os of the overall system can not directly be provided at each FPGA.

```
while DFG is not completely assigned do
   compute time frames (TF) and forces for each operation;
   determine operations (op) which can be assigned at current c-step;
   if not all critical op can be assigned then insert c-step; re-eval TF;
   for all ready operations in ascending force order do
      call multi-stage estimation function;
      store best assignment;
   od
od
```

Algorithm 2. Assignment of operations to functional units

5 Scheduling under I/O constraints

Depending on the initial partitioning, I/O operations are inserted between operations that are assigned to different partitions. The I/O operations are annotated with the actual interchip delay of the target architecture and are scheduled similarly to functional operations. Since I/O data transfers can have unit or combinational delay, the scheduling algorithm checks, if the overall delay of chained functional and I/O operations scheduled in a single cycle, exceeds the given clock frequency. Then the data transfer is registered and postponed to the next clock cycle. Additionally, with respect to given pin constraints, pins are assigned to multiple I/O operations and data transfer serialization is carried out by the scheduling algorithm. If the pin constraints can not be satisfied, yet, a new clock cycle is inserted.

List scheduling is used as a scheduling algorithm, driven by a global estimation function based on the probability of scheduling operations to control steps, which additionally evaluates possible assignments of operations to functional unit types [14]. Whereas no changes are needed for the probability calculation of functional operations, several extensions are needed to cope with pin constraints. The I/O operations op_{io} are represented by a separate distribution function $D_{io}(t)$ which has to incorporate the word width $width(op_{io})$ of each I/O operation and can be calculated by

$$D_{io}(t) = \sum_{op_{io} \in OP_{io}} width(op_{io}) \cdot p_{op_{io}}(t).$$

This distribution function denotes the overall pin consumption within a given clock step under an assumed uniform pin utilization within each time frame without consideration of any pin constraints. Due to the quite different time frame density in each clock cycle, the distribution function can demand much more or much less pins than provided by all FPGAs. In order to take pin constraints into account, each interchip data transfer has to be divided into a send and a receive operation. An access to an external I/O requires either only a send or only a receive operation. Hence, for all send and receive operations of each FPGA, the following *pin constraint condition* has to be satisfied:

$$\sum_{\forall op_{io} \in ReadySet_{io}(t) : sender(op_{io}) \in F_i \vee receiver(op_{io}) \in F_i} \left\lceil width(op_{io}) / \left| T_{op_{io}} \right| \right\rceil \leq IO(F_i), \forall F_i \in \boldsymbol{F}$$

where $ReadySet_{io}(t)$ denotes the set of all I/O operations, which can be scheduled at the current clock step t, $IO(F_i)$ returns the number of pins of FPGA F_i, and $sender(op_{io})$ and $receiver(op_{io})$ refer to the FPGA containing the send or receive operation op_{io}, respectively.

Note that the time frames $\left| T_{op_{io}} \right|$ are gradually reduced during scheduling. The final time frame represents the number of clock steps used by a serialized data transfer. In each clock step of the time frame, $\left\lceil width(op_{io}) / \left| T_{op_{io}} \right| \right\rceil$ data bits of the I/O operations can be transferred. In the following algorithm, only the scheduling extensions concerning I/O constraints are presented. Since the amount of serialized data transfers should be limited, the objective of scheduling is to postpone I/O operations, with the globally lowest force, until the demanded number of pins can be allocated. If unused pins ($IO_{remained}(F_i)$) remain, then serialize that I/O operation, that provides the minimal force sum of the length $width(op_{io}) / IO_{remained}(F_i)$, in order to include the entire serialization period. Note that the operation force outside its time frame is defined to be *infinite*. The data serialization of an I/O operation ends in that cycle, that provides a sufficient number of pins to which the remaining bits of the I/O operation can be mapped, depending on the current forces of all I/O operations. Afterwards, the actual subword width of the serialized data transfer can finally be fixed. This is shown in the procedure handle_pin_constraints (Algorithm 3), which is invoked in each iteration of the list scheduling algorithm.

```
procedure handle_pin_constraints
while pin constraint condition are not fulfilled do
   if all op_io on critical path then insert c-step; re-eval. time frames;
   postpone, or expand serialization of, op_io with globally lowest force;
od
map remaining op_io to suitable pins of the involved FPGAs;
if unconsidered pins remain then
   serialize op_io with min. force sum of length width(op_io)/IO_remained(F_i);
fi
```

Algorithm 3. Scheduling extension to handle pin constraints

6 Experimental results

In order to enable a comparison with related approaches concerning multiple-chip architectures, we disabled in our approach data transfer serialization and perform only a manual distribution of functional units according to the results of the other approaches. Since, only bi-partitioning results have been presented so far, Table 1 is restricted to bi-partitioning of several benchmarks. The latency is assumed to be one clock cycle for the adder and two clock cycles for the pipelined multiplier. The delay of interchip connection is statically fixed to one clock cycle. The CPU time is related to a Sun Ultra 1 with 167 MHz.

Table 1. Restricted approach for results comparison

Example	Partition 1	Partition 2	c-steps	data transfers	CPU (sec)
EWF	1 + , 1 *	1 + , 1 *	18 + 1	1 × 16 bit	2
EWF	2 + , 1 *	2 + , 1 *	17 + 1	1 × 16 bit	2
AR	1 + , 1 *	1 + , 1 *	14 + 2	2 × 16 bit	1
AR	1 + , 2 *	1 + , 2 *	11 + 2	2 × 16 bit	1
HAL	1 – , 2 *	1 + , 1 >	6 + 0	0	< 1

Secondly, experimental results of our approach with respect to the constraints of a real FPGA-based emulation platform [16] are presented. Each module of the emulator consists of four Xilinx 4025 FPGAs, where each FPGA is connected via a 77 bit bus with two of its neighbors and have a 88 bit external bus. Additionally, the mapping of a second module with similar structure, but composed of four Xilinx 4013 FPGAs with 60 bit interchip buses and 64 bit external buses, are provided. The used FUs have been generated by a module generator for the Xilinx XC4000 family, where an 16 bit adder needs 9 CLBs and an 16 bit pipelined multiplier 250 CLBs. Interchip data transfers between two multiplications require one additional clock cycle and all other interchip data transfers can be chained within a single clock cycle. Based on different FU allocations, the number of used FPGAs, the FU distribution to the FPGAs, and the latency and the number of external data transfers are presented in Table 2, with and without scheduling extensions. By applying interconnection sharing and data transfer serialization, designs which initially exceeds the given pin constraints (✗) could now be implemented (✓).

Table 2. Extended approach applied to different examples

FPGAs	Resources				logic	w/o extensions		with extensions	
	FPGA 1	FPGA 2	FPGA 3	FPGA 4	trans-fers	c-steps	transfers	c-steps	transfers
EWF									
1 × 4025	2 + , 1 *	–	–	–	0	19	0	19	0
2 × 4025	2 + , 1 *	2 + , 1 *	–	–	22	17	3	17	1
AR									
1 × 4025	2 + , 2 *	–	–	–	0	14	0	14	0
2 × 4025	1 + , 2 *	1 + , 2 *	–	–	45	11	5 ✗	11	2 ✓
2 × 4013	1 + , 1 *	1 + , 1 *	–	–	22	14	4	14	2
4 × 4013	1 + , 1 *	1 + , 1 *	1 *	1 *	77	11	12 ✗	11	4 ✓
DCT									
2 × 4025	2+/-, 2 *	2+/-, 2 *	–	–	34	9	12 ✗	9	2 ✓
2 × 4013	2+/-, 1 *	2+/-, 1 *	–	–	43	12	12 ✗	12	$2\frac{1}{2}$ ✓

The maximum number of data transfers that have to be executed concurrently in one c-step are shown in the columns “transfer” in Table 2. The fraction $2\frac{1}{2}$ comes from applying data transfer serialization and shows the benefit of this extension since one of the data transfers is performed in two c-steps. The column “logic” shows results obtained from logic partitioning by a min-cut algorithm. This serves not for a direct competition with our approach but it points out the advantages of our integrated partitioning approach.

7 Conclusion

In this paper a new approach on high-level synthesis for multi-FPGA based emulation has been presented. During synthesis, functional units are distributed, operations are partitioned, and interchip data transfers are inserted. Additionally, the interchip data transfers are scheduled, and serialized automatically, in order to maximize the circuit performance under the constraints of the given target architecture. By treating interchip data transfers similar to CDFG operations, optimizations like interconnection sharing or serialization of data transfers can be applied. This approach can easily be extended to handle multi-process specifications and to support the sharing of interchip connections belonging to different processes by applying the techniques presented in [17] and [18].

8 References

[1] F. M. Johannes: *Partitioning of VLSI Circuits and Systems.* Proceedings of DAC, 1996.

[2] F. Vahid, T. Le, Y. Hsu: *Functional Partitioning Improvements over Structural Partitioning for Packaging Constraints and Synthesis-Tool Performance.* Transactions on Design Automation of Electronic Systems, vol. 3, no. 2, 1998.

[3] R. Camposano, J. van Eijndhoven: *Partitioning a Design in Structural Synthesis.* Proceedings of ICCD, 1987.

[4] R. Camposano, R. Brayton: *Partitioning Before Logic Synthesis.* Proceedings of ICCAD, 1987.

[5] M. McFarland, T. Kowalski: *Incorporating Bottom-Up Design into Hardware Synthesis.* IEEE Transactions on CAD, vol. 9, pp. 938-950, 1990.

[6] E. Lagnese, D. Thomas: *Architectural Partitioning for System Level Synthesis of Integrated Circuits.* IEEE Transactions on CAD, vol. 10, no. 7, pp. 847-860, 1991.

[7] R. Gupta, G. DeMicheli: *Partitioning of Functional Models of Synchronous Digital Systems*. Proceedings of ICCAD, 1990.

[8] F. Vahid: *A Three-Step Approach to the Functional Partitioning of Large Behavioral Processes.* Proceedings of ISSS, 1998.

[9] W. Wong, R. Jain: PARAS: *System-Level Concurrent Partitioning and Scheduling.* Proceedings of ICCAD, 1995.

[10] C. Gebotys: *Optimal Synthesis of Multichip Architectures.* Proceedings of ICCAD, 1992.

[11] K. Küçükçakar, A. Parker: *A Methodology and Design Tools to Support System-Level VLSI Design.* IEEE Transactions on VLSI, vol.3, no. 3, pp 355-369, 1995.

[12] Y.-H. Hung, A. Parker: *High-Level Synthesis with Pin Constraints for Multiple-Chip Designs.* Proceedings of DAC, 1992.

[13] P. Gutberlet, J. Müller, H. Krämer, W. Rosenstiel: *Automatic Module Allocation in High-Level Synthesis.* Proceedings of EURO-DAC, 1992.

[14] W. Rosenstiel, H. Krämer: *Scheduling and Assignment in High-Level Synthesis.* In R. Camposano, W. Wolf: High-Level VLSI Synthesis, Kluwer, 1991.

[15] P. Paulin, J. Knight: *Force-Directed Scheduling for the Behavioral Synthesis of ASICs.* IEEE Transactions on CAD, vol. 8, no. 6, pp. 661-679, 1989.

[16] G. Koch, U. Kebschull, W. Rosenstiel: *A Prototyping Environment for Hardware/Software Codesign in the COBRA Project.* Proceedings of International Workshop on Hardware/Software Codesign, 1994.

[17] O. Bringmann, W. Rosenstiel: *Resource Sharing in Hierarchical Synthesis.* Proceedings of ICCAD, 1997.

[18] O. Bringmann, W. Rosenstiel, D. Reichardt: *Synchronization Detection for Multi-Process Hierarchical Synthesis.* Proceedings of ISSS, 1998.