Hardware/Software Partitioning using Integer Programming

(Extended Version)

Ralf Niemann

Lehrstuhl Informatik XII

University of Dortmund

Report No. 586

September 1995

Hardware/Software Partitioning using Integer Programming

Ralf Niemann

Lehrstuhl Informatik XII

University of Dortmund

Report No. 586

September 1995

Abstract

One of the key problems in hardware/software codesign is hardware/software partitioning. This paper describes a new approach to hardware/software partitioning using integer programming (IP). The advantage of using IP is that optimal results are calculated respective to the chosen objective function. The partitioning approach works fully automatic and supports multi-processor systems, interfacing and hardware sharing. In contrast to other approaches where special estimators are used, we use compilation and synthesis tools for cost estimation. The increased time for calculating the cost metrics is compensated by an improved quality of the estimations compared to the results of estimators. Therefore fewer iteration steps of partitioning are needed. The paper will show that using integer programming to solve the hardware/software partitioning problem is feasible and leads to promising results.

Contents

1 Introduction

Embedded systems typically consist of *application specific* hardware parts and programmable parts, i.e. processors like DSPs, core processors or ASIPs. In comparison to the hardware parts, the software parts can be developed and modified much easier. Thus, software is less expensive in terms of costs and development time. Hardware however, provides better performance. For this reason a system designer's goal is to design a system which fulfills all performance constraints by using as few as possible hardware. Hardware/software codesign deals with the problem of designing embedded systems, where automatic partitioning is one key issue. This paper describes a new approach in hardware/software partitioning for multi-processor systems working fully automatic. It uses integer programming (IP) to solve the partitioning problem optimally (or if desired, nearly optimally with decreased calculation time) for the chosen objective function. The cost model is not calculated by estimators like other approaches, because the quality of estimations is often bad and estimators do not concern compiler effects. In our approach the tools (a compiler for the software parts and a high-level synthesis tool for the hardware parts) are used instead of special estimators. The disadvantage of an increased runtime for calculating the cost metrics is compensated by a better quality of the cost metrics compared to the results of estimators. Further, better cost metrics lead to fewer partitioning iterations.

The outline of the paper is as follows: Chapter 2 gives an overview of related work in the field of hardware/software partitioning. In chapter 3 our own approach to partitioning is presented. A formalization of the hardware/software partitioning problem follows in chapter 4. Section 5 describes the problem by an IP-model. After experimental results of solving these IP-models have been presented in chapter 6, a conclusion is given in chapter 7 .

2 Related Work

There are only few approaches considering hardware/software partitioning. One of these is the COSYMA system ([EHB93],[HEY+95]), where hardware/software partitioning is based on simulated annealing using estimated costs. The partitioning algorithm is *software-oriented*, because it starts with a first non-feasible solution consisting only of software components. In an *inner loop partitioning (ILP)* software parts of the system are iteratively realized in hardware until all timing constraints are fulfilled. To handle discrepancies between estimated and real execution time, an *outer loop partitioning (OLP)* restarts the *ILP* with adapted costs ([HE94]). The *OLP* is repeated until all performance constraints are fulfilled.

Another hardware/software paritioning approach is realized in the *VULCAN* system ([GCJDM92]). This approach is *hardware-oriented*. It starts with a complete hardware solution and iteratively moves parts of the system to the software as long as the performance constraints are fulfilled. In this approach performance satisfiability is not part of the cost function. For this reason the algorithm will easily trap in a local minimum.

The approach of Vahid [VGG94] uses a relaxed cost function to satisfy performance in an inner partitioning loop and to handle hardware minimization in an outer loop. The cost function consists of a very heavily weighted term for performance and a second term for minimizing hardware. The authors present a *binary-constraint search algorithm* which determines the smallest size constraint (by binary search) for which a performance satisfying solution can be found by the partitioning algorithm. The algorithm minimizes hardware, but not execution time.

Kalavade and Lee [KL94] present an algorithm (GCLP) that determines for each node iteratively the mapping to hardware or software. The GCLP algorithm does not use a hardwired objective function, but it selects an appropriate objective according a global time-criticality measure and another measure for local optimality. The results are close to optimal and the runtime grows quadratically to the number of nodes. This approach has been extended to solve the extended partitioning problem [KL95] including the implementation selection problem.

Eles [EPD94] presents a two-stage partitioning approach, where in the first step a VHDL system specification is partitioned into two sets of candidates for hardware and software using profiling and user-interaction. In the second step a process graph is constructed and partitioned into hardware and software parts using a simulated-annealing algorithm [PK93].

Jantsch [JEO+94] presents a partitioning approach where hardware candidates are pre-selected using profiling. All of these selected hardware candidates realize a system speedup of greater than 1. The goal is to speed-up a system by incorporating hardware. A key feature is a memory allocation method which minimizes the interface traffic between hardware and software. The disadvantage of this approach is that hard timing constraints can not be guranteed because the cost model is based on profiling.

3 Hardware/Software Partitioning Approach

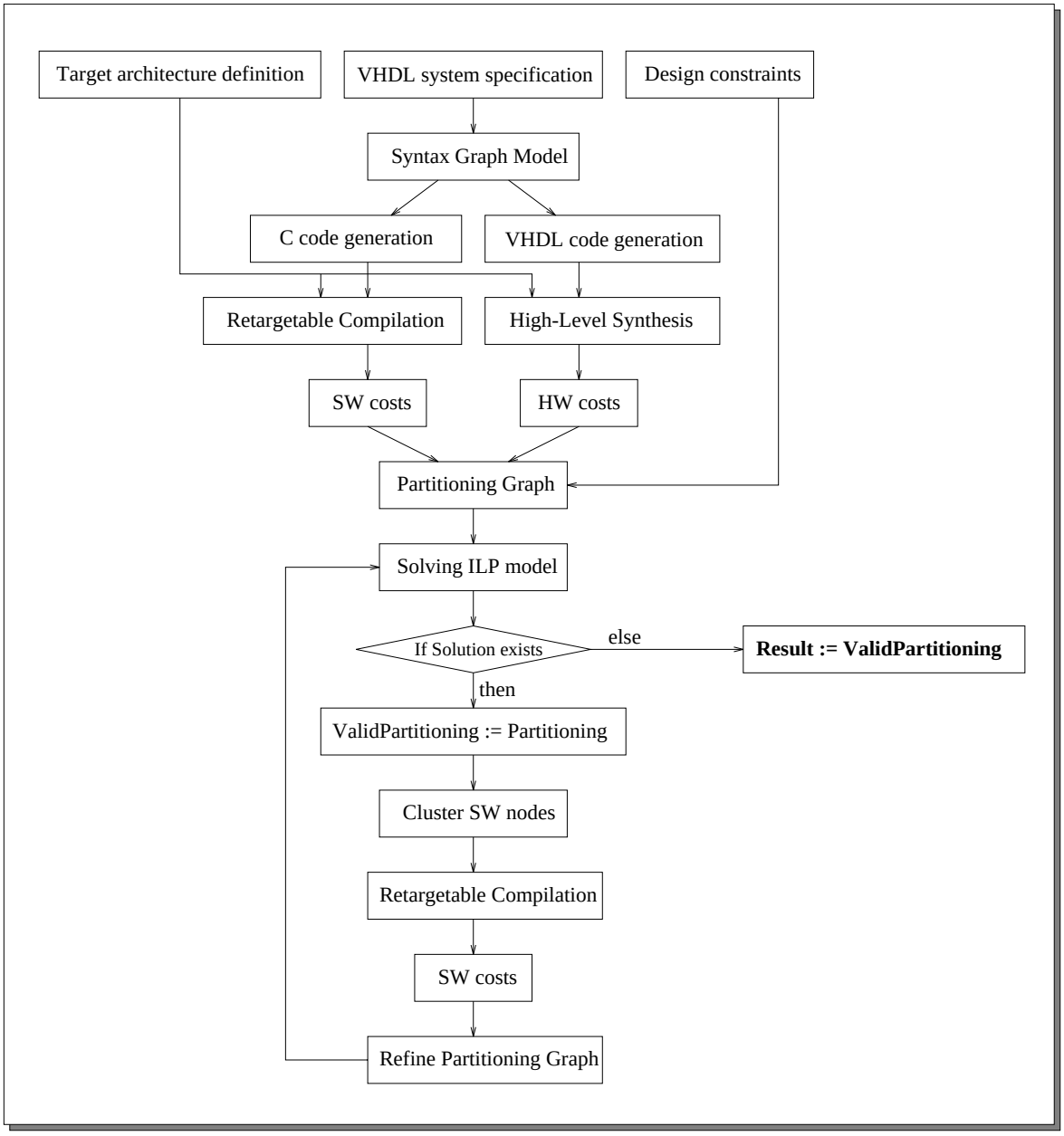

Figure 1: Hardware/Software Partitioning

Our hardware/software partitioning approach is depicted in figure 1. The designer has to define the following:

1. The **target architecture** has to be specified by defining the set of processors for the software parts and the component library to synthesize the hardware parts

of the embedded system.

2. The **system** has to be defined in VHDL as a set of interconnected instances of entities.

3. The **design constraints** have to be determined, containing performance constraints (timing) and resource constraints (area, memory).

Then, the VHDL specification is compiled into an internal syntax graph model. For each entity of this model, software source code (C or DFL) and hardware source code (VHDL) is generated. The software parts are compiled and the hardware parts are synthesized by a high-level synthesis tool (OSCAR [LMD94]). The results are software cost metrics (software execution time, memory usage) and hardware cost metrics (hardware execution time, area) for the entities. The disadvantage of an increased runtime for calculating the cost metrics is compensated by two facts:

- a better quality of the cost metrics compared to the results of estimators,
- better cost metrics lead to fewer partitioning iterations.

After the compilation/synthesis phase a partitioning graph is generated. Nodes represent the entities of the system and edges represent the interconnections between them. The nodes are weighted with the hardware and software costs, the edges are weighted with interface costs which occur if an interface would be realized between the nodes of the edge. The interface costs are approximated by the number and type of data flowing between both nodes. The user-defined design constraints are also matched to the graph. Thus, the partitioning graph includes all information needed for partitioning. The partitioning graph is then transformed into an IP-model, which is the key issue of this paper. The calculated design is optimal for the generated cost model, but nevertheless it is possible to improve the design, because sharing between different instances of same entities is considered, but not sharing effects between different entities. This disadvantage can be solved by an iterative partitioning approach. We use a software oriented approach, because compilation is faster than synthesis and software oriented approaches seem to be superior to hardware oriented approaches (see [VGG94]).

Sets of nodes which have been mapped on the same processor are clustered, and new cost metrics are calculated for them. The partitioning graph is transformed by replacing each cluster by a new node attached with the new cost metric. Then, the redefined graph is repartitioned. This iteration will be repeated until no solution is found. The last valid partitioning represents the resulting design. The clustering technique is illustrated in figure 2.

Example 1:

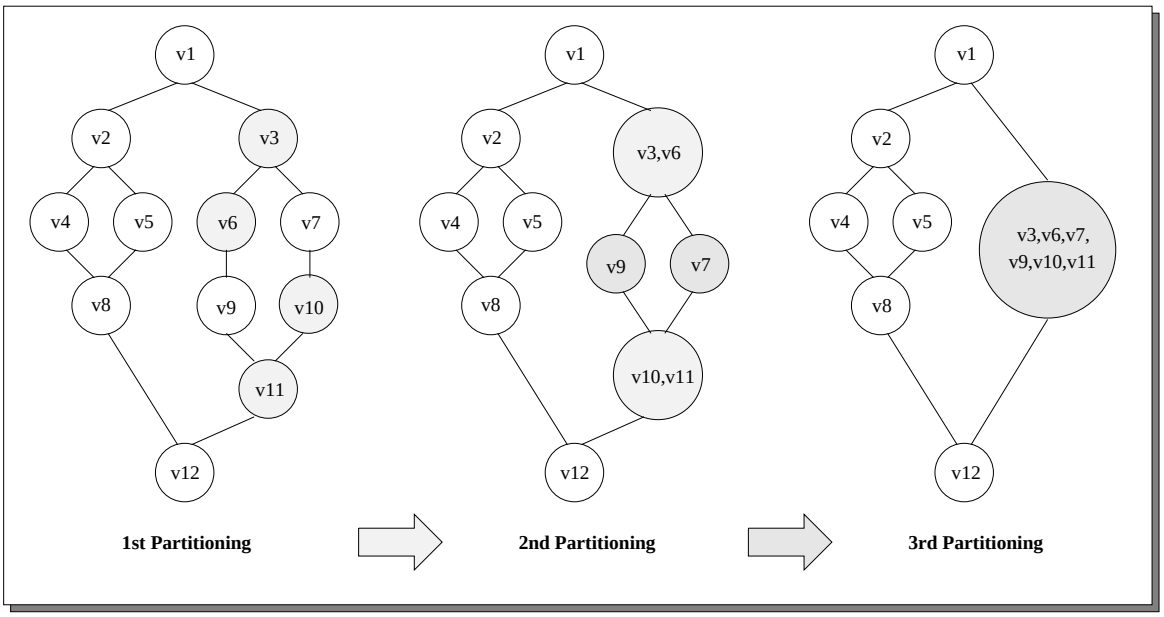

Figure 2: Partitioning refinement

The first partitioning iteration results in 4 software nodes (v_3,v_6,v_{10},v_{11}). The nodes v_3, v_6 and v_{10}, v_{11} are clustered. After the second iteration it is now possible to execute v_6,v_7 on the processor, so the new cluster contains v_3,v_6,v_7,v_9,v_{10},v_{11}. In the third iteration no more nodes can be pushed from hardware to software.

4 Formulation of the Hardware/Software Partitioning Problem

This chapter introduces a formulation of the hardware/software partitioning problem. This formulation is necessary to simplify the description of the problem with the help of an IP-model. We have to define the target architecture and the system which has to be partitioned.

Definition 4.1 *The* **target architecture** *consists of an ASIC h, a set of processors $\mathcal{P} = \{p_1, \ldots, p_{n_P}\}$, external memory and busses between them. The set of target architecture components is defined as:*

$$\mathcal{TA} = \{h\} \cup \mathcal{P} \tag{1}$$

To simplify the notations in the following chapters, let the ASIC be the first element of $\mathcal{TA}$ with index 0, followed by the processors:

$$ta_0 := h; ta_k := p_k, \forall k \in \{1, \ldots, n_P\}$$

A system that has to be realized on the target architecture consists of different instances of entities and interconnections between them. The formal definition looks as follows:

Definition 4.2 *A* **system** *is defined as a tuple*

$$\mathcal{S} = (\mathcal{E}, V, E, I)$$

with the following definitions:

$\mathcal{E} = \{en_1, \ldots, en_{n_\mathcal{E}}\}$	*set of entities,*
$V = \{v_1, \ldots, v_{n_V}\}$	*set of nodes, representing instances of entities,*
$E \subset V \times V$	*set of edges, representing interconnections between instances,*
$I : V \rightarrow \mathcal{E}$	$I(v_j) = en_l$ *defines that* v_j *is an instance of* en_l.

The following cost metrics are defined for each entity en_l: $\underline{c^a(en_l)}$ represents the hardware area, $\underline{c^{th}(en_l)}$ the hardware execution time, $\underline{c^{dm}(en_l)}$ the used software data memory, $\underline{c^{pm}(en_l)}$ the used software program memory and $\underline{c^{ts}(en_l)}$ the software execution time. The costs $\underline{c^a(v_j)}$, $\underline{c^{th}(v_j)}$, $\underline{c^{dm}(v_j)}$, $\underline{c^{pm}(v_j)}$ and $\underline{c^{ts}(v_j)}$ for the instances

v_j of an entity en_l are equal to the costs of en_l. The following interface costs for an edge $e = (v_1, v_2)$ are considered: $\underline{ci^a(e)}$ defines the additional hardware area and $\underline{ci^t(e)}$ defines the communication time for e.

A **design** represents the realization of a system $\mathcal{S}$ on a target architecture $\mathcal{TA}$. The **design quality** can be expressed by the following **design metrics**: $\underline{C^a(S)}$ represents the hardware area of $\mathcal{S}$, $\underline{C^{pm}(S)}$ the used software program memory of $\mathcal{S}$, $\underline{C^{dm}(S)}$ the used software data memory of $\mathcal{S}$ and $\underline{C^t(S)}$ the total execution time of $\mathcal{S}$. The set of **design constraints** $\mathcal{C}$ consists of $\underline{MAX^a(S)}$, $\underline{MAX^{pm}(S)}$, $\underline{MAX^{dm}(S)}$ and $\underline{MAX^t(S)}$ according to the design metrics of $\mathcal{S}$.

Definition 4.3 *The* **hardware/software partitioning problem** *is the problem of finding a mapping* $map : V \rightarrow \mathcal{TA}$ *in such a way that all performance and resource constraints are fulfilled and the design costs are minimized.*

The definitions will be used in the following example:

Example 2:

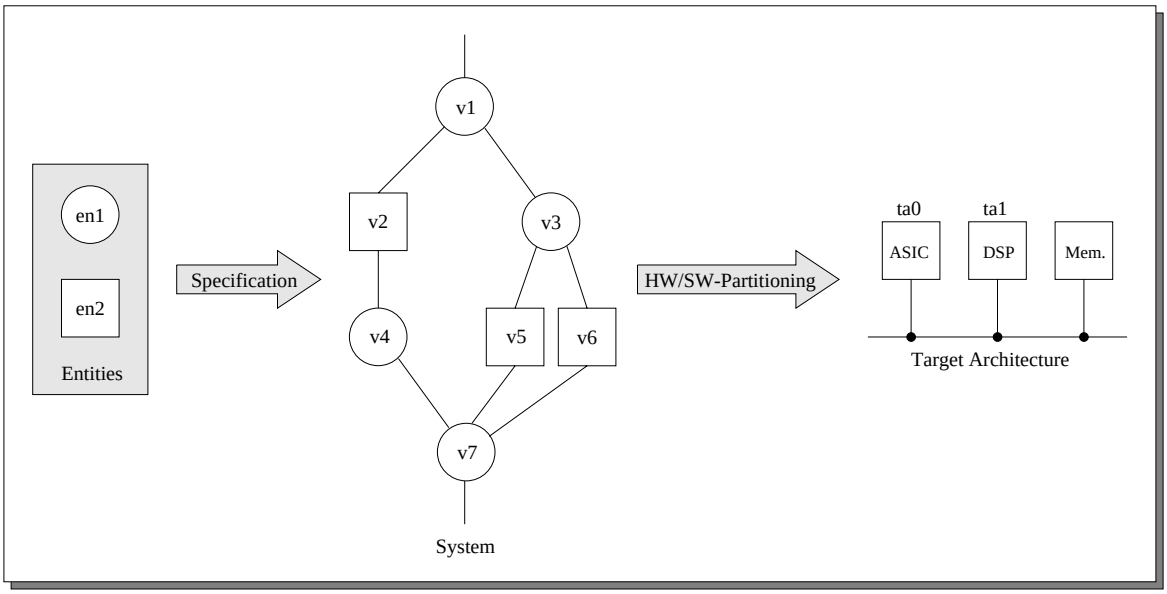

Figure 3: Unpartitioned system

In figure 3 a system is specified consisting of 2 entities en_1 (circle), en_2 (box) and 7 instances $v_1, \ldots, v_7$ of these entities. This system will be partitioned for a target architecture containing one ASIC, one DSP, memory and a bus connecting these components.

5 The IP-Model

Optimization problems can be solved optimally by using integer programming (IP). This paper will show that our IP-model is able to solve the hardware/software partitioning problem with the following characteristics:

- optimal solution for a objective function,
- support of multiprocessor systems,
- timing constraints are guaranteed by scheduling the nodes,
- interface costs are considered,
- instances of the same architecture can be shared on hardware,
- if the user wants to interact the design process, user-defined constraints can easily be adapted to the IP-model.

The following paragraphs describe the IP-model which has been used to perform hardware/software partitioning with these characteristics. To describe the IP-model the following notations are necessary:

Definition 5.1 *Let* $J = \{1, \ldots, n_V\}$ *represent the indices of* $v_j \in V$.
Let $K = \{0, \ldots, n_P\}$ *represent the indices of elements* $ta_k \in \mathcal{TA}$.
Let $L = \{1, \ldots, n_\mathcal{E}\}$ *represent the indices of elements* $en_l \in \mathcal{E}$.
Let $c^x_{l,k}$ *be the cost metric* $c^x(en_l)$ *on target architecture component* ta_k.
Let $c^x_{j,k}$ *be the cost metric* $c^x(v_j)$ *on target architecture component* ta_k.
Let C^x_k *be the design metric* $C^x(S)$ *on* ta_k *of* $\mathcal{S}$.
Let MAX^x_k *be the design constraint* $MAX^x(S)$ *on* ta_k *of* $\mathcal{S}$.
Let T^S_j *be the execution starting time of node* v_j.
Let T^D_j *be the execution time of node* v_j.
Let T^E_j *be the execution ending time of node* v_j.

5.1 The Decision Variables

Our IP-model uses the following 0/1-variables:

Definition 5.2 *Let the following 0/1-variables be defined as:*

$$x_{j,0} = \begin{cases} 1 & : \ v_j \text{ is executed unshared on } ta_0, \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$y_{j,k} = \begin{cases} 1 & : \ v_j \text{ is executed shared on hardware } ta_0, \\ 1 & : \ v_j \text{ is executed on processor } ta_k (k \geq 1), \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$sh_{l,k} = \begin{cases} 1 & : \ en_l \text{ is executed shared on hardware } ta_0, \\ 1 & : \ en_l \text{ is executed on processor } ta_k (k \geq 1), \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$i_{j_1,j_2} = \begin{cases} 1 & : \ \text{an interface is needed between } v_{j_1} \text{ and } v_{j_2}, \\ 0 & : \ \text{otherwise.} \end{cases}$$

$$b_{j_1,j_2,k} = \begin{cases} 1 & : \ v_{j_1} \text{ and } v_{j_2} \text{ are executed on different components,} \\ 1 & : \ v_{j_1} \text{ ends before } v_{j_2} \text{ starts on } ta_k, \\ 0 & : \ \text{otherwise.} \end{cases}$$

Example 3:

The result of hardware/software partitioning of the system depicted in figure 3 is shown in figure 4. Gray shaded nodes are realized in hardware. Shared nodes are enclosed by a dashed line. The following table shows the 0/1-variables for executing nodes shared or not shared on hardware or software.

HW/SW	shared	$v_j \rightarrow$ var	v_1	v_2	v_3	v_4	v_5	v_6	v_7
HW (ta_0)	no	$x_{j,0}$	0	0	1	0	0	0	0
HW (ta_0)	yes	$y_{j,0}$	0	0	0	1	1	1	1
SW (ta_1)		$y_{j,1}$	1	1	0	0	0	0	0

The nodes v_1 and v_2 are executed on the processor ta_1 ($y_{1,1} = y_{2,1} = 1$). Therefore $sh_{\underline{1},1} = sh_{\underline{2},1} = 1$, because v_1 is of entity type $en_{\underline{1}}$ and v_2 is of entity type $en_{\underline{2}}$. v_3

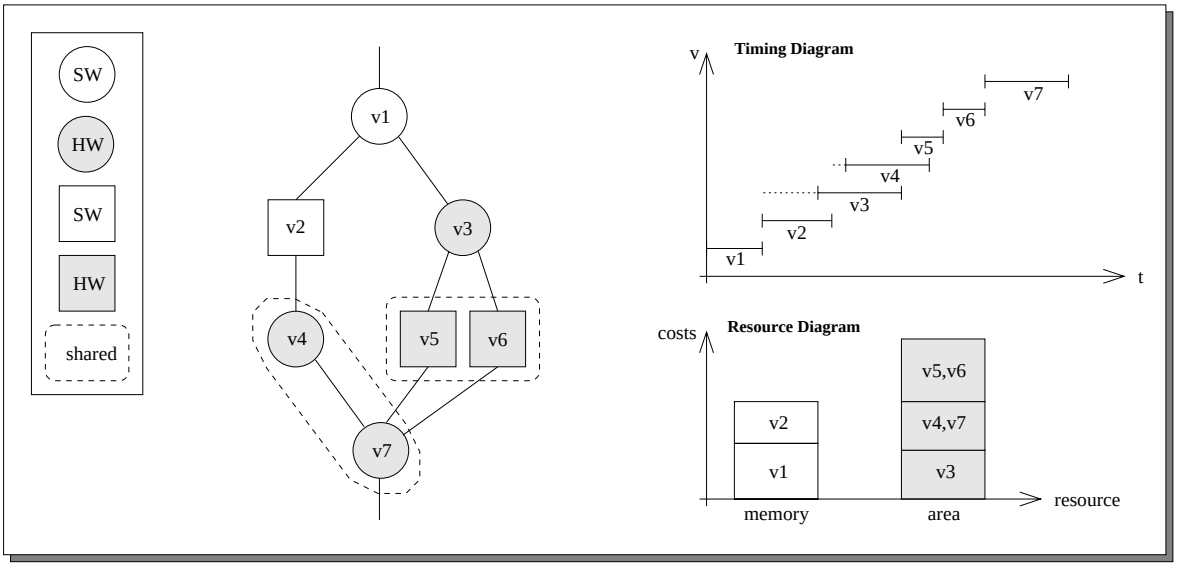

Figure 4: Partitioned system

is executed unshared on the hardware ($x_{3,0} = 1$). The other four nodes $v_4, \ldots, v_7$ are executed shared on the hardware. In total, 2 interfaces are needed: $i_{2,4} = i_{1,3} = 1$. The timing diagram shows that unshared nodes in hardware (v_3) can be executed in parallel to instances of the same entity (v_4). Shared nodes (v_5, v_6) have to be sequentialized, but result in less hardware area as shown in the resource diagram.

5.2 The Constraints

The following constraints have to be fulfilled:

1. **General Constraints**:

   Each node v_j is executed exactly on one target architecture component ta_k.

$$\forall j \in J: \qquad x_{j,0} + \sum_{k \in K} y_{j,k} = 1 \tag{2}$$

2. **Resource Constraints**:

   The values for used data memory C_k^{dm} (eq. 3) and program memory C_k^{pm} (eq. 4) on each processor ta_k may not exceed a given maximum. The used hardware area C_0^a (eq. 5) is the sum of hardware area of unshared instances, shared entities, and the total interface area CI_0^a (eq. 16). C_0^a may not exceed a given maximum.

$$\forall k \in K \backslash \{0\}: \quad C_k^{dm} = \sum_{l \in L} sh_{l,k} * c_{l,k}^{dm} \leq MAX_k^{dm} \tag{3}$$

$$\forall k \in K \backslash \{0\}: \quad C_k^{pm} = \sum_{l \in L} sh_{l,k} * c_{l,k}^{pm} \leq MAX_k^{pm} \tag{4}$$

$$C_0^a = \sum_{j \in J} x_{j,0} * c_{j,0}^a + \sum_{l \in L} sh_{l,0} * c_{l,0}^a + CI_0^a \leq MAX_0^a \tag{5}$$

3. **Timing Constraints**:

The timing costs cannot be calculated by accumulating the execution time of the nodes, because nodes, that are not shared on the ASIC can be executed in parallel. To determine the starting time and ending time for each node, scheduling has to be performed. The execution time T_j^D (eq. 6) of v_j is either the hardware or the software execution time. The ending time T_j^E (eq. 7) is the sum of starting time T_j^S and execution time T_j^D. The starting time T_j^S (eq. 8) of nodes have to be in their ASAP/ALAP-range which can be calculated in a preprocessing step. Data dependencies (eq. 9) have to be considered for all edges $e = (v_{j_1}, v_{j_2})$ including interface communication time T_{j_1,j_2}^I of equation 14. The system execution time C^t (eq. 10) is the maximum of all ending times and may not violate the constraint.

$$\forall j \in J: \quad T_j^D = x_{j,0} * c_{j,0}^{th} + y_{j,0} * c_{j,0}^{th} + \sum_{k \in K \backslash \{0\}} y_{j,k} * c_{j,k}^{ts} \tag{6}$$

$$\forall j \in J: \quad T_j^E = T_j^S + T_j^D \tag{7}$$

$$\forall j \in J: \quad ASAP(v_j) \leq T_j^S \leq ALAP(v_j) \tag{8}$$

$$\forall e = (v_{j_1}, v_{j_2}) \in E: \quad T_{j_2}^S \geq T_{j_1}^E + T_{j_1,j_2}^I \tag{9}$$

$$\forall j \in J: \quad T_j^E \leq C^t \leq MAX^t \tag{10}$$

5.3 Interfacing

An interface has to be realized for an edge $e = (v_{j_1}, v_{j_2})$, if v_{j_1} and v_{j_2} are realized on different target architecture components.

Example 4:

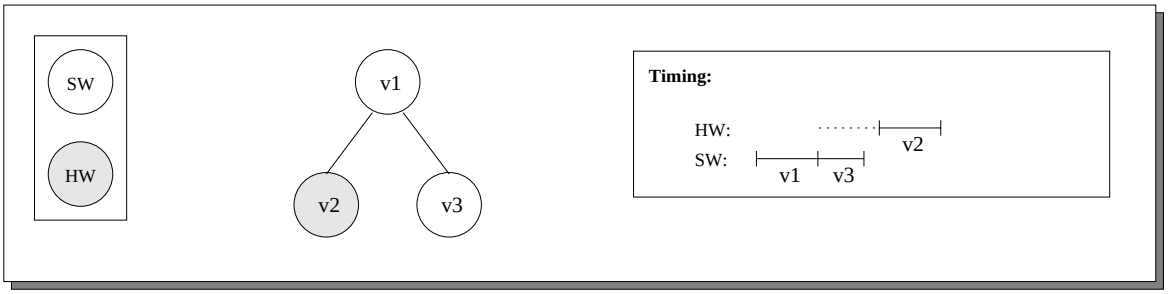

Figure 5: Interface

In figure 5 an interface is needed between v_1 and v_2, because v_1 is realized in software and v_2 in hardware. The interface causes a delay between the ending time of v_1 and the starting time of v_2 which is needed for communication between the processor and the hardware.

With help of the interface 0/1-variable i_{j_1,j_2} the following interface costs can be calculated: interface execution time $T^I_{j_1,j_2}$ (eq. 14), interface hardware area $A^I_{j_1,j_2}$ (eq. 15), and the area of all interfaces CI^a_0 (eq. 16).

$\forall e = (v_{j_1}, v_{j_2}) \in E$:

$$i_{j_1,j_2} \geq x_{j_1,0} + y_{j_1,0} + \sum_{k \in K \setminus \{0\}} y_{j_2,k} - 1, \tag{11}$$

$$i_{j_1,j_2} \geq x_{j_2,0} + y_{j_2,0} + \sum_{k \in K \setminus \{0\}} y_{j_1,k} - 1, \tag{12}$$

$$i_{j_1,j_2} \geq \sum_{k_1 \in K \setminus \{0\}} \sum_{k_2 \in K \setminus \{0\}, k_2 \neq k_1} y_{j_1,k_1} + y_{j_2,k_2} - 1, \tag{13}$$

$$i_{j_1,j_2} \rightarrow minimize$$

$$T^I_{j_1,j_2} = i_{j_1,j_2} * ci^t_{j_1,j_2} \tag{14}$$

$$A^I_{j_1,j_2} = i_{j_1,j_2} * ci^a_{j_1,j_2} \tag{15}$$

$$CI^a_0 = \sum_{e=(v_{j_1},v_{j_2}) \in E} A^I_{j_1,j_2} \tag{16}$$

5.4 Sharing

An entity en_l is shared on hardware ta_0 (eq. 17), if at least two nodes v_{j_1}, v_{j_2} which are instances of entity en_l are executed shared on ta_0. An entity en_l is shared on processor ta_k (eq. 18), if at least one instance of entity en_l is executed on ta_k.

$$\forall l \in L : \forall j_1, j_2 \in J : I(v_{j_1}) = I(v_{j_2}) = en_l : \quad sh_{l,0} \geq y_{j_1,0} + y_{j_2,0} - 1 \tag{17}$$

$$\forall k \in K \backslash \{0\} : \forall l \in L : \forall j \in J : I(v_j) = en_l : \quad sh_{l,k} \geq y_{j,k} \tag{18}$$

5.5 Scheduling

Two nodes v_{j_1}, v_{j_2} which can be executed in parallel have to be sequentialized, if

- v_{j_1} and v_{j_2} are executed on the same processor or
- v_{j_1} and v_{j_2} are shared on the hardware.

To sequentialize two nodes v_{j_1}, v_{j_2} on a target architecture component ta_k, the binary decision variables $b_{j_1,j_2,k}$ and $b_{j_2,j_1,k}$ are used.

Example 5:

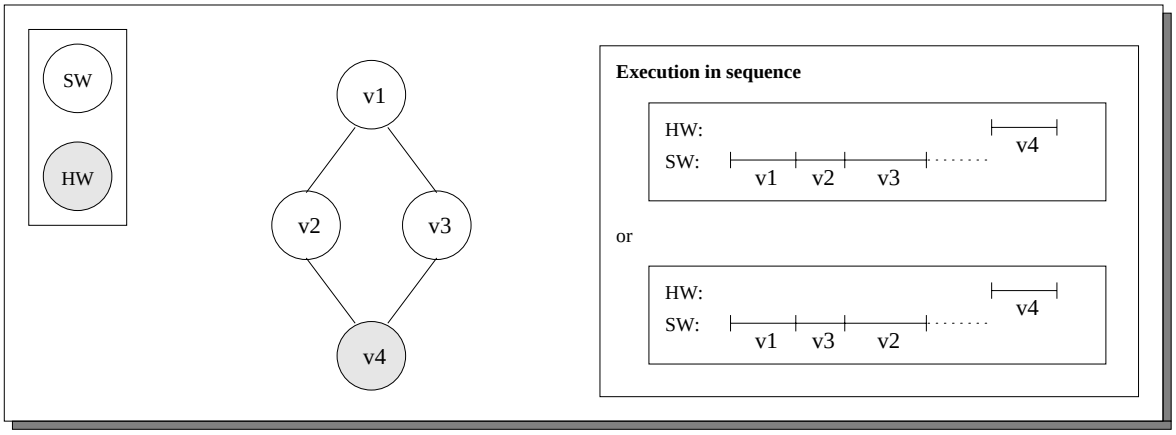

Figure 6: Scheduling

In figure 6 v_2 and v_3 have to be sequentialized, because both are executed on the same processor. Therefore, the starting time of v_2 has to be greater or equal than the ending time of v_3 or vice versa.

If two nodes v_{j_1}, v_{j_2} are sequentialized, then the scheduling variables $b_{j_1,j_2,k}$ and $b_{j_2,j_1,k}$ have to be different, otherwise both are 1. With $b_{j_1,j_2,k}$ (eq. 19-22) nodes can be sequentialized (eq. 23,24).

$$b_{j_1,j_2,k} + y_{j_1,k} \geq 1 \quad (19)$$

$$b_{j_1,j_2,k} + y_{j_2,k} \geq 1 \quad (20)$$

$$b_{j_1,j_2,k} + b_{j_2,j_1,k} \geq 1 \quad (21)$$

$$b_{j_1,j_2,k} + b_{j_2,j_1,k} + y_{j_1,k} + y_{j_2,k} \leq 3 \quad (22)$$

$$\forall k \in K: \quad T^S_{j_1} \geq T^E_{j_2} - \infty * b_{j_1,j_2,k} \quad (23)$$

$$T^S_{j_2} \geq T^E_{j_1} - \infty * b_{j_2,j_1,k} \quad (24)$$

5.6 Heuristic Scheduling

Optimal scheduling of the nodes is a great problem, because the number of the 0/1-variables $b_{j_1,j_2,k}$ can grow quadratically in the number of nodes. An idea to solve this problem is to execute partitioning while iterating the following steps:

1. Solve an IP-model for the hardware/software mapping with help of approximated time values.
2. Solve an IP-model for calculating an exact schedule with nodes mapped to hardware or software.
3. If the resulting total time violates the timing constraint, repeat the first two steps with a timing constraint that is tighter than the approximated total time of step 1. (see figure 7).

Example 6:

The first partitioning results in an approximated execution time which fulfills the given timing contraint. However, the exact execution time violates this constraint. For this reason, a second partitioning with a new timing constraint is executed. This new constraint is tighter than the approximation of the first partitioning. The second partitioning results in a decreased approximated execution time. The exact execution time of the second partitioning fulfills the original timing constraint. Therefore, the second partitioning represents the solution.

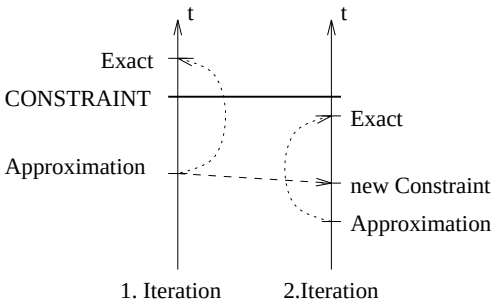

Figure 7: Heuristic scheduling

The following constraints are used additionally to the equations 6-10 to approximate time values:

- The starting time of a node v_j is equal or greater than the accumulated software execution times of all predecessor nodes v_j (eq. 25) and the accumulated hardware execution times of all shared predecessor nodes of v_j (eq. 26).

$$\forall k \in K \backslash \{0\} : \forall j \in J : \quad T_j^S \geq \sum_{i \in J, v_i \in Pred(v_j)} y_{i,k} * c_{i,k}^{ts} \tag{25}$$

$$\forall j \in J : \quad T_j^S \geq \sum_{i \in J, v_i \in Pred(v_j)} y_{i,0} * c_{i,0}^{th} \tag{26}$$

- The starting time of a node v_j is equal or greater than the sum of the ending time of the dominator node v_i of v_j and the software execution times on processor ta_k of all nodes on the paths between v_i and v_j (eq. 27). Equation 28 defines the same constraint for the hardware execution times of all shared nodes on the paths between v_i and v_j.

$$\forall j, j' \in J : v_{j'} \in Dominator(v_j) :$$

$$\forall k \in K \backslash \{0\} : \quad T_j^S \geq T_{j'}^E + \sum_{i \in J, v_i \in Path(v_{j'}, v_j)} y_{i,k} * c_{i,k}^{ts} \tag{27}$$

$$T_j^S \geq T_{j'}^E + \sum_{i \in J, v_i \in Path(v_{j'}, v_j)} y_{i,0} * c_{i,0}^{th} \tag{28}$$

Example 7:

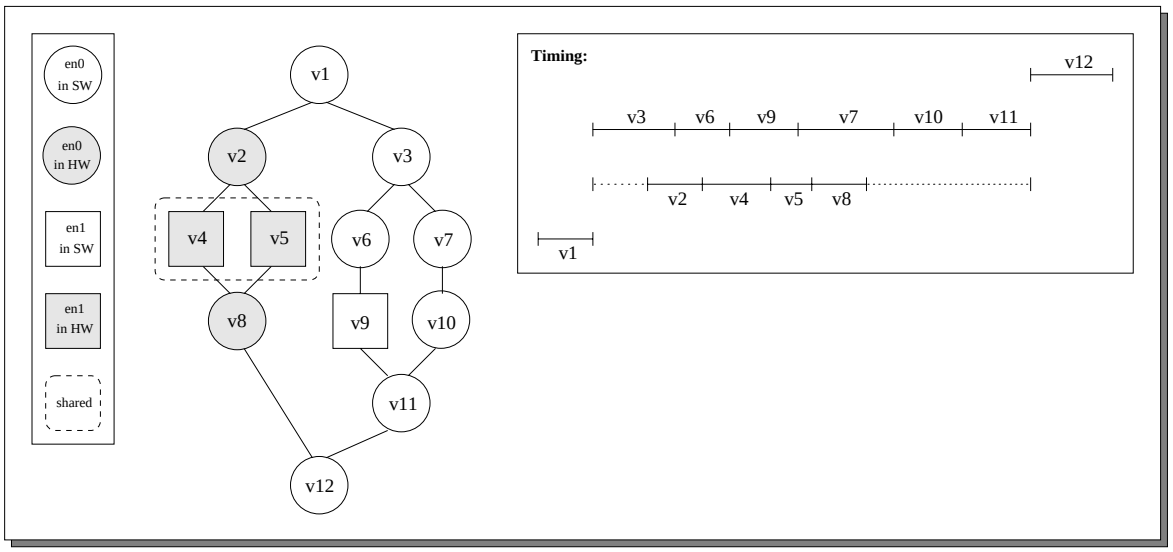

Figure 8: Time approximation used in heuristic scheduling

The partitioning graph in figure 8 contains the following dominator nodes: $Dominator(v_8) = v_2$, $Dominator(v_{11}) = v_3$, $Dominator(v_{12}) = v_1$. For this reason the starting time of v_8 is greater or equal than the sum of the ending time of v_2 and the hardware execution times of v_4 and v_5, because v_4 and v_5 are shared. The starting time of v_{11} is greater or equal than the sum of the ending time of v_3 and the software execution times of v_6, v_7, v_9, v_{10}, because these nodes are realized on the same processor and have to be executed sequentialized.

6 Results

The interesting parameter for partitioning is the number of nodes n which have to be partitioned. For this reason, we have developed some examples containing a lot of instances of small entities. The heuristic partitioning approach can be evaluated by examining the deviation between the exact and the approximated solution and by the different runtimes of solving the IP-models. If the heuristic partitioning approach does not consider interfacing, then the results are always exact, and therefore optimal. If interfacing is considered however, then the approximated system execution time may differ from the exact value. We have calculated the exact and the approximated system execution time, considering interfacing, for 6 different systems. For each system solutions have been calculated for a set of constraints. In figure 9 it is shown that the

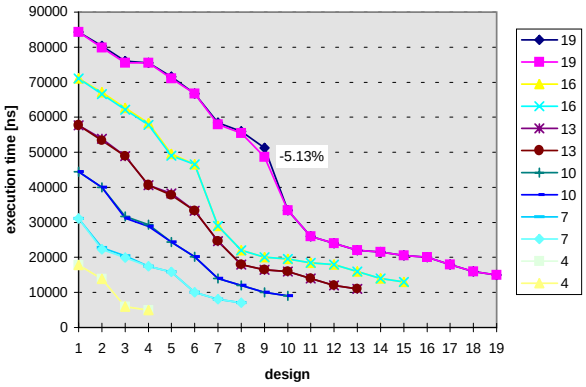

Figure 9: Exact and approximated system execution time

approximated execution time is equal or very close to the exact value. In figure 10 the maximal and the average deviations of both execution times for all 6 systems are depicted. The maximal deviation between the exact and the approximated execution time is 5.13%, the average deviation is smaller than 1% for all examined systems. The runtimes (CPU seconds of a Sun SPARCstation20) to solve these examples considering interfacing and sharing are depicted in figure 11. The maximal runtime of the 6 examined systems has been 29 seconds. Compared to the heuristic approach, the optimal approach (see table 1) has a drastic increased runtime. Thus, the heuristic approach is superior to the optimal approach, because the results are always nearly optimal and the runtimes have been drastically reduced.

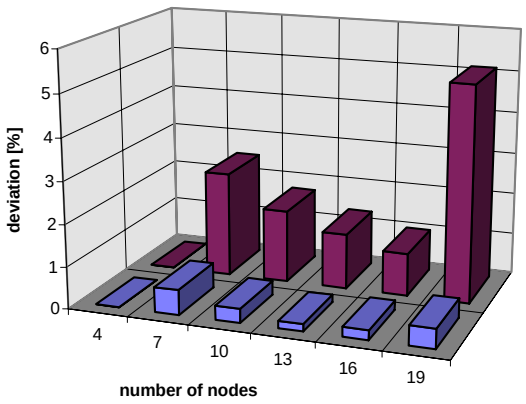

Figure 10: Maximal and average deviations

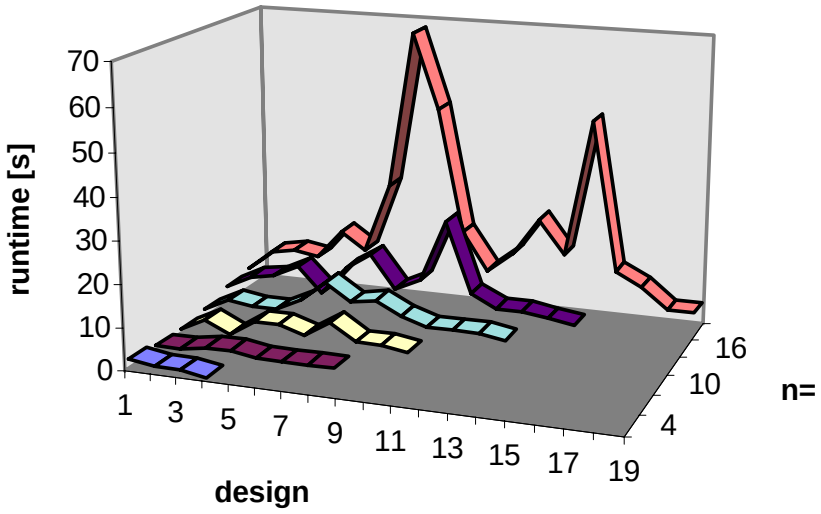

Figure 11: Runtimes of the heuristic approach

n	Method	1	2	3	4	5	6	7	8
4	optimal	2	3	3	1	-	-	-	-
4	heuristic	2	1	1	0	-	-	-	-
7	optimal	9	32	59	246	135	44	12	8
7	heuristic	1	1	2	2	1	1	1	1

Table 1: Runtimes of the exact and the heuristic approach

7 Conclusion

This paper presents a new approach to full-automated hardware/software partitioning supporting multi-processor systems, interfacing and hardware sharing. The partitioning approach itself is based on integer programming leading to optimal results. In contrast to other approaches, where hardware and software costs are estimated, our approach follows the idea of 'using the tools' for cost estimation. The disadvantage of an increased calculation time is compensated by better metrics. The presented partitioning results are very promising, because (nearly) optimal results are calculated in short time. Future work will deal with the iterative partitioning approach to improve the design successively. Design studies of real system level examples will be performed to examine our approach and the idea of 'using the tools' for cost estimation.

References

[EHB93] Rolf Ernst, Jörg Henkel, and Thomas Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test*, Vol.12, pages 64–75, 1993.

[EPD94] Petru Eles, Zebo Peng, and Alexa Doboli. VHDL system-level specification and partitioning in a hardware/software co-synthesis environment. *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 49–55, 1994.

[GCJDM92] Rajesh K. Gupta, Claudionor Nunes Coelho Jr., and Giovanni De Micheli. Synthesis and simulation of digital systems containing interacting hardware and software components. *29th ACM, IEEE Design Automation Conference*, pages 225–230, 1992.

[HE94] D. Henkel J. Herrmann and R. Ernst. An approach to the adaption of estimated cost parameters in the cosyma system. *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 100–107, 1994.

[HEY+95] Jörg Henkel, Rolf Ernst, Wei Ye, Michael Trawny, and Thomas Benner. Cosyma: Ein system zur hardware/software co-synthese. *GME Fachbericht Nr. 15 Mikroelektronik*, pages 167–172, 1995.

[JEO+94] Axel Jantsch, Peeter Ellervee, Johnny Öberg, Ahmed Hemani, and Hannu Tenhunen. Hardware/software partitioning and minimizing memory interface traffic. *European Design Automation Conference (EURO-DAC)*, pages 226–231, 1994.

[KL94] Asawaree Kalavade and Edward A. Lee. A global critically/local phase driven algorithm for the constrained hardware/software partitioning problem. *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 42–48, 1994.

[KL95] Asawaree Kalavade and Edward A. Lee. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, 1995.

[LMD94] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming. *Proceedings of the EURO-DAC*, pages 90–95, 1994.

[PK93] Zebo Peng and Krzysztof Kuchcinski. An algorithm for partitioning of application specific systems. *Proceedings of the European Conference on Design Automation (EDAC)*, pages 316–321, 1993.

[VGG94] Frank Vahid, Jie Gong, and Daniel Gajski. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. *European Design Automation Conference (EURO-DAC)*, pages 214–219, 1994.