, , 1–34 ()

An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming

RALF NIEMANN AND PETER MARWEDEL
DEPT. OF COMPUTER SCIENCE XII, UNIVERSITY OF DORTMUND, D-44221 DORTMUND, GERMANY

niemann@ls12.informatik.uni-dortmund.de,marwedel@ls12.informatik.uni-dortmund.de

Received May 1, 1991

Editor:

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 for a 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 values for the cost metrics is compensated by an improved quality of the values. Therefore, fewer iteration steps for partitioning are needed. The paper presents an algorithm using integer programming for solving the hardware/software partitioning problem leading to promising results.

Keywords: hardware/software codesign, hardware/software partitioning, embedded systems, mixed integer linear programming

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 fulfilling all performance constraints and using a minimum amount of 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. The approach is based on integer programming (IP) to solve the partitioning problem. A formulation of the IP-model will be introduced in detail. The drawback of solving IP-models often is a high computation time. To reduce the computation time, an algorithm using IP has been developed which splits the partitioning approach in two phases. In a first phase, a mapping of nodes to hardware or software is calculated by estimating the schedule times for each node with heuristics. During the second phase a correct schedule is calculated for the resulting HW/SW-mapping of the first phase. It will be shown that this *heuristic scheduling* approach strongly reduces the computation time while the results are nearly optimal for the chosen objective function.

Another new feature of our approach is the cost estimation technique. The cost model is not calculated by estimators like in other approaches, because the quality of estimations is often poor and estimators do not consider compiler effects. In our approach, a compiler and a high-level synthesis tool are used instead of special estimators. The disadvantage of an increased runtime for calculating values for the cost metrics is compensated by a higher precision of these values. A higher precision leads to fewer partitioning iterations.

The outline of the paper is as follows: Section 2 gives an overview of related work in the field of hardware/software partitioning. Our system specification method is introduced in section 3. In section 4 our own approach to partitioning is presented. A formulation of the hardware/software partitioning problem follows in section 5. Section 6 describes the IP-model of the problem. Experimental results of solving these IP-models are presented in section 7 and a conclusion is given in section 8.

2. Related Work

There are many approaches to hardware/software partitioning. One of these is the COSYMA system [3], 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 [8]. The *OLP* is repeated until all performance constraints are fulfilled.

Another hardware/software partitioning approach is realized in the *VULCAN* system [5]. 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 can easily be trapped in a local minimum.

The approach of Vahid [16] 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. The partitioning algorithm minimizes hardware, but not execution time.

Kalavade and Lee [11] 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 optimum. 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 [12] including the implementation selection problem.

Eles [4] 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 [15].

Jantsch [10] presents a partitioning approach where hardware candidates are preselected 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 guaranteed because the cost model is based on profiling.

3. System Specification

One of the key problems in hardware/software codesign is specification of large systems. Many system specification languages have been developed in the last years. One of the most frequently used ones is VHDL, because many CAD tools supporting VHDL exist. In our approach, we also specify systems in VHDL. In [2], [7] the advantages and disadvantages of several system specification languages have been compared and the results for VHDL are promising.

To specify a system that has to be partitioned, the designer has to define the following:

1. The **target technology** has to be specified by defining the set of processors for the software parts and the component library for synthesizing the hardware parts of the embedded system.

2. The **system** has to be defined in VHDL as a set of interconnected instances of components (behavioural VHDL-entities).

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

In our approach the target technology, the system, and the design constraints are specified by using the specification tool *COSYS* [1] which is part of the codesign tool *COOL* [2]. *COSYS* is a graphical VHDL-based interface for hierarchical system specification. In the following, specifying systems with *COSYS* will be illustrated by an MPEG audio system.

First, the designer defines **behavioural entities** using VHDL source code. These behavioural entities, called **components**, are instantiated to form **structural entities**. This is done by connecting **instances** of these components by wires (VHDL signals). Structural entities may also be instantiated. Thus, *COSYS* allows the designer to describe the system **hierarchically**. In figure 1 the specification of a hierarchical system is illustrated.

Example 1:

> The system *mpeg_audio* depicted in figure 1 realizes an MPEG audio encoder and decoder. The upper part of the system represents the encoder that encodes incoming PCM audio samples. The result is an encoded bit-stream of the MPEG audio format. The lower part of the system realizes the decoder that decodes MPEG audio bit-streams into PCM samples. The structural entity *quantizer_coding* is instantiated in this hierarchical specification. *Quantizer_coding* is defined with help of two behavioural entities. It contains an instance of component *quantizer* and another instance of component *coding* which have been specified in VHDL code.

The partitioning approach for these specified systems will be described in the following sections.

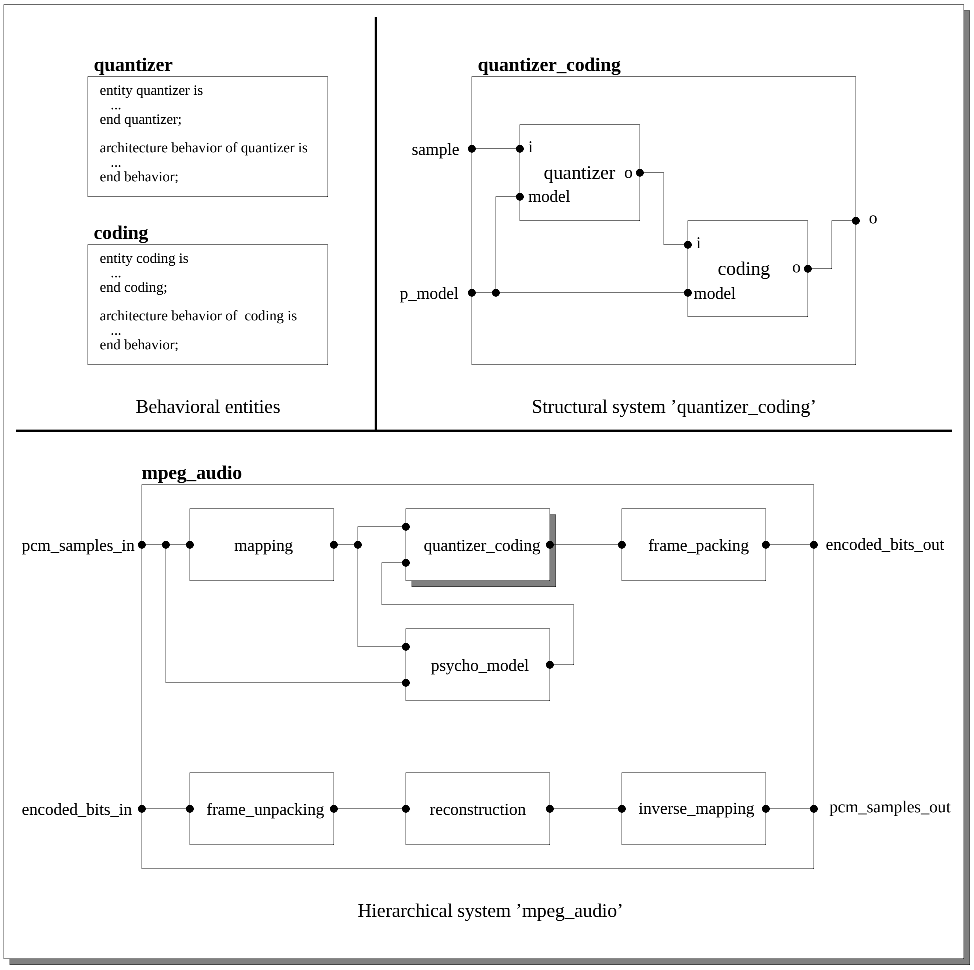

Figure 1. Hierarchical system specification of an MPEG audio system

4. Hardware/Software Partitioning Approach

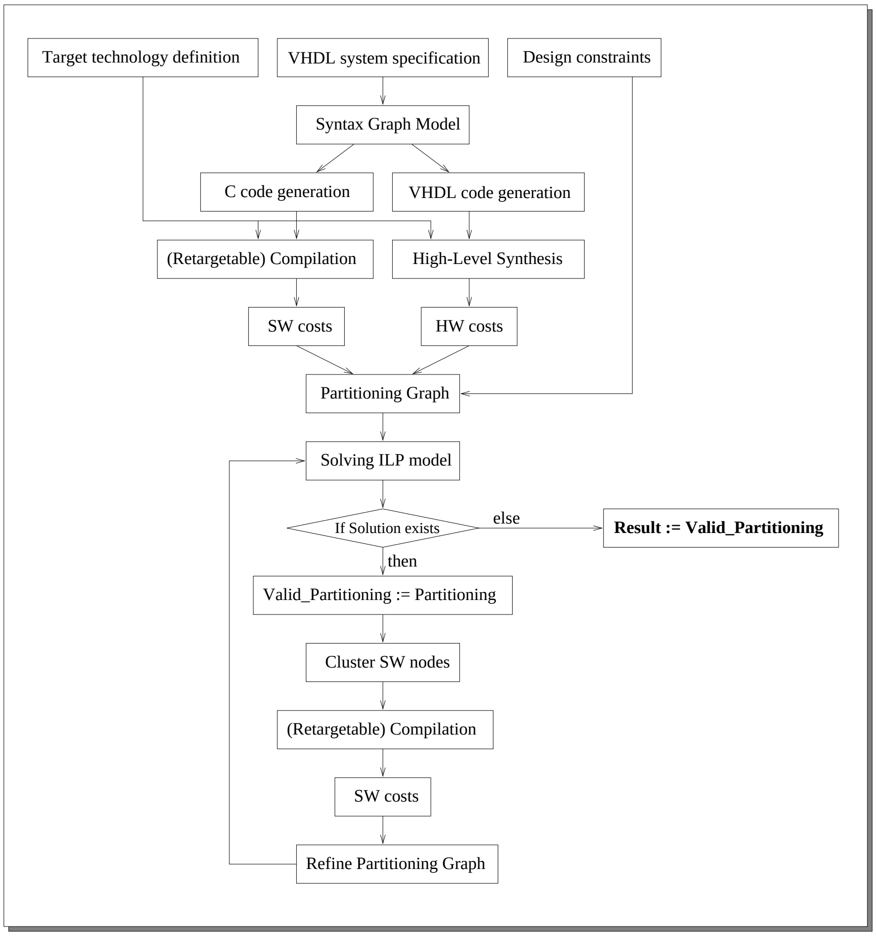

Figure 2. Hardware/Software Partitioning

After the system has been specified with *COSYS*, the VHDL specification is compiled into an internal syntax graph model. For each component (behavioural VHDL-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 [13]). The results are values for software cost metrics (software execution time, memory usage) and values for hardware cost metrics (hardware execution time, area) for the components. The disadvantage of an increased runtime for calculating the cost metrics by running compilers and synthesizers is compensated by a better quality. Moreover, a higher

precision of the cost values leads to fewer partitioning iterations. After the compilation/synthesis phase, a partitioning graph is generated in two steps. First, the hierarchy of the system is flattened. Then, a partitioning graph is created in which each node of the graph represents an instance of a component in the flattened system. Edges of the partitioning graph represent the wires between these instances. In figure 3 the partitioning graph is calculated for a hierarchical system.
Example 2:

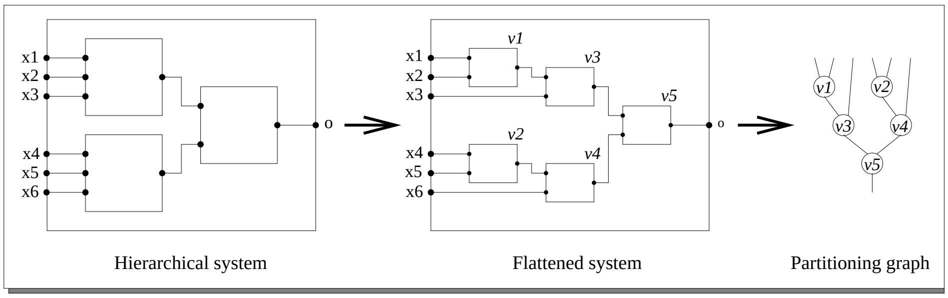

Figure 3. Calculation of the partitioning graph

In the first step, the structural entities are flattened resulting in a set of instances of components. Then for each instance ($v_1 \dots v_5$) a node is added to the partitioning graph. The edges between the nodes represent the wires between the instances.

Nodes are weighted with hardware and software costs, edges are weighted with interface costs which reflect the cost of hardware/software interfaces. Interface costs are approximated by the number and type of data flowing between both nodes. User-defined design constraints are also attached 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. Afterwards, the model is solved by an IP-solver. The calculated design is optimal for the chosen objective function using the generated cost model, but nevertheless it is possible to improve the design, because although sharing effects between different instances of the same components is considered, sharing effects between different components is not. This limitation can be removed 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 [16]).

Sets of nodes which have been mapped on the same processor are clustered. For each cluster, a new cost metric is calculated by compiling all nodes of the cluster together. Then, the partitioning graph is transformed by replacing each cluster by a new node with the new cost metric attached. Finally, 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 4.

Example 3:

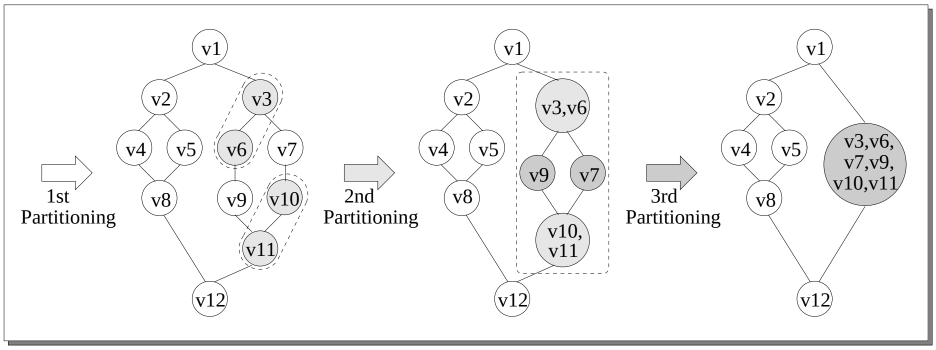

Figure 4. Partitioning refinement

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_7,v_9 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 moved from hardware to software.

5. Formulation of the HW/SW Partitioning Problem

This section 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 system which has to be partitioned and the target technology used to implement the system.

5.1. Target Technology and System Specification

Definition 1 *A* **target technology** $\mathcal{T}$ *is defined as a tuple*

$$\mathcal{T} = (\mathcal{V}, \mathcal{E}), \qquad \mathcal{V} = \mathcal{H} \cup \mathcal{P} \cup \mathcal{M}, \qquad \mathcal{E} \subseteq \mathcal{PS}(\mathcal{V}) \setminus \big\{\{v\} \mid v \in \mathcal{V}\big\}$$

containing all target technology components and interconnections. The target technology components $\mathcal{V}$ are defined as a set of hardware components (ASICs) $\mathcal{H} = \{h_1, \ldots, h_{n_\mathcal{H}}\}$, processors $\mathcal{P} = \{p_1, \ldots, p_{n_\mathcal{P}}\}$ and memories $\mathcal{M} = \{m_1, \ldots, m_{n_\mathcal{M}}\}$. The target technology interconnections $\mathcal{E}$ are defined as a set of busses $\mathcal{E} = \{e_1, \ldots, e_{n_\mathcal{E}}\}$ connecting these components (at least 2) where $\mathcal{PS}(\mathcal{V})$ represents the power set of $\mathcal{V}$.

Example 4:

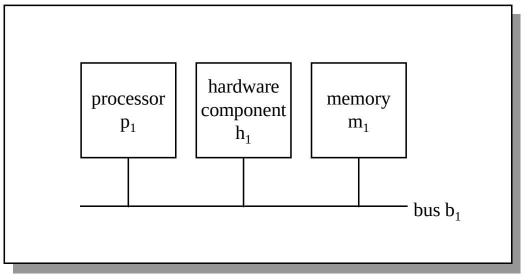

Figure 5. Target technology

In figure 5 an example for a target technology is given. It contains a processor p_1, a hardware component h_1, external memory m_1 and a bus b_1 connecting p_1, h_1 and m_1.

A system that has to be mapped to the target technology consists of several instances of different system components and interconnections between them. The formal definition is as follows:

Definition 2 *A* **system** $\mathcal{S}$ *is defined as a 4-tuple*

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

with the following definitions:

$C = \{c_1, \ldots, c_{n_C}\}$	*set of system components,*
$V = \{v_1, \ldots, v_{n_V}\}$	*set of nodes, representing instances of system components,*
$E \subseteq V \times V$	*set of edges, representing interconnections between nodes,*
$I : V \to C$	$I(v_i) = c_l$ *defines that* v_i *is an instance of component* c_l*.*

Example 5:

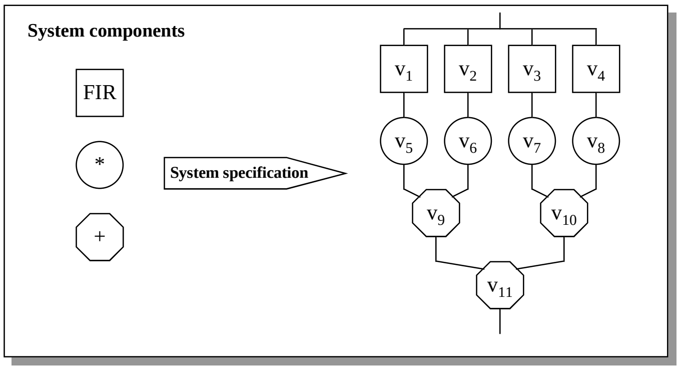

Figure 6. System specification

In figure 6 a 4-band-equalizer is specified. It consists of 3 system components: an FIR-filter, a multiplier and an adder. The equalizer is specified by using 4 instances of the FIR-filter ($v_1 \ldots v_4$), 4 instances of the multiplier ($v_5 \ldots v_8$) and 3 instances of the adder ($v_9 \ldots v_{11}$). This 4-band-equalizer is a well suited example to demonstrate the scheduling, hardware sharing and interfacing problem. Therefore, it will be used in the rest of the paper as a demonstrator example.

5.2. Hardware and Software Implementation

Parts of the system may be implemented in hardware or in software. The main difference between implementing system instances v_{i_1}, v_{i_2} on a processor or on a hardware component is that v_{i_1}, v_{i_2} can not be executed in parallel on a processor. On the software side, a system component c_l is implemented as a function on a processor. Each system instance v_i of c_l which is mapped to the processor uses a corresponding function call for this function. On the hardware side however, two instances v_{i_1} and v_{i_2} of c_l may be executed in parallel on a hardware component. Therefore, it is possible that v_{i_1} and v_{i_2} are mapped to different hardware instances of c_l. The following definition will define the different implementation possibilities.

Definition 3
Let $\mathcal{S} = (C, V, E, I)$ be a system.
Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be a target technology. $\mathcal{V} = \mathcal{H} \cup \mathcal{P} \cup \mathcal{M}$
Let $p_k \in \mathcal{V}$ be a processor and $h_k \in \mathcal{V}$ a hardware component.
The sets of possible **hardware implementations** $Impl^{hw}(c_l, h_k)$ *and* **software implementations** $Impl^{sw}(c_l, p_k)$ *for a system component c_l are defined as:*

$$\begin{aligned} Impl^{hw}(c_l, h_k) &= \{h_{l,1,k} \ldots h_{l,N,k} | \ N = | \{v_i \mid I(v_i) = c_l\} \mid \} \\ Impl^{sw}(c_l, p_k) &= \{p_{l,k}\} \end{aligned}$$

The sets of possible hardware implementations $Impl^{hw}(S, \mathcal{T})$ and software implementations $Impl^{sw}(S, \mathcal{T})$ for $\mathcal{S}$ on $\mathcal{T}$ are then defined as:

$$\begin{aligned} Impl^{hw}(\mathcal{S}, \mathcal{T}) &= \bigcup_{c_l \in C, h_k \in \mathcal{H}} Impl^{hw}(c_l, h_k) \\ Impl^{sw}(\mathcal{S}, \mathcal{T}) &= \bigcup_{c_l \in C, p_k \in \mathcal{P}} Impl^{sw}(c_l, p_k) \end{aligned}$$

Example 6:

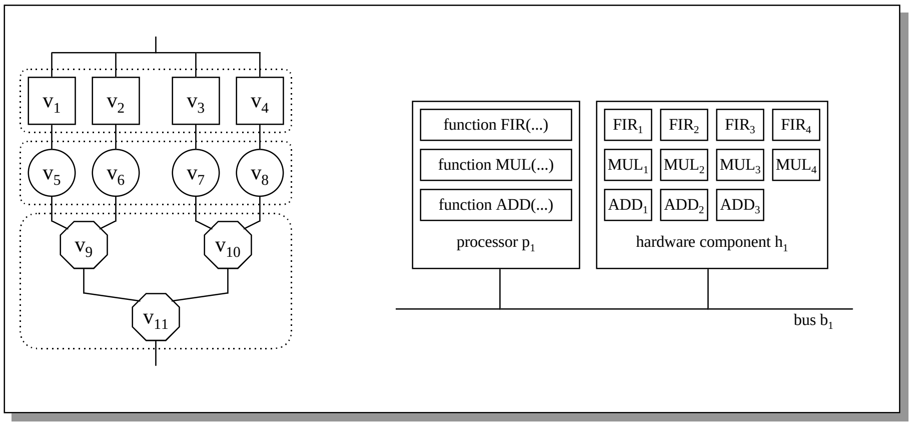

Figure 7. Hardware/software implementations

The possible hardware/software implementations for the 4-band-equalizer are depicted in figure 7. Three functions may be implemented in software, one for an FIR-filter (FIR), one for multiplying (MUL) and one function for adding (ADD). On the hardware side, 4 hardware instances of an FIR-filter may be needed. In such a case, the highest speed can be reached, because all 4 hardware instances are able to work in parallel. Finally, the hardware may contain a maximum of 4 hardware multipliers and 3 hardware adders.

5.3. Cost Model

Hardware/software partitioning algorithms need **cost metrics** for the nodes and the edges of the system to evaluate different partitionings. The values for these cost metrics are calculated for the possible hardware and software implementations of system components. This is done during the compilation and synthesis phase, described in section 4. In our approach, we partition systems based on the following cost metrics:

Definition 4
Let $\mathcal{S} = (C, V, E, I)$ be a system and $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ a target technology.
Let $c \in C$ be a system component and $e \in E$ an edge of system $\mathcal{S}$.
Let $p \in \mathcal{V}$ be a processor, $h \in \mathcal{V}$ a hardware component and $b \in \mathcal{E}$ a bus of $\mathcal{T}$.
The **cost metrics** *are defined as follows:*

$c^{dm}(c,p)$	*represents the software data memory required by c on p,*
$c^{pm}(c,p)$	*the software program memory required by c on p,*
$c^{ts}(c,p)$	*the software execution time required by c on p,*
$c^{a}(c,h)$	*the hardware area required by c on h,*
$c^{th}(c,h)$	*the hardware execution time required by c on h,*
$ci^{a}(e,b)$	*the additional interface hardware area required by e on b and*
$ci^{t}(e,b)$	*the additional interface communication time required by e on b.*

These costs are also defined for instances v of these system components c. They are denoted by $c^{dm}(v,p)$, $c^{pm}(v,p)$, $c^{ts}(v,p)$, $c^{a}(v,h)$ and $c^{th}(v,h)$ for a system component c. The costs for different instances of the same system component are obviously equal:

Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be a target technology.
Let $p \in \mathcal{V}$ be a processor and $h \in \mathcal{V}$ a hardware component of $\mathcal{T}$.

$$\forall p \in \mathcal{V} : I(v_{i_1}) = I(v_{i_2}) \Rightarrow c^{ts}(v_{i_1},p) = c^{ts}(v_{i_2},p) \wedge c^{dm}(v_{i_1},p) = c^{dm}(v_{i_2},p) \wedge c^{pm}(v_{i_1},p) = c^{pm}(v_{i_2},p) \quad (1)$$

$$\forall h \in \mathcal{V} : I(v_{i_1}) = I(v_{i_2}) \Rightarrow c^{th}(v_{i_1},h) = c^{th}(v_{i_2},h) \wedge c^{a}(v_{i_1},h) = c^{a}(v_{i_2},h) \quad (2)$$

According to the cost metrics definition for system components, we can define the resource costs for each target technology component. The resource costs of a target technology component t_k represent the sum of cost metrics used by the nodes mapped to t_k.

Definition 5
Let $\mathcal{S} = (C, V, E, I)$ be a system and $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ a target technology.
Let $p \in \mathcal{V}$ be a processor, $h \in \mathcal{V}$ a hardware component and $b \in \mathcal{E}$ a bus of $\mathcal{T}$.
The **resource costs** *required for implementing $\mathcal{S}$ on target technology components*

are defined as follows:

$C^{dm}(p)$	*represents the software data memory required on p,*
$C^{pm}(p)$	*the software program memory required on p,*
$C^{a}(h)$	*the hardware area required on h and*
$CI^{a}(b)$	*the additional interface hardware area required for b.*

For each of these resource costs maximum values can be defined. These values are called **resource constraints** for the target technology, e.g. the maximal number of CLBs of an FPGA or the amount of internal memory of a processor. They are denoted by $MAX^{dm}(p)$, $MAX^{pm}(p)$ and $MAX^{a}(h)$.

A **design** represents the realization of a system $\mathcal{S}$ on a target technology $\mathcal{T}$. The **design quality** can be expressed by evaluating the resource costs. The results are the following **design costs**:

Definition 6 *The* **design costs** *of a system $\mathcal{S}$ are defined as follows:*

$C^{dm}(S)$	*represents the used software data memory,*
$C^{pm}(S)$	*the used software program memory,*
$C^{a}(S)$	*the hardware area and*
$C^{t}(S)$	*the total execution time.*

The design costs may also be constrained, e.g. the total execution time of a system has to fulfill a timing constraint to guarantee real-time conditions. These **design constraints** are denoted by $MAX^{dm}(S)$, $MAX^{pm}(S)$, $MAX^{a}(S)$ and $MAX^{t}(S)$. With help of this complex cost model, the hardware/software partitioning problem can be defined as follows.

5.4. Hardware/Software Partitioning

The task of the hardware/software partitioning problem is to map nodes to target technology components and edges (if communication is necessary) to busses of the target technology (see figure 8). The goal of partitioning algorithms is to minimize the design costs, while meeting all requirements. The design costs are calculated with help of a given cost model.

Definition 7
Let $\mathcal{S} = (C, V, E, I)$ be a system.
Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be a target technology. $\mathcal{V} = \mathcal{H} \cup \mathcal{P} \cup \mathcal{M}$
Let $Impl^{hw}(\mathcal{S}, \mathcal{T})$ be the set of possible hardware implementations of $\mathcal{S}$ on $\mathcal{T}$.
Let $Impl^{sw}(\mathcal{S}, \mathcal{T})$ be the set of possible software implementations of $\mathcal{S}$ on $\mathcal{T}$.
The **hardware/software partitioning problem** *is defined as the problem of finding a mapping from $\mathcal{S}$ to $\mathcal{T}$ given by two mapping functions:*

$$\begin{aligned} mv: &\quad V \rightarrow Impl\_V \subseteq Impl^{hw}(\mathcal{S}, \mathcal{T}) \cup Impl^{sw}(\mathcal{S}, \mathcal{T}) \\ me: &\quad E \rightarrow Impl\_E \subseteq \mathcal{E} \end{aligned}$$

such that

$$mv(v_i) = \begin{cases} p_{l,k} \in Impl^{sw}(c_l, p_k) & , \textit{ if } v_i \textit{ is implemented by function} \\ & \quad p_{l,k} \textit{ on } p_k \textit{ calculating } c_l = I(v_i) \\ h_{l,j,k} \in Impl^{hw}(c_l, h_k) & , \textit{ if } v_i \textit{ is implemented by the } j\textit{-th} \\ & \quad \textit{instance } h_{l,j,k} \textit{ of } c_l = I(v_i) \textit{ on } h_k. \end{cases}$$

$$me(e_i) = \begin{cases} b_k \in \mathcal{E} & , \textit{ if } e_i \textit{ is needed to realize an interface on bus } b_k \\ \emptyset & , \textit{ if no interface is needed for } e_i \end{cases}$$

and design costs are minimized and resource and design constraints are met.

The following example illustrates this complex definition of the problem.
Example 7:

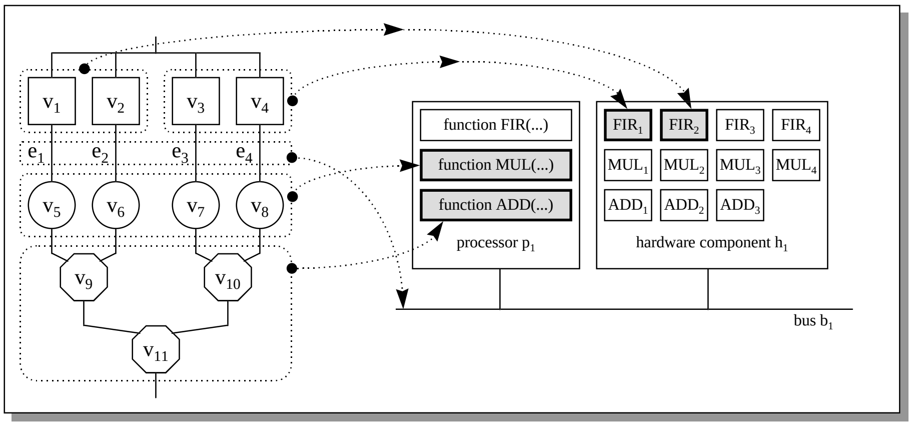

Figure 8. Hardware/software partitioning

The hardware/software partitioning problem is illustrated in figure 8. Two FIR-filters (v_1, v_2) are mapped to a first hardware instance (FIR_1) on h_1. Two other FIR-filters (v_3, v_4) are mapped to a second hardware instance (FIR_2). The multipliers $(v_5, \ldots, v_8)$ are implemented as a function MUL on processor p_1. The function ADD on p_1 implements the adders $(v_9, \ldots, v_{11})$. The results of the FIR-filters are calculated by h_1 and have to be transported to p_1. Therefore the edges $e_1, \ldots, e_4$ realize the interfaces on bus b_1. In summary, two instances of an FIR-filter are implemented by hardware instances on h_1, and two functions (MUL, ADD) are implemented in software on p_1.

6. The IP-Model

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

- optimal solution for an objective function,
- support for multiprocessor and multi-ASIC target technologies,
- timing constraints are guaranteed by scheduling the nodes,
- bus conflicts are prevented by scheduling communication events on edges,
- interface costs are considered,
- instances of the same system component can share their implementation on hardware,
- interactive support for user-defined constraints.

The following paragraphs describe the IP-model for performing hardware/software partitioning with these characteristics. To simplify the description of the IP-model, the following notations are used:

Definition 8 *Sets of nodes and edges*
Let $\mathcal{S} = (C, V, E, I)$ be a system.

$$\begin{array}{lcll}
pred_nodes(v \in V) & = & \{w & \mid \exists p : p = (w, \ldots, v)\} \\
succ_nodes(v \in V) & = & \{w & \mid \exists p : p = (v, \ldots, w)\} \\
pred_edges(v \in V) & = & \{e & \mid e = (x, y) \cup y \in pred_nodes(v)\} \\
succ_edges(v \in V) & = & \{e & \mid e = (x, y) \cup x \in succ_nodes(v)\} \\
pred_edges(e \in E) & = & \{f & \mid e = (v, w) \cup f \in pred_edges(v)\} \\
succ_edges(e \in E) & = & \{f & \mid e = (v, w) \cup f \in succ_edges(w)\} \\
instances_of(c \in C) & = & \{v & \mid I(v) = c\} \\
share_nodes(v \in V) & = & \{w & \mid w \neq v \wedge I(v) = I(w)\} \\
schedule_nodes(v \in V) & = & \{w & \mid w \neq v \wedge w \notin \{pred_nodes(v) \cup succ_nodes(v)\}\} \\
schedule_edges(e \in E) & = & \{f & \mid f \neq e \wedge f \notin \{pred_edges(e) \cup succ_edges(e)\}\} \\
path_nodes(v_1, v_2 \in V) & = & \{w & \mid w \in succ_nodes(v_1) \wedge w \in pred_nodes(v_2)\} \\
path_edges(v_1, v_2 \in V) & = & \{e & \mid e \in succ_edges(v_1) \wedge e \in pred_edges(v_2)\} \\
dominator_nodes(v \in V) & = & \{w & \mid \forall p : p = (s, \ldots, v) \wedge pred_nodes(s) = \emptyset : w \in p\}
\end{array}$$

Furthermore, the following *indices* and *variables* are used:

Definition 9 *Indices*

$L = \{1, \ldots, n_{\mathcal{C}}\}$	*indices for system components $c_l \in \mathcal{C}$,*
$I = \{1, \ldots, n_V\}$	*indices for nodes $v_i \in V$,*
$J \in N_0^+$	*indices for hardware instances of system components,*
$KH = \{1, \ldots, n_{\mathcal{H}}\}$	*indices for hardware components $h_k \in \mathcal{H}$,*
$KP = \{1, \ldots, n_{\mathcal{P}}\}$	*indices for processors $p_k \in \mathcal{P}$,*
$KB = \{1, \ldots, n_{\mathcal{E}}\}$	*indices for busses $b_k \in \mathcal{E}$.*

Definition 10 *Variables for costs and constraints*
Let $\mathcal{S} = (C, V, E, I)$ be a system and $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ a target technology.
Let $c_l \in C$ be a system component, $v_i \in V$ a node and $e \in E$ an edge of system $\mathcal{S}$.
Let $p_k \in \mathcal{V}$ be a processor, $h_k \in \mathcal{V}$ a hardware component and $b_k \in \mathcal{E}$ a bus of $\mathcal{T}$.

$c^{ts}_{l,k}$, $c^{dm}_{l,k}$, $c^{pm}_{l,k}$	*cost metrics* $c^{ts}(c_l, p_k)$, $c^{dm}(c_l, p_k)$, $c^{pm}(c_l, p_k)$,
$c^{th}_{l,k}$, $c^{a}_{l,k}$	*cost metrics* $c^{th}(c_l, h_k)$, $c^{a}(c_l, h_k)$,
$c^{ts}_{i,k}$, $c^{dm}_{i,k}$, $c^{pm}_{i,k}$	*cost metrics* $c^{ts}(v_i, p_k)$, $c^{dm}(v_i, p_k)$, $c^{pm}(v_i, p_k)$,
$c^{th}_{i,k}$, $c^{a}_{i,k}$	*cost metrics* $c^{th}(v_i, h_k)$, $c^{a}(v_i, h_k)$,
$ci^{t}_{i_1,i_2,k}$, $ci^{a}_{i_1,i_2,k}$	*cost metrics* $ci^{t}(e, b_k)$, $ci^{a}(e, b_k)$ *for* $e = (v_{i_1}, v_{i_2})$,
C^{dm}_k, C^{pm}_k	*resource costs* $C^{dm}(p_k)$, $C^{pm}(p_k)$,
C^{a}_k	*resource costs* $C^{a}(h_k)$,
CI^{a}_k	*resource costs* $CI^{a}(b_k)$,
MAX^{dm}_k, MAX^{pm}_k	*resource constraints* $MAX^{dm}(p_k)$, $MAX^{pm}(p_k)$,
MAX^{a}_k	*resource constraints* $MAX^{a}(h_k)$,
C^t, C^{dm}, C^{pm}, C^a	*design metrics* $C^t(S)$, $C^{dm}(S)$, $C^{pm}(S)$, $C^a(S)$,
MAX^t, MAX^a	*design constraints* $MAX^t(S), MAX^a(S)$
MAX^{dm}, MAX^{pm}	*design constraints* $MAX^{dm}(S), MAX^{pm}(S)$
T^S_i	*starting time of node* v_i,
T^D_i	*execution time of node* v_i,
T^E_i	*ending time of node* v_i,
$TI^S_{i_1,i_2}$	*starting time of edge* $e = (v_{i_1}, v_{i_2})$,
$TI^D_{i_1,i_2}$	*execution time of edge* $e = (v_{i_1}, v_{i_2})$,
$TI^E_{i_1,i_2}$	*ending time of edge* $e = (v_{i_1}, v_{i_2})$.

6.1. The Decision Variables

The IP-model needs decision variables for defining mapping, scheduling, sharing and interfacing constraints. Thus, the solution of the IP-model is driven by the following variables:

Definition 11

$$x_{i,j,k} = \begin{cases} 1 : & v_i \textit{ is mapped to the } j\textit{-th hardware instance of } c = I(v_i) \\ & \textit{on hardware component } h_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$X_{i,k} = \begin{cases} 1 : & v_i \textit{ is mapped to hardware component } h_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$Y_{i,k} = \begin{cases} 1 : & v_i \textit{ is mapped to processor } p_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$Z_{i_1,i_2} = \begin{cases} 1 : & \textit{an interface is needed between } v_{i_1} \textit{ and } v_{i_2}, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$z_{i_1,i_2,k} = \begin{cases} 1 : & \textit{communication between } v_{i_1}, v_{i_2} \textit{ is realized on bus } b_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$nx_{l,j,k} = \begin{cases} 1 : & \textit{at least 1 instance of } c_l \textit{ is mapped to the } j\textit{-th} \\ & \textit{hardware instance of } c_l \textit{ on } h_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$NY_{l,k} = \begin{cases} 1 : & \textit{at least 1 instance of } c_l \textit{ is mapped to processor } p_k, \\ 0 : & \textit{otherwise.} \end{cases}$$

$$NX_{l,k} \;:\; \textit{number of hardware instances of } c_l \textit{ realized on } h_k.$$

$$b_{i_1,i_2} = \begin{cases} 1 : & v_{i_1} \textit{ ends before } v_{i_2} \textit{ starts,} \\ 0 : & \textit{otherwise.} \end{cases}$$

$$bi_{i_1,i_2} = \begin{cases} 1 : & \textit{communication time for } e_{i_1} \textit{ ends before } e_{i_2} \textit{ starts,} \\ 0 : & \textit{otherwise.} \end{cases}$$

Example 8:

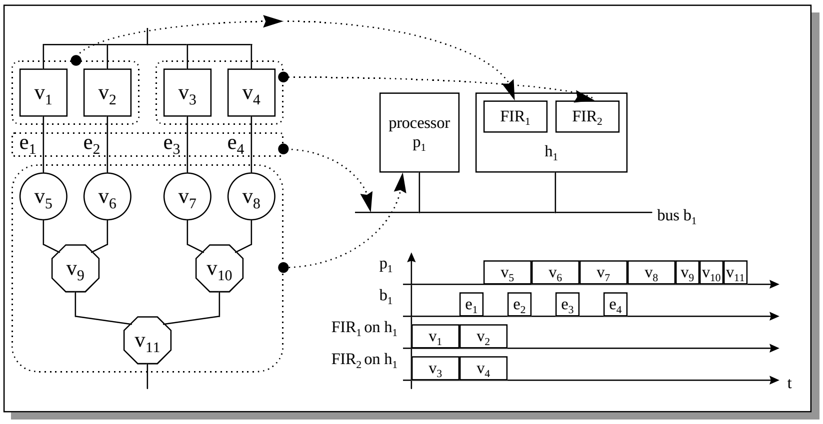

Figure 9. Hardware sharing

To visualize usage of these variables, figure 9 shows a partitioning of the 4-band-equalizer. $v_1 \ldots v_4$ are mapped to hardware component h_1 ($X_{1,1} = \ldots = X_{4,1} = 1$). All other nodes $v_5 \ldots v_{11}$ are mapped to processor p_1 ($Y_{5,1} = \ldots = Y_{11,1} = 1$). This mapping forces interfaces for edge $e_1 = (v_1, v_5), \ldots, e_4 = (v_4, v_8)$ ($Z_{1,5} = \ldots = Z_{4,8} = 1$) to be needed, because data has to be transported from h_1 to p_1. Therefore, $e_1 \ldots e_4$ are mapped to bus b_1 ($z_{1,5,1} = \ldots = z_{4,8,1} = 1$). To reduce the amount of hardware area, v_1 and v_2 share the same hardware instance of an FIR-filter on h_1 ($x_{1,1,1} = x_{2,1,1} = 1$). v_3 and v_4 share the second one ($x_{3,2,1} = x_{4,2,1} = 1$).
The variables get the following values: $nx_{1,1,1} = nx_{1,2,1} = 1, nx_{1,3,1} = nx_{1,4,1} = 0$, because only the first two hardware instances of four possible FIR-filters (c_1) are required on h_1 ($NX_{1,1} = 2$). All multiplications (c_2) and adders (c_3) are realized on processor p_1. Therefore, one function is needed for multiplying ($NY_{2,1} = 1$) and another function is needed for adding ($NY_{3,1} = 1$) incoming values. The timing diagram shows a possible schedule for this partitioning. In the depicted case $b_{5,6} = 1$, because v_5 is executed before v_6 on p_1; $bi_{1,2} = 1$, because the transfer for e_1 is executed before the transfer for e_2.

6.2. The Constraints

The following constraints have to be fulfilled:

1. **General Constraints**: Each node v_i is executed exactly on one target technology component t_k, a processor or a hardware component (eq.5). If a system component c_l has been realized on a processor p_k, then it is not necessary to implement it more than once (eq.6), because it can be implemented as one function and several function calls (see definition 3). Therefore, the number $NY_{l,k}$ is calculated by equations 7 and 8. In contrast to the binary variable $NY_{l,k}$, the number of hardware instances $NX_{l,k}$ of a system component c_l realized on a hardware component h_k may be greater than one. If no hardware sharing is considered, then $NX_{l,k}$ is equal to the sum of system instances of c_l that have been mapped to h_k (eq.9).

$$\forall i \in I : \forall k \in KH : \quad X_{i,k} \leq 1 \tag{3}$$
$$\forall i \in I : \forall k \in KP : \quad Y_{i,k} \leq 1 \tag{4}$$
$$\forall i \in I : \quad \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1 \tag{5}$$

$$\forall l \in L : \forall k \in KP : \quad NY_{l,k} \leq 1 \tag{6}$$
$$\forall l \in L, \forall i : I(v_i) = c_l, \forall k \in KP : \quad NY_{l,k} \geq Y_{i,k} \tag{7}$$
$$\forall l \in L, \forall k \in KP : \quad NY_{l,k} \leq \sum_{i:I(v_i)=c_l} Y_{i,k} \tag{8}$$
$$\forall l \in L, \forall k \in KH : \quad NX_{l,k} = \sum_{i:I(v_i)=c_l} X_{i,k} \tag{9}$$

2. **Resource Constraints**: The area C_k^a (eq.10) used on a hardware component h_k is calculated by accumulating the costs for all hardware instances of system components realized on h_k. The amount of used memory (eq.11,12) on a processor p_k is calculated by summing up the costs for implementing these system components as functions. The resource costs may not violate their resource constraints.

$$\forall k \in KH : \quad C_k^a = \sum_{l \in L} NX_{l,k} * c_{l,k}^a \leq MAX_k^a \tag{10}$$
$$\forall k \in KP : \quad C_k^{dm} = \sum_{l \in L} NY_{l,k} * c_{l,k}^{dm} \leq MAX_k^{dm} \tag{11}$$
$$\forall k \in KP : \quad C_k^{pm} = \sum_{l \in L} NY_{l,k} * c_{l,k}^{pm} \leq MAX_k^{pm} \tag{12}$$

3. **Design Constraints**: The design costs for the complete system are calculated by accumulating the resource costs required by the components of the target technology (eq.13-15). These design costs may not exceed their given design constraints. The required hardware area includes additional hardware CI_k^a used for interfaces (eq.13). If interfacing is not considered, $CI_k^a = 0$ for all busses b_k. The design costs C^t will be described separately.

$$C^a = \sum_{k \in KH} C_k^a + \sum_{k \in KB} CI_k^a \leq MAX^a \quad (13)$$

$$C^{dm} = \sum_{k \in KP} C_k^{dm} \leq MAX^{dm} \quad (14)$$

$$C^{pm} = \sum_{k \in KP} C_k^{pm} \leq MAX^{pm} \quad (15)$$

4. **Timing Constraints**:
The timing costs cannot be calculated by accumulating the execution time of the nodes, because two nodes v_1, v_2 can be executed in parallel if they do not share the same resources and if there is no path from v_1 to v_2 and vice versa. To determine the starting time and ending time for each node, scheduling has to be performed. The execution time T_i^D (eq.16) of v_i is either a hardware or a software execution time. The ending time T_i^E (eq.17) of v_i is the sum of starting time T_i^S and execution time T_i^D. The system execution time C^t (eq.18) is the maximum of the ending times of all nodes v_i and may not violate the global design timing constraint. Data dependencies (eq.19) have to be considered for all edges $e = (v_{i_1}, v_{i_2})$ including interface communication time TI_{i_1,i_2}^D of equation 35. If interfacing is not considered, $TI_{i_1,i_2}^D = 0$. The starting times T_i^S (eq.20) of nodes have to be in their ASAP/ALAP-range which can be calculated in a preprocessing step.

$$\forall i \in I: \quad T_i^D = \sum_{k \in KH} X_{i,k} * c_{i,k}^{th} + \sum_{k \in KP} Y_{i,k} * c_{i,k}^{ts} \quad (16)$$

$$\forall i \in I: \quad T_i^E = T_i^S + T_i^D \quad (17)$$

$$\forall i \in I: \quad T_i^E \leq C^t \leq MAX^t \quad (18)$$

$$\forall e = (v_{i_1}, v_{i_2}) \in E: \quad T_{i_2}^S \geq T_{i_1}^E + TI_{i_1,i_2}^D \quad (19)$$

$$\forall i \in I: \quad ASAP(v_i) \leq T_i^S \leq ALAP(v_i) \quad (20)$$

6.3. Hardware Sharing

If hardware sharing is considered, then it is not sufficient to model bindings between nodes v_i and hardware components h_k with help of the binary variable $X_{i,k}$. In order to consider hardware sharing, the binding of v_i to the j-th hardware instance (of system component $c_l = I(v_i)$) contained in h_k has to be modelled. This binding is modelled using the binary binding variable $x_{i,j,k}$ (eq.21).

Example 9:

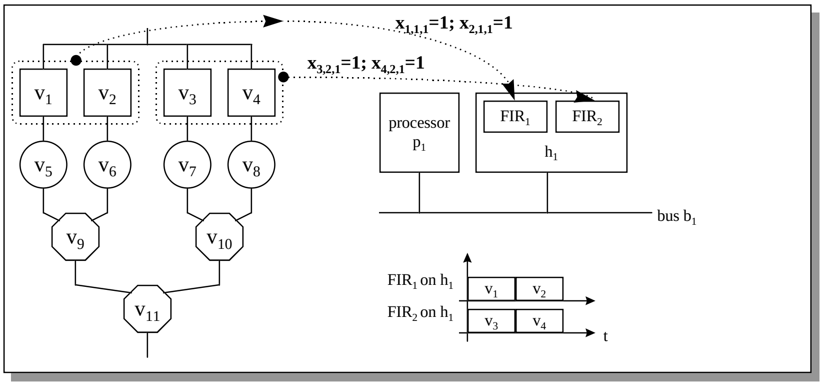

Figure 10. Hardware sharing

In figure 10 an example is given for sharing hardware resources to minimize the amount of hardware area. The system instances v_1, v_2 of an FIR-filter are mapped to the first hardware instance FIR_1 of an FIR-filter on hardware component h_k ($x_{1,1,1} = x_{2,1,1} = 1$). v_3 and v_4 share the second hardware instance ($x_{3,2,1} = x_{4,2,1} = 1$). Therefore, 4 system instances are realized by two hardware instances of FIR-filters on h_k. The timing diagram shows that v_1, v_2 and also v_3, v_4 have to be scheduled. But both hardware instances FIR_1 and FIR_2 are able to work in parallel on h_1.

A node v_i is realized on h_k, if v_i is bound to one hardware instance of system component $c_l = I(v_i)$ on h_k (eq.22). If at least one instance v_i of system component c_l is bound to the j-th hardware instance of c_l on h_k, then $nx_{l,j,k} = 1$ (eq.23,24). The number $NX_{l,k}$ of used hardware instances of c_l on h_k is calculated by accumulating the variables $nx_{l,j,k}$ (eq.25).

$\forall k \in KH : \forall l \in L : N = \mid instances\_of(c_l) \mid :$

$$\begin{array}{rrcll}
\forall i : I(v_i) = c_l : \forall j \in \{1, \ldots, N\} : & x_{i,j,k} & \leq & 1 & (21) \\
\forall i : I(v_i) = c_l : & X_{i,k} & = & \sum_{j=1}^{N} x_{i,j,k} & (22) \\
\forall i : I(v_i) = c_l : \forall j \in \{1, \ldots, N\} : & nx_{l,j,k} & \geq & x_{i,j,k} & (23) \\
\forall j \in \{1, \ldots, N\} : & nx_{l,j,k} & \leq & \sum_{i:I(v_i)=c_l} x_{i,j,k} & (24) \\
 & NX_{l,k} & = & \sum_{j=1}^{N} nx_{l,j,k} & (25)
\end{array}$$

If hardware sharing is not considered, then equation 9 is used instead of equations 21-25.

6.4. Interfacing

An interface has to be realized for an edge $e = (v_{i_1}, v_{i_2})$, if v_{i_1} and v_{i_2} are realized on different target technology components.
Example 10:

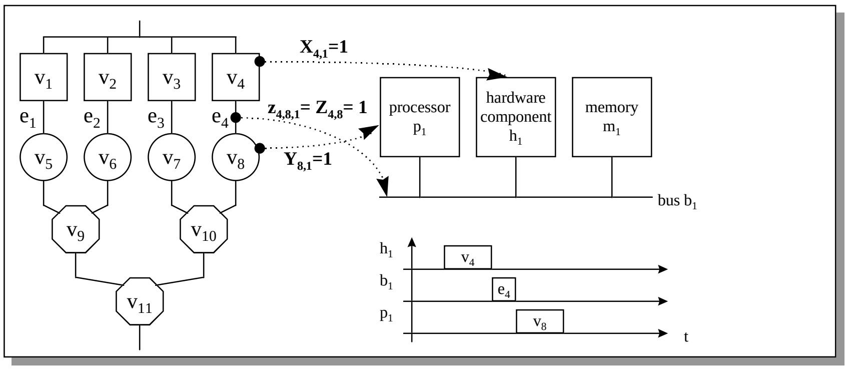

Figure 11. Interface

In figure 11 an example is given for a required interface. v_4 has been mapped to hardware component h_1 ($X_{4,1} = 1$) and v_8 to processor p_1 ($Y_{8,1} = 1$). Therefore, the output data of v_4, calculated on h_1 has to be moved to v_8, implemented on p_1. Thus, an interface is needed for e_4, indicated by $Z_{4,8} = 1$. For this reason, edge $e_4 = (v_4, v_8)$ is mapped to bus b_1 ($z_{4,8,1} = 1$), to realize the data transfer. The timing diagram shows that v_8 starts after the data has been transfered from v_4 using b_1.

An interface is needed between two nodes v_{i_1}, v_{i_2}, indicated by $Z_{i_1,i_2} = 1$, if they are mapped to different target technology components (eq.27-31). With help of the interface binding variable $z_{i_1,i_2,k}$, a bus is selected realizing the data transfer from v_{i_1} to v_{i_2} (eq.33). The additional amount of hardware area CI^a_k and the communication delay $TI^D_{i_1,i_2}$ used for an interface are calculated in equations 34-35. Additional constraints for the starting and ending time of communication are added in equations 36-39.

$\forall e = (v_{i_1}, v_{i_2}) \in E:$

$$
\begin{array}{lrcll}
 & Z_{i_1,i_2} & \leq & 1 & (26)\\
\forall k \in KH: & Z_{i_1,i_2} & \geq & X_{i_1,k} - X_{i_2,k} & (27)\\
\forall k \in KH: & Z_{i_1,i_2} & \geq & X_{i_2,k} - X_{i_1,k} & (28)\\
\forall k \in KP: & Z_{i_1,i_2} & \geq & Y_{i_1,k} - Y_{i_2,k} & (29)\\
\forall k \in KP: & Z_{i_1,i_2} & \geq & Y_{i_2,k} - Y_{i_1,k} & (30)\\
 & Z_{i_1,i_2} & \rightarrow & minimize & (31)\\
\forall k \in KH: & z_{i_1,i_2,k} & \leq & 1 & (32)\\
 & Z_{i_1,i_2} & = & \sum_{k \in KB} z_{i_1,i_2,k} & (33)
\end{array}
$$

$\forall e = (v_{i_1}, v_{i_2}) \in E:$

$$
\begin{array}{lrcll}
\forall k \in KB: & CI_k^a & = & \sum_{e \in E} z_{i_1,i_2,k} * ci_{i_1,i_2,k}^a & (34)\\
 & TI_{i_1,i_2}^D & = & \sum_{k \in KB} z_{i_1,i_2,k} * ci_{i_1,i_2,k}^t & (35)\\
 & TI_{i_1,i_2}^E & = & TI_{i_1,i_2}^S + TI_{i_1,i_2}^D & (36)\\
 & TI_{i_1,i_2}^S & \geq & T_{i_1}^E & (37)\\
 & TI_{i_1,i_2}^E & \leq & T_{i_2}^S & (38)\\
 & ASAP(e_{i_1}) & \leq & TI_{i_1,i_2}^S \leq ALAP(e_{i_1}) & (39)
\end{array}
$$

6.5. Scheduling

Two nodes v_{i_1}, v_{i_2} which can be executed in parallel have to be sequentialized, if

- v_{i_1} and v_{i_2} are executed on the same processor or
- v_{i_1} and v_{i_2} share the same hardware instance on the same hardware component.

To sequentialize two nodes v_{i_1}, v_{i_2}, the binary decision variable b_{i_1,i_2} is used.

Two edges e_{i_1}, e_{i_2} have to be sequentialized, if e_{i_1} and e_{i_2} represent interfaces and both edges use the same bus to realize the communication. In this case, the binary decision variable bi_{i_1,i_2} is used to schedule e_{i_1} and e_{i_2}. The following example will illustrate all situations where scheduling constraints are required.

Example 11:

In figure 12 all three possibilities are depicted when scheduling constraints are required.

1. v_7 and v_8 are mapped to the same processor p_1: Then, v_7 has to be executed before v_8 ($b_{7,8} = 1$) or v_8 before v_7 ($b_{8,7} = 1$).
2. v_3 and v_4 are mapped to the same hardware instance of an FIR-filter on h_1: Therefore v_3 and v_4 have to be scheduled.

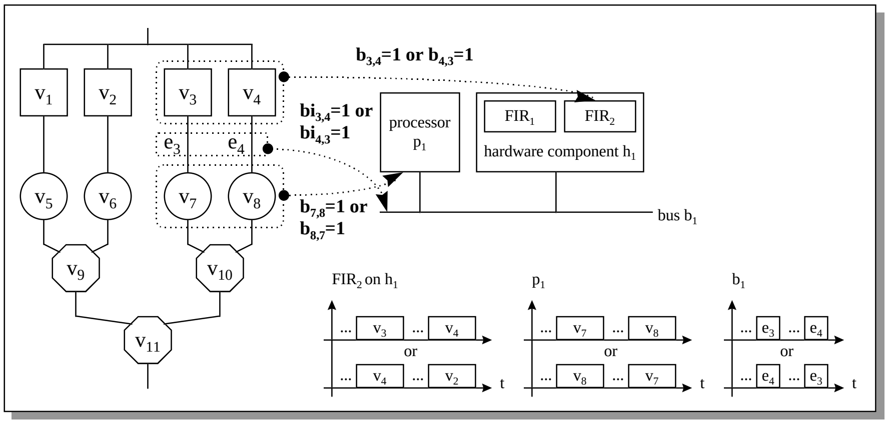

Figure 12. Scheduling

3. e_3 realizes an interface for transferring data from v_3 (on p_1) to v_7 (on h_1). e_4 realizes an interface between v_4 and v_8. Both edges have been mapped to bus b_1 to transferring the data. For this reason, the communication times of e_3 and e_4 have to be scheduled. If e_3 is scheduled before e_4, then the schedule variable $bi_{3,4} = 1$, otherwise $bi_{4,3} = 1$.

The following constraints are necessary, to schedule nodes and edges:

$\forall k \in KP, \forall v_{i_1}, v_{i_2} \in V : v_{i_1} \in schedule\_nodes(v_{i_2})$

$$T^E_{i_1} \leq T^S_{i_2} + (3 - b_{i_1,i_2} - Y_{i_1,k} - Y_{i_2,k}) * C_1 \tag{40}$$
$$T^E_{i_2} \leq T^S_{i_1} + (2 + b_{i_1,i_2} - Y_{i_1,k} - Y_{i_2,k}) * C_2 \tag{41}$$

$\forall l \in L :\mid instances\_of(c_l) \mid \geq 2 :$
$\forall k \in KH, \forall v_{i_1}, v_{i_2} \in V : v_{i_1}, v_{i_2} \in instances\_of(c_l)$

$$T^E_{i_1} \leq T^S_{i_2} + (3 - b_{i_1,i_2} - x_{i_1,j,k} - x_{i_2,j,k}) * C_3 \tag{42}$$
$$T^E_{i_2} \leq T^S_{i_1} + (2 + b_{i_1,i_2} - x_{i_1,j,k} - x_{i_2,j,k}) * C_4 \tag{43}$$

$\forall k \in KB, \forall e_{i_1} = (v_{i_{11}}, v_{i_{12}}), e_{i_2} = (v_{i_{21}}, v_{i_{22}}) \in E : e_{i_1} \in schedule\_edges(e_{i_2}) :$

$$TI^E_{i_{11},i_{12}} \leq TI^S_{i_{21},i_{22}} + (3 - bi_{i_1,i_2} - z_{i_{11},i_{12},k} - z_{i_{21},i_{22},k}) * C_5 \tag{44}$$
$$TI^E_{i_{21},i_{22}} \leq TI^S_{i_{11},i_{12}} + (2 + bi_{i_1,i_2} - z_{i_{11},i_{12},k} - z_{i_{21},i_{22},k}) * C_6 \tag{45}$$

The idea of these constraints is equivalent in all three cases. For this reason, only the constraints 40 and 41 for scheduling nodes v_{i_1} and v_{i_2} using the same processors p_k are described in the following. If v_{i_1} and v_{i_2} have to be scheduled $(Y_{i_1,k} = Y_{i_2,k} = 1)$, then one of the following conditions has to be fulfilled:

1. v_{i_1} is executed before v_{i_2} $(b_{i_1,i_2} = 1) \Rightarrow T^E_{i_1} \leq T^S_{i_2}$, or
2. v_{i_2} is executed before v_{i_1} $(b_{i_1,i_2} = 0) \Rightarrow T^E_{i_2} \leq T^S_{i_1}$.

This fact is modelled by the constraints defined in equations 40 and 41:

$Y_{i_1,k} = Y_{i_2,k} = 1$	b_{i_1,i_2}	equation 40	equation 41
yes	0	$T^E_{i_1} \leq T^S_{i_2} + C_1$	$T^E_{i_2} \leq T^S_{i_1}$
yes	1	$T^E_{i_1} \leq T^S_{i_2}$	$T^E_{i_2} \leq T^S_{i_1} + C_2$
no	$0, 1$	$T^E_{i_1} \leq T^S_{i_2} + n_1 * C_1, n_1 \geq 1$	$T^E_{i_2} \leq T^S_{i_1} + n_2 * C_2, n_2 \geq 1$

If v_{i_1} and v_{i_2} have to be scheduled $(Y_{i_1,k} = Y_{i_2,k} = 1)$, only one of equations 40 and 41 results in hard constraints. If $b_{i_1,i_2} = 0$, equation 40 has no effect and if $b_{i_1,i_2} = 1$ equation 41 can be ignored. If either $Y_{i_1,k} = 0$ or $Y_{i_2,k} = 0$, both constraints have no effect, if C_1 and C_2 are dimensioned correctly. It can be shown, that C_1 and C_2 have the following lower bounds:

Let $MaximalExecutionTime(v_i) = Max \left\{ \{c^{ts}_{i,k_1} \mid k_1 \in KP\} \cup \{c^{th}_{i,k_2} \mid k_2 \in KH\} \right\}$

1. $C_1 = \lceil ALAP(v_{i_1}) + MaximalExecutionTime(v_{i_1}) - ASAP(v_{i_2}) \rceil$, because

$$\begin{aligned} T^E_{i_1} &\leq T^S_{i_2} + C_1 \\ &\leq T^S_{i_2} + ALAP(v_{i_1}) + MaximalExecutionTime(v_{i_1}) - ASAP(v_{i_2}) \\ &\leq ALAP(v_{i_1}) + MaximalExecutionTime(v_{i_1}) \qquad \Box \end{aligned}$$

2. $C_2 = \lceil ALAP(v_{i_2}) + MaximalExecutionTime(v_{i_2}) - ASAP(v_{i_1}) \rceil \ldots \Box$

6.6. Heuristic Scheduling

Resource constrained scheduling is a NP-complete problem [6]. Therefore, it is clear that solving the scheduling problem optimally can not be done efficiently. For this reason, we have developed an algorithm using integer programming that solves the partitioning problem 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 a valid 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 13).

Example 12:

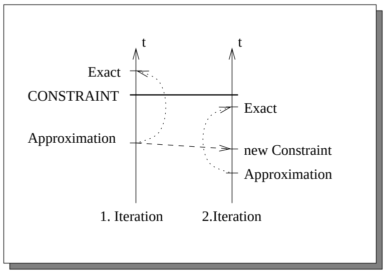

Figure 13. Heuristic scheduling

> The first partitioning results in an approximated execution time which fulfills the given timing constraint. 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.

The following constraints are used in addition to the equations 16-20 to approximate time values:

- **Predecessor nodes:**
  A node v is ready to start, if all its predecessors have finished their execution. The effect of being forced to schedule some of these predecessor nodes can be exploited to estimate the starting time of v.
  Example 13:

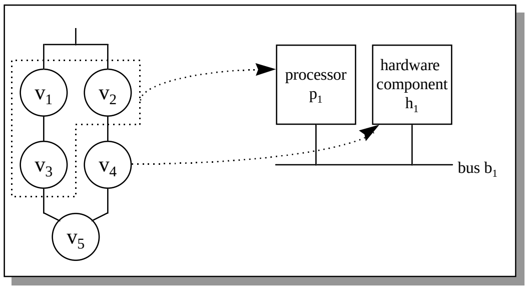

Figure 14. Using predecessor nodes in heuristic scheduling

> In figure 14 the starting time of v_5 is at least the sum of execution times of v_1, v_2, v_3, because all 3 nodes have been mapped to p_1 and have to be scheduled.
> $\Rightarrow \qquad T_5^S \geq c_{1,1}^{ts} + c_{2,1}^{ts} + c_{3,1}^{ts}$.

The starting time of a node v_i is equal or greater to the accumulated software execution times of all predecessor nodes v_i (eq.46) on a processor p_k. Similar constraints can be added if hardware sharing (eq.47) and/or interfacing (eq.48) are considered.

$\forall i_1 \in I : \forall l \in L : N = \mid instances\_of(c_l) \mid$:
Let $LV = pred\_nodes(v_{i_1})$ and $LE = pred\_edges(v_{i_1})$:

$$\forall k \in KP : \quad T^S_{i_1} \geq \sum_{v_{i_2} \in LV} Y_{i_2,k} * c^{ts}_{i_2,k} \tag{46}$$

$$\forall j \in \{1,\ldots,N\} : \forall k \in KH : \quad T^S_{i_1} \geq \sum_{\substack{v_{i_2} \in LV, \\ I(v_{i_2})=c_l}} x_{i_2,j,k} * c^{th}_{l,k} \tag{47}$$

$$\forall k \in KB : \quad T^S_{i_1} \geq \sum_{e=(v_{i_2},v_{i_3}) \in LE} z_{i_2,i_3,k} * ci^t_{i_2,i_3,k} \tag{48}$$

- **Dominator nodes:**
  Another possibility to estimate the starting time of a node v is to look at dominator nodes. A dominator node w of v is a node, such that each path to v contains w (see definition 8). v is able to start if dominator w and all nodes between w and v have been executed.
  Example 14:

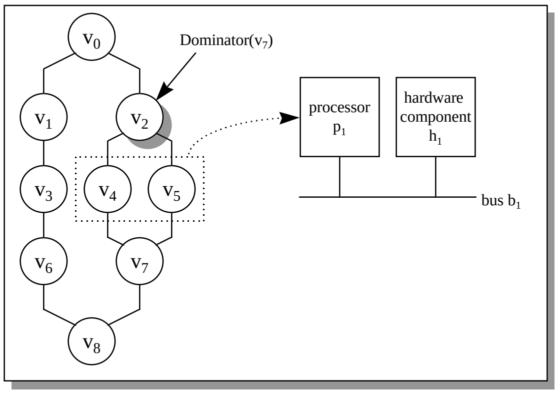

Figure 15. Using dominator nodes in heuristic scheduling

In figure 15 the starting time of v_7 is at least the sum of the ending time of v_2 and the execution times of v_4, v_5, because v_4, v_5 have to be scheduled after executing v_2.
$\Rightarrow \qquad T^S_7 \geq T^E_2 + c^{ts}_{4,1} + c^{ts}_{5,1}$.

The starting time of a node v_{i_1} is equal or greater to the sum of the ending time of the dominator node v_{i_0} of v_{i_1} and the software execution times on processor

p_k of all nodes on the paths between v_{i_0} and v_{i_1} (eq.49). Equation 50 defines the same constraint for the hardware execution times of all shared nodes on the paths between v_{i_0} and v_{i_1}. Equation 51 defines the equivalent constraint considering communication times of required interfaces.

$\forall i_0, i_1 \in I : v_{i_0} \in dominator\_nodes(v_{i_1}) : \forall l \in L : N = \mid instances\_of(c_l) \mid :$
Let $LV = path\_nodes(v_{i_0}, v_{i_1})$ and $LE = path\_edges(v_{i_0}, v_{i_1})$:

$$\forall k \in KP : \quad T^S_{i_1} \geq T^E_{i_0} + \sum_{v_{i_2} \in LV} Y_{i_2,k} * c^{ts}_{i_2,k} \tag{49}$$

$$\forall j \in \{1, \ldots, N\} : \forall k \in KH : \quad T^S_{i_1} \geq T^E_{i_0} + \sum_{\substack{v_{i_2} \in LV, \\ I(v_{i_2}) = c_l}} x_{i_2,j,k} * c^{th}_{l,k} \tag{50}$$

$$\forall k \in KB : \quad T^S_{i_1} \geq T^E_{i_0} + \sum_{e=(v_{i_2}, v_{i_3}) \in LE} z_{i_2,i_3,k} * ci^t_{i_2,i_3,k} \tag{51}$$

7. Results

To evaluate the quality of our partitioning approach we have done an application study in the area of audio algorithms. We have implemented systems between 6 and 29 nodes (between 6 and 37 edges) which have to be partitioned. The partitioning results in this section have been calculated for the following systems:

- n-band-equalizers with $n \leq 7$,
- a system called *audiolab* including a mixer, a fader, an echo, an equalizer and a balance ruler, and
- an MPEG audio encoder (layer II).

These systems were mapped to a target architecture (see figure 16) containing a SPARC processor, an ASIC manufactured in a 1μ CMOS technology (COMPASS library) and external memory. A bus connects these components.

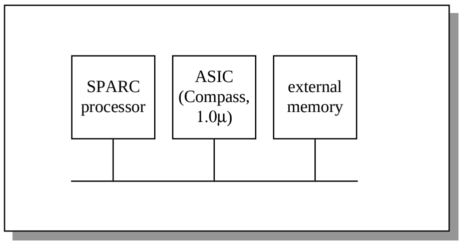

Figure 16. Target architecture

All calculated partitionings consider interface costs and hardware sharing effects between nodes. The IP-models were solved by using the IP-solver package OSL [3] from IBM. The computation times of the examples represent CPU seconds on a RS6000. The heuristic partitioning approach can be evaluated by examining

- the quality and
- the computation time

compared to optimal solutions.

The quality of the heuristic approach can be derived from the deviation between the exact and the approximated solutions. Two equalizers, a 2-band- and a 3-band-equalizer, were partitioned with the optimal and the heuristic approach. For each system, solutions were calculated for a set of 8 timing constraints, resulting in a set of designs ranging from a complete software to a complete hardware solution.

The hardware area deviation is zero for both benchmarks. The total system execution time differs between both approaches (see figure 17). The optimal approach

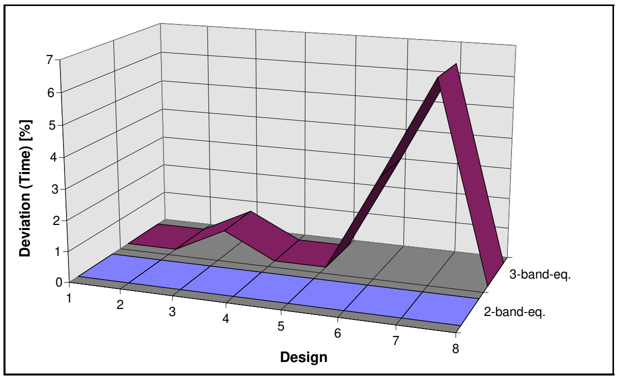

Figure 17. Deviation System Execution Time (exact/heuristic approach)

calculates a mapping first using a minimal amount of hardware area, and then minimizes the system execution time. The heuristic approach tries to minimize the hardware area and finding a valid schedule in a second step. For this reason, the resulting system execution times may differ. The deviation in our experiments is not greater than 6.4%. The average deviation is smaller than 1%.

The main difference of both approaches is the computation time (see figure 18). The computation time for both benchmarks is below one second for all designs using

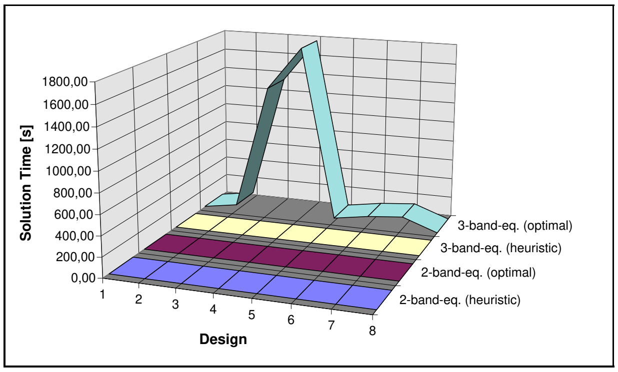

Figure 18. Computation Time (exact/heuristic approach)

the heuristic approach. The computation time calculating the optimal solution is 1773 seconds in the worst case. It becomes clear that solving the hardware/software partitioning problem optimally is not applicable to systems with a larger number of instances.

In figure 19 an overview of partitioning all benchmarks using the heuristic approach is given.

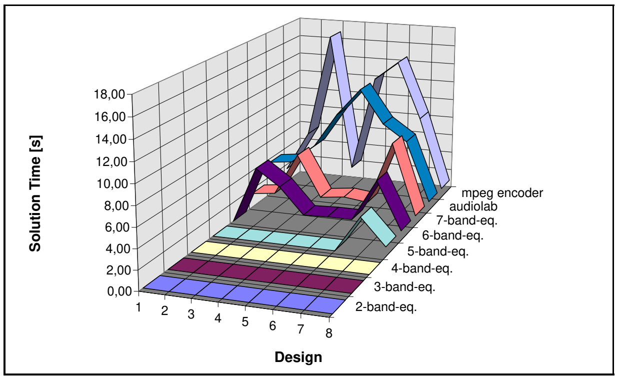

Figure 19. Computation Time (heuristic approach)

The largest computation time is 16.9 seconds for partitioning the *mpeg* system (containing 29 nodes and 37 edges).

Clearly, the heuristic approach is more practical than the optimal approach, because the results are always nearly optimal and the computation times are significantly lower.

Finally, the trade-off between hardware area and system execution time is demonstrated for the *audiolab* system (containing 25 nodes and 31 edges). 8 different partitionings (see figure 20) have been calculated for 8 different timing constraints.

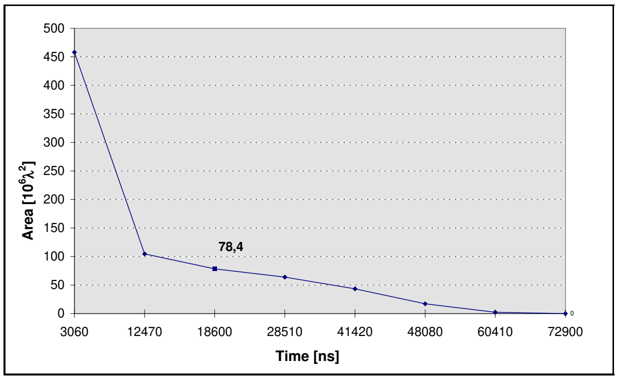

Figure 20. Area/Time Curve for the Audiolab System

A pure software realization of the *audiolab* system would result in a system execution time of 72900 ns, but this solution is too slow. The fastest realization would have a system execution time of 3060 ns, but it would be a complete hardware realization using $457,9 * 10^6\lambda^2$ chip area. This solution is a too expensive. The best solution is a hardware/software solution which fulfills the timing constraint of 22675 ns (44.1 kHz sample frequency) with a minimal amount of hardware. The calculated solution has a system execution time of 18600 ns and would require $78,4 * 10^6\lambda^2$ chip area.

8. Conclusion

This paper presents a new approach of fully-automated hardware/software partitioning supporting multi-processor systems, interfacing and hardware sharing. An algorithm has been developed, which is able of solving the hardware/software partitioning problem using integer programming and leading to (nearly) 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 and therefore fewer iteration steps. The presented results are very promising, because nearly optimal results are calculated in short time. Future work will deal with further refinement of the IP-model for target architecture selection and design studies of other system level examples.

Notes

1. *COSYS*: (Codesign System specification tool)
2. *COOL*: (Codesign Tool)
3. OSL : Optimal Subroutine Library

References

1. A. Bender. Design of an Optimal Loosely Coupled Heterogeneous Multiprocessor System. *European Design & Test Conference (ED&TC)*, pages 275–281, 1996.
2. W. Ecker. Using VHDL for HW/SW Co-Specification. *International Conference on Computer-Aided Design (ICCAD)*, pages 500–505, 1993.
3. R. Ernst, J. Henkel, and T. Benner. Hardware-software Cosynthesis for Microcontrollers. *IEEE Design & Test*, Vol.12, pages 64–75, 1993.
4. P. Eles, Z. Peng, and A. 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.
5. R.K. Gupta, C. Coelho, and G. De Micheli. Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components. *29th ACM, IEEE Design Automation Conference*, pages 225–230, 1992.
6. M.R. Garey and D.S. Johnson. Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM J. Comput.*, pages 397–411, 1975.
7. D. Gajski, F. Vahid, S. Narayan, and J. Gong. Specification and Design of Embedded Systems. *Prentice-Hall*, 1994
8. 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.
9. J. Henkel, R. Ernst, W. Ye, M. Trawny, and T. Benner. COSYMA: Ein System zur Hardware/Software Co-Synthese. *GME Fachbericht Nr. 15 Mikroelektronik*, pages 167–172, 1995.
10. A. Jantsch, P. Ellervee, J. Öberg, A. Hemani, and H. Tenhunen. Hardware/Software Partitioning and Minimizing Memory Interface Traffic. *European Design Automation Conference (EURO-DAC)*, pages 226–231, 1994.
11. A. Kalavade and E.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.
12. A. Kalavade and E.A. Lee. The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection. *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, 1995.
13. 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.
14. R. Niemann, and P. Marwedel. Hardware/Software Partitioning using Integer Programming. *European Design & Test Conference (ED&TC)*, pages 473–479, 1996.
15. Z. Peng and K. Kuchcinski. An Algorithm for Partitioning of Application Specific Systems. *Proceedings of the European Conference on Design Automation (EDAC)*, pages 316–321, 1993.
16. F. Vahid, J. Gong, and D. Gajski. A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning. *European Design Automation Conference (EURO-DAC)*, pages 214–219, 1994.

Received Date
Accepted Date
Final Manuscript Date