A System Level HW/SW Partitioning and Optimization Tool

M. Schwiegershausen H. Kropp P. Pirsch
University of Hannover
Laboratorium für Informationstechnologie
Schneiderberg 32, 30167 Hannover, Germany
schwiege@mst.uni-hannover.de

Abstract

This paper presents a system level HW/SW partitoning methodology and its implementation as CAD tool for the optimization of heterogeneous multiprocessor systems. Starting from modelling of the signal processing scheme and of the available processor resources, performance and expense measures are estimated for a finite set of processor modules. Based on these measurements, a numerical optimization can be carried out by using mixed integer linear programming as mathematical framework, leading to a heterogeneous system, which is optimal in terms of area expense and throughput rate.

1 Introduction

The domain of advanced, digital signal processing (DSP) systems is characterized by an increase concerning computational power and throughput rate requirements. A typical DSP system consists of several algorithms with rather different requirements for each single algorithm. Examples can be found in video coding, with source rates ranging from 1.5 Mbyte/s for visual telephony [1] (H.261, CIF 10 Hz) up to 15.6 Mbyte/s for MPEG-2 [2] (CCIR-ITU 601). One suitable way to achieve compact VLSI realizations with high throughput meeting these requirements are hardware architectures, consisting of dedicated modules on one side and programmable units on the other side. These composite architectures are therefore referred to as *heterogeneous* multiprocessor systems.

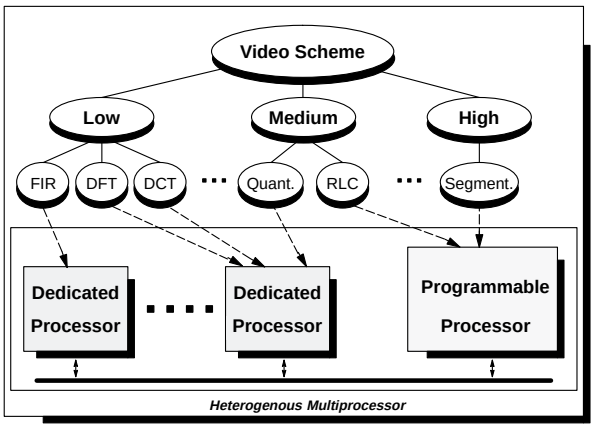

Fig. 1: Heterogeneous Multiprocessor

Herein, the dedicated modules are highly suitable for performing tasks like FIR filtering, FFT, etc., with a predefined sequence of operations, thus often called low-level algorithms. On the other hand, programmable modules, like RISC-based architectures are more appropriate for the implementation of high-level algorithms, like segmentation, feature extraction, due the large amount of data- and control-dependencies between successive operations of the algorithm. Furthermore, often medium-level algorithms are also part of a composite signal processing scheme. In this case both, dedicated as well as programmbale modules are appropriate for an implementation (see Fig. 1).

Nevertheless, it is conceivable that for each single algorithm of a composite scheme a large multitude of dedicated as well as programmable processor alternatives can be found. Thus, the question arises, which combination of processors leads to an optimal heterogeneous multiprocessor system for a given signal processing scheme. Due to the large multitude of possible combinations, it is impossible for the human designer to choose manually the best one. Therefore, a systematic design methodology for the optimization of heterogeneous systems is mandatory, including the partitioning of a composite DSP scheme into hardware, i.e. dedicated or programmable processors.

Today's state-of-the-art CAD environments provide no sufficient support with respect to the derivation of heterogeneous systems. There are DSP environments like *CATHEDRAL II, 2nd* [4] available today, in order to derive VLIW architectures addressing medium throughput applications. However, advanced signal processing applications demand for architectures providing extremely high computational and throughput rates. Furthermore, DSP prototyping and codesign systems like *PTOLEMY* [5], and *GRAPE-II* were developed, with a focus either on simulation, software synthesis, or on prototyping aiming at target platforms consisting of FPGAs or commercial DSPs. To summarize, to our opinion, none of these environments sufficiently supports the designer in the domain of heterogeneous multiprocessor system development. Thus, a new methodology for the partitioning and optimization at the system level has been developed and will be presented.

The organization of this paper is as follows: Section 2 presents the proposed methodology. First results, using a video coding scheme as case study are presented in Section 3. Section 4 gives some hints on

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

the implementation as CAD tool. Concluding remarks are provided in Section 5.

2 The CAD-Tool HMOPS

The investigated approach HMOPS[1] consists of three main steps. It starts from modelling of algorithms and architectures in a domain-specific way. Then, a set of performance and expense measures is estimated for the implementation of algorithms on dedicated as well as programmable modules. Finally, an overall optimization is performed. The single steps of our tool are explained in more detail below.

2.1 Modelling Algorithms & Architectures

Due to the different ways of hardware implementation of a specific algorithm, we choose a description for algorithms and architectures which is independently from each other. As mentioned before, in composite video signal processing schemes, the different kinds of algorithms, can be distinguished in regular low–level algorithms on one side and non-regular, data dependent medium–level algorithms on the other side. Typical examples of low–level algorithms are filtering (FIR, IIR, etc.), transform (DCT, DFT, etc.), and motion estimation (BMA, etc.). On the other side adaptive quantization (Q, etc.), run and variable length coding (RLC, VLC) are typically classified as medium–level algorithms. All these algorithm classes can be subdivided hierarchically in different specific algorithms. Therefore, we use a hierarchical description of algorithms in form of a tree (Fig. 2).

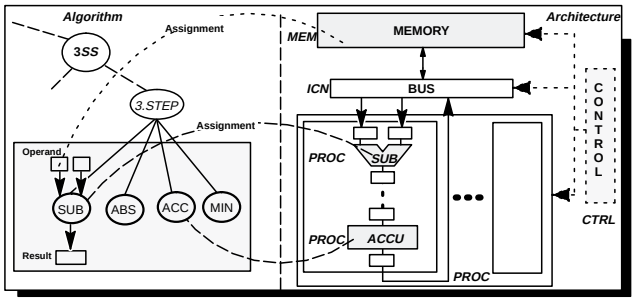

Fig. 2: Hierarchical Description and Assignment

With respect to the design space exploration of heterogeneous systems, it is desireable to characterize each node of the tree by a predefined set of parameters. To describe for example possible algorithm classes, the set Θ was introduced, including all relevant parameters of an algorithm class. For example, in case of a 2–dimensional filter algorithm, like FIR four parameters are mandatory, i.e. $\Theta = \{T_W, T_H, K_W, K_H\}$. In more detail, T_W designates the width and T_H the height for an rectangle image block of pixels to be filtered by a FIR algorithm. Additionally, K_W, K_H are necessary, in order to specify the width and height for the size of the kernel window, to be passed over the rectangle image block. To clarify the meaning of the four parameters they are visualized in Figure 3. Furthermore, for any regular low-level algorithm class a similar set of parameters can be derived.

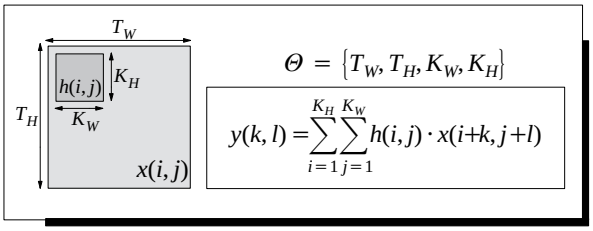

Fig. 3: Parameter Set Θ for 2D-FIR Filtering

Related to the description of algorithms, architectures are described in a tree, too. Contrary to the nodes of the algorithm–tree, the hardware units has to be distinguished in more detail, especially in case of programmable architectures. Therefore, our description is based on a generic multiprocessor model [7]. Depending on the degree of detailization, a global module like a processor presents the top level of the architecture tree (Fig. 8). It continues with lower levels until basic blocks $\mathcal{BB}$ like ALUs, ACCUs, registers, etc. are reached. In each level, any node (module) of the architecture–tree is declared by its type as processing unit (PROC), memory (MEM), interconnection network (ICN), or control unit (CTRL). Whereas in programmable processors all possible unit–types are used, in case of dedicated (array) processors, the most important unit–type is the PROC-module. Contrary to that, the control part is often neglectible. Distinguishing dedicated and programmable modules, in this paper we use for dedicated processor elements the abbreviation $\mathcal{PE}$, respectively.

Another important parameter, the formation (module order) of single modules, characterize the way of data execution. Possible module orders are: sequential(SEQ), parallel (PAR), pipeline (PIPE), and array (ARRAY) structures. Due to the fact, that especially for dedicated processors we only need PROC-module-types ($\mathcal{PE}$s), the module order SEQ, PAR, PIPE, and ARRAY is used to distinguish the dedicated architectures in more detail. With other words, the type of a dedicated processor is identical with its module order. Additionally parameters of the algorithm or architecture have to be specified in the next section with the aim of estimate performance measures.

2.2 Estimating Performance and Expense

Since we are interested in the best combination of dedicated and programmable processors, it is necessary – prior to the optimization – to characterize each single processor alternative by some performance and expense measures . The most relevant measures which have to be estimated for dedicated as well as programmable processors in our approach are:

- Computation Time τ
  The computation time τ corresponds to the number of clock cycles, necessary to compute once any specific video processing algorithm $\mathcal{ALGO}$[2]

[1] Part of the methodology described is currently under development as a CAD tool called HMOPS (**H**eterogeneous **M**ultiprocessor **OP**timization and **S**ynthesis)

[2] This type of style denotes an abstract object like an algorithm, task or a processing element.

using a processor $\mathcal{P}$. To put it another way, τ denotes the time interval between i) reading the first input value and ii) producing the last output value.

- PIPELINE INTERVAL α
  In contrast to this, the pipeline interval α reflects the ability of a processor in terms of pipelining, when the same algorithm $\mathcal{ALGO}$ is computed repeatedly by a processor $\mathcal{P}$, each time with a new input data set. Thus, α denotes the number of clock cycles after which a new invocation of the algorithm with a new input data set can be computed again.

For the sake of clarity, these performance measures are again visualized in Figure 4.

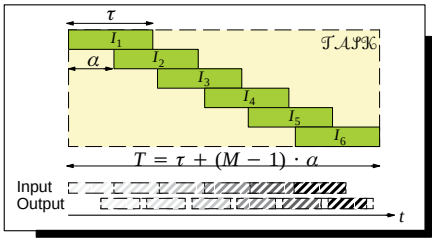

Fig. 4: Overlapped and pipelined Execution

$I_1, \ldots, I_6$ represent consecutive invocations of the same algorithm, but with a new input data set. The reading of the input and the writing of the output data blocks are marked as dashed rectangles in the timing diagram.

Besides these performance parameters, describing a processor $\mathcal{P}$ in terms of computation time and pipeline interval, the most relevant expense parameter is the silicon area of a processor alternative:

- AREA EXPENSE A
  The area expense A is an estimate in terms of transistor expense or silicon area, respectively. It can be estimated considering the particular areas of the ALUs, registers, etc. Their area expense has to be accumulated bottom-up along the architecture tree. So, the effort for the calculation of the area expense is relatively low.

As mentioned before, in case of dedicated (array) processors, the most important unit is the processing unit, consisting of an array of regular processing nodes or elements, $\mathcal{PE}$ in short terms. In contrast to this, for a programmable processor the processing part, the memory part, the controlling part, as well as the interconnection network are important hardware units in terms of the overall area expense.

To characterize any processor alternative, we are now tackled with the problem of estimating these performance and expense measures $\mathcal{M} = \{\tau, \alpha, A\}$ for different dedicated and programmable modules. Clearly, these measures have to be estimated prior to the overall optimization. Nevertheless, from an estimation point of view it is reasonable to distinguish between dedicated and programmable processors, as described next.

2.2.1 Dedicated Processors

In case of dedicated (array) processors, which are preferably suitable for the implementation of regular low-level algorithms, the performance and expense measures can be even calculated deterministically. Opposite to programmable processors, there is no estimation required. This is due to the fact, that in case of regular low-level algorithms performance measures like computation time τ and pipeline interval α only depend on the algorithm class $\mathcal{ALGO}$ and its characteristics like the total number of operations, the maximum achievable concurrency of operations, etc.

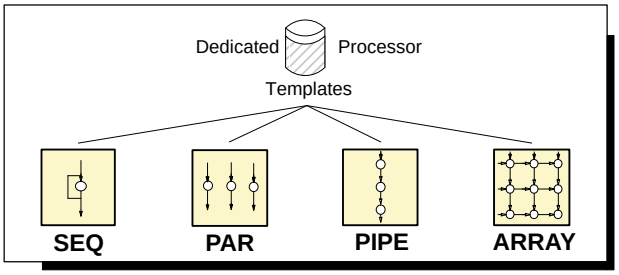

Fig. 5: Dedicated Processor Types

Nevertheless, in order to restrict the possibly large design space to a finite set of dedicated architectural alternatives, a processor library with four different dedicated processor types for each (low-level) algorithm class was developed. The library contains different *templates* of processor modules, whereas the templates differ with respect to their ability of computational concurrency, i.e. pipelining and parallel processing. Fig. 5 shows the different dedicated processor types, called SEQ, PAR, PIPE, and ARRAY. For example, SEQ denotes a processor type, performing all operations of the algorithm sequentially with one dedicated $\mathcal{PE}$, leading to a low area expense at the cost of a large execution time. In contrast to this, the processors of type PAR and PIPE provide the possibility to execute the operations of the algorithm in parallel or in a pipelined mode. The processor of type ARRAY provides both parallel and pipelined execution of operations by a two dimensional grid of processing elements. Consider, for example a filtering algorithm 2D-FIR with an image block size of 8×8 pixels and a kernel window of 3×3 filter coefficients, i.e. $\Theta = \{T_W, T_H, K_W, K_H\} = \{8, 8, 3, 3\}$.

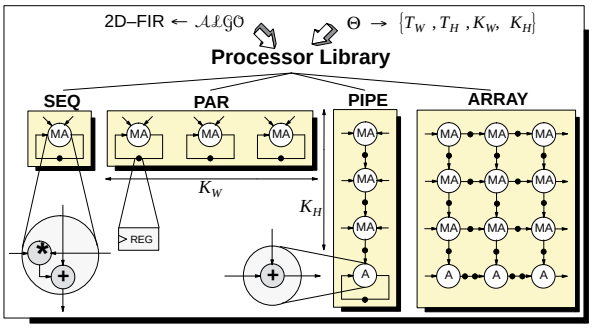

Fig. 6: Dedicated Processor Types for 2D-FIR Filtering

Then, the alternative dedicated architectures of

type SEQ, PAR, PIPE, and ARRAY are sketched as schematic on a register transfer level in Fig. 6 with the two different $\mathcal{PE}$s (MA, A) emphasized, whereas MA stands for multiplication and addition, and A stands for addition.

It can be shown, that the performance attributes of the architecture type SEQ, PAR, PIPE, and ARRAY strongly related to the parameter set Θ, i.e. T_W, T_H, K_W, K_H. This is also true for the expense measure, e.g. silicon area A. So, the performance and expense measures of such dedicated architectures can always be calculated deterministically with respect to the class $\mathcal{ALGO}$ and the associated parameter set Θ.

2.2.2 Programmable Processors

Considering the flexiblity an algorithm can be executed on a programmable processor, the evaluation of the total execution time T is much more difficult, than on dedicated processors. In case of programmable hardware, for an exact evaluation of the execution time a complete simulation on register transfer level is necessary. This leads to a high effort, especially, at an early design-stage. Furthermore, the programm instructions have to be known. Due to these facts, we use an estimation approach, which is based on the algorithm modelling and on the knowledge concerning clock cycles for basic operations and memory accesses of a programmable processor.

For our estimation approach the detailed instruction or programm code of the processed algorithm is not required. This leads to a moderate effort for modelling the execution time of one algorithm on a programmable hardware. The strategy is as follows: an assignment has to be performed, to determine the way an algorithm, for example the block matching algorithm three step search ($3SS$), is computed by a module (see Fig. 2). To evaluate the time for arithmetical and logical operations of the algorithm, those operations are matched to suitable PROC-modules. Furthermore, necessary clock cycles for memory accesses are considered, if the operands and results of each operation are assigned to some proper MEM-modules, too.

Considering the supposition that the instruction code is not taken into account, the actual approach for estimating the execution time does not consider the controlling. This simplification is admissible, because controlling mainly occurs simultaneously to the computation and the data input- and output-transfer. Then, the execution time T of an algorithm is composed of the time for execution of basic operations T_{op} and the communication time $T_{I/O}$. The specific time intervall T_{op} is evaluated with regard to the number of clock cycles for one operation on the assigned module, the number of operations, and possible parallel execution. In contrast to that, $T_{I/O}$ depends on the access time T_{acc} for the first address of a data block and the transfer time $T_{transfer}$ which denotes the number of clock cycles to transfer the rest of the data block from memory into a module or vice versa. This leads to $T_{I/O} = T_{acc} + T_{transfer}$. The transfer time itself depends on the size of the data blocks and on the bandwidth between the PROC- and the MEM-modules or between different MEM-modules (e.g. 1.- and 2. level cache). To calculate the total execution time T, it should be mentioned, that operations and I/O occured concurrently. Therefore, an overlapping time $T_{overlap}$ is introduced, leading to $T = T_{op} + T_{I/O} - T_{overlap}$. Opposite to T_{op} and $T_{I/O}$ the exact evaluation of $T_{overlap}$ supposes the knowledge at which time a computation and communication process begins and at which time it ends. Due to the fact, that a necessary scheduling could not be specified at an early design stage, we developed a method [7] to estimate the time $T_{overlap}$ and the total execution time T as accurate as possible avoiding scheduling techniques.

The evaluation of T starts at the lowest level of the algorithm tree, which means T is evaluated for each single operation, first. Then these time durations are taken into account to evaluate the execution times of the upper tasks. The evaluation repeats along the algorithm tree from bottom to top until the top task-node attains. To model possible access conflicts on modules, possible miss penalties for memory accesses, etc., additional runs are necessary. In order to derive T in each run, the execution time is recalculated for each task of the algorithm-tree.

2.3 Optimization

Finally, after the possible design space for a composite video signal processing scheme has been explored based on a finite set of dedicated and programmable processors, it is necessary to select from the large amount of alternative modules the most suitable combination for an optimal overall multiprocessor system. In order to judge a certain solution, the performance and expense measures, which were derived during the estimation step, are used as input to the optimization procedure. Furthermore, the cost function of the optimization procedure has to take into consideration all those parameters, having an impact on the achievable performance in terms of computation time T and periodic computation time P as well as on the hardware expense in terms of silicon area A. Based on the cost function above, the optimization procedure has to determine which is the most suitable processor type for any task, and which is the best temporal order concerning task execution and data transfer. It can be shown, that this leads to a combinatorial optimization problem. One basic approach using mixed integer linear programming (MILP) in order to tackle this kind of optimization problems can already be found in [8]. But we extended these models by inclusion of 1) periodic and overlapped processing of tasks and 2) a parametrizable library with different dedicated and programmable processor alternatives. For more details concerning our MILP model the interested reader is referred to [9].

Nevertheless, in order to solve our optimization problem using MILP, an efficient solution strategy is necessary. Branch-and-bound is known to be suitable for solving an ILP or a mixed ILP. Therefore, we have implemented a specific branch-and-bound strategy in Fortran 77 for our MILP formulation, by adapting the

BLAS[3] routines as part of the LAPACK[4] package to our needs. Futhermore, for the sake of flexibility, we developed a special parser, which reads a textual description of an MILP and automatically generates all matrices and vectors needed by the LAPACK routines.

3 Case Study: Video Encoding

To demonstrate the feasibility of the proposed system level methodology for a typical DSP system, the encoder of the hybrid video codec scheme H.261 [1] was chosen.

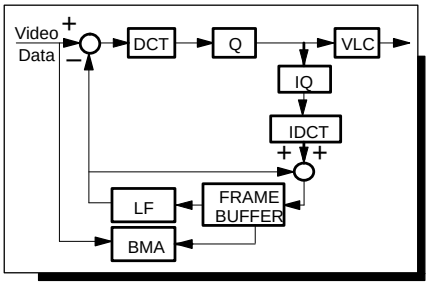

Fig. 7: Video Encoder Scheme for H.261

The H.261 scheme is used for data reduction necessary to transmit video data on a line with $p \times 64$kbit/s ($1 \leq p \leq 30$), e.g. video telephone, video conferencing, or even multimedia. H.261 consists of several regular low-level tasks as well as irregular medium level tasks, see Fig. 7. As architectural example we chose the programmable video signal processor AxPe640V [3] and additional dedicated processor models as base for our optimization. The AxPe640V consists of a RISC kernel and a more dedicated coprocessor named Low Level Coprocessor (LCP). The AxPe640V is able to perform the coding of the H.261 scheme in real time, at a maximum frame rate of 10 Hz. Nevertheless, the question arises if an improvement in terms of throughput rate of the AxPe640V is possible by an extension with dedicated modules.

In the following sections we model the algorithm tree as well as the architecture tree, first. Then the results of the estimation approach for dedicated processor modules and the AxPe640V are shown. Finally, the actual optimization results are presented.

3.1 Modelling

Applying our system level partitioning and optimization tool means to start from the modelling. The modelling consist i) of the hierarchical algorithm tree, denoting the composite video signal processing scheme (H.261) on one side, and ii) of the hierarchical description of the available processor modules on the other side. Concerning the specification of the architecture tree (Fig. 8), we decided to present only the tree for the AxPe640V programmable video signal processor. For the sake of simplicity, the architecture trees of all dedicated processor modules are not presented here.

3.2 Estimation

Based on the set of dedicated modules presented earlier as well as on the programmable video signal processor AxPe640V, we estimated the performance and expense measures for all processor modules, which are currently supported. Concerning the computation time τ measured in number of clock cycles, the results of the estimation for FIR, DCT, and IDCT are shown in Fig. 9.

[3]BLAS: Basic Linear Algebra Subprograms
[4]LAPACK: Linear Algebra PACKage

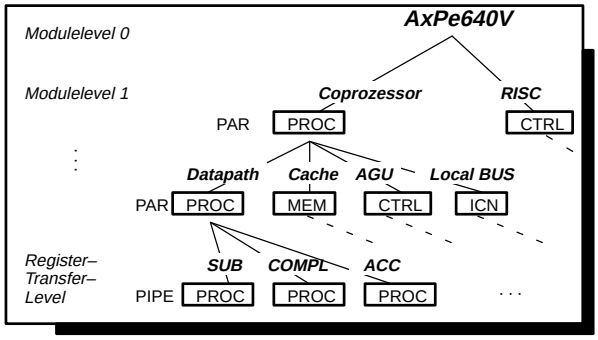

Fig. 8: The AxPe640V as Architecture Tree

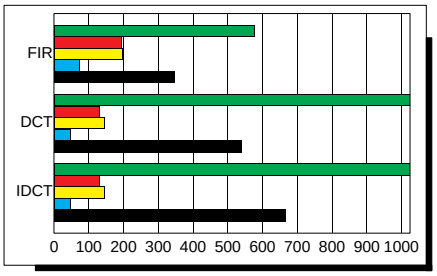

Fig. 9: Estimated Computation Time τ

The length of the different bars corresponds to the number of clock cycles, when executing the algorithm on the related architecture. For each algorithm or row of the chart, the four top bars correspond to the execution time of the dedicated modules, i.e. SINGLE, PAR, PIPE, and ARRAY. The last bar of each row corresponds to the estimated execution time for the AxPe640V. The accuracy of the execution time estimation for the programmable module various from 1% up to 7%. This demonstrates, that our estimation approach is suitable to evaluate the execution time for programmable processors.

3.3 Optimization

Finally, the optimization was carried out. Based on the estimation step, the MILP model was derived, leading to 117 variables and 208 restrictions, which is of rather moderate size for any LP solver. We experienced, that optimal solutions could be derived within a few CPU seconds. In order to direct the search during the optimization, we assumed four different design priorities with respect to the weights of the system parameters A, T, P, Y.

3.3.1 Results

In a first approach, we examined if it is reasonable to improve the given programmable architecture module (AxPe640V) by additional dedicated modules, as mentioned before. Herein, our goal was to derive a heterogeneous system primarily with minimum area

expense A and additionally small latency T and computation period P, i.e. $A \succ T, P \succ Y$.

Based on these assumptions, the optimization was carried out, providing the partitioning and the task scheduling (see Fig. 10).

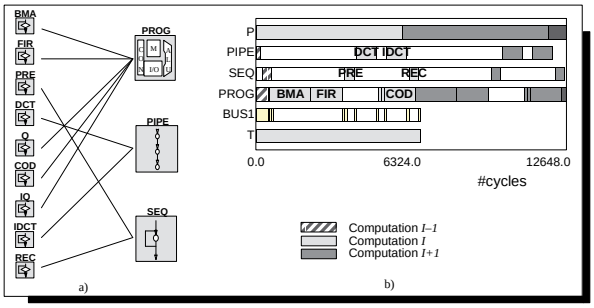

Fig. 10: a) Partitioning b) Task Scheduling

The derived heterogeneous system consists of the programmable module AxPe640V, and furthermore two dedicated modules of type SEQ and PIPE. As can be seen, the medium-level tasks as well as some of the low-level tasks are mapped onto the AxPe640V. This example reveals, that additional dedicated modules improve the performance in terms of throughput rate of the overall system, with a small area overhead ($\Delta A = 3\%$), compared to a single AxPe640V solution.

4 Implementation

The presented system level partitioning and optimization tool is currently under development as a protoype CAD implementation in COMMON LISP/CLOS [5] on Sparc Stations. First of all the set of algorithms for the composite image processing scheme can be selected. Then, the set of tasks, their data- and control dependencies are specified by the user. Afterwards, the estimation is performed, calculating performance and expense measures. Finally, the optimization is performed. The results of the optimization can be analyzed using a graphical user interface (GUI), based on CLUE/CLX [6].

5 Conclusion

In this paper, a system level HW/SW partitoning methodology and its implementation as the CAD tool HMOPS for the optimization of heterogeneous multiprocessor systems has been presented. Starting from a hierarchical modelling of algorithms and architectures, an approach for estimating performance and expense measurements, like execution time and processor area was developed. Based on these measurements, it becomes possible to optimize heterogeneous multiprocessor systems for composite DSP schemes with respect to maximal throughput rate and minimal area expense.

Acknowledgements

The work presented is supported by the Deutsche Forschungsgemeinschaft, DFG, under contract numbers Pi–169/4 and Pi–169/5. We are grateful to H. Jeschke for providing us with his library as part of the performance estimation approach.

[5] CLOS Specification, ANSI, X3J13

[6] Common Lisp User Interface Environment/X

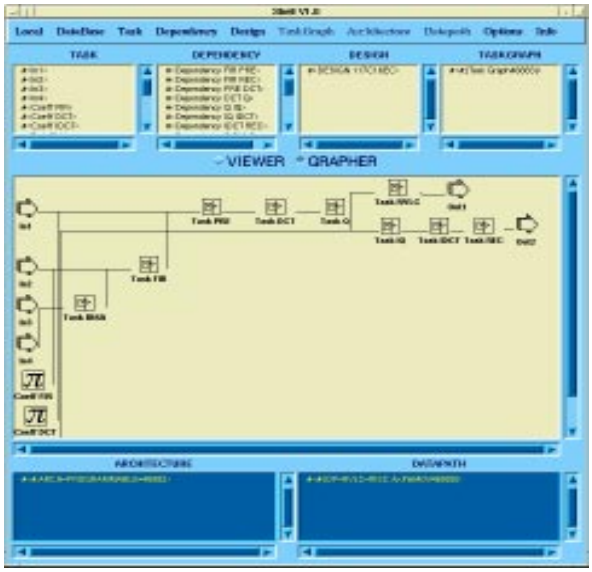

Fig. 11: HMOPS Graphical User Interface (GUI)

References

[1] ITU-T Recommendation H.261, "Video Codec for Audiovisual Services at p×64 kbit/s", 1993.

[2] ISO–IEC IS 13818, "Generic coding of moving pictures and associated audio", 1994.

[3] Philips, "AxPe640V Programmable Video Signal Processor", Preliminary Specification, Vol. 1-2, 1994.

[4] J. Rabbaey, H. de Man, J. Vanhoof, G. Goosens, F. Cathoor, "CATHEDRAL II: A Synthesis System for Multiprocessor DSP Systems", in *Silicon Compilation*, Addison-Wesley, 1988, pp. 311-360.

[5] J.L. Pino, S. Ha, E.A. Lee, J.T. Buck, "Software Synthesis for DSP Using Ptolemy", E.E. Swartzlander, jr., and S.Y. Kung, ed., *Journal of VLSI Signal Processing*, Kluwer Academic Publishers, vol. 9, no. 1-2, 1995, pp. 7-21.

[6] R. Lauwereins, M. Engels, M. Ade, J.A. Peperstraete, "Grape-II: A System-Level Prototyping Environment for DSP Applications", *IEEE Computer*, IEEE CS Press, vol. 28, no. 2, 1995, pp. 35-43.

[7] H. Kropp, M. Schwiegershausen, P. Pirsch, "A CAD Tool for the Optimization of Video Signal Processor Architectures", *Proc. ICASSP '96*, 1996, Vol. 2, pp. 1244-1249.

[8] S. Prakash, A.C. Parker, "Synthesis of Application Specific Heterogeneous Multiprocessor Systems", *J. of Parallel & Distributed Computing*, no. 16, 1992, pp. 338-351.

[9] M. Schwiegershausen, M. Schönfeld, P. Pirsch, "Mapping complex image processing algorithms onto heterogeneous multiprocessors regarding architecture dependent parameters", in *Algorithms and Parallel VLSI Architectures III*, M. Moonen, F. Catthoor, editors, Amsterdam: Elsevier Science, Chapter 30, 1995, pp. 353-364.