An Evolutionary Approach to System-Level Synthesis

Jürgen Teich, Tobias Blickle, Lothar Thiele

***Abstract*— In this paper, we consider *system- level synthesis* as the problem of optimally mapping a task-level specification onto a heterogeneous hardware/software architecture. This problem requires (1) the *selection of the architecture* (allocation) including general purpose and dedicated processors, ASICs, buses and memories, (2) the *mapping* of the algorithm onto the selected architecture in space (binding) and time (scheduling) and (3) the *design space exploration* with the goal to find a set of implementations that satisfy a number of constraints on cost and performance. Here, a new *graph-based mapping* model is introduced to specify the task of system-synthesis as an optimization problem. An *Evolutionary Algorithm* is adapted to solve this problem and is applied to explore the design space of video-codec implementations.**

I. Introduction

A. Optimization Methodology

The proposed optimization methodology treats the problem of optimizing the mapping of a large-grain dataflow graph-based specification onto a heterogeneous hardware / software architecture. This problem requires (1) the *selection of the architecture* (allocation) among a specified set of possible architectures, (2) the *mapping* of the algorithm onto a selected architecture in space (binding) and time (scheduling), and (3) the design space exploration with the goal to find the set of optimal implementations that satisfy a number of constraints on cost and performance.

Our approach provides a) *a new system-level algorithm/ architecture model* for heterogeneous hardware/software systems, and b) applies Evolutionary Algorithms to system-level synthesis showing that they can perform steps (1)-(3) in a single optimization run. Evolutionary Algorithms turn out to be a good candidate for system-level synthesis, because they a) iteratively improve a *population* (set) of implementations, b) they do not require the quality (cost) function to be linear (e.g., area-time product). In case *Pareto-ranking* is used as the fitness function, the design spaces can be explored in a single optimization run. Finally, c) they are known to work "well" on problems with large and non-convex search spaces, and d) they require a reasonable amount of run-time due to their heuristic nature.

B. Existing Approaches to System-Level Synthesis

There exist already many different approaches to system-level synthesis. Some approaches focus on *control-dominant systems*, e.g., [9], [6], and [10]. Other approaches deal with *dataflow-dominant designs* like [1], and [8]. The term system synthesis, however, is weakly defined. In some approaches, the result of the synthesis procedure is the description of a *dedicated control & data path in VLSI*, e.g., [13], [12], [14] or a *multi-chip dedicated VLSI architecture*, e.g., [11]. In other cases, it is a mixed *hardware/software architecture.* In some of these approaches, the architecture is already fixed, see, e.g., [6], or [8] and the synthesis problem is the problem of partitioning functionality into hardware and software blocks, see, e.g., [3], [13], [12], [14]. Some approaches do not handle communication as part of the synthesis problem [4].

The authors are with the Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), CH-8092 Zürich, Switzerland, email: teich@tik.ee.ethz.ch. In Proc. 5th Int. Workshop on Hardware/Software Codesign, braunschweig, Germany, March 1997,pp. 167-171, © IEEE.

II. The Problem of System Synthesis

A. Graph-Based Specification

Definition 1 (Specification Graph) *A* specification graph *is a graph* $G_S = (V_S, E_S)$ *consisting of* D *dependence graphs* $G_i(V_i, E_i)$ *for* $1 \leq i \leq D$ *and a set of* mapping edges E_M. *In particular,* $V_S = \bigcup_{i=1}^{D} V_i$, $E_S = \bigcup_{i=1}^{D} E_i \;\cup\; E_M$ *and* $E_M = \bigcup_{i=1}^{D-1} E_{Mi}$, *where* $E_{Mi} \subseteq V_i \times V_{i+1}$ *for* $1 \leq i < D$.

Consequently, the specification graph consists of several layers of dependence graphs and mapping edges which relate the nodes of two neighboring dependence graphs. These layers correspond to levels of abstractions, for example algorithm description (problem graph), architecture description (architecture graph) and system description (chip graph). The edges represent user-defined mapping constraints in the form of a relation: 'can be implemented by'.

Example 1 *Fig. 2 shows an example of a specification graph derived from a given problem graph (Fig. 1a)), an architecture graph (Fig. 1c)) derived from Fig. 1b) and and a chip graph (Fig. 1e) derived from Fig. 1d). The edges between the two subgraphs are the additional edges* E_{M1} *and* E_{M2} *that describe all possible mappings. For example, operation* v_1 *can be executed only on* v_{RISC}. *Operation* v_2 *can be executed on* v_{RISC} *or* v_{HWM2}. *The role of dotted arcs and nodes is explained later. Communication* v_7 *can be executed by* v_{BR1} *or within* v_{RISC} *or* v_{HWM1}. *The specification allows the* RISC *processor, the hardware modules* HWM1, HWM2 *and the communication modules* BR1, BR2 *to be implemented in* CHIP1. *The communication* BR1 *can either be handled by* CHIP1 *or by the off-chip bus* OCB.

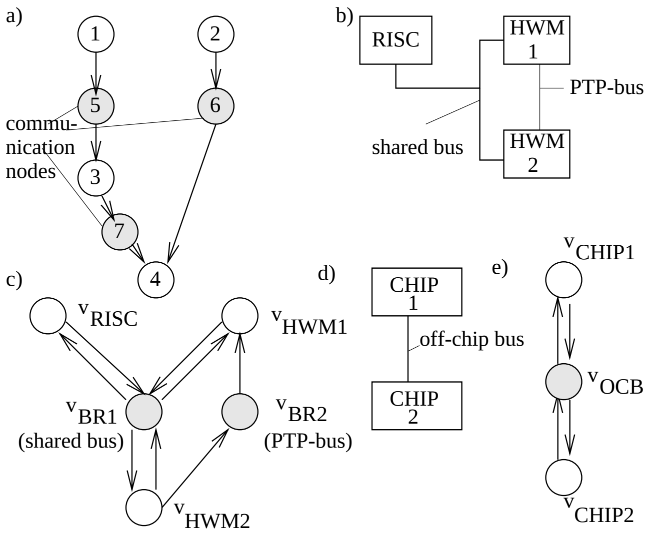

Fig. 1. Example of a problem graph, an architecture graph and a chip graph

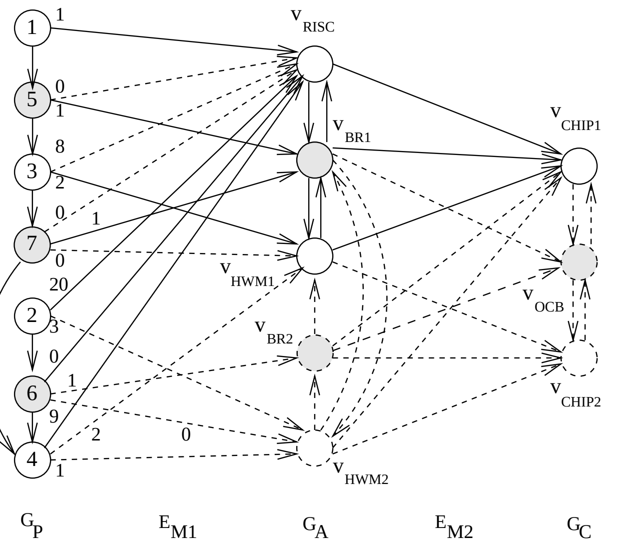

Fig. 2. An example of a specification graph.

Hence, a specification graph allows a flexible expression of the expert knowledge about useful architectures and mappings. In order to describe a concrete mapping, i.e., an *implementation*, the term *activation* of nodes and edges of a specification graph is defined. Based on this, *allocation*, *binding* and *scheduling* will formally be defined in the next section.

Definition 2 (Activation) *The* activation *of a specification graph $G_S(V_S, E_S)$ is a function $a : V_S \cup E_S \mapsto \{0, 1\}$ that assigns to each edge $e \in E_S$ and each node $v \in V_S$ the value* 1 *(*activated*) or* 0 *(*not activated*).*

The activation of a node or edge of a dependence graph describes its use. In the examples presented so far, all nodes and edges of the problem graph contained in G_S have been selected, i.e., activated. The determination of an implementation can be seen as the task of assigning activity values to each node and each edge of the architecture graph and/or chip graph. An activated mapping edge represents the fact that the source node is implemented on the target node.

B. System Synthesis

Definition 3 (Allocation) *An* allocation *α of a specification graph is the subset of all activated nodes and edges of the dependence graphs, i.e., $\alpha = \alpha_V \cup \alpha_E$ with $\alpha_V = \{v \in V_S | a(v) = 1\}$ and $\alpha_E = \bigcup_{i=1}^{D} \{e \in E_i | a(e) = 1\}$*

Definition 4 (Binding) *A* binding *β is the subset of all activated mapping edges, i.e., $\beta = \{e \in E_M | a(e) = 1\}$.*

Definition 5 (Feasible Binding) *Given a specification G_S and an allocation α. A* feasible binding *β is a binding that satisfies the following criteria: a) Each activated edge $e \in \beta$ starts and ends at an activated node, i.e. $\forall e = (v, \tilde{v}) \in \beta : v, \tilde{v} \in \alpha$, b) for each activated node $v \in \alpha_V$ with $v \in V_i$, $1 \le i < D$ exactly one outgoing edge $e \in V_M$ is activated, i.e. $| \{e \in \beta \,| e = (v, \tilde{v}), \tilde{v} \in V_{i+1}\} | = 1$, and c) for each activated edge $e = (v_i, v_j) \in \alpha_E$ with $e \in E_i$, $1 \le i < D$ either both operations are mapped onto the same node, i.e. $\tilde{v}_i = \tilde{v}_j$ with $(v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$ or there exists an activated edge $\tilde{e} = (\tilde{v}_i, \tilde{v}_j) \in \alpha_E$ with $\tilde{e} \in E_{i+1}$ to handle the communication associated with edge e, i.e. $(\tilde{v}_i, \tilde{v}_j) \in \alpha_E$ with $(v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$.*

The calculation of a feasible binding and therefore the test of an allocation for feasibility is hard. This result will influence the coding of allocation and binding in the EA.

Theorem 1 *The determination of a feasible binding, is NP-complete.*

Proof: By polynomial reduction of BOOLEAN SATISFIABILITY, see [2]. ∎

Finally, it is necessary to define a *schedule*. Here, we use the execution time delay $delay(v, \beta)$ of an operation associated to node v of a problem graph G_P. Note that the execution time depends on a particular binding β.

Definition 6 (Schedule) *Given specification G_S containing a problem graph $G_1 = G_P$, a feasible binding β, and a function delay which determines the execution time $delay(v, \beta) \in \mathbb{Z}^+$ of a node $v \in V_P$. A* schedule *is a function $\tau : V_P \mapsto \mathbb{Z}^+$ that satisfies for all edges $e = (v_i, v_j) \in E_P$: $\tau(v_j) \ge \tau(v_i) + delay(v_i, \beta)$.*

Now, given a specification graph G_S a *(valid) implementation* is a triple (α, β, τ) where α is an allocation, β is a feasible binding, and τ is a schedule.

Example 2 *Fig. 2 shows an implementation of the specification shown in the same figure. The nodes and edges not allocated are shown dashed, as well as the edges $e \in E_M$ that are not activated. The allocation of nodes is $\alpha_V = \{v_{RISC}, v_{HWM1}, v_{BR1}, v_{CHIP1}\}$ and the binding is $\beta = \{(v_1, v_{RISC}), (v_2, v_{RISC}), (v_3, v_{HWM1}), (v_4, v_{RISC}), (v_5, v_{BR1}), (v_6, v_{RISC}), (v_7, v_{BR1}), (v_{RISC}, v_{CHIP1}), (v_{BR1}, v_{CHIP1}), (v_{HWM1}, v_{CHIP1})\}$. This means that all architecture components are bound to CHIP1. A schedule is*

$\tau(v_1) = 0$, $\tau(v_2) = 1$, $\tau(v_3) = 2$, $\tau(v_4) = 21$, $\tau(v_5) = 1$, $\tau(v_6) = 21$, $\tau(v_7) = 4$.

C. The Tasks of System Synthesis

With the model introduced above the task of system synthesis can be formulated as an optimization problem.

Definition 7 (System Synthesis) *The task of* system synthesis *is the following optimization problem:*

minimize $h(\alpha, \beta, \tau)$,
subject to

α *is a feasible allocation,*
β *is a feasible binding,*
τ *is a schedule, and*
$g_i(\alpha, \beta, \tau) \geq 0,\ i \in \{1, \ldots, q\}$.

The constraints on α, β and τ define the set of valid implementations. Additionally, there are functions $g_i, i = 1, \ldots, q$ that together with the objective function h describe the optimization goal.

III. Optimization Method and Design Space Exploration

In this section, the application of an Evolutionary Algorithm (EA) is described to solve the problem of system synthesis as specified by Definition 7. An EA works on *populations of individuals* J_i, $i = 1, \cdots, N$ where N is called *size of the population*, and each individual codes an implementation of the problem graph including an architecture and a mapping of nodes in the problem graph in space (binding) to that architecture. The EA consists of an optimization loop that applies the principles of *reproduction*, *crossover* and *mutation* to the strings that code implementations. The purpose is to iteratively find better populations: Each individual in an actual population P_k is ranked by evaluation of a *fitness function* that gives a measure how good an implementation is in terms of cost and performance, etc. The EA terminates after a certain number k_{max} of generated populations and outputs those implementations with the best fitness values.

In our case, the EA is responsible for the determination of the allocation and of the binding. As there are good heuristics known for scheduling, it is not necessary to load the EA with this task. This is why the schedule for the fixed architecture obtained by the EA is computed by a standard heuristic scheduler. This division of work is depicted in Fig. 3.

A. Coding of Implementations

A specification graph G_S may consist of D subgraphs $G_i, 1 \leq i \leq D$ corresponding to $D-1$ mapping tasks. In order to handle these mapping tasks the allocation and binding of the levels is done sequentially from level 1 to level $D-1$. In each level the following steps are executed: First, the allocation of nodes V_{i+1} is decoded from the individual, next the binding of the edges $e \in E_{Mi}$ is performed. In Fig. 4, the encoding of an allocation in the chromosome of an indivudual is shown.

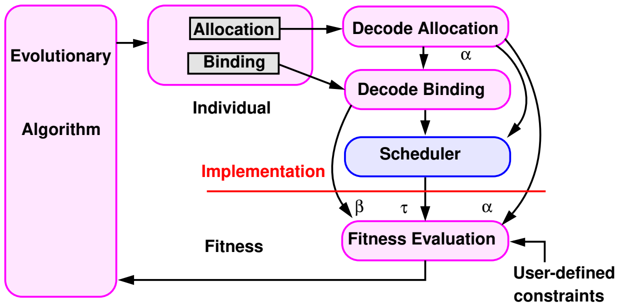

Fig. 3. The decoding of an individual to an implementation.

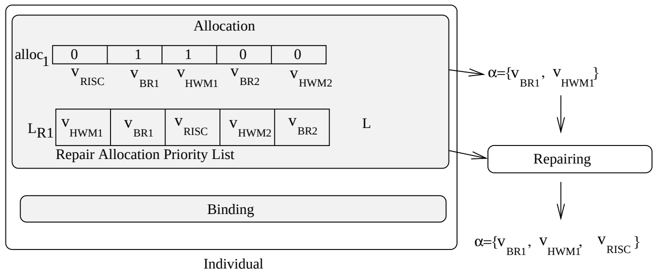

Fig. 4. The coding and decoding of an allocation

Now, if allocations and bindings are (at least in some sense) randomly chosen, many are infeasible (77% in the given example). Punishing these "bad" individuals with a penalty value would in certain cases not prevent a needle-in-the-haystack search of the EA. On the other hand, one could "repair" invalid allocations and bindings with some mechanism and incorporate domain knowledge in these repair mechanisms. However, the determination of a feasible allocation or binding is NP-complete, so that this would result in solving a NP-complete task for every individual to be repaired. These considerations lead to the following compromise: The randomly generated allocations of the EA are partially repaired using the following heuristic: For all nodes $v \in V_i : |\{\tilde{v} \in V_{i+1} | (v, \tilde{v}) \in E_{Mi} \wedge a(\tilde{v}) = 0\}| = 0$, one node $v \in V_{i+1}$ is added to the allocation in order to make this node bindable. The choice is determined by the first element of a priority list L_{Ri} that is not yet element of the current allocation (see, e.g. in Fig. 4 where v_{RISC} is added to α in order to make node v_1 and v_2 bindable). The lists L_{Ri} are also transformed by the EA to obtain good orders of what element to add to an allocation in case repairing is necessary. In the example, the number of infeasible allocations was reduced to 6.5%. Details on intelligent coding, binding and decoding mechanisms can be found in [2]. One iteration of the loop results in a feasible allocation and binding of the nodes and edges of G_i to the nodes and edges of G_{i+1}. If no feasible binding is found the decoding of the individual is aborted. The scheduler chosen here is a modified list scheduler, see, e.g., [5] that schedules the nodes of G_P and obtains the execution delays and resource bindings for each node from the EA as described above. The scheduler performs latency minimization under resource constraints. The scheduler returns the latency of the schedule and the start times of all nodes. Then, the complete fitness function can be evaluated for each individual.

B. EA Parameters

For design space exploration, the selection method should maintain a high diversity in the population. Therefore, we use *restricted tournament selection*. The specific encoding of an individual makes special crossover and mutation schemes necessary. In particular, for the allocation α, uniform crossover is used that randomly swaps a bit between two parents with a probability of 50%. For different priority lists, order based crossover is applied. The probability of crossover is 50%. A mutation of an allocation α consists in simply swapping the allocation bit with a probability of 50%. Mutation is applied to 20% of the individuals of a population.

C. The Choice of the Fitness Function

According to Definition 7, the system-synthesis task was described as an optimization problem. As the particular optimization goal depends on the specific design problem, the objective function h has to be designed individually for each problem. But the restriction to a feasible allocation and binding needs to be considered in any case. The fitness function f to be minimized is

$$f(J) = \begin{cases} x_a(J)p_a + x_b(J)p_b & : \quad x_a(J) = 0 \vee x_b(J) = 0 \\ h'(J) & : \quad else \end{cases} \tag{1}$$

The p values are the penalty terms, i.e., p_a is the penalty term for an infeasible allocation and p_b for an infeasible binding. The boolean variables x denote whether the corresponding constraint is violated or not, e.g., $x_a(J) = 0$ if the allocation is feasible and $x_a(J) = 1$ if the allocation is infeasible. The values for the penalty terms p_a and p_b should be chosen such that any infeasible allocation or binding has a worse fitness value than any feasible implementation. The modified objective function $h'(J)$ has to reflect the additional constraints g_i.

In system-synthesis, usually many different criteria have to optimized like for example cost, data-throughput, power consumption, and maintainability. The concept of *Pareto-optimality* gives a measure of concurrently comparing implementations to several criteria.

Definition 8 *A point J_i is* dominated *by point J_k if point J_k is better than point J_i on all criteria, denoted as $J_i \succ J_k$. A point (implementation) is said to be* Pareto-optimal *if it is not dominated by any other point.*

In [7], a Pareto-ranking scheme is proposed for multimodal optimization. We obtain

$$h'(J) = \sum_{i=1,\ldots N} \begin{cases} 1 & : \quad J_i \prec J \\ 0 & : \quad else \end{cases} \tag{2}$$

All Pareto-points in the population have an (optimal) fitness of zero. Note that the fitness of an individual depends on the population.

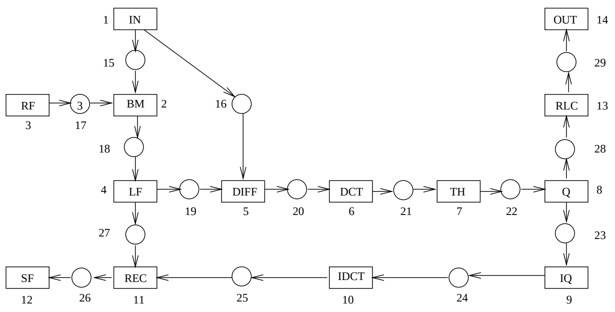

Fig. 5. Problem graph of the video coder. Functional nodes are symbolized by squares, communication nodes by circles.

Example 3 *As an example for Pareto-optimization consider the case of two-dimensional optimization with the criteria cost $c(\alpha, \beta)$ and period $P(\tau)$. One obtains as fitness function:*

$$h'(J) = \sum_{i=1,\ldots,N} \begin{cases} 1 & : \quad (c(J) > c(J_i) \wedge P(J) \geq P(J_i)) \\ 1 & : \quad (c(J) \geq c(J_i) \wedge P(J) > P(J_i)) \\ 0 & : \quad else \end{cases} \tag{3}$$

Here, $c(J)$ is used as an abbreviation of $c(\alpha(J), \beta(J))$ and the dependence of α and β on the individual J is explicitly denoted by $\alpha(J)$, $\beta(J)$. Similar $P(J)$ stands for $P(\tau(J))$.

IV. Case Study

We explain our methodology using the example of a video codec for image compression using the H.261 standard. The problem graph of the coder is shown in Fig. 5. The synthesis problem is restricted to single level of hierarchy, i.e., only the mapping of the problem graph G_P to the architecture graph G_A is examined. Motion estimation is done by the block matching operation (BM), the block subtraction is named DIFF (difference) and the block addition REC (recover). Additionally, the quantization is split up into threshold calculation (TH) and quantization (Q).

The graph and a similar graph for the decoder (not shown here) is mapped onto a target architecture shown in Fig. 6. The architecture consists of three shared bus with different data rates, two memory modules (a single and a dual ported memory), two programmable RISC processors, a signal processor (DSP), several predefined hardware modules (namely a block matching module (BMM), a module for performing DCT/IDCT operations (DCTM), an subtract/adder module (SAM) a Huffman coder (HC), and I/O devices (INM and OUTM).

Possible mappings and delays of all modules are shown in Table 1. Due to space requirements, the mappings of communication nodes to resources are not specified here.

The number of possible bindings are $1.9 \cdot 10^{27}$. Hence, the intractability of enumerative or exact methods to explore the search space.

The cost of an implementation is simply reflected by the sum of the costs of the allocated hardware components. The cost of each module is given in brackets in Fig. 6.

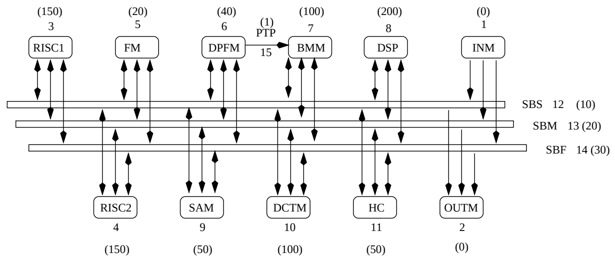

Fig. 6. Architecture template of architectures to implement the problem graph of the video codec. The values in brackets give the basic cost c_b of each resource.

Node #	Operation	Resource/ Delay	Resource/ Delay	Resource/ Delay
1	IN	INM/0		
31	IND	INM/0		
14	OUT	OUTM/0		
36	OUTD	OUTM/0		
2	BM	BMM/22	DSP/60	RISC/88
3	RF	FM/0	DPFM/0	
37	RFD	FM/0	DPFM/0	
12	SF	FM/0	DPFM/0	
39	SFD	FM/0	DPFM/0	
4	LF	HC/2	DSP/3	RISC/9
38	LFD	HC/2	DSP/3	RISC/9
5	DIFF	SAM/1	DSP/2	RISC/2
6	DCT	DCTM/2	DSP/4	RISC/8
10	IDCT	DCTM/2	DSP/4	RISC/8
34	IDCTD	DCTM/2	DSP/4	RISC/8
7	TH	HC/2	DSP/8	RISC/8
8	Q	HC/1	DSP/2	RISC/2
9	IQ	HC/1	DSP/2	RISC/2
33	IQD	HC/1	DSP/2	RISC/2
11	REC	SAM/1	DSP/2	RISC/2
35	RECD	SAM/1	DSP/2	RISC/2
13	RLC	HC/2	DSP/8	RISC/8
32	RLD	HC/2	DSP/8	RISC/8

Table 1. The mapping of the functional nodes to architectural nodes.

Now, we aim to perform a design space exploration. For this purpose, the Pareto-ranking function introduced in Example 3 is used. The fitness value directly gives the number of implementations that dominate an individual J. With a population size of $N = 100$, the Pareto-set found in a single optimization run after 200 generations is shown in Table 2 and depicted in Fig. 7. In general, there is no evidence that the Pareto-points obtained above are the true Pareto-points of the problem. Surprisingly, all points in Table 2 turn out to be true Pareto-points of the problem. Fig. 8 shows the architecture corresponding to the fastest design point found. The minimal period of the implementation is obviously determined by the execution time of the block matching module (BMM).

V. Conclusions

The optimization methodology presented here is part of the CodeSign framework currently in development. The framework uses object-oriented high-level Petri-nets as the kernel language for the specification of heterogeneous systems. The complete framework including also interfaces to common design languages and code generation tools will be described in a separate paper.

	J_1	J_2	J_3	J_4	J_5	J_6
Period P	22	42	54	78	114	166
Cost c	350	340	330	280	230	180

Table 2. Implementations obtained as Pareto-points in a single run of the Evolutionary Algorithm.

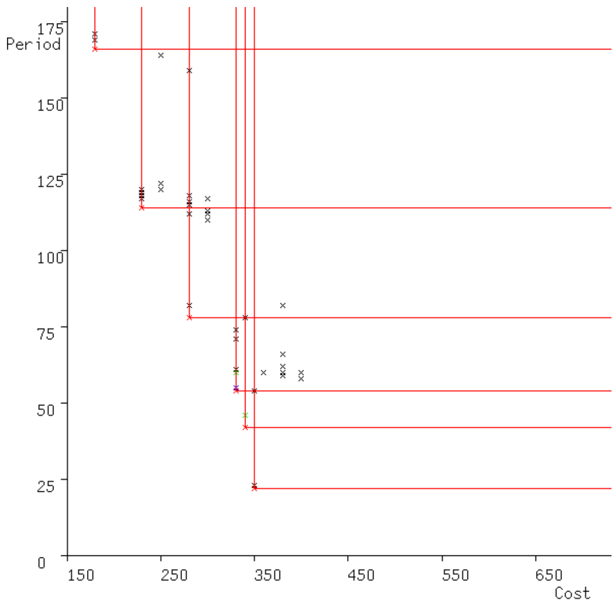

Fig. 7. Design space with Pareto-points of Table 2

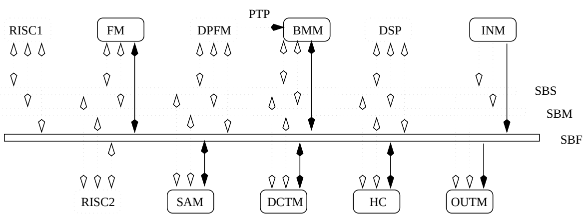

Fig. 8. Architecture of the fastest implementation

References

[1] E. Barros and W. Rosenstiel. A method for hardware software partitioning. In *Proc. 1992 COMPEURO: Computer Systems and Software Engineering*, pages 580–585, The Hague, Netherlands, May 1992.

[2] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. Technical Report 16, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, April 1996. http://www.tik.ee.ethz.ch/ Publications/ TIK-Reports/ TIK-Report16.ps.gz.

[3] R. Camposano and R. K. Brayton. Partitioning before logic synthesis. In *Proc. ICCAD*, 1987.

[4] J. G. D'Ambrosio and X. Hu. Configuration-level hardware/software partition for real-time embedded systems. In *Proc. of CODES/CASHE'94, Third Intl. Workshop on Hardware/Software Codesign*, pages 34–41, Grenoble, France, September 1994.

[5] G. DeMicheli. *Synthesis and Optimization of Digital Circuits.* Mc Graw Hill, New York, 1994.

[6] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, pages 64–75, December 1994.

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[8] R. Gupta and G. De Micheli. System-level synthesis using re-programmable components. In *Proc. of the European Conference on Design Automation (EDAC)*, pages 2–7, 1992.

[9] W. Hardt and R. Camposano. Specification analysis for hw/sw-partitioning. In *Proc. GI/ITG Workshop Application of formal Methods during the Design of Hardware Systems*, pages 1–10, Passau, Germany, March 1995.

[10] T. B. Ismail, K. O'Brien, and A. A. Jerraya. Interactive system-level partitioning with PARTIF. In *Proc. of the European Conference on Design Automation (EDAC)*, pages 464–473, 1994.

[11] K. Küçükçakar and A. C. Parker. A methodology and design tools to support system-level VLSI design. *IEEE Trans. on VLSI systems*, 3(3):355–369, September 1995.

[12] E. D. Lagnese and D. E. Thomas. Architectural partitioning for system level synthesis of integrated circuits. *IEEE Trans. on CAD*, 10(7):847–860, July 1991.

[13] M. C. McFarland. Using bottom-up design techniques in the synthesis of hardware from abstract behavioral descriptions. In *Proc. 23rd Design Automation Conference*, pages 474–480, June 1986.

[14] F. Vahid and D. Gajski. Specification partitioning for system design. In *Proc. 29th Design Automation Conference*, pages 219–224, Anaheim, CA, June 1992.