PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 46, NO. 3–4, PP. 123–136 (2002)

HIGH-LEVEL SYNTHESIS USING PREDEFINED IP-S

Péter ARATÓ, Tibor KANDÁR, Zoltán MOHR and Tamás VISEGRÁDY

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
H–1117 Budapest, Magyar tudósok körútja 2, Hungary

Received: June 6, 2003

Abstract

In this paper, an algorithm is presented for decomposing a system into IP (Intellectual Property) functional units. The system to be decomposed is characterized by a complete cover of the set of its behavioral datapath operations. Such a cover is obtainable at the allocation stage of a high-level synthesis procedure. Each block of the cover represents a subset of behavioral operations, which are non-concurrent, i.e., are executable by the same real resource (processor). Each IP – as a real resource – is assumed to be specified also by a subset of behavioral operations, the execution of which is possible and preferred by applying this IP. The quality constraints of the decomposition are handled as a weighted composition of several criteria, which may characterize a solution (degree of reuse, weighted sum of several cost parameters, etc.). Since the problem is NP-complete [7], the quality of the results is illustrated and evaluated on a widely used benchmark example of practical size.

Keywords: system-level synthesis, high-level synthesis, IP-based design, hardware/software codesign, reuse.

1. Introduction

The latest development in system-level synthesis and hardware/software codesign involves the need for methodologies using beneficially complex adaptable and reconfigurable functional units called IP-s (Intellectual Property) as building blocks [16], [17]. One of the most crucial steps of this design procedure is basically a special decomposition algorithm constructing an architecture from predefined IP-s and the communication between them [9], [12], [13]. IP vendors provide a growing variety of products specified in catalogues also on a behavioral level [16]. Although this specification is an exact definition of IP-behavior, it is not easy to use it in formulating a decomposition algorithm. Therefore, a proper transformation of the specification seems to be useful for direct interfacing to the preceding stages of system-level synthesis. These complex functional units are mostly communication interfaces (like serial UART, SPI, I2C, etc.), signal processing functions, (like FIR, IIR filter blocks, FFT/DCT transformers, Viterbi decoders, etc.), system level functions (like DMA controller, MMU, interrupt controller, etc.). A subclass of IP units can be considered and handled as complex units being able to perform a set of RTL level operations (like adding, shifting, XOR, storing, etc.). The algorithms presented in the paper focus on this subset of IP-s. Since the behavioral system

description generally starts with a dataflow graph or a high-level language representation [4], [5], [10], [3], [6], [8], therefore, so-called behavioral operations are always assumed as atomic behavioral units [14], [11], [7]. High-level synthesis steps yield proper subsets of non-concurrent behavioral operations by executing the scheduling and allocation algorithms [15], [1], [2], [3], [7]. These subsets are to be mapped in IP-s as real resources. Thus, it seems to be beneficial to specify each IP also by a subset of behavioral operations, the execution of which is possible and preferred by applying this IP. The decision on this executability is made by the designer upon considering first of all the suitability and adaptability of an IP [17], [16]. The requirements on speed, communication cost, complexity of control and reusability, etc. may set strict conditions and might exclude some behavioral operations from the subset of the preferred ones in spite of the adaptability of an IP [3], [7].

In this paper, an algorithm is presented for decomposing a system characterized by subsets of behavioral operations. These subsets represent a complete cover of all behavioral operations [11], [3], [7] of the system and the target architecture is to be constructed by applying executing IP-s selected from a predefined set of IP-s specified as outlined above. Since finding the optimal decomposition is an NP-complete problem [7], the quality of the results are illustrated and evaluated on a widely used benchmark example of practical size (MARS cipher) by constructing a weighted composition of several criteria, which may characterize a solution.

In Chapter 2, the algorithm DECIP (DEComposition into predefined IP-s) is described. In Chapter 3, the results are illustrated for a benchmark problem. The conclusions and some further research aims are summarized in Chapter 4.

2. The Algorithm DECIP

2.1. The Basic Problem to be Solved

Let a complete cover M be assumed on the set E of behavioral operations. The blocks (subsets) of such a cover are obtainable, for example, as the maximal compatibility classes of non-concurrent operations by scheduling and allocation from a high-level synthesis method [11], [3], [7]. Based on the cover, each behavioral operation is to be allocated in one of the IP-s from a predefined set I. In other words, a proper complete partition P on the set E of elementary operations is to be found starting from the cover M. An executing IP unit should be selected to each block of this partition P, under the assumptions and conditions as follow:

1. Each IP from a predefined set is specified by those behavioral operations, the execution of which is possible and preferred by this IP.
2. As few IP-s as possible are to be used from their predefined set.
3. As few as possible types of IP-s are to be used (the reuse of IP-s is preferred).
4. The criteria for selecting the executing IP-s should easily be combined and composed by applying weight factors.

2.2. *Notations for the Basic Algorithm*

$E : (e_1, \ldots, e_i, \ldots, e_N)$	Set of behavioral operations
$M : (M_1, \ldots, M_r, \ldots, M_k)$	Complete cover of E (e.g. the maximal compatibility classes obtained by the allocation)
$c(M_r)$	Relative cost of M_r
$I : (I_1, \ldots, I_s, \ldots, I_j)$	Set of IP-s predefined for application
$c(I_s)$	Relative cost of I_s
$R : (R_1, \ldots, R_s, \ldots, R_j)$	Set of those – not necessarily disjoint – subsets of behavioral operations in E, whose execution is possible and preferred by applying IP-s $I_1, \ldots, I_s, \ldots, I_j$, respectively
$S : (\ldots, I_h, \ldots, I_v, \ldots, I_q, \ldots)$	Set of executing IP-s selected for application
$n(I_s)$	Number of I_s copies selected for application
$R_{s,z}$	Disjoint subsets of R_s containing those elementary operations, for the execution of which the z^{th} copy of I_s is selected $((1 \leq s \leq j), (1 \leq z \leq n(I_s)))$
P	Executing partition on E, (the blocks are all subsets $R_{s,z}$)
W_{IPCost}	Relative weight factor constant for $c(I_s)$
W_γ	Relative weight factor constant for γ_s, where $\gamma_s = \sum_r \lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R$
W_{IPSort}	Relative weight factor constant for N_s, where $N_s = \begin{cases} 1, & \text{if } n(I_s) > 0 \\ 0, & \text{if } n(I_s) = 0 \end{cases}$
w_s	Weight function value for I_s

2.3. *Description of the Algorithm DECIP*

START
$\forall n(I_s := 0)$
$S := \varnothing$
while $M \neq \varnothing$
do {
$\alpha = \max\{\lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R\}$
for $\forall R_s$: $\gamma_s = \sum_r \lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R$
determine $\gamma_{\max}$ (the maximal γ_s value)
for $\forall R_s$: $w_s = -W_{\text{IPCost}} \dfrac{c(I_s)}{\max\{c(I_k), I_k \in I\}} + W_\gamma \dfrac{\gamma_s}{\gamma_{\max}} + W_{\text{IPSort}} N_s$

for $\forall M_r$: $\delta_r = |M_r|$
determine $\delta_{\min}$ (the minimal δ_r value)
selsect one R_s, for which:
$\exists M_r : |M_r \cap R_s| = \alpha$ and
$w_s = w_{\max}$ and
$\delta_r = \delta_{\min}$
$S := S \cup I_s$
$n(I_s) := n(I_s) + 1$
$R_{s,n(I_s)} := M_r \cap R_s$
neglect $\forall e_i$ from M if $e_i \in R_{s,n(I_s)}$
}
STOP

The convergence of algorithm DECIP is obvious, since the size of M is reduced in each cycle and an empty set M is obtained at the end. The speed of convergence is strongly influenced by the heuristic steps of selecting R_s from the different possibilities according to the criteria (α, $w_{\max}$, $\delta_{\min}$). The w_s values can be varied by adjusting the weight factor constants (W_{IPCost}, W_γ, W_{IPSort}). In this way, the selection strategy for R_s can be influenced and tested by trials as shown later. The number of these choices strongly depends on the magnitude (number of blocks, $|M|$) of the initial cover M. The increasing value of $|M|$ may involve a higher degree of overlapping between the blocks of the initial cover, which also rapidly increases the possible variations at selecting R_s. By a proper algorithm for reducing the initial cover M, this difficulty in computation can be avoided.

2.4. Algorithm REDIN (for REDucing the INitial cover M)

Notations for REDIN

z_i	the number of occurrence of e_i in actual M
m_r	the weighted sum of z_i-s in M_r, (weights are the relative costs of e_i-s)
g	the desired grade of reduction of $\|M\|$
$s(M)$	the actual set of M_r-s with the smallest m_r-s
$M(red)$	the reduced M

The Algorithm

START
if $|M| < g$ then STOP
$M(red) := \varnothing$
$j := 1$

calculate z_i-s for each e_i and m_r for each M_r
do {
determine $s(M)$
select an M_r from $s(M)$ and remove it from actual M
$M(red) := M(red) \cup M_r$
if $e_i \in M_r$, then $z_i := 0$
recalculate m_r-s and $s(M)$ for the actual M
$j := j + 1$
} while $M(red)$ is not a complete cover
while $j \leq g$ {
select an M_r from $s(M)$ and remove it from M
$M(red) := M(red) \cup M_r$
recalculate z_i-s, m_r-s and $s(M)$ for the actual M
$j := j + 1$
}
$g := j - 1$
STOP

There is still a selection step in each cycle of REDIN, but the number of elements in $s(M)$ is very limited in practical problems. Therefore, to choose an M_r does not mean a large number of variations.

3. Benchmark Solutions

In this section, the solution of a practical benchmark problem (cipher algorithm MARS) is presented for illustrating the basic modes of algorithm DECIP. The problem description starts with constructing an elementary operation graph (EOG) [4], [11], [3], [6]. The maximal compatibility classes of non-concurrent operations (set M) are generated by the high-level synthesis CAD tool PIPE developed at the Department of Control Engineering and Information Technology, Technical University of Budapest [3].

Steps of the solution:

1. Determining behavioral operation types for the problem to be solved. The number of types is assumed to be five for algorithm MARS.
2. Constructing the EOG. Algorithm MARS requires 416 behavioral operations.
3. For generating the set of IP-s predefined for application (set I), algorithm DECIP can be used in two different execution modes as follows:

   Mode 1: Selecting from a predefined set of IP-s, which can be taken from catalogues, or assumed to be generated by CAD tools. We have simulated an available set of IP-s by using the XILINX Foundation Series CAD tool. The IP-s generated in this way are specified by composing the behavioral operations applied in Step 1.

Mode 2: Approximating the best IP behaviors by composing initial fictive IP-s from the behavioral operations used in the EOG description of the problem to be solved.

4. Determining the execution times of behavioral operation types based on the IP-s found or generated in Step 3. If a behavioral operation is executable by several IP-s, then the longest execution time will be assumed.
5. Executing PIPE for determining an initial cover M. Since tool PIPE is dedicated for synthesis of pipelined systems, the desired restarting period is also an input parameter. Non-pipeline mode can be forced if the latency time of EOG is given as restarting period [3]. However, involving also the pipeline possibility, the restarting period is set to 200 clock cycles, which is approximately the half of the latency time.
6. Constructing the input parameters for DECIP.
7. Executing DECIP.

3.1. Input Parameters for DECIP

As it has been shown in Section 1, algorithm DECIP requires the following input parameters:

Behavioral Operations

The types of behavioral operations (and their parameters) used in MARS algorithm are summarized in *Table 1*. To obtain proper practical values for the execution times, each type of behavioral operations has been generated experimentally by the XILINX Foundation Series software tool. A clock frequency of 32 MHz is assumed, and the execution times of the behavioral operations are handled as the number of clock periods required for execution.

Functions $sbox$, $sbox0$, $sbox1$ represent algorithmic components of MARS cipher. These components are assumed to be implemented in memory-type IP-s. In most cases these components are simple look-up tables.

Compatibility Classes (Set M)

The graph representation of algorithm MARS, the operation types and their execution times (*Table 1*) are input parameters for design tool PIPE. The result of the allocation step is a complete partition on the set (E) of behavioral operations. Each block of this partition consists of pair-wise non-concurrent operations. This partition can be used as initial cover (M) for DECIP. Since the blocks of the partition are disjoint, there is no need to execute algorithm REDIN before DECIP in this case.

Table 1. Execution times of behavioral operations

Operation type name	Required IP function	Number of occurrence	Maximal frequency (MHz)	Execution time, t_i
Mult	multiplying	16	2.823	12
Add	adding	72	7.290	5
Sub	subtracting	24	7.290	5
Xor	XOR	80	36.621	1
shr8	shifting	24	32.424	1
Shl	shifting	32	32.424	1
shl5	shifting	32	32.424	1
shl8	shifting	24	32.424	1
shl13	shifting	32	32.424	1
sbox	storing	16	34.904	1
sbox0	storing	32	34.904	1
sbox1	storing	32	34.904	1

Predefined Set I of IP-s to Be Applied and their Costs

The composition of this set is made differently in the two execution modes of DECIP as illustrated later.

Weight Factors (W_{IPCost}, W_{γ}, W_{IPSort})

Since these factors stress only the relative weights of the different criteria expressed by values between 0 and 1, their task can be fulfilled also in the range of value between 0 and 1. For illustrating the influence of the experimental weight factors, a range of their values from 0 to 0.4 in steps of 0.1 is examined. In this way, the relative weights can be set for 1 to 5 and 125 variants of weight factors can be examined. By calculating with different compositions of weight factor values, algorithm DECIP can be adjusted for the character of each problem to be solved. In this way, the selecting procedure of the algorithm is influenced in order to approach to the optimal strategy for the given problem.

By scanning with discrete values of weight factors, global optimum may be hidden. Therefore, proper distances and dominance have to be established experimentally.

Obviously, other compositions of weight factors, range and step size may be checked in the same way. For instance, one of the most serious difficulties of applying IP-s is establishing of the proper communication between them. The complexity, cost and execution time of communication are important parameters for specifying and selecting IP-s. The communication problem is out of the scope of this

paper, but algorithm DECIP can be adjusted also for considering communication effects by introducing additional weight factors and special criteria.

3.2. *Selecting from an Available IP Set (Mode 1)*

In this execution mode, an available set of IP-s is assumed and the optimal selection from this set is approached. We have simulated experimentally an available IP set by generating them using the CAD tool XILINX Foundation. The IP-s obtained in this way, are considered as products taken from a catalogue. Each IP is specified by the behavioral operations, for which it is generated. In other words, these are the operations, the execution of which is possible and preferred by the IP. In *Table 1*, these specifying operations (i.e., the elements of set R) are listed for each IP. The other parameters of the IP-s are shown in *Table 2*. For simplicity, the relative cost of IP-s ($c(I_s)$) is assumed to be identical to the number of CLB-s.

Table 2. Illustration of simulated available IP-s

IP name (set I)	IP specification	Possible and preferred operation types (types of elements in subsets R_s)	Max. frequency (MHz)	Cost ($c(I_s)$) (based on the number of CLBs)
Multiplier	multiplying	Mult	2.823	840
Adder	adding	Add	7.290	32
Subtracter	subtracting	Sub	7.290	32
Logic	XOR	Xor	36.621	9
Shift	shifting	shr8, shl, shl5, shl8, shl13	32.424	32
AU	adding, subtracting	add, sub	7.290	72
Memory	storing	sbox, sbox0, sbox1	34.904	528

Thus, all input parameters are given for executing DECIP. Let a cost function C be defined as follows:

$$C = \sum_s c(I_s)n(I_s).$$

A special radar diagram is shown in *Fig. 1* for evaluating the results. The cost values and the number of IP-s selected for application ($|S|$) are illustrated in a proper normalized way for a better presentation. Let an efficiency factor F be defined as follows:

$$F = \frac{|S| + C}{2}.$$

The minimal value of F can be considered as a good compromise between cost and reuse.

Where the cost is small, there $n(I_s)$ is large and vice versa. Therefore, the curve of the efficiency factor becomes generally smooth (see *Fig. 1* and *Fig. 2*).

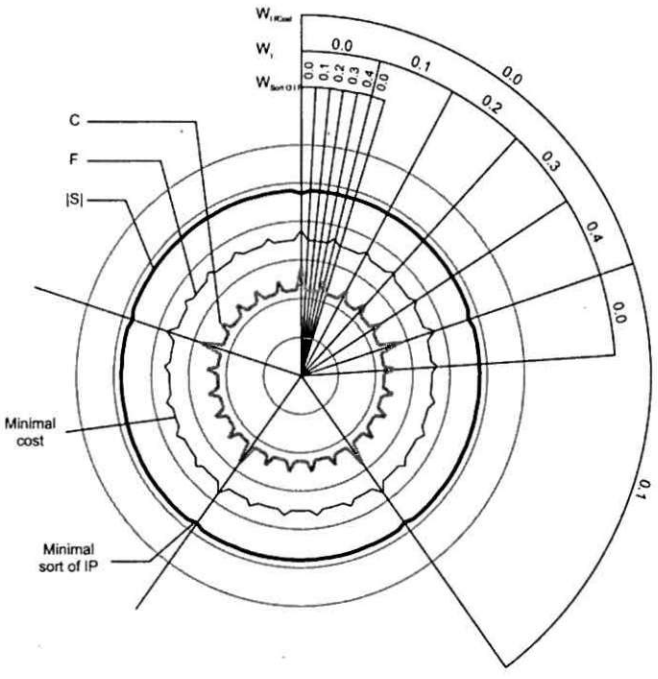

Fig. 1. The cost, $|S|$ values and efficiency factor in case of selecting from an available IP set

Table 3 contains the selected IP set S, which has the minimal cost (this can be observed on the radar diagram as the location of smallest value of C). In this case, the use of all types of predefined IP-s is necessary.

If reuse is preferred to cost, then at least 5 types of IP-s are needed, as shown in *Table 4*. In this case the cost is larger, so this solution requires more CLBs.

3.3. Approximating the Best IP Behaviors (Mode 2)

In this execution mode, the types of IP-s (set I) are assumed not to be taken from catalogue, but to be determined and constructed from compositions of behavioral operations applied in the behavioral description (EOG) of the problem to be solved. Thus, an initial set of such fictive IP-s is considered as set I in the first step. These fictive IP-s are specified by combinations of their behavioral operations. The parameters of these fictive IP-s are estimated for experimental use, as if they were

Table 3. Results with minimal cost

Selected IP-s set S	Number of IP-s $n(I_s)$	Cost C
Multiplier	10	8400
Adder	5	160
Subtracter	3	96
Logic	16	144
Shift	15	480
AU	8	576
Memory	15	7920
		17776

Table 4. Results if reuse is preferred to cost

Selected IP-s set S	Number of IP-s $n(I_s)$	Cost C
Multiplier	10	8400
Logic	16	144
Shift	15	480
AU	16	1152
Memory	15	7920
		18096

obtained by using the XILINX Foundation Series software tool with the same results as in the previous section.

Each combination of five behavioral operations (multiplication, addition, subtraction, xor and shift) generated previously for basic IP-s is assumed to specify a fictive IP. The number of CLB-s of multifunctional IP-s is estimated as the sum of the number of CLB-s obtained for the individual components in the previous section. (*Table 2*). These parameters of the initial fictive IP set are shown in *Table 5*.

The results of DECIP are illustrated in a radar diagram (*Fig. 2*) introduced in the previous section.

The minimal cost and the best reuse are represented by extreme values on the diagram. The corresponding IP sets selected from the initial fictive IP-s of *Table 5* are illustrated in *Table 6* and *Table 7*, respectively.

It can be seen by comparing *Tables 3* and *6*, that the cost is lower in the case of multifunctional IP-s.

Table 7 shows that the selecting procedure provides the trivial solution, if reuse is preferred in this case.

Table 5. Illustration of generated (fictive) IP-s

IP name (set I)	IP specification	Possible and preferred operation types (types of elements in subsets R_s)	Cost ($c(I_s)$) (based on the number of CLBs)
Memory	storing	sbox, sbox0, sbox1	528
ip0	multiplying	Mult	840
ip1	adding	Add	32
ip2	subtracting	Sub	32
ip3	XOR	Xor	9
ip4	shifting	shr8, shl, shl5, shl8, shl13	32
ip5	multiplying, adding	mul, add	872
ip6	multiplying, subtracting	mul, sub	872
ip7	multiplying, XOR	mul, xor	849
ip8	multiplying, shifting	mul, shr8, shl, shl5, shl8, shl13	872
ip9	adding, subtracting	add, sub	64
ip10	adding, XOR	add, xor	41
ip11	adding, shifting	add, shr8, shl, shl5, shl8, shl13	64
ip12	subtracting, XOR	sub, xor	41
ip13	subtracting, shifting	sub, shr8, shl, shl5, shl8, shl13	64
ip14	XOR, shifting	xor, shr8, shl, shl5, shl8, shl13	41
ip15	multiplying, adding, subtracting	mul, add, sub	904
ip16	multiplying, adding, XOR	mul, add, xor	881
ip17	multiplying, adding, shifting	mul, add, shr8, shl, shl5, shl8, shl13	904
ip18	multiplying, subtracting, XOR	mul, sub, xor	881
ip19	multiplying, subtracting, shifting	mul, sub, shr8, shl, shl5, shl8, shl13	904
ip20	multiplying, XOR, shifting	mul, xor, shr8, shl, shl5, shl8, shl13	881
ip21	adding, subtracting, XOR	add, sub, xor	73
ip22	adding, subtracting, shifting	add, sub, shr8, shl, shl5, shl8, shl13	96
ip23	adding, XOR, shifting	add, xor, shr8, shl, shl5, shl8, shl13	73
ip24	subtracting, XOR, shifting	sub, xor, shr8, shl, shl5, shl8, shl13	73
ip25	multiplying, XOR, adding, subtracting	mul, add, sub, xor	913
ip26	multiplying, adding, subtracting, shifting	mul, add, sub, shr8, shl, shl5, shl8, shl13	936
ip27	multiplying, adding, XOR, shifting	mul, add, xor, shr8, shl, shl5, shl8, shl13	913
ip28	multiplying, shifting, subtracting, XOR	mul, sub, xor, shr8, shl, shl5, shl8, shl13	913
ip29	adding, shifting, subtracting, XOR	add, sub, xor, shr8, shl, shl5, shl8, shl13	105
ip30	multiplying, XOR, adding, shifting, subtracting	mul, add, sub, xor, shr8, shl, shl5, shl8, shl13	945

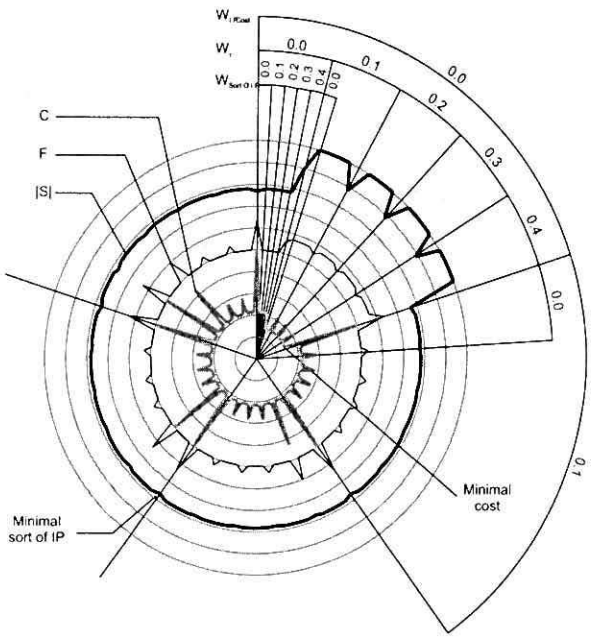

Fig. 2. The cost, $|S|$ values and efficiency factor for approximating the best IP behaviors

Table 6. Results with minimal cost for approximating the best IP behaviors

Selected IP-s Set S	Number of IP $n(I_s)$	IP specification	Cost of IP $c(I_s)$	Total cost C
ip12	1	subtracting, XOR	41	41
ip23	1	adding, XOR, shifting	73	73
ip24	2	subtracting, XOR, shifting	73	146
ip27	4	multiplying, adding, XOR, shifting	913	3652
ip29	2	adding, shifting, subtracting, XOR	105	210
ip30	6	multiplying, XOR adding, shifting, subtracting,	945	5670
Memory	15	storing	528	7920
				17712

The experimental version of algorithm DECIP has been written in Visual Basic Script and its running time on a Pentium PC (300 MHz, 64M RAM) was 7 hours for MARS benchmark in both modes.

Table 7. Results if reuse is preferred for approximating the best IP behaviors

Selected IP-s Set S	Number of IP $n(I_s)$	IP specification	Cost of IP $c(I_s)$	Total cost C
ip30	16	multiplying, XOR adding, shifting, subtracting,	945	15120
Memory	15	storing	528	7920
				23040

4. Conclusions and Further Research

The algorithm DECIP presented in this paper decomposes a system characterized by subsets of behavioral operations specifying the problem to be solved by the system. The target architecture consists of executing IP-s selected from a predefined set of IP-s. Each IP is assumed to be specified by behavioral operations, the execution of which is possible and preferred by this IP. Algorithm DECIP selects the executing IP-s by varying weight factor values of several criteria in order to adjust the selection procedure to the character of the problem to be solved. In Mode 2 of DECIP, an initial set of fictive IP-s specified by combinations of behavioral operations can also be handled, in order to approximate to the best executing IP-behaviors.

Algorithm MARS is used as benchmark of practical size for illustrating the performance of DECIP.

The communication between IP-s is crucial in system-level synthesis. Algorithm DECIP can be adjusted to consider communication parameters by introducing additional weight factors and special criteria for selection. However, this extension is not elaborated yet, and it is a subject of further research.

Further extension of criteria and their weight factors would be necessary for taking into consideration the different execution times of the same operations in different IP-s. Slower execution of some operations may be allowed if, for example reuse is the most important aim of optimization.

Many practical IP-s are specified not only by a set of behavioral operations, but also by the execution order of the operations (e.g. filters, ALU, DCT, etc.). Such IP-s might be strongly preferred if the problem to be solved contains similar parts in operation order. Handling such IP specifications also requires modification of DECIP in further research.

Another further research aim is to build in some adaptivity for changing the weight factors automatically by a learning procedure. In this case, the character of problem to be solved has to control somehow the adaptive learning algorithm during the execution of DECIP.

Acknowledgements

The research work of the authors has been supported by the grants OTKA T030178 and FKFP 0416/97 at the Department of Control Engineering and Information Technology, Technical University of Budapest.

References

[1] ARATÓ, P. – BÉRES, I, A Compatibility-Based Allocation Method in High-Level Synthesis, *Periodica Polytechnica, El. Eng.*, 1996.

[2] ARATÓ, P. – BÉRES, I. – RUCINSKI, A. – DAVIS, R. – TORBERT, R., A High-Level Datapath Synthesis Method for Pipelined Structures, *Microelectronics Journal*, Elsevier Science Ltd., **25** (1994), pp. 237–247.

[3] ARATÓ, P. – VISEGRÁDY, T. – JANKOVITS, I., *High-Level Synthesis of Pipelined Datapaths*, John Wiley & Sons, Chichester, United Kingdom. first edition 2001,

[4] CAMPOSANO, R., From Behaviour to Structure: High-Level Synthesis, *IEEE Design and Test of Computers*, **10** (1990), pp. 8–19.

[5] CAMPOSANO, R. – ROSENSTIEL, W., Synthesizing Circuits from Behavioural Descriptions, *IEEE Transactions on Computer Aided Design*, **2** (1989), pp. 171–180.

[6] CAMPOSANO, R. – WOLF, W., *High-Level VLSI Synthesis*, Kluwer Academic Publisher, 1991.

[7] DE MICHELI, G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[8] GAJSKI, D., *High-Level Synthesis*, Kluwer Academic Publisher, 1992.

[9] GAJSKI, D. D. – DOEMER, R. – ZHU, J., IP-centric Methodology and Design with the SpecC Language, *NATO ASI System-Level Synthesis*, August, 1998.

[10] HWANG, C.–T. – LEE, J.–H. – HSU, Y.–C., A Formal Approach to the Scheduling Problem in High-Level Synthesis, *IEEE Transactions on Computer Aided Design*, **10** (1991), pp. 464–475.

[11] IEEE, Special Issue on High-Level Synthesis, *IEEE Transactions on Very Large Scale Integration Systems*, **1** No. 3 (1993).

[12] JERRAYA, A., Multilanguage Specification for System Design, In: *System-Level Synthesis, NATO Science Series*, Kluwer Academic Publisher, 1999.

[13] JERRAYA, A. A., *Behavioral Synthesis and Component Reuse with VHDL*, Kluwer Academic Publisher, 1997.

[14] PARK, N. – PARKER, A., SHEWA: A Program for Synthesis of Pipelines, *Proceedings of the 23rd Design Automation Conference*, 1986, pp. 454–460.

[15] PAULIN, P.G. – KNIGHT, J. P., Force-Directed Scheduling for the Behavioural Synthesis of ASICs, *IEEE Transactions on Computer Aided Design*, **6** (1989), pp. 661–679.

[16] ROLF, E., Embedded System Architectures, *System-Level Synthesis, NATO Science Series*, Kluwer Academic Publisher, 1999.

[17] STAUNSTRUP, J. – WOLF, W., *Hardware/Software Co-Design: Principles and Practice*, Kluwer Academic Publisher, 1997.