Semantic model for high-level synthesis of dataflow pipelines

Prihozhy A.A., Karasik O.N., Frolov O.M.
Information technologies and robotics department
Belarusian national technical university
Minsk, Republic of Belarus
Email: prihozhy@yahoo.com

***Abstract* — A semantic model of computational hardware and software pipelines has been developed. Several relations, graphs and logic inference rules constitute a basis for the construction and high-level synthesis of dataflow pipelines. The behavioral specification pipelining tool is capable of optimizing parallel implementations of logic inference and knowledge dynamic processing algorithms.**

Keywords* — *semantic model*; *pipeline*; *knowledge processing*; *high level synthesis*; *optimization

Pipelining is an efficient way of increasing the operating frequency and throughput of data intensive digital systems in various application fields. Among them, pipelining of knowledge representation and processing tools as well as logic inference tools is the most important task. A pipelined system is usually described in an appropriate programming or hardware description language. Pipelining can be seen as a transformation of a source behavioral specification into pipeline-stage-fragments that are executed in time-sliced fashion.

Complex digital systems are typically characterized by irregular structures, thus it is impossible to perform a straightforward mapping of the specification into a pipeline implementation. Therefore, this paper develops an efficient semantic model of pipelining designs that imply several "low cost" chained operators in one basic processing block. The model takes into account key parameters of the behavioral elements including the variable sizes, the operator delays, the relations on the set of variables and operators, and the behavior of mutually exclusive branches.

Various languages and representations are used for describing pipelines: the concurrent language, CAL [1], programming language, C/C++ [2], data flow graphs [3], signal flow graphs [4], transactional specifications [5], and other notations. A pipeline system is characterized by several parameters such as the clock cycle time, stage cycle time, number of pipeline stages, latency, data initiation interval, frequency and throughput.

The pipeline high-level synthesis algorithms as follows have been proposed: list scheduling [6], force directed scheduling [7], iterative modulo scheduling [8], speculative loop pipelining [9], and integer linear programming [10]. The loop winding method [11], percolation based scheduling [12], loop rotation scheduling [3], pipeline vectorization method [2] and modulo scheduling followed by stage scheduling [13] aim at pipelining loops. The macro pipelining based scheduling technique [14] is capable of pipelining heterogeneous multiprocessor systems. A pipeline decomposition tree based scheduling framework is presented in [15]. The cost-optimized algorithm for the selection of components without sharing resources in the pipeline is presented in [16].

Since modern technology provides large amounts of available resources, faster and larger pipelines for knowledge processing without (or with minimal) sharing of resources can be synthesized with advantages in performance [17-19]. In order to realize this challenge, a systematization of knowledge on pipeline construction, synthesis and optimization has to be conducted.

This paper is organized as follows. Section II presents the semantic model of the behavioral specification under pipelining. Section III describes the semantic model of computational pipelines. Section IV presents the semantic model of pipeline high level synthesis and optimization. The last section concludes the paper.

I. Semantic Model of Behavioral Specification under Pipelining

A. *Behavioral specification for pipelining*

The system behavior that is under pipelining is represented as a program in a system representation language. The key parts of the representation are variables, operators and relations. Each variable is characterized with a type and a size. The set of operators includes logic scalar and vector operators, arithmetic operators and others. The assignment, conditional and loop instructions allow to represent any computational behavior of the system under pipelining.

Rule 1. The pipeline synthesis and optimization is performed from a system behavior and constraints on pipeline parameters.

B. *Control-data flow graph*

The control-data flow graph (CDFG) is a result of translation of the behavioral specification into an intermediate representation. The original control dominated CDFG is not efficient for pipeline high-level synthesis. It should be transformed to a data flow graph (DFG) that is more convenient for pipelining. The transformation is based on splitting and eliminating control structures as shown in [17] and on rules 2-5.

Rule 2. If the behavioral description contains loops then it is transformed to a single loop with an infinite iteration scheme, one linear basic block and *break* instructions inside it.

Rule 3. If the behavioral description is a branched one then it is transformed to a sequence of short *if-then* instructions with an assignment inside which are considered as data flow elements.

Rule 4. If an assignment instruction contains more than one operator in the right part expression then it is transformed to a sequence of simpler assignments by adding intermediate variables.

Rule 5. The mixed control/data flow graph is transformed to a pure data flow graph and a set of relations.

An example of pure data flow graph is shown in fig.1. The graph consists of the variables and operators that are reported in fig.2. The operators are located on the levels according to the data dependences and critical path.

C. *Data dependency relation and graph*

Let V be a set of variables and N be a set of operators. A set of input variables of operator $i=1,\ldots,n$ is denoted as $in(i)\subseteq V$ and a set of its output variables is denoted as $out(p)\subseteq V$. From these sets, a set $cons(v)\subseteq N$ of consumers and a set $prod(v)\subseteq N$ of producers is being computed for each variable $v\in V$. The data dependences among operators are represented with a binary matrix (relation and graph) D whose rows and columns correspond to operators. An element $d_{ij}=1$ if operator j is data dependent on operator i. For CDFG with loops the graph D is cyclic, otherwise it is acyclic.

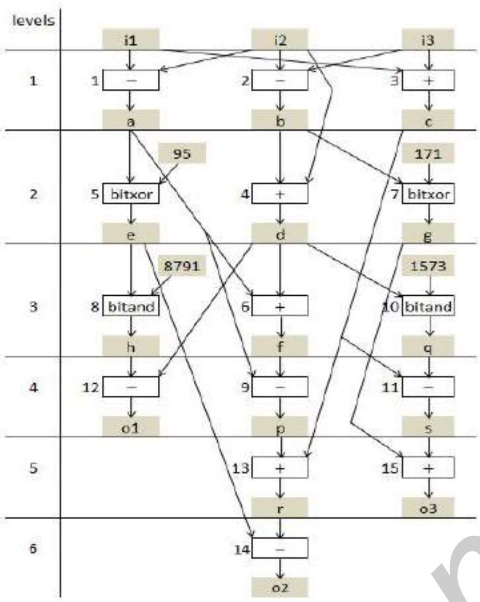

Fig.1. Example data flow graph

Operators			Variables			
No	Type	Relative delay	No	Name	Mode	Size
1	–	2.20	1	i1	in	16
2	–	1.65	2	i2	in	12
3	+	2.00	3	i3	in	12
4	+	1.75	4	a	loc	16
5	bitxor	0.10	5	b	loc	10
6	+	2.00	6	c	loc	13
7	bitxor	0.10	7	d	loc	14
8	bitand	0.10	8	e	loc	18
9	–	2.20	9	f	loc	16
10	bitand	0.10	10	g	loc	6
11	–	1.93	11	h	loc	18
12	–	2.48	12	p	loc	13
13	+	1.62	13	q	loc	14
14	–	2.48	14	r	loc	13
15	+	1.25	15	s	loc	10
			16	o1	out	17
			17	o2	out	14
			18	o3	out	10

Fig.2. Operators and variables of example data flow graph

Rule 6. The data dependency graph is constructed taking into account the input and output variables of operators, feedbacks in CDFG with loops and mutually exclusive execution conditions.

D. *Operator precedence relation and graph*

The operator precedence relation P describes a partial order on the set of operators.

Rule 7. The partial order is derived from the analysis of data dependences between operators in DFG taking into account the orthogonality of test variables in conditional statements.

The operator direct precedence relation P_{direct} is computed as a minimal anti-transitive relation of the transitive closure P_{trans} of relation P (fig.3). This relation represents the direct precedence graph as well. The graph can be cyclic or acyclic. It describes a mixed sequential-parallel execution of operators and short conditional instructions.

$$P_{trans}=\begin{vmatrix}
0&0&0&0&1&1&0&1&1&0&0&1&1&1&0\\
&0&0&1&0&1&1&0&1&1&1&1&1&1&1\\
&&0&0&0&0&0&0&0&0&1&0&1&1&1\\
&&&0&0&1&0&0&1&1&1&1&1&1&1\\
&&&&0&0&0&1&0&0&0&1&0&1&0\\
&&&&&0&0&0&1&0&0&0&1&1&0\\
&&&&&&0&0&0&0&0&0&0&0&1\\
&&&&&&&0&0&0&0&1&0&0&0\\
&&&&&&&&0&0&0&0&1&1&0\\
&&&&&&&&&0&1&0&0&0&1\\
&&&&&&&&&&0&0&0&0&1\\
&&&&&&&&&&&0&0&0&0\\
&&&&&&&&&&&&0&1&0\\
&&&&&&&&&&&&&0&0\\
&&&&&&&&&&&&&&0
\end{vmatrix}$$

Fig.3. Transitive matrix P_{trans} for the example dataflow graph

Rule 8. The minimal anti-transitive precedence relation speeds up the optimization process in high-level synthesis.

E. *Operator delays*

Timing models and delay estimation techniques for operators depend on the implementation platform: ASIC, FPGA, multi-core processor etc. The rules as follow are used.

Rule 9. Timing models of operators that are executed on words of bit depend on the operator type, operands width and implementation style.

Rule 10. Time delays of operators that are implemented on a LUT-based FPGA are measured in LUT (lookup table) levels and are estimated through bit-level interpretation of word operators and decomposing them into logic LUT-fragments.

F. *Longest delay paths in operator precedence graph*

The lengths of longest paths between operators in the operator precedence graph constitute a basis for realizing pipeline constraints. A matrix L represents the lengths for all operator pair. As the precedence graph is DAG for a non-loop behavior, matrix L can be computed in a polynomial time. For P_{trans} shown in Fig.3 and its elements described in fig.2, the matrix L is given in Fig.4.

Rule 11. Additive timing models are used in many cases of pipeline implementation. More complex and precise timing models of operators and paths are used in several design flow, in particular, for FPGA.

II. SEMANTIC MODEL OF COMPUTATIONAL PIPELINES

A. *Classification of pipelines*

Fig.5 and 6 show two architectures of hardware pipelines and fig.7 shows architecture of a software pipeline. The number of clock cycles within one stage is called a pipeline initiation interval (II). The increase of II dramatically influences the resource sharing.

Rule 12. If the goal is to minimize the resources by sharing, II is increased. It costs a growth in the hardware pipeline latency and a reduction in the system throughput.

$$L = \left|\begin{array}{ccccccccccccccc}
2.20 & 0.00 & 0.00 & 0.00 & 2.30 & 4.20 & 0.00 & 2.40 & 6.40 & 0.00 & 0.00 & 4.88 & 8.02 & 10.50 & 0.00 \\
 & 1.65 & 0.00 & 3.40 & 0.00 & 5.40 & 1.75 & 0.00 & 7.60 & 3.50 & 5.43 & 5.88 & 9.23 & 11.70 & 6.68 \\
 & & 2.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 3.93 & 0.00 & 3.63 & 6.10 & 5.18 \\
 & & & 1.75 & 0.00 & 3.75 & 0.00 & 0.00 & 5.95 & 1.85 & 3.78 & 4.23 & 7.57 & 10.05 & 5.03 \\
 & & & & 0.10 & 0.00 & 0.00 & 0.20 & 0.00 & 0.00 & 0.00 & 2.68 & 0.00 & 2.58 & 0.00 \\
 & & & & & 2.00 & 0.00 & 0.00 & 4.20 & 0.00 & 0.00 & 0.00 & 5.82 & 8.30 & 0.00 \\
 & & & & & & 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.35 \\
 & & & & & & & 0.10 & 0.00 & 0.00 & 0.00 & 2.58 & 0.00 & 0.00 & 0.00 \\
 & & & & & & & & 2.20 & 0.00 & 0.00 & 0.00 & 3.83 & 6.30 & 0.00 \\
 & & & & & & & & & 0.10 & 2.03 & 0.00 & 0.00 & 0.00 & 3.28 \\
 & & & & & & & & & & 1.93 & 0.00 & 0.00 & 0.00 & 3.18 \\
 & & & & & & & & & & & 2.48 & 0.00 & 0.00 & 0.00 \\
 & & & & & & & & & & & & 1.63 & 4.10 & 0.00 \\
 & & & & & & & & & & & & & 2.48 & 0.00 \\
 & & & & & & & & & & & & & & 1.25
\end{array}\right|$$

Fig.4. Matrix *L* of longest path lengths for the example dataflow graph

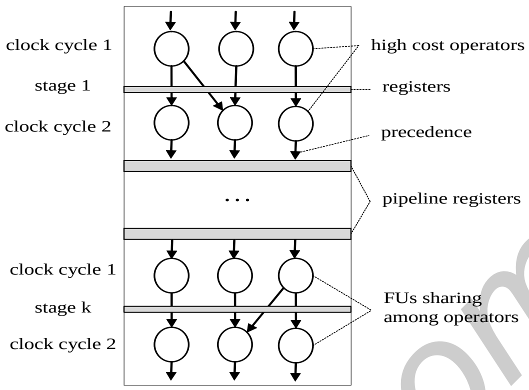

Fig.5. Hardware pipeline with two clock cycles per stage and resource sharing

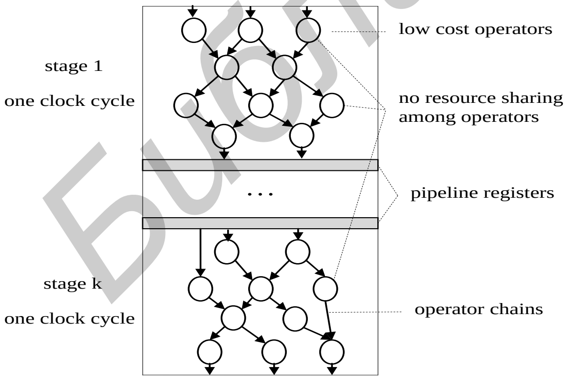

Fig.6.Hardware pipeline with one clock cycle per stage, operator chaining and without resource sharing

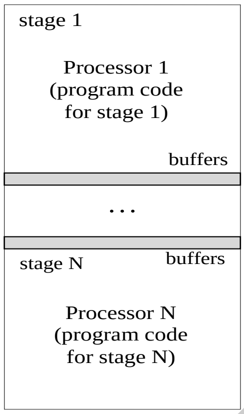

Fig.7.Software pipeline that consists of stages which are assigned a program code that is executed on a processor

Rule 13. If the goal is to minimize the hardware pipeline latency and maximize the throughput, II is decreased. Pipelines with one clock cycle per stage use operator chains within one stage and do not use resource sharing.

Rule 14. If the goal is to maximize the throughput of software pipeline, the program code is partitioned for the execution on processors which run in the time-sliced fashion.

Rule 15. If the goal is to optimize the pipeline, the tasks as follows are to be solved: choosing the number of stages and the pipeline initiation interval; selection of operator implementations, assignment of operators to stages and clock cycles, minimization of buffer sizes and minimization of the pipeline latency.

B. *Pipeline stage time*

In a hardware synchronous pipeline, the stage time, T_{stage} is evaluated in the number of clock cycles multiplied by the clock period. In pipeline optimization, the time is often considered as a constraint that essentially influences the resulting design throughput and load of equipment. In a software asynchronous pipeline, the stage time is the program code maximum execution time in a stage on the corresponding processor over all stages. If the data buffers which are inserted in between two stages are implemented as FIFOs, the stage time can vary over stages and data sets.

Rule 16. The pipeline stage time and the number of stages are mutually dependent values. The larger stage time implies the fewer number of stages.

C. *Operator conflict relation and graph*

For two operators *i* and *j*, if the value of $l_{i\,j}$ in matrix *L* is larger than T_{stage}, we say there is a pipeline stage conflict between these operators. To overcome this conflict, the operators must be assigned to different pipeline stages. The conflict relation and graph is described with a binary matrix, *C*. To speed up the pipeline optimization process, *C* is replaced with its minimal anti-transitive version which is computed from the transitive closure of *C* and contains the minimum number of value 1. Fig.6 presents the operator conflict relation for the example matrix, *L* and T_{stage}=3.825.

In a software pipeline, operators *i* and *j* have a conflict if the execution time of *i* and *j* plus the execution time of all operators which are successors of *i* and predecessors of *j* exceed the pipeline stage time.

$$C = \begin{vmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & & & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
 & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
 & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & & 0 & 0 & 0 & 0 \\
 & & & & & & & & & & & & 0 & 1 & 0 \\
 & & & & & & & & & & & & & 0 & 0 \\
 & & & & & & & & & & & & & & 0
\end{vmatrix}$$

Fig.6. Operator conflict relation C for the matrix L and T_{stage}=3.825

Rule 17. The conflict relation C is a basis for the estimation of the minimum number of pipeline stages and for the generation of a tremendous number of alternative but functionally equivalent hardware and software pipelines.

D. *Mapping of operators onto pipeline stages*

The mapping is described with a function *stage*: $N \rightarrow S$ where N is the set of operators and S is the set of stages. According to the mapping, s=*stage*(p) means that s is the stage which the operator p is assigned to.

Rule 18. The number of different solution, *stage* is equal to the number of different valid pipelines that are feasible and legal for the stage time T_{stage}.

III. SEMANTIC MODEL OF PIPELINE HIGH LEVEL SYNTHESIS AND OPTIMIZATION

A. *Determining the number of pipeline stages*

Rule 19. The number l of pipeline stages is determined by the length of a longest path in the operator conflict graph, C. The length is measured in the number of edges.

For a l-stage pipeline a minimum stage time is denoted $T_{\text{stage}}(l)$. The stage time for l stages is larger than the stage time for l+1 stages. Therefore all pipelines which are generated for the stage time in the range form $T_{\text{stage}}(l+1)$ to $T_{\text{stage}}(l)$ have the same number l of stages as shown in fig.7.

B. *ASAP and ALAP pipeline schedules*

The as soon as possible (ASAP) schedule assigns operators to the earliest pipeline stages and the as late as possible (ALAP) schedule assigns operators to the latest stages. Fig.8 and fig.9 show these schedules for the example dataflow graph.

Rule 20. ASAP and ALAP determine the mobility of each operator over pipeline stages.

Rule 21. ASAP and ALAP give the fastest pipeline schedule without sharing resources.

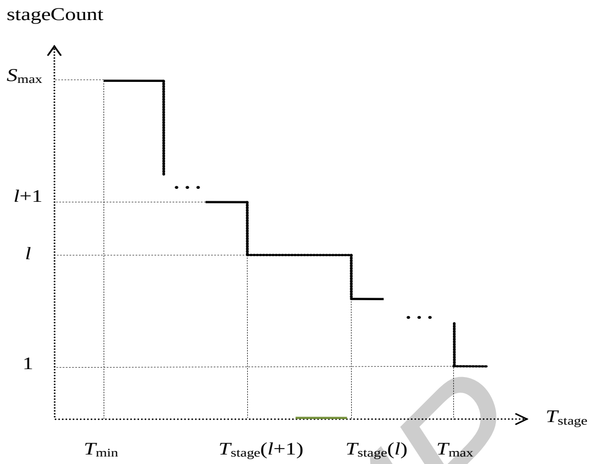

Fig.7.Number of pipeline stages versus stage time

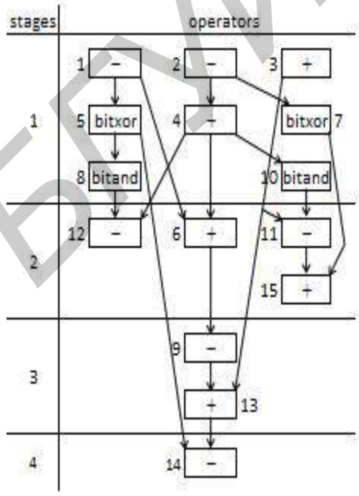

Fig.8.ASAP 4-stage pipeline schedule for the example dataflow graph and T_{stage}=3.825

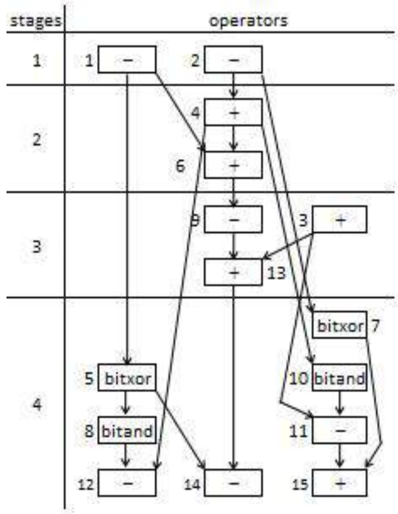

Fig.9. ALAP 4-stage pipeline schedule for the example dataflow graph and T_{stage}=3.825

Rule 21. ASAP and ALAP give the fastest pipeline schedule without sharing resources.

Rule 22. ASAP and ALAP do not yield the minimum overall pipeline buffer size.

C. A set of pipelines with the same stage time

A huge set of pipelines with the same stage count can be generated from the same operator conflict graph.

Rule 23. The number of feasible valid pipelines is estimated as μ^n where μ is the average operator mobility and n is the number of operators.

Rule 24. Heuristic optimization techniques must be used for large pipelined designs.

D. Overall pipeline buffer size minimization

The *lifetime*(v) of variable v over pipeline stages is determined by the difference of the earliest stage of its producers and the latest stage of its consumers (fig.10). Two and more producers must be conditional, *if* c_1 *then* v:=e_1; *end* … *if* c_k *then* v:=e_k; *end* with orthogonal test variables $c_1 \ldots c_k$ and expressions $e_1 \ldots e_k$.

Rule 25. The size of all buffers that represent v in a pipeline is computed as *size*(v)×*lifetime*(v). The overall buffer size is the sum of buffer sizes over all variables. This is true for both hardware and software pipelines.

Rule 26. In asynchronous pipelines the overall buffer size increases against the synchronous pipelines as each buffer is replaced with a FIFO.

E. Pipeline optimization algorithms

Exact and heuristic algorithms have been developed to optimize the dataflow pipelines. They assume the functional units and their parameters have been selected for the operator implementation and assume the processor parameters have been selected for the program code execution.

Rule 27. The algorithm of searching for the shortest path in the operator conflict graph minimizes the number of pipeline stages.

Rule 28. The overall buffer size minimization is a hard combinatorial problem that is solved by exact algorithms for small designed and heuristic algorithms for large designs.

Rule 29. The exact algorithm finds an optimal solution *stage* by means of logic inference with backtracking.

Rule 30. The heuristic algorithm finds a suboptimal solution *stage* by means of exploiting pipeline heuristics.

Fig.11 shows an optimal 4-stage pipeline schedule for the example data flow graph. This schedule consumes 13 pipeline registers (167 bit) while ASAP (fig.8) consumes 17 registers (247 bit) and ALAP (fig.9) consumes 16 registers (216 bit).

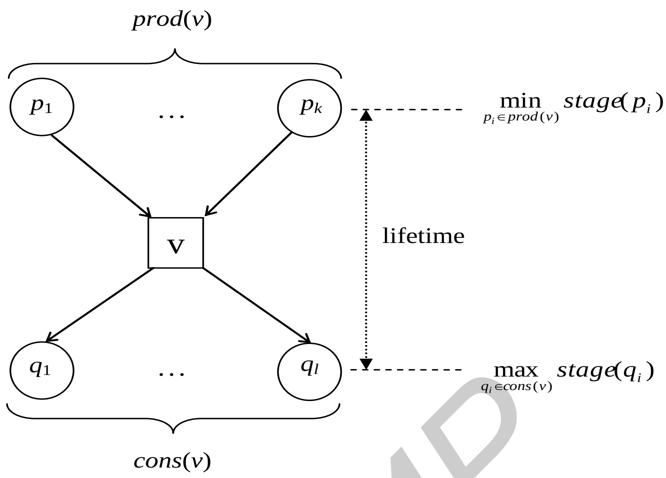

Fig.10. Lifetime of variable v over pipeline stages

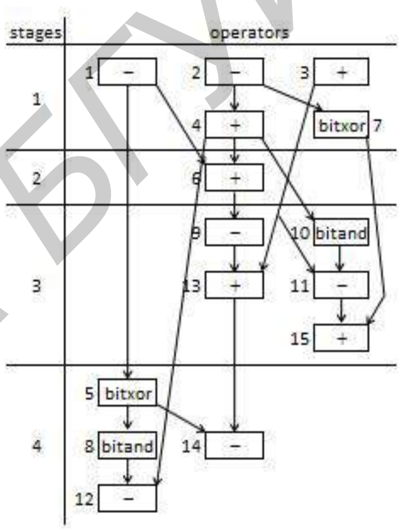

Fig.11. Optimal 4-stage pipeline schedule for the example dataflow graph and T_{stage}=3.825

F. Experimental results

The experiments have been conducted on designs from industry and on randomly generated designs. The proposed exact and heuristic algorithms of dataflow pipeline optimization yield much better results against ASAP and ALAP. They gain up to twice over ASAP and ALAP with respect to the overall buffer size. The exact algorithm is able to yield a solution for pipelines with 100 operators and 5 stages. The heuristic algorithm loses the exact one and gives only 2% larger buffer size on average over the exact algorithm. At the same time the heuristic algorithm is capable of handling large designs which consist of thousands operators and is capable of generating many-stage pipelines which consist of tens stages.

IV. Conclusion

This paper presents a semantic model for high-level synthesis and optimization of dataflow pipelines. Several objects, relations and graphs lie in the basis of this model, that are constructed in accordance with the set of inference rules which are formulated in this paper. Different architectures of hardware and software pipelines are analyzed and different optimization parameters and criteria are considered. Knowledge on the pipeline high-level synthesis and optimization techniques are represented with rules which allow the implementation of the synthesis by means of logic inference and heuristics exploration. The semantic model and pipelining tool aim at the parallelization and speeding up the knowledge acquisition and processing as well as increasing the throughput of the logic inference and knowledge manipulation tools.

REFERENCES

[1] Eker, J. CAL Language Report: Specification of the CAL Actor Language / J. Eker and J. Janneck // University of California-Berkeley, December 2003.

[2] Weinhardt, M. Pipeline vectorization / M. Weinhardt and W. Luk // Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 20, no. 2, pp. 234–248, Feb. 2001.

[3] Chao, L.-F. Rotation scheduling: a loop pipelining algorithm / L.-F. Chao, A. LaPaugh, and E.-M. Sha // Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 16, no. 3, pp. 229–239, Mar 1997.

[4] Jun, H.-S. Design of a pipelined datapath synthesis system for digital signal processing / H.-S. Jun, S.-Y. Hwang // Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 12, no. 3, pp. 292–303, September 1994.

[5] Nurvitadhi, E. Automatic pipelining from transactional datapath specifications / E. Nurvitadhi, J. Hoe, T. Kam, and S. Lu // Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 30, no. 3, pp. 441–454, March 2011.

[6] Park, N. Sehwa: A software package for synthesis of pipelines from behavioral specifications / N. Park and A. C. Parker // IEEE Trans. Computer-Aided Design, vol. 7, pp. 358–370, March 1988.

[7] Paulin P. G., Force-directed scheduling for the behavioral synthesis of ASIC's / P. G. Paulin and J. P. Knight // IEEE Trans. Computer-Aided Design, vol. 8, pp. 661–679, June 1989.

[8] Sun, W. FPGA pipeline synthesis design exploration using module selection and resource sharing / W. Sun, M. Wirthlin, and S. Neuendorffer // Trans. Comp.-Aided Des. Integrated Cir. Sys., vol. 26, no. 2, 2007, pp. 254–265.

[9] Oh, S. Speculative loop pipelining in binary translation for hardware acceleration / S. Oh, T. G. Kim, J. Cho, E. Bozorgzadeh // Trans. Comp.-Aided Des. Integ. Cir. Sys., vol. 27, no. 3, pp. 409–422, March 2008.

[10] Cong, J. An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation / J. Cong and Z. Zhang // Design Automation Conference (DAC), Jul. 2006.

[11] Girczyc, E. M. Loop winding–a data flow approach to functional pipelining / E. M. Girczyc // Proc. of the IEEE ISCAS, May 1987, pp. 382–385.

[12] Potasman, R. Percolation based synthesis / R. Potasman, J. Lis, A. Aiken, A. Nicolau // Proc. 27th Design Automation Conf., 1990, pp. 444–449.

[13] Eichenberger, A.E. Stage Scheduling: A Technique to Reduce the Register Requirements of a Modulo Schedule / A.E. Eichenberger, E.S. Davidson // Proc. 28th Int. Symp. on Micro architecture, 1995, pp. 338-349.

[14] Banerjee, S. Macro pipelining based scheduling on high performance heterogeneous multiprocessor systems / S. Banerjee, T. Hamada, P. Chau, and R. Fellman // IEEE Trans. Signal Processing, vol. 43, no. 6, pp. 1468-1484, June 1995.

[15] Ko, D.-I. The pipeline decomposition tree: an analysis tool for multiprocessor implementation of image processing applications / D.-I. Ko and S. S. Bhattacharyya // Proc. CODES+ISSS '06: 4th Int. Conf. on Hardware/ software codesign and system synthesis, 2006, pp. 52-57.

[16] Bakshi S. Component Selection for High-Performance Pipelines / S. Bakshi, D. Gajski // IEEE Trans. VLSI Syst., Vol. 4, No. 2, 1996, pp. 181-194.

[17] Prihozhy, A. High-level Synthesis through Transforming VHDL Models / A. Prihozhy // System-on-Chip Methodologies and Design Languages, Kluwer Academic Publishers, 2001, pp.135-146.

[18] Prihozhy, A. Synthesis and Optimization of Pipelines for HW Implementations of Dataflow Programs / A. Prihozhy, E. Bezati, H. Rahman, M. Mattavelli. // IEEE Trans. on CAD of Integrated Circuits and Systems, Vol. 34, No. 10, 2016, pp. 1613-1626.

[19] Rahman, H. Pipeline Synthesis and Optimization of FPGA-Based Video Processing Applications with CAL / H. Rahman, A. Prihozhy, M. Mattavelli // EURASIP Journal on Image and Video Processing, Vol. 2011:19, pp. 1–28.

СЕМАНТИЧЕСКАЯ МОДЕЛЬ ВЫСОКОУРОВНЕВОГО СИНТЕЗА КОНВЕЙЕРОВ ПО ПОТОКОВЫМ ОПИСАНИЯМ

Прихожий А.А., Карасик О.Н., Фролов О.М.

Разработана семантическая модель вычислительных аппаратных и программных конвейеров. Отношения, графы и правила логического вывода составляют базис построения и высокоуровневого синтеза конвейеров для обработки потоков данных. Отношения и графы представляют конвейер на всех этапах, начиная со спецификации и кончая реализацией. Правила логического вывода представляют процесс трансформации поведенческого описания в структурное описание конвейера. Инструментальная система конвейеризации поведенческих спецификаций обладает возможностями оптимизации параллельных потоковых реализаций алгоритмов логического вывода и динамической обработки знаний.