AUTOMATIC EXPLORATION APPROACH FOR SCHEDULING AND REGISTERS OPTIMIZATION IN HIGH-LEVEL SYNTHESIS

Rogério Xavier de Azambuja — *UFSC[1] – ULBRA Santa Maria[2]*
Dione Jonathan Ferrari — *UFSC[1]*
Luiz C. V. dos Santos — *UFSC[1]*

[1]UFSC – Universidade Federal de Santa Catarina
INE/CTC/LAPS – P.O. Box 476 – 88010-970 – Florianópolis/SC, Brazil
{xavier, dione, santos}@inf.ufsc.br

[2]ULBRA Santa Maria – Universidade Luterana do Brasil
P.O. Box 21834 – 97020-001 – Santa Maria /RS – Brazil

ABSTRACT

The objective of this paper is to present the solution for two classical High-Level Synthesis problems through an approach oriented to the automatic exploration of several alternative solutions. The first of them is the problem of scheduling operations under physical resource constraints, respecting its precedence order. The second problem is the respective allocation of registers, where it is determined how many registers are necessary in the digital circuit to store all values produced by some operations until they are consumed by others. This paper describes the implementation of the constructive approach and shows promising experimental results.

KEYWORDS

High-Level Synthesis, Scheduling, Register allocation.

RESUMO

O objetivo deste artigo é apresentar a solução para dois problemas clássicos da Síntese de Alto Nível através de uma abordagem orientada à exploração automática de soluções alternativas. O primeiro é o problema de escalonamento de operações sob restrição de recursos físicos, respeitando sua ordem de precedência. O segundo problema é a respectiva alocação de registradores, cuja solução determina quantos registradores são necessários no circuito digital para armazenar todos os valores produzidos por algumas operações até serem consumidas por outras. Este artigo descreve a implementação da abordagem construtiva e mostra resultados experimentais promissores.

PALAVRAS-CHAVE

Síntese de Alto Nível, Escalonamento, Alocação de registradores.

AUTOMATIC EXPLORATION APPROACH FOR SCHEDULING AND REGISTERS OPTIMIZATION IN HIGH-LEVEL SYNTHESIS

Rogério Xavier de Azambuja — *UFSC[1] – ULBRA Santa Maria[2]*
Dione Jonathan Ferrari — *UFSC[1]*
Luiz C. V. dos Santos — *UFSC[1]*

[1]UFSC – Universidade Federal de Santa Catarina
INE/CTC/LAPS – P.O. Box 476 – 88010-970 – Florianópolis/SC, Brazil
{xavier, dione, santos}@inf.ufsc.br

[2]ULBRA Santa Maria – Universidade Luterana do Brasil
P.O. Box 21834 – 97020-001 – Santa Maria /RS – Brazil

ABSTRACT

The objective of this paper is to present the solution for two classical High-Level Synthesis problems through an approach oriented to the automatic exploration of several alternative solutions. The first of them is the problem of scheduling operations under physical resource constraints, respecting its precedence order. The second problem is the respective allocation of registers, where it is determined how many registers are necessary in the digital circuit to store all values produced by some operations until they are consumed by others. This paper describes the implementation of the constructive approach and shows promising experimental results.

1. INTRODUCTION

Several applications like mobile phones, household appliances, consumer electronics, etc., are based on embedded computing systems. A typical embedded system consists of a processor, memory and possibly an application-specific integrated circuit (ASIC).

Starting from a specification, the design of an ASIC consists of some development stages: The *synthesis*, the *validation* and the *test of functioning*. In this process, the major step is *High-Level Synthesis* (HLS), which is the automatic synthesis of the architectural structure of a digital circuit from an algorithmic specification of its behavior [3]. Behavior is described using a *hardware description language* (HDL).

Among the main High-Level Synthesis steps are *allocation*, which determines how many resources are needed and *scheduling*, which defines when operations are executed. *Operations* (addition, subtraction, comparison, etc.) are associated with *resources* that can be classified in *functional units* (adder, ALU, multiplier,...), *storage units* (register, memory,...) and *interconnection units* (bus, multiplexers,...).

When the number of resources of each type is defined before synthesis, we say that the digital system must satisfy *resource constraints.*

Data-dependencies within a HDL description impose *precedence constraints*. Such constraints are modeled by a *data-flow graph* (DFG), where the vertices represent operations and the edges represent data-dependencies.

The result of the HLS is a *datapath* and a *controller*, which are modeled by distinct graphs: the *data-path graph* (DPG) and the *state-machine graph* (SMG), respectively. The DPG models a network of functional units and registers, showing the occupation of resources by operations. The SMG describes a symbolic state machine for the controller. It shows in which state each operation is executed. The Figure 1 shows an overview of HLS.

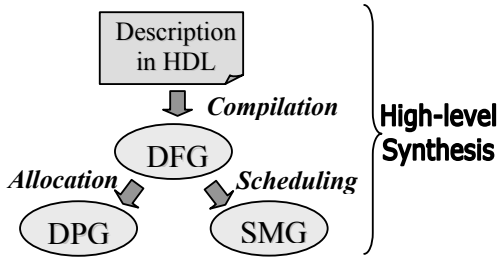

Figure 1: High-Level Synthesis Modeling.

This paper tackles two classical HLS problems: the scheduling of operations and the allocation of registers. Both problems are solved by an approach oriented to the automatic exploration of several alternative solutions.

Above are summarized the main contributions of this paper:

- The approach used to solve the problems of scheduling and allocation is not based on the use of heuristics that limit the generation of only one solution,

but allows the *exploration of several alternative solutions*, with the objective to obtain solutions of better quality.

- Instead of limiting to schedule operations along the time, the scheduler presented here, associates operations directly with *states*, constructing the symbolic state machine of the controller progressively during the scheduling.This allows the use of the number of states as metric to evaluate the quality of the solutions, what is not supported by methods where the scheduling is treated as an order in a lineal sequence of steps.

The remainder of this article is organized of the following form: The Section 2 presents the modeling of the proposed problems and presents one brief bibliographical revision. The Section 3 discusses a general vision of the adopted approach, showing the used techniques. The Section 4 describes the implementation of the approach, the accomplished experiments and the obtained results. The conclusions and the perspectives of continuity of the research in futures works are discussed in the Section 5.

2. MODELING AND PROBLEM FORMULATION

2.1. Basic definitions

Definition 1 - A polar *data flow graph* DFG(V,E) is a directed graph where each vertex $v_i \in V$ represents an operation and where each edge $(v_i, v_j) \in E$ represents a data dependence between v_i e v_j. The poles v_0 e v_n are called *source* and *sink*, respectively.
Definition 2 - A polar *state-machine graph* SMG(S,T) is a directed graph where each vertex vertex $s_i \in S$ represents a state and where each edge $(t_i, t_j) \in T$ represents a transition between states s_i e s_j. The poles s_0 e s_n are called *source* and *sink*, respectively.
Definition 3 - A polar *data-path graph* DPG(C,W) is a directed graph where each vertex vertex $c_i \in C$ represents a functional unit and where each edge $(w_i, w_j) \in W$ represents a interconnection between units c_i e c_j. The poles c_0 e c_n are called *source* and *sink*, respectively.
Definition 4 - A resource constraint vector **a** is a vector where each component a_k represents the number of functional units available of a given type $k \in \{1, 2, ... n_{res}\}$.
Definition 5 – Let $s_o, s_1, ... , s_k, ... , s_n$ be successive states in the SMG. The latency λ of the SMG is equal to the number n-1 of states between the source s_0 and the sink s_n.
Definition 6 - The chromatic number χ is the minimum number of colors needed to solve a vertex-coloring problem of a given graph.

2.2. The Scheduling Problem

Scheduling consists in finding an ordering in time for the operations such that precedence and resource constraints are both satisfied. The goal is to find a solution to minimize the latency λ.

The Figure 2 shows a behavioral description and its corresponding DFG. Assuming the following restriction of resources: one instance of the operator ALU and two multipliers, a possible solution for the example of Figure 2a can be found in Figure 2b. Horizontal lines depict different clock *cycles*. Each clock cycle represents the duration of a *state*. The execution of an operation in a resource requires an interval known as *execution delay*, which is expressed in terms of clock cycles. The poles of the DFG represent primary inputs and outputs and have zero execution delay. The time when an operation finishes is given as its initial execution time plus its execution delay.

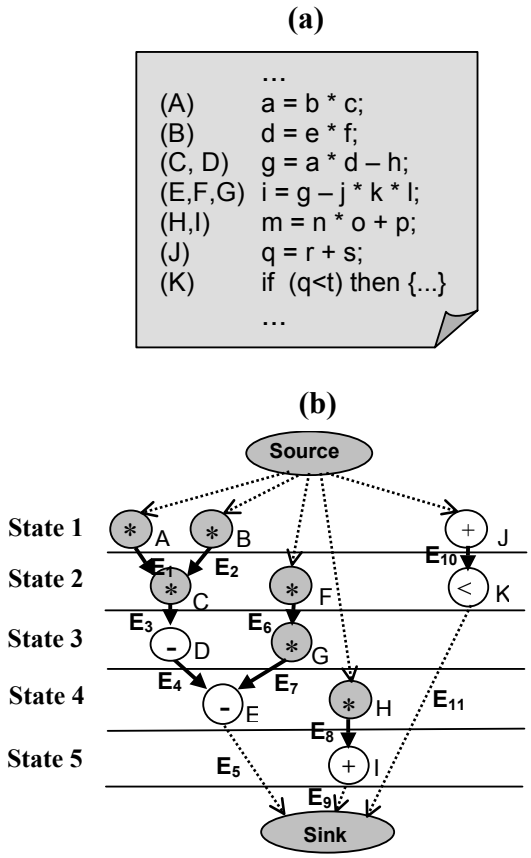

Figure 2: (a) Behavioral description and (b) Scheduled DFG ($\lambda = 5$).

2.3. The Register Allocation Problem

Given an edge of a DFG, it is associated with a *value* which is produced by the vertex at its tail and consumed by the vertex at its head. Given a state the values that must be consumed in the next state must be stored in *registers*. Therefore, each value can be associated with a *lifetime interval*, which starts in the state in which it is produced

and finishes in the state where it is at last consumed. If two or more values have disjoint lifetime intervals, they can share a same register in distinct time intervals. Therefore, it is possible to minimize the amount of needed registers, reducing circuit area and fabrication costs.

Initially, it is necessary to associate each DFG edge with a value. Primary inputs lifetime intervals are omitted since its values are supposed to be always available. Figure 2b shows a scheduled DFG where the edges are labeled. The underlying assumption is that values must be preserved until the fifth cycle only. It is possible to find the minimum number of registers by means of vertex-coloring. Each color corresponds to a distinct register. Therefore, the chromatic number χ is equal to the minimum number of registers.

An efficient algorithm to obtain χ is the so-called left-edge algorithm [4]. Figure 3a shows the lifetime intervals for the edges labeled in Figure 2b. When two or more intervals are disjoint, they can occupy the same register and the respective values are said to be *compatible*. Incompatibility can be represented by means of a conflict graph, which is shown in Figure 3b, where each vertex represents a lifetime interval and each edge represents a conflict between incompatible values.

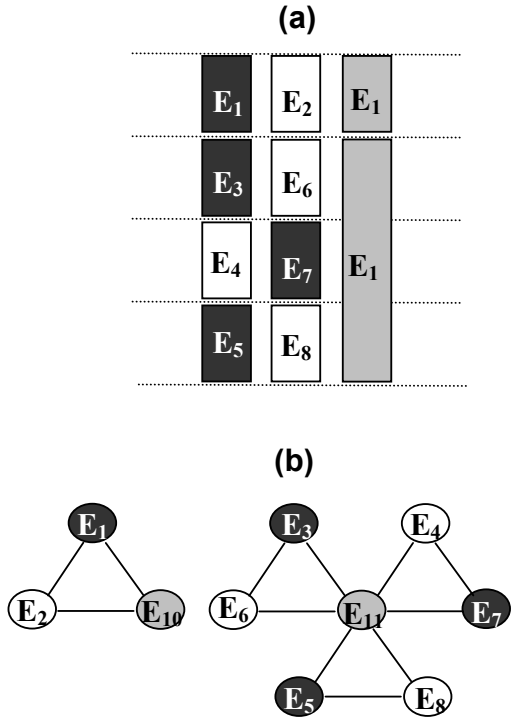

Figure 3: Lifetime intervals and their conflict graph for the example in Figure 2b.

2.4. Brief Literature Review

On the one hand, resource constrained scheduling is an intractable problem, which means that no polynomial algorithm is expected to always found an optimal solution. On the other hand, given a schedule, register allocation can be solved by polynomial algorithms in some special cases [4]. For this reason, the use of exact algorithms is limited to small instances of the scheduling problem, otherwise the computational effort would be prohibitive. An example of an exact method is Integer Linear Programming (ILP) [4]. A second approach is the use of approximate algorithms, which may come up with acceptable solutions yet not always optimal. In this case, operations are scheduled according to a priority list, which is obtained by means of an heuristic criterion. Examples of heuristic scheduling techniques are *List Scheduling* and *Force-Directed Scheduling* [4]. These techniques can generate high-quality solutions. However, their deficiency is that, since they generate a single solution, if it turned to be of unacceptable quality, there would be no way of exploring alternative solutions of better quality. Since embedded systems tend to be specified with *tight* constraints, it becomes important to support *design space exploration*. This motivates an approach oriented to the exploration of alternative solutions, such an approach is described in the next section.

3. AN APPROACH ORIENTED TO AUTOMATIC EXPLORATION

To increase the chances of finding a good solution within an acceptable time, it is desirable to explore several solutions, but without falling on an exhaustive search. This approach tries to make a trade-off between search time and solution quality. It is possible to explore several alternative solutions by defining a priority encoding to break ties in operation selection during scheduling [10]. A priority encoding is a permutation $\prod$ of the operations in the DFG. The relative position of operations in set $\prod$ is associated with their priority. For instance, if operations u and v compete for the same resource and u precedes v in the permutation, then operation u is selected to occupy the resource.

Different priority encodings result in solutions with possibly distinct costs. Therefore, the cost optimization can be supported by automatic monitoring costs of different explored solutions and by finally choosing the one with lowest cost. In the literature, several methods can be found to generate priority encodings (e.g. genetic algorithms, simulated annealing, etc.). To build solutions from priority encodings, we use the algorithm proposed by Aiken et al.[1], which was adapted to explore several alternative solutions [10].

Our approach consists of an *explorer* engine and a *constructor* engine, as shown in Figure 4. The explorer

restricts itself to generate priority encodings ∏ and analyze the corresponding solution cost. The constructor builds up a solution for each ∏ and returns its cost to the explorer.

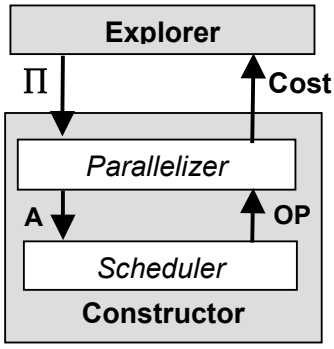

Figure 4: An overview of our approach.

The constructor consists of a *parallelizer* and a *scheduler*. The parallelizer creates a current state (s_k) where operations are to be scheduled. When all possible operations are scheduled within the current state, the parallelizer creastes a next state (s_{k+1}). In this way, the SMG is created step-by-step until there are no more operations to be scheduled. The parallelizer keeps a set of operations ready to be schedule on entry to each state, which are called *ready operations*. The set of ready operations associated with state s_k is denoted by A_k. The constrained resources require selection of operations of higher priority. The set A_k is implemented so as to maintain its elements ordered according to the priority encoding ∏.

The scheduler selects from the elements of A_k, the operations to be scheduled in state s_k, according to the priority ∏ and the available number of resources. It returns a set of scheduled operations in the current state, represented by OP_k. That is repeated until all the available resources are busy in the state s_k or until A_k is empty, indicating that all the operations of DFG were scheduled.

A great advantage is the separation of different functionalities in distincts engines. This allow further tuning, since one engine can be modified or replaced without the need to change the other. (For instance, an explorer based on genetic algorithms could be replaced by another based on simulated annealing; the constructor could be modified to include various kinds of pruning techniques). Another advantage is that the approach allows to trade search time against solution quality, since the number of explored solutions can be, in general, controlled by the parameters of the search method.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

4.1. Implementation

A prototype was developed in the frame of the so-called OASIS Project: "Modeling, Synthesis and Optimization of Architectures for Digital Systems." The target platform is a microcomputer PC under Linux Operating System. The adopted language is C++.

To solve the scheduling problem, Algoritm 1 was implemented.

```
ScheduleState (sk, Ak, a, ∏) {
    OPk = ∅;
    vi = SelectOperation (Ak, a, ∏);
    while (vi ≠ none) do {
        schedule vi in sk;
        OPk = OPk ∪ {vi};
        delete vi from Ak;
    }
    retorn (OPk);
}
Schedule(a, ∏) {
    λ = 0; create initial state s0 in the SMG;
    A0 = {vi| vo is the only predecessor of vi in the DFG};
    next = s0;
    while (next ≠ none) do{
        sk = next;
        OPk = ScheduleState (sk, Ak, a, ∏);
        λ = λ + 1;
        P = FindReadyOperations (OPk); P = P ∪ Ak;
        If (P ≠ ∅) {
            Create a new state sk+1 in the SMG;
            Ak+1 = P;
            next = sk+1;
        } else { next = none; }
    }
    retorn (λ);
}
```

Algorithm 1: Scheduling Algorithm.

Given a priority encoding ∏, Algorithm 1 creates a SMG out of a DFG under resource constraints **a**, by calling procedure **Schedule(a, ∏)**. The algorithm schedules a current state s_k by associating with it as many operations as can be accommodated within the available resources during a clock cycle. When no more operations can be scheduled in s_k, the next state s_{k+1} is created, which will become the current state in the next loop iteration. Therefore, operations are scheduled in successive states until there are no more operations to be scheduled. As a state corresponds to a clock cycle, the latency λ is incremented of one at each new scheduled state. The scheduling of the current state is performed by function **ScheduleState (s_k, A_k, a, ∏)**, which schedules operations in A_k, while there are ready operations or available resources. Such function returns the set of operations scheduled in the current state (OP_k). The selection of one operation is performed by function **SelectOperation(A_k,**

a, ∏), which returns na operation $v_i \in A_k$ with higher priority ∏ that satisfies the resource constraints **a**. After a state s_k is scheduled, new operations can become ready, as a result of the operations scheduled in that state (OP_k). Such operations are evaluated by function **FindReadyOperations (OP_k)**.

To solve the registers allocation problem, the left-edge algorithm were implemented, it is well-known and can be found in [4].

4.2. Experimental Results

For our experiments, two classical examples of the HLS literature were choosen: **fdct** [8] and **wdelf** [5]. Tables 1 and 2 summarize our results for scheduling. Observe that the values of λ obtained in our approach are similar or better than those obtained by other methods. To obtain our values, 100 solutions were explored and the solution with the best latency was taken. The time to build and explore the 100 solutions depends on the constraints and varies from 5 to 20s on a Pentium III, 450 MHz. The execution delay considered for the operations of ALUs is unitary and for the multiplications it is the same to two clock cycles.

Table 1: Latency for example fdct.

MUL	ALU	λ in [6] Optimum Solution	λ in ours approach	λ in [6] Genetics Algorit.	λ in [6] List Sched.
8	4	8	8	8	8
5	4	10	10	10	10
4	3	11	11	11	13
4	2	13	13	13	15
3	2	14	15	14	17
2	2	18	19	18	21
2	1	26	26	26	27
1	1	34	34	34	40

Table 2: Latency for example wdelf.

MUL	ALU	λ in [6] Optimum Solution	λ in ours approach	λ in [6] Genetics Algorit.	λ in [6] List Sched.
3	3	17	17	17	17
2	2	18	18	18	19
1	2	21	21	21	22
1	1	28	28	28	28

Tables 3 and 4 compare our results for register optimization with other methods. Our values for χ overestimate the amount of registers up to a maximum of 20% for the example fdct, (Table 3) and a maximum of 50% for the example wdelf (Table 4). These results point out the need of a more elaborate exploring scheme. The explorer should guide the search towards solutions with same latency, but with distinct number of registers. Such a refinement would probably mitigate (or hopefully avoid) register overestimation.

Table 3: Number of registers for fdct.

MUL	ALU	Our approach		Genetics Algorit. in [7]		List Scheduler in [7]	
		λ	χ	λ	χ	λ	χ
3	2	15	12	14	10	17	12
2	2	19	12	18	13	21	11

Table 4: Number of registers for wdelf.

MUL	ALU	Our approach		Symbolic in [8]		TASS in [2]	
		λ	χ	λ	χ	λ	χ
3	3	17	11	17	10	17	9
2	2	18	11	18	10	18	8
1	2	21	9	21	10	21	6
1	1	28	9	28	10	28	6

5. CONCLUSIONS AND FUTURE WORK

The results obtained are of a good quality, when compared with other methods reported in the literature. However, as expected, the optimal solutions is not found in all cases, since we are not using exact techniques, which could lead to an exhaustive search of solutions. We use polynomial algorithms based on lists ordered by heuristic criteria. Although we can not guarantee an optimal solution, our approach allows a trade-off between solution quality and acceptable runtimes.

In the future, we intend to improve the generation of priority encodings so as to use evolutionary techniques for obtaining the ordering in ∏. This will allow the generation of encodings that progressively lead to better solutions, so as to obtain convergence towards the optimal solution. We believe that a more elaborate exploring scheme can allows us to reach lower values of χ, avoiding the over-estimates in Tables 3 and 4.

6. REFERENCES

[1] AIKEN, A.; NICOLAU, A. "A Resource-Constrained Software Pipelining", IEEE Transactions on Parallel and Distributed Systems, vol. 6, nº 12, december of 1995.

[2] AMELLAL, Said; KAMINSKA, Bozena "Functional Synthesis of Digital Systems with TASS", IEEE Transactions on Computer-Aided Design, vol. 13, nº 5, pp. 537-552, may of 1994.

[3] CAMPOSANO, R. "Path-based Scheduling for Syntesis", IEEE Trans. On Computer-Aided Design, vol 10, nº 1, january of 1991.

[4] De MICHELI, Giovanni “Synthesis and Optimization of Digital Circuits”, Mc Graw-Hill, USA, 1994.

[5] DEWILDE et. al. “Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms”, in S.Y. Kung, H.J. Whitehouse and T. Kailath, *VLSI and Modern Signal Processing*. Prentice Hall, USA, 1985.

[6] HEIJLIGERS, M.J.M.; CLUITMANS, L.J.M. & Jess, J.A.G. “High-Level Synthesis Scheduling and Allocation using Genetic Algorithms”, Proceedings of the Asia and South Pacific Design Automation Conference, pp. 61-66, 1995.

[7] HEIJLIGERS, M.J.M. “The Application of Genetic Algorithms to High-Level Synthesis”, PhD. Thesis, Eindhoven University of Technology, The Netherlands, pp. 116, 1996.

[8] MALLON, D. J.; DENYER, P. B. “A New Approach to Pipeline Optimization”, Proc. EDAC’90m OO, 1990.

[9] RADIVOJEVIC, I.; BREWER, F. “A New Symbolic Technique for Control Dependent Scheduling”, IEEE Transactions on Computer-Aided Design, vol. 15, nº 1, p. 53 , 1996.

[10] SANTOS, Luiz C. V. dos “Exploiting instruction-level parallelism: a construtive approach”, Eindhoven University of Technology, PhD. Thesis, Eindhoven, 1998.