Phase Coupled Operation Assignment for VLIW Processors with Distributed Register Files

Marco Bekooij Philips Research
Prof. Holstlaan 4
Eindhoven, The Netherlands

Jochen Jess University of Technology
Den Dolech 2
Eindhoven, The Netherlands

Jef van Meerbergen
Philips Research
Prof. Holstlaan 4
Eindhoven, The Netherlands

1. ABSTRACT

The ever increasing complexity of signal processing applications and the desire to reduce the time to market demands efficient compilation techniques for programmable Digital Signal Processors (DSPs). Because the instruction sets of VLIW processors are more regular and orthogonal then the instruction sets of the traditional DSPs, they tend to be more compiler friendly. However in order to improve the maximal clock frequency, the power efficiency and the code density, several register files are used in the newer generation of VLIW processors instead of just one large register file. The use of several register files and partial connected networks leads, for example, to the problem that a result stored in a register file can not be accessed by all the functional units in the processor. This makes the assignment of operations to functional units a difficult task for the compiler.

This paper describes the constraint analysis based operation assignment techniques intended to deal with processors with distributed register files and partially connected networks. The assignment techniques have been implemented in our code generation tool FACTS [8]. This tool is intended for the generation of an operation assignment, a register binding and a schedule of folded loops that satisfy the specified timing constraints. Our approach is based on satisfaction of constraints which makes it different from the optimisation based operation assignment techniques [4] [2] [3] which are known from literature. The operation assignment technique is based on the modeling of the assignment search space in a conflict graph. Pruning of this conflict graph prevents decisions that inevitably lead to solutions that do not satisfy the timing constraints. If after pruning infeasibility is detected, backtracking of assignment decisions is performed. In order to obtain a tight coupling between the assignment phase and the schedule phase information is derived from the conflict graph which is used to prune the schedule search space and information from the schedule search space is incorporated in the conflict graph. Automatic insertion of copy operations for moving intermediate values from one register file to another register files is not supported. However the use of a shared global bus in the processor guarantees that at least one direct communication path from a producing functional unit to a consuming functional unit exists and therefore the use of copy operations is not necessary.

ISSS'01, October 1-3, 2001, Montréal, Québec, Canada.

2. INTRODUCTION

In this section a motivating operation assignment example is described. Also the target processor family for which our operation assignment techniques are intended is defined. At the end of this section it is described how the assignment techniques are incorporated in our code generation tool FACTS.

2.1 Motivating example

In Figure 1 a Data Flow Graph (DFG) and a data-path of a processor is shown, which will be used to illustrate some aspects that make operation assignment a difficult task for the compiler. The shared resources in the data-path of this processor are the functional units (FUs), the registers in the register files, the connections in the connection network and the input ports and output ports of these building blocks. These resources can be used in parallel and are controlled by VLIW-instructions. In order to make the example easy to understand, the number of resources in the processor, and the number of operations and data-dependencies in the DFG, are kept small.

Assignment of the operations in the DFG to the functional units of the target processor should be done in such a way that a communication channel exists. A communication channel is a path from the output port of the functional unit that computes the result to the input port of the functional unit that uses this result. This path passes the connection network and a register file. For the DFG and the architecture in this example, the constraint that there must exist a communication path from the producing FU to the consuming FU, has consequences for the assignment of the operations $n0$ and $n1$ to the functional units. For example operation $n1$ must be assigned to FU_2 if operation $n0$ is assigned to FU_2. In the case that the DFG is folded and both operations $n0$ and $n1$ are executed in the same cycle then operation $n0$ can only be assigned to FU_1 because operation $n1$ must be executed on FU_2.

The basic block to be assigned and scheduled is represented by a DFG, which is defined as follows:

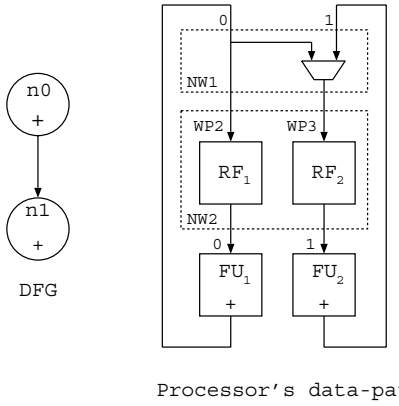

Figure 1: A example DFG and the data-path of the target VLIW-processor.

Definition 1: (Data Flow Graph). A data flow graph DFG is a triple $(V, E_d \cup E_s, w)$

- V is the set of vertices (operations),
- $E_d \subseteq V \times V$ is the set of data precedence edges,
- $E_s \subseteq V \times V$ is the set of sequence precedence edges, and
- $w : E_d \cup E_s \rightarrow Z$ is a function describing the timing delay (in clock cycles) associated with each precedence edge.

When an operation in the DFG is executed on an FU it consumes intermediate values via the input ports of this FU. In the case the input ports of the FU are not equivalent then the port via which the intermediate value must be consumed is specified. If the output ports of an FU are not equivalent the also the output port via which the intermediate value must be produced is specified.

The operation assignment problem that is addressed can be stated as follows:

Problem 1: Given a DFG and a port specification, find an assignment of operations to resources and intermediate values to register files in the target processor architecture such that a schedule can be constructed in which the specified global throughput and schedule length constraints are respected and, if required, loop-folding [5] is applied.

2.2 Processor template

The addressed operation assignment techniques are intended to be applied in a retargetable compiler for a family of DSP cores derived from the template shown in Figure 2.

The for operation assignment relevant characteristics of this template are:

- A distributed Register File (RF) architecture,
- Several potentially equivalent Functional Units (FU) that can execute an operation of a certain type,
- A partially connected Connection Network (CN) between the outputs of the unit clusters and the write ports of register files. This network may contain shared busses,
- An orthogonal instruction encoding.

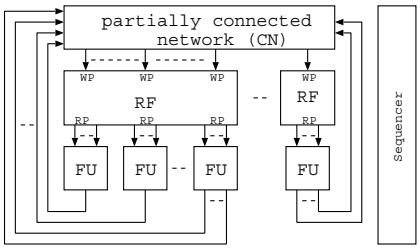

Figure 2: Target digital signal processor template

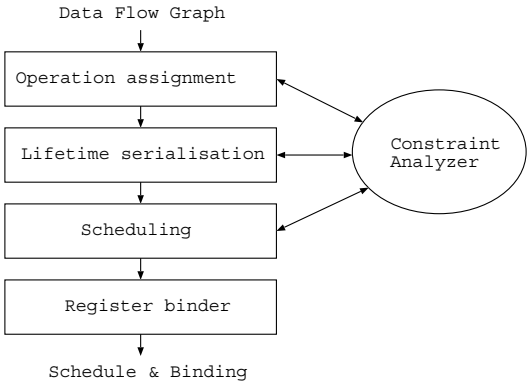

Figure 3: FACTS: Code generation phases

2.3 Code generation phases

The code generation techniques has been implemented in our research tool FACTS. Code generation takes place in an operation assignment phase, a register serialisation phase, a scheduling phase, and a register binding phase (see Figure 3). These code generation phases are re-targeted to a specific processor instance by means of a machine description file. During the operation assignment phase, operations are assigned to functional units. During the lifetime serialisation phase, operations are serialised in such a way that the register file capacity constraints are satisfied after scheduling [6]. During the scheduling phase the start times of operations are determined. After scheduling the register binder selects for every intermediate value a register in which it will be stored.

During the operation assignment phase it is for every operation in the data-flow graph decided, on which functional unit in the processor it will be executed. After an operation assignment decision is taken by the *operation assigner* (see Figure 4) the consequences of this decision are evaluated by the constraint analyzer. Back-tracking is performed in the case that the constraint analyzer detects infeasibility, where after another assignment decision is taken. In the case the constraint analyzer does not detect infeasibility the next operation is assigned till every operation is assigned to a functional unit.

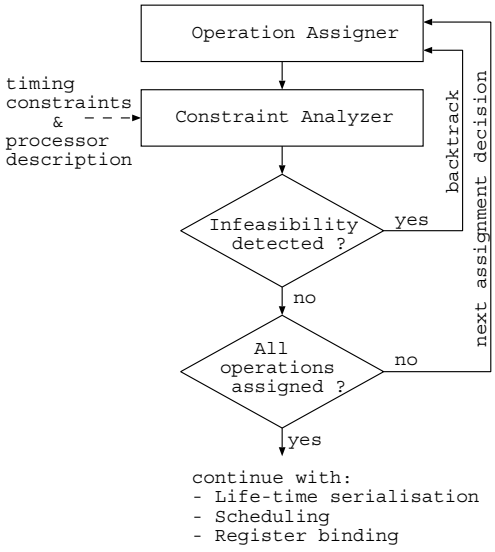

Figure 4: Operation assignment flow-diagram.

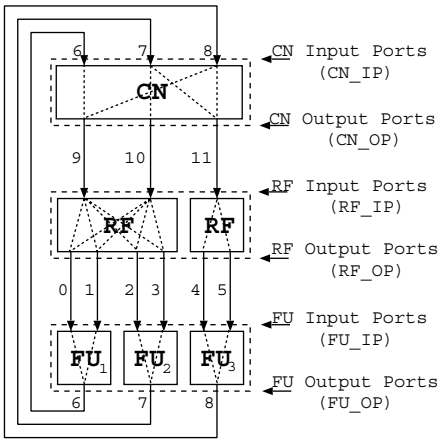

Figure 5: VLIW-Processor instance.

3. ASSIGNMENT SEARCH SPACE MODELING

Our operation assignment search space model is based on the notion that a processor architecture can be seen as a collection of networks (the dashed boxes in Figure 5). The output of a network in the processor is connected to the input of another network in the processor. The selection of a network input port determines the output ports through which potentially a result can be produced. The other way around is also true, that is, the selection of an output port through which a result must be produced determines through which ports the input data can be provided. These mutual dependencies between input ports and output ports can be modeled in a conflict graph. This conflict-graph is used to find an assignment of operations in a DFG to the functional units such that there exist a communication path, as specified by data dependency edges in the DFG, through the connection network and the register files in the target processor.

Figure 6 depicts a network in which there does not exist a connection between input port 0 and output port 3. This

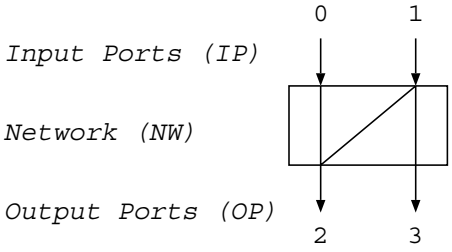

Figure 6: Network in which no connection exist between input port 0 and output port 3.

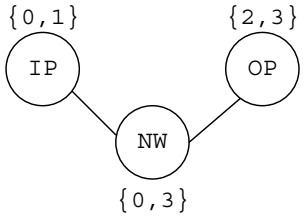

Figure 7: Conflict graph in which the same restrictions are expressed as in Figure 6.

has the consequence that if port 0 is chosen as input port then output port 3 is not reachable and vice versa. This restriction is modeled in the conflict graph shown in Figure 7. The IP and OP node in this graph are associated with respectively the input ports and the output ports of the network. The numbers in the sets next to the nodes indicate the ports that could be selected if there are no additional restrictions which is only the case if the network is fully connected. These numbers correspond also to the colors that potentially can be assigned to the nodes by the coloring algorithm. The third node in the graph, which is labeled with NW, enforces the restriction that the choice to color IP node with color 0 has as a consequence that OP node cannot be colored with color 3. All other combinations of colors can be assigned by a coloring algorithm. The restriction on the colors that can be assigned to the IP and OP node is enforced by the NW node in the following way. In the case the IP node is given color 1 then the edge between the IP and the NW node enforces that color 1 cannot be assigned to the NW node. The NW node must therefore be colored with 3. If the NW node is colored with 3 then the edge between the NW node and the OP node enforces that color 2 is the only color that can be assigned to the OP node. In a similar way the assignment of color 3 to the OP node has as consequence that color 0 cannot be assigned to the IP node. The NW nodes can be used in the same way to model in a conflict graph restrictions imposed by networks with an arbitrary interconnection pattern between a number of input ports and output ports. A NW node is needed for every combination of an input port and an output port for which there is no connection in the network. Therefore, the worst case number of needed NW nodes in the conflict graph per network equals $\#IP \times \#OP$ where #IP and #OP are respectively the number of input-, and output ports of the network.

A conflict graph can be used, as described above, to derive a valid assignment for the operations in the DFG and

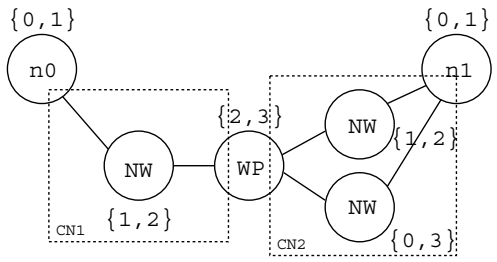

Figure 8: Conflict graph in which the communication paths in the architecture are modeled.

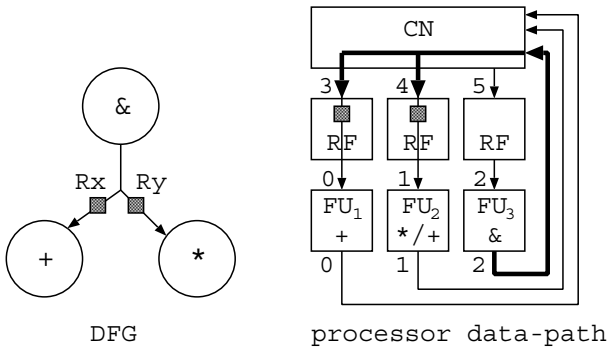

Figure 9: Multi-casting of an intermediate result to two register files

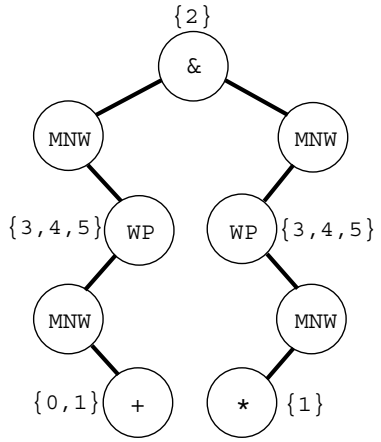

Figure 10: The conflict graph in which the communication paths from the producing operations to the consuming operations are modeled.

the processor shown in Figure 1. The nodes $n0$ and $n1$ in the conflict graph shown in Figure 8 correspond to operations $n0$ and $n1$ in the DFG. Every number in the depicted sets above these nodes correspond to one of the two output ports of network 2 (NW2) in Figure 1. Because both ports are connected to only one functional unit these numbers correspond also to a specific functional unit. The data produced by operation $n0$ should, according to the data edge in the DFG, be consumed by operation $n1$. A value produced by the functional unit on which $n0$ is executed should pass network 1 to a write port (WP) of a register file. In the conflict graph there is a node labeled with WP that corresponds to this write port. The write ports in the processor are numbered 2 and 3 which are also the numbers in the set above this WP node. The NW nodes in the conflict graph model that certain connections do not exist between the input ports and the output ports of NW1 or between the input ports and the output ports of network 2 (NW2). The existence of a valid coloring of this graph which only makes use of the colors in the sets, reveals that there exist an assignment which respects the constrains imposed by the interconnect. From the colors assigned by the graph coloring algorithm the assignment of the operations can be derived.

4. MULTI-CASTING

In Figure 9 a DFG is shown in which a value produced by an operation, is consumed by two operations. If these consuming operations are executed on functional units that do not have access to the same register file then the intermediate result must be stored in two register files. We assume that the target processor supports multi-casting that is the storage of an intermediate result that is produced by an functional unit in more than one register file in the same cycle. Multi-casting becomes a decision problem for the compiler if the number of register files in which an intermediate result must be stored depends on the assignment of the consuming operations. In that case the number of occupied register file write ports depends on the assignment of the consuming operations to the functional units. For example, if the consuming operations in Figure 9 are assigned to functional unit FU1 and FU2, multi-casting must be applied and two write ports are used while if both operations are assigned to functional unit FU2 the intermediate value must be stored once in the register file and only one write port is used. Whether multi-casting should be applied or not depends on the global timing constraints and the data-path of the processor.

Multi-casting is handled in our operation assignment technique as follows. In the conflict graph it is modeled that for every consumption of an intermediate result there must exist a connection between the output of the producing functional unit and the write port of the register file from which the intermediate value is consumed. The conflict graph for the DFG and architecture of Figure 9 is shown in Figure 10. The MNW nodes and the tick edges in this figure are used as short-hand notation and represent the necessary NW nodes with edges.

Care is taken that no conflicts are added between the write ports involved in multi-casting. This way the write port nodes in the conflict graph can get the same color which corresponds to the situation where both write port usage are assigned to the same write port in the processor. Only in the case that after pruning it is detected that different write ports must be used then the resource usage of the write ports is taken into account by timing constraint analysis.

5. SHARED GLOBAL BUS

In Figure 11a a DFG and a data path of a target processor is shown. In this processor a partial connected network is applied that connects the outputs of functional units to a write port of a register file. The DFG in this figure can not be executed on the target processor. The reason is that the multiply operation must be executed on FU2 and the

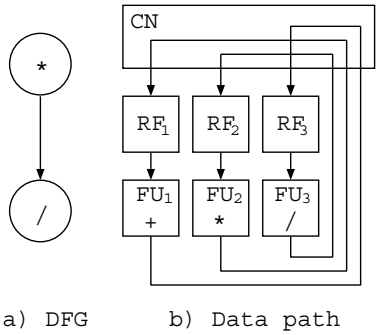

Figure 11: DFG (a) which can not be executed without an additional copy operation on the data path (b) of the target processor.

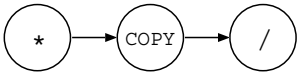

Figure 12: A DFG in which a copy operation is inserted such that it can be executed on the processor in Figure 11.

division operation on FU3. Therefore the result produced by FU2 must be stored in register file RF3 such that FU3 can read this result. However there is no connection in the network such that the value produced by FU2 can be stored in the register file RF3.

A solution is to modify the DFG by inserting an copy operation as shown in Figure 12. This copy operation is, for example, implemented as an addition with zero. It is used to copy the intermediate value from RF2 to RF3.

However the use of a copy operations was only possible because FU2 can execute operations that implement copying of an intermediate value without changing it. Also the insertion of the copy operation introduces latency and uses a functional unit during a cycle which can lead to violation of the global timing constraints.

An alternative for the use of copy operations is the use of a global shared bus as is shown in Figure 13. Via this bus the result produced by a functional unit can be stored in every register file. By applying multi-casting it is also possible to store the same intermediate value in several register files, if required.

Care should be taken that the global bus does not become the bottleneck that limits the computational performance of the processor. Therefore a network should be applied that contains an appropriate amount of other connections.

6. PRUNING AND PHASE-COUPLING

In order to prevent operation assignment decisions that inevitably lead to infeasibility, the colors in the sets of the nodes in the conflict graph are pruned. Pruning algorithms are used to remove the colors from the sets in the conflict graph, that given the colors that can be assigned to adjacent vertices, cannot be assigned to the vertex under consideration. A simple example of a pruning rule is that if only one color can be assigned to a vertex, then this color cannot be assigned to any adjacent vertex.

Pruning is also applied to establish a tight coupling between assignment and scheduling. If after pruning to a node only one color can be assigned, then the corresponding operation in the DFG can be executed by only one specific resource in the processor. If an operation can only be executed on one specific resource out of a set of equivalent resources in the processor's data-path then an additional resource usage is added in the internal DFG representation. An example of how the resource usage of one adder out of 4 adders is specialized to a specific adder in the data-path is shown in Figure 14. The dashed edges in this figure are sequence edges to enforce that the resource usage $+_A$ is scheduled in the same cycle as the addition. The additional resource usage is evaluated by the BSG constraint analysis technique [7] and potentially results in a reduction of the schedule search space.

In the case the constraint analyzer determines that two operations of the same type must be executed in the same cycle then an edge between the corresponding nodes in the conflict graph is added. This will guarantee that these nodes will be assigned a unique color which corresponds to the assignment of only one resource usage (e.g. an operation) to a resource (e.g. a functional unit). By adding these edges in the conflict graph, information from the schedule search space is modeled in the assignment search space.

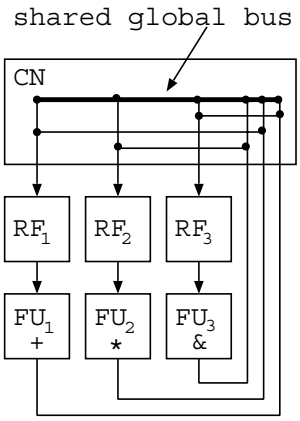

Figure 13: Processor with a shared global communication bus which guarantees the existence of a connections such that copy operations are not required.

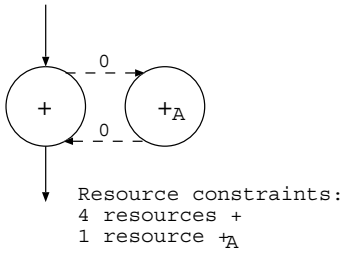

Figure 14: Resource usage specialization modeled in an internal DFG representation.

7. RESULTS

The template of the target architecture is shown in Figure 15 and the obtained results in Table 1. This table shows

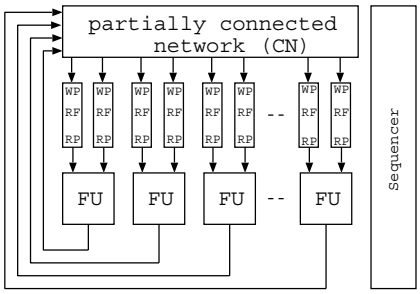

Figure 15: Template of the target VLIW-processor architecture.

the number of operations V_a and data edges E_d in the DFG, the lower bound (lb.) estimates of the Initiation Interval (II) and Latency (L) and the obtained II and L. The lower bound estimates assume a fully connected processor network. The last two columns contain the number of assignment decisions and backtracks. In experiment 1,2,3 the number of unit clusters was increased such that an II of one was obtained. Experiment 6 and 8 shows that a relatively large number of decisions is needed to obtain an operation assignment. In experiment 10,11,12,13 a significant number of connections in the connection network of the processor were removed which causes an increases in the number of backtracks to obtain a feasible result.

The results in Table 1 indicate that the use of a strongly distribute register file architecture hardly effect the throughput of folded loops. The throughput is only reduced in the case a significant number of connections are remove from the connection network.

The results were obtained by appling FACTS in the A|RT designer environment of Frontier Design [1]. In order to evaluate only the effect of the use of a partially connected network and a distributed register file architecture on the schedule quality the number of registers in the register files were not taken into account in these experiments.

To obtain a compilation time of less than 10 seconds on a 600 MHz Pentium processor for our benchmark set, the assignment conflict graph was not colored but only pruned after every assignment decision. Backtracking was performed in the cases that after pruning it was detected that not any color could be assigned to a certain node in the conflict graph.

The effect on the schedule search space is taken into account during assignment, but because the schedule search space is only pruned there is no guarantee that after assignment a schedule, that satisfies the timing constraints, indeed exists. However our experience is that often a schedule does exist. This was also the case for the examples in Table 1.

8. FUTURE WORK

In the near future the effect of the number of registers and the number of read and write ports of the register files on the operation assignment and schedule quality will be evaluated. Techniques to reduce the size of the conflict graph will be implemented in order to reduce the compilation time for DFGs with more than 100 operations and processors with more than 20 functional units.

no	Example	V_a	E_d	II lb.	II	L lb.	L	d.	b.
1	Fir	7	9	3	3	4	4	0	0
2	Fir	7	9	2	2	4	4	0	0
3	Fir	7	9	1	1	4	4	0	0
4	Sum	9	10	3	3	5	6	0	1
5	Biquad	15	19	7	7	8	8	0	0
6	Biquad	15	19	4	4	8	8	10	0
7	Mac-loop	25	31	9	9	10	10	0	0
8	Mac-loop	25	31	3	3	4	4	24	0
9	FFT	49	60	12	12	21	21	0	0
10	Fir	7	9	3	3	4	4	0	0
11	Biquad	15	19	7	10	8	13	0	30
12	Mac-loop	13	17	2	2	4	4	11	3
13	Mac-loop	13	17	1	1	3	3	5	2

Table 1: Operation assignment and scheduling results for inner-loops.

9. CONCLUSIONS

This paper describes how operation assignment is performed of potentially folded loops on VLIW processors with a distributed register file architecture and partially connected network under tight timing and resource constraints. Our method is based on the modeling of the assignment search space in a conflict graph. A tight coupling with the scheduling phase is obtained by the derivation of the usage of additional resources from a pruned conflict graph. Multi-casting of intermediate results can be modeled in the conflict graph and exploited by this operation assignment technique. The use of copy operations to move data values from one register file to another file is currently not supported but also not necessary in the case a global bus in the processor is applied.

10. REFERENCES

[1] *Frontier Design, http://www.frontierd.com.*

[2] S. Bashford and R. Leupers. Constraint driven code selection for fixed-point dsps. Proceedings Design Automation Conference.

[3] E. de Kock. *Video Signal Processor Mapping.* PhD thesis, Eindhoven University of Technology, 1995.

[4] J. Ellis. *Bulldog: A Compiler for VLIW Architectures.* ACM Doctor Dissertation Awards, MIT, Cambridge, 1986.

[5] G. Goossens. *Optimisation Techniques for Automated Synthesis of Application-Specific Signal-Processing Architectures.* PhD thesis, Katholieke Universiteit Leuven, 1989.

[6] B. Mesman, C. Alba-Pinto, and K. van Eijk. Efficient scheduling of dsp code on processors with distributed register files. San Jose, 1999. Int. Symp. on System Synthesis.

[7] A. Timmer. *From design space exploration to code generation.* PhD thesis, Eindhoven University of Technology, 1995.

[8] K. van Eijk, B. Mesman, C. Alba-Pinto, Q. Zhao, M. Bekooij, J. van Meerbergen, and J. Jess. Constraint analysis for code generation: Basic techniques and applications in facts. volume 5. ACM Transactions on Design Automation of Electronic Systems, 2000.