# Exploitation of component information in a RAM-based architectural synthesis system

Peter Marwedel, Birger Landwehr
Universität Dortmund,
Informatik XII
D-44221 Dortmund, Germany
e-mail: *marwedel@ls12.informatik.uni-dortmund.de*

**Abstract**

This paper describes how the TODOS microarchitecture synthesis system[1] uses information about available library components during the synthesis phases. TODOS stands for "TOp Down Synthesis". TODOS is an extension of the work described in Marwedel (1986). TODOS takes advantage of this information right from the beginning and contains an assignment algorithm considering more component-specific library details than other algorithms. Special care is taken about RAMs as library elements. Possible multiple concurrent accesses are considered in the scheduling and the assignment phases. Possibilities for scheduling reads and writes with common addresses in the same control step are exploited. The assignment algorithm simultaneously generates bindings to ALUs, immediate control fields and memory ports. The paper shows that some control steps do not influence the generated data path. Excluding these control steps from the assigment phase speeds up this phase. An even more important speedup is obtained by using special simplifying rules for the assignment problem at hand.

.

## 1 Introduction

Microarchitecture- or high-level-synthesis (HLS) is concerned with the mapping of behavioral descriptions onto register-transfer-level structural descriptions implementing that behavior. The resulting structure is intended to be implemented using components from a technology-specific library.

In contrast to many other HLS systems, our TODOS system favors RAMs over separate registers. The advantages RAMs include the following:

- RAM-based designs usually require a simpler interconnection scheme (see e.g. section 8).
- If layout is taken into account, RAM-based designs are even more efficient. The reason is that RAMs usually come with full-custom layout quality. Rauscher et al. (1992) mention the use of RAMs as an important reason for the fact that designs generated by TODOS compete favourably with manual designs.

[1]This is a reprint of a contribution for a book edited by G. Saucier. Copyright restrictions of the publisher (Chapman & Hall) apply. This research was partially supported by projects BMFT-NT 2850 6 ("SALLY") and ESPRIT-BRA 6855 ("LINK").

- Furthermore, RAMs are easier to test than dedicated registers, because they are usually very well-connected to the other components.
- An increasing number of cell libraries now contains multi-port memories. These multi-port memories provide the bandwidth that applications require. Hence, RAM-based architectures are becoming more and more interesting.

A good link between cell libraries and the HLS system is generally considered to be one of the important features that have to be implemented before HLS systems will be commercially successful. This includes a good link to available RAM generators.

For RAM-based synthesis, the way in which the characteristics of available RAMs are taken into account, is very crucial. If these characteristic are taken into account only during the final phases of HLS, there is a large danger of suboptimal designs. For example, the scheduling phase should generate control steps not containing "too many" references to variables. Otherwise, even sophisticated assignment algorithms cannot guarantee any constrained number of memory ports. Furthermore, the assignment of operations to operators and the assignment of variable references to memory ports should be done concurrently. Otherwise, there would be a risc of generating a suboptimal number of interconnections.

In this paper, we will describe how the different tasks have been integrated in our TODOS tool. This paper is structured as follows: Section 2 describes related work. Section 3 presents the information made available in the input to our TODOS algorithm. Section 4 explains our method for variable to storage binding. Section 5 focusses on scheduling and how information about RAMs is used during that phase. Allocation is briefly described in section 6. Section 7 contains the main contribution of this paper. In this section, we describe the assignment phase and the special techniques that we have developed for integrating operator and RAM port assignment. Results and summary are given in sections 8 and 9, respectively.

## 2 Related Work

For a general survey of approaches for architectural synthesis see McFarland (1990). In most approaches, register-based target architectures have been generated. The use of RAMs, despite its well-known advantages, has been described only in few papers.

Some authors (see e.g. Peng (1994)) just consider single-port RAMs. Balakrishnan et al. (1988) describe the assignment of variables to multi-port RAMs. In their algorithm, access requirements for variables are analysed and used in order to map a compatible subset of variables to a RAM. This process is repeated for remaining variables, if such variables exist. The mapping problem is modeled as an IP-problem. In Kim (1993), Kim and Liu map variables (or registers) to RAMs, too. In contrast to Balakrishnan, several RAMs can be considered concurrently. However, the authors assume that both the schedule and the assignment of operations to operators is given. Generating the assignment of operations to operators and of memory accesses to memory ports sequentially may result in non-optimal interconnection patterns. In Ahmad (1991), Ahmad and Chen present a post processor for simultaneous assignment of operators and memory ports. The assignment problem is modeled as an IP-problem. In this approach, the scheduling does not take information about available RAMs into account.

In all the cases mentioned above, bindings to RAMs and RAM-ports are generated for a fixed given schedule. It is easy to imagine that this potentially results in suboptimal solutions.

Moreover, most existing algorithms use separate phases for binding arithmetic operations to ALUs and for binding variable references to ports. Also, none of the approaches considers interconnections required for generating constants. Suboptimal interconnection schemes may be the result. In this paper, we will describe how the different tasks have been integrated in our TODOS tool.

## 3 Input to TODOS

The input to TODOS consists of the behavior of the system to be designed, the behavior of the library components, and additional user-defined bindings.

The description of the *system behavior* differs from that of other HLS algorithms only by features which are not relevant for the purpose of this paper. The interested reader is therefore requested to refer to other papers on the synthesis from our language MIMOLA (see Marwedel (1986)).

*Example:* As a simple running example of the specified system behavior, we will use the following two assignments:

```
a:= (b - 2 * i) +1;
i:= 0;
```

The description of *library components* contains the functions performed by these components, the codes required for mode control and information about the required wiring of the ports.

Unfortunately, the meaning of the term *port* in the sense of *memory ports* is not consistent with its use in VHDL. In order to access a memory location, a set of (VHDL-) ports (data, address, control and clock ports) is usually required. We will call such a set of VHDL-ports a *meta port* whenever confusion could occur.

For TODOS, we assume that control steps are free of subcycles and that therefore a single clock phase is sufficient. All ports have to remain allocated during each of the control steps. Due to that assumption, we distinguish between four different types of meta ports:

1. *Input meta ports*

   These can be used only to *write* data.

2. *Output meta ports*

   These can be used only to *read* data.

3. *Bidirectional meta ports*

   These can be used to *read* and *write* data, but only one of these operations can be performed in a control step. Since read and write operations are mutually exclusive, they can share the same data port.

4. *Shared i/o meta ports*

   These can be used to *read* and *write* data in the *same* control step.

   Due to the absence of time-multiplexing within control steps, shared i/o meta ports require separate data inputs and output ports. Their address input is not time-multiplexed within a control step. Shared i/o meta ports can be used to read and write during the same control step, provided both operations are for the same address.

*Example:* For the purposes of our running example, let us assume that the libary contains a three-port memory called `ThreePortRAM`. We assume that `ThreePortRAM` has one input (meta) port `p1`, one output (meta) port `p3`, and a shared i/o (meta) port `p2`.

Modules like `ThreePortRAM` denote library components. Such components can be *instantiated.* User-instantiated components will never be deleted by TODOS. Such components therefore form a *minimum set of components* which will always be present in the final generated structure.

Storage components *must* be instantiated by the user. The reason behind this requirement is that there are usually only few choices for RAMs. Choosing RAMs early in the design process results in more information about the library being available to the scheduler. This, in turn, results in a controlled number of accesses to storage components per control step. If required, designs can be repeated with different RAMs. Note that TODOS allows the user to instantiate several RAMs and that the approach of TODOS is not limited to the single RAM that will be used in the example.

A subset of the locations of instances of RAMs can be *designated as locations for user-defined variables.* This is an example of the *user-defined bindings* between hardware resources and the input behavior.

Now that we have described the input to TODOS, we are going to describe the synthesis process with emphasis on handling of RAMs.

## 4 Variable to storage binding

In order to guarantee that the final structure does not require more accesses to variables than possible with the given RAMs, TODOS performs the variable to storage binding for user-defined variables quite early.

*Example:* Assuming that `reg` has been declared as an instance of `ThreeportRAM` and that `reg` is the designated memory for variables, TODOS would transform the variables of our running example into:

```
reg[0]:= (reg[1] - 2 * reg[2]) + 1;  -- a => reg[0]; b => reg[1];
reg[2]:= 0;                          -- i => reg[2]
```

## 5 Scheduling

The idea underlying scheduling in TODOS is that an even distribution of operations over control steps like the one generated by force-directed scheduling (Paulin (1987)) is not a primary design goal. The primary goal is to create the fastest possible design under a given set of design constraints. Therefore, it is important not to generate control steps for which the constraints are not met. The constraints should be made as tight as possible. Stok (1991) comes to the same conclusion:

> The best way to seems to develop algorithms, that can deal with both strong hardware bounds and strong time constraints. They can produce good results taking advantage of the reduction in search space provided by these hard bounds.

This requirement is fulfilled by the TODOS ASAP scheduler, which relies on strong resource constraints. If the assignment of an operation to the current control step would violate the design constaints, its assignment is deferred. In the context of this paper, we just consider checking (meta)

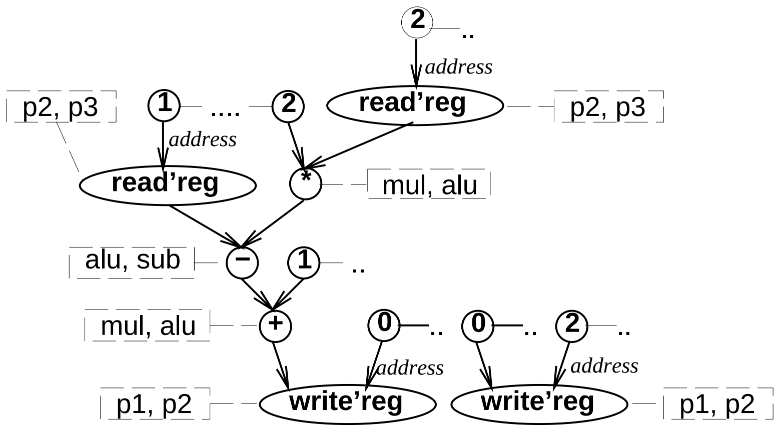

Figure 1: Internal representation of the running example

port constraints for RAMs. This check is complicated by the fact that shared address meta ports can be assigned to two operations in the same control step if their address is identical.

*Example:* For the running example, TODOS would find that all memory references can be assigned to the same control step, because `reg` has enough meta ports. As a result, TODOS would assign all operations to the same control step. If we had put a tighter constraint on the number of ports, more control steps had been generated.

This consideration of library information during scheduling is not possible with any HLS system we are aware of. In fact, for available systems, the number of variable references per control step is even unbounded.

## 6 Allocation

Next to scheduling, TODOS performs allocation of arithmetic components (ALUs). Automatic allocation of ALUs in TODOS is based upon the IP-formulation described in detail in Marwedel (1990).

*Example:* For the running example, we need to implement `+`, `-`, and `*` operations in the same control step. Given a library with types `aluType` (able to implement all three operations), `subType` (able to implement `-`) and `mulType` (able to implement `*`, `+` and `DIV`), TODOS could allocate, for example, one instance of each of the component types. Let us assume, they will be called `alu`, `sub`, and `mul`, respectively.

## 7 Assignment

### 7.1 Problem description

The assignment phase in TODOS is special in that it simultaneously generates bindings for ALUs, immediate control fields and RAM meta ports. In this context, the term *immediate control field* denotes a set of controller output bits used to generate constants which are present in the behavioral description and address constants generated during variable to storage binding.

*Example:* Consider our running example. In the internal representation of this example, all operations, including reads and writes, are made explicit (see fig. 1). This figure just describes the *data flow* within a single control step and in the general case, there would be one such figure for each control step. In this figure, operations are shown in bold. For reads and writes, the address is used as one of the arguments. For each of the operations, there is a set of *candidate hardware resources*. In fig. 1 candidate hardware resources are shown with thin lines. For reads and writes we have candidate meta ports, for `+,-` and `*` we have `alu, sub` and `mul` as candidate resources and for all the constants, we have candidate immediate control fields (all fields of the width of the constant, not shown in fig. 1). Hardware resources are represented by dashed lines, if they are not yet assigned to operations and by non-dashed lines otherwise.

The question is: which assignment of resources to operations minimizes interconnect? The assignment is not immediately obvious (at least not to the computer) and several assignments could be tried.

Simultaneous assignment for (meta) ports, ALUs and immediate fields is essential, because the generation of these assignments in sequence would be unable to optimize interconnect. If, for example, we would generate the bindings for ALUs first, we would have no information about the (meta) ports used to read the arguments and to store the results. Hence, we do not partition the assignment phase into subphases for ALU assignment, port assignment and interconnect assignment.

## 7.2 General assignment algorithm

The general assignment algorithm is based upon the observation that –except for the first few control steps– new control steps require only a small number of new interconnections. Therefore, we are using the following approach: We compute a lower bound on the number of missing interconnections. Starting with that lower bound, we execute a branch and bound procedure trying to assign resources to each of the operations, counting the number of required new interconnections. If the number of new connections exceeds the current lower bound, the assignment procedure is aborted and restarted with an increased lower bound. This process terminates as long as the scheduler has not ignored resource constraints.

Two special features are implemented:

- Commutative operations are exploited
- Common subexpressions can be computed either once or several times, depending on what fits resource constraints and required interconnections better. This feature is not implemented in any other HLS system.

## 7.3 Simplification rules

### 7.3.1 Removal of redundant control steps

The TODOS assignment algorithm simultaneously generates bindings which are normally generated in sequential order. As a consequence, the complexity is very high. In fact, in the worst case, the running time of the assignment algorithm is an exponential function of the number of operations. In order to solve practical cases in reasonable time, control steps are considered step after step. Motivated by the fact that control steps with a large number of operations have the largest impact on the resulting data path, TODOS *sorts* control steps with respect to their number of operations. Assignment then proceeds from the "largest" control step towards the "smallest" control step.

In order to reduce the running time as well as storage requirements, one simplification is used: many of the control steps are equivalent in the sense that they will generate the same data path.

*Example:* All control steps containing just transfers between two storage units are data-path equivalent, regardless of source and destination addresses. For TODOS, the controller is responsible for generating these addresses.

It is sufficient to consider only one representative for each set of equivalent control steps. This idea is implemented in TODOS as follows:

*Definition:* Control steps $i$ and $j$ are *data-path equivalent* $\Longleftrightarrow$ their flow graphs –except for the value of constants– are identical.

This definition is conservative in the sense that it guarantees that TODOS generates the same data path for both control steps. The definition could, however, be extended to reduce the set of control steps that have to be considered further.

The actual procedure for control step ordering uses a list `csl` of control steps, sorted by the number of operations. Each newly generated control step is checked against all control steps with the same number of operations and ignored if data-path equivalent to an existing control step.

### 7.3.2 Exploitation of resource set cardinalities

The running time of the assignment algorithm remains high, even if we consider only single control steps at a time. This is caused by the fact that in practical cases we found up to 80 operations per control step (this includes reads, writes, concatenations, constants and arithmetic operations). The assignment problem becomes seemingly complex especially if RAMs with several (meta) ports are used. Analysing practical cases, we found that the running time could be reduced by orders of magnitude, if the special structure of the assignment algorithm at hand is exploited using a few simplification rules. With these rules, the running time becomes acceptable for all the cases that we have studied so far.

*Definition:* Operation $o$ is said to be in relation $\approx$ with operation $o'$ $\Longleftrightarrow$ there exists a resource $r$ which is able to execute both $o$ and $o'$.

*Definition:* Relation $\approx^*$ is the transitive closure of relation $\approx$.

These relations are used in the following procedure which is iterated until no further simplification is possible:

```
FOR ALL operations $op$ of the current control step DO
  BEGIN
    $O := \{o|o \approx* \ op\}$;
    $R := \{r|r$ can execute at least one operation in $O\}$;
    IF $|R| = |O|$ AND $\exists r' \in R : r'$ can only perform operation $o'$
      THEN assign $r'$ to $o'$
      ELSE IF $|R| \ < |O|$ THEN --more operations than resources; shared ports required
        IF NumberOfSharedReads= $|R|$ -- all shared ports required
          THEN reduce resource candidates of shared READS and WRITES to shared ports.
    IF $|$resource candidates for $op| = 1$ THEN assign resource to $op$.
  END;
```

*Example:* For the running example, the steps of the algorithm would be as follows:

- Let *op* denote the second `write`-operation. The set $O$ of operations will then contain all `reads` and `writes`, because the p2 is able to perform both operations, implying that the assignment of these operations is not independent. The corresponding set $R$ contains the three meta ports.

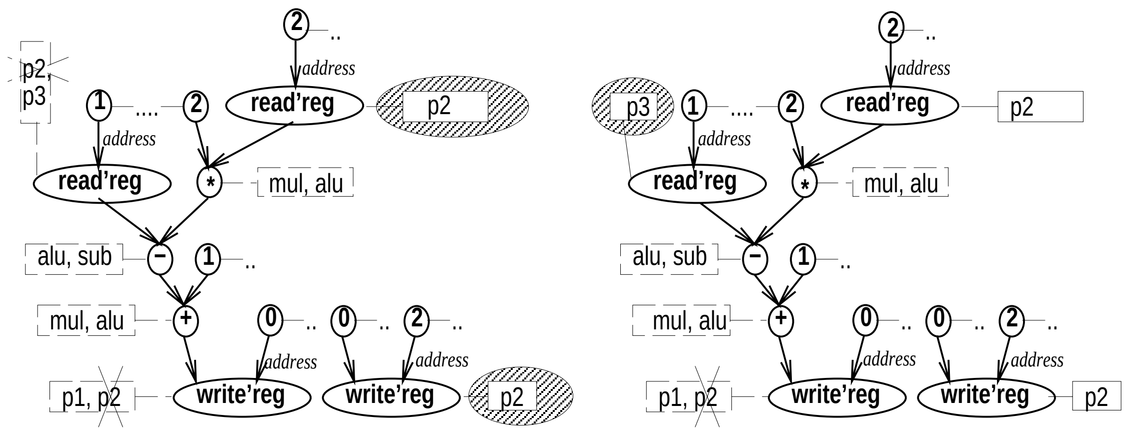

Figure 2: Implied assignments

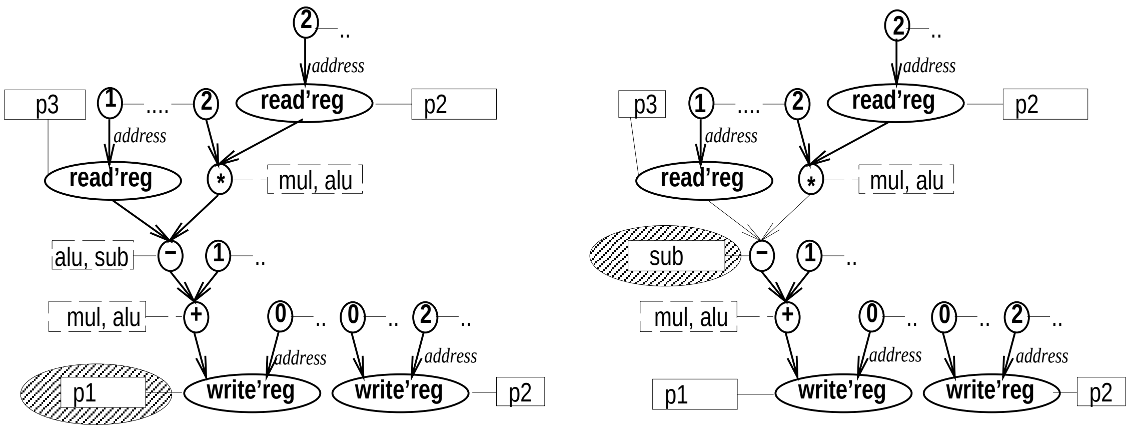

Figure 3: More implied assignments

Since $|R| < |O|$, some of the reads and writes must have a common address (otherwise the scheduler is faulty). Since exactly one read and one write have a common address (2 in this case) and since there is exactly one shared port, reading and writing of location 2 will be assigned to the shared port (see highlighted areas in fig. 2 (left)).

- The next operation considered will the `read'reg(1)`. Only a single resource (`p3`) is left for this operation and it is assigned (see highlighted area in fig. 2 (right)).
- The same arguments hold for the first write operation and it is bound to `p1` (see highlighted area in fig. 3 (left)).
- Next, operation `-` could be considered. `-` is in relation $\approx^*$ with the other operations and cannot be assigned independently. $O$ will be set to `+, -, *`. $R$ will contain `mul, alu, sub`. We will have $|R| = |O|$. Hence, all resources within $R$ are required. Since `sub` is only able to perform `-`, it will be assigned (see highlighted area in fig. 3 (right)). Note that we have concluded, that we cannot use the subtraction mode of `alu`. This decision would not have been possible with a covering approach.
- Finally, resource assignments for operations `+` and `*` will be made such that interconnections between resources will be minimized, taking predefined connections into account if such con-

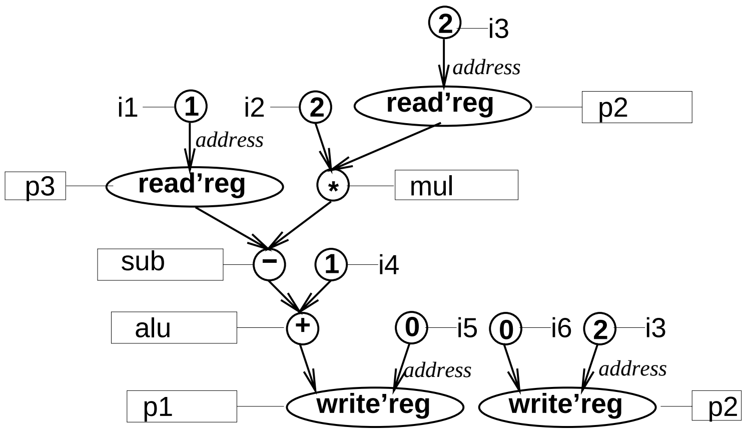

Figure 4: Final assignments

nections exist. Concurrently, the same will be done for the immediate fields `i1` to `i6`. Fig. 4 contains possible assignments.

## 8 Results

Table 1 shows that the PDP-8, if synthesized with TODOS, actually contains a reduced number of input ports if a RAM-based architectural style is chosen. The RAM based design was automatically generated with TODOS. Due to the limited support of register-based architectures in TODOS, the register-based architecture was generated after manually binding variables to registers.

Table 1: Comparison between register- and RAM-based structures generated by TODOS

| *Example* | *Input ports* | | *input ports ∗ bitwidth* | |
|---|---|---|---|---|
| | *register-based* | *RAM-based* | *register-based* | *RAM-based* |
| PDP-8 | 58 | 51 | 796 | 603 |

Table 2 shows that 4 out of 5 examples contain data-path redundant control steps. From the table, it can be seen that large examples usually contain a higher percentage of data-path redundant control steps. Hence, this type of redundancy cannot be exploited for the usual small HLS benchmarks. It is, however, important for real applications. `Diffeq` is the standard differential equation solver, `Ref` and `page68` are available from the authors. `PDP-8` and `mergesort` are standard algorithms although we did not use the standard HLSW benchmark sources.

Fig. 5 shows how different constraints for the number of memory ports can be used to generate the fastest design under those constraints. A single RAM with multiple shared i/o (meta) ports has been used. The left part of fig. 5 shows the mergesort example. The right part shows results for the elliptical wave filter example (see e.g. Tsai (1992) for a flow graph). It is important to analyze the solutions a) and b). If the selected library component has 5 (meta) ports, TODOS immediately generates a solution with 10 control steps (solution a) ). Time-constrained scheduling with post-synthesis RAM generation could end up with solution b), which is as fast as solution a) but more expensive. The results cannot be directly compared with other synthesis systems because TODOS

Table 2: Effect of eliminating data-path redundant control steps

| *Example* | *Total number of steps* | *Redundant steps (absolute)* | *Redundant steps (in % )* |
|---|---|---|---|
| Diffeq | 1-12 (several constraints) | 0 (for every set of constraints) | 0 % |
| Ref | 60 | 40 | 66.7 % |
| page68 | 86 | 63 | 73.2 % |
| PDP-8 | 101 | 47 | 46.5 % |
| mergesort | 161 | 134 | 83.2 % |

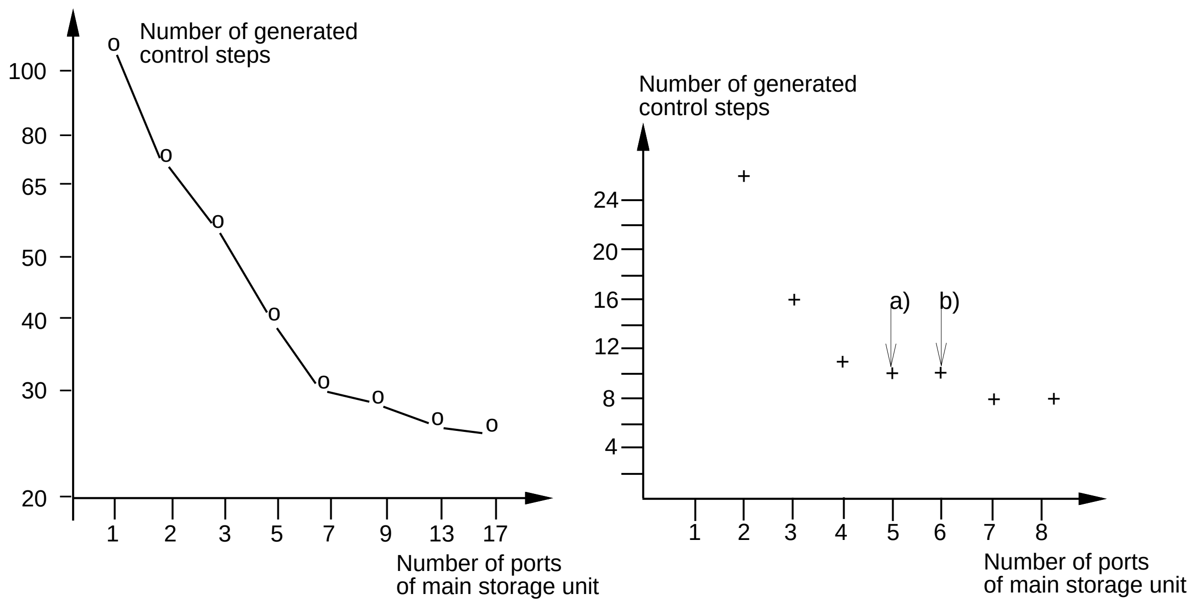

Figure 5: Resource constrained scheduling for mergesort (left) and the elliptical wave filter (right)

unfortunately does not support multi-cycle operations and the elliptical wave filter example has been partially prescheduled to a minimum of 8 control steps. Nevertheless, the essential point is clear: scheduling should take (meta) port constraints into account.

## 9 Summary

This paper describes various aspects of handling RAM library components in the TODOS high-level synthesis system. The contributions of this paper are in the following areas:

- TODOS uses resource constrained scheduling taking characteristics available RAMs into account.
- The assignment phase in TODOS integrates operator-, port-, and immediate field assignments.
- The paper introduces the notion of data-path redundant control steps. Redundant control steps are eliminated in order to reduce the running time of the assignment algorithm.
- Furthermore, information about how to reduce the complexity of memory port assignment is given. The important case of *shared* I/O-ports is considered.

## References

[1] I. Ahmad and C.Y. Chen. Post-processor for data path synthesis using multiport memories. *Int. Conf. on Computer-Aided Design (ICCAD)*, pages 276–9, 1991.

[2] A.K. Agrawala and T.G. Rauscher. *Foundations of Microprogramming.* Acadamic Press, New York, 1976.

[3] M. Balakrishnan, A.K. Majumdar, D.K. Banerji, J.G. Linders, and J.C. Majithia. Allocation of multiport memories in data path synthesis. *IEEE Trans. on Computer-Aided Design*, pages 536–40, 1988.

[4] N. Hendrich, J. Lohse, and R. Rauscher. Prototyping of microprogrammed VLSI-circuits with MIMOLA and SOLO-1400. *EUROMICRO*, 1992.

[5] T. Kim and C.L. Liu. Utilization of memories in data path synthesis. *30th Design Automation Conference*, pages 298–302, 1993.

[6] P. Marwedel. A new synthesis algorithm for the MIMOLA software system. *23rd Design Automation Conf.*, pages 271–7, 1986.

[7] P. Marwedel. Matching system and component behavior in MIMOLA synthesis tools. *Proc. 1st EDAC*, pages 146–56, 1990.

[8] M.C. McFarland, A.C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proc. of the IEEE, Vol. 78*, pages 301–18, 1990.

[9] P.G. Paulin and J.P. Knight. Force-directed scheduling in automatic data path synthesis. *24th Design Automation Conference*, 1987.

[10] Z. Peng and K. Kuchinski. Automated transformation of algorithms into register-transfer level implementations. *IEEE Trans. on Computer-Aided Design*, pages 150–66, 1994.

[11] L. Stok. *Architectural Synthesis and Optimization of Digital Systems.* PhD thesis, Technische Universiteit Eindhoven, 1991.

[12] F.-S. Tsai and Y-C. Hsu. STAR: An automatic data path allocator. *IEEE Trans. on Computer-Aided Design*, pages 1053–64, 1992.